SpectralRadar SDK

5.2

Generated by Doxygen 1.8.11

Contents

1 Spectral Radar SDK

1.1 SpectralRadar SDK License

By using the Thorlabs SpectralRadar SDK you agree to the terms and conditions detailed in the license agreement provided here: THORLABS SpectralRadar SDK License Agreement (PDF reader required). If this link does not work, you will also find this license agreement in Start Menu -> All Programs -> Thorlabs -> SpectralRadar-SDK.

1.2 Introduction

This document gives an introduction into using the ANSI C Spectral Radar SDK and demonstrates the use of the most important functions.

1.2.1 Overview

The ANSI C Spectral Radar SDK follows an object-oriented approach. All objects are represented by pointers where appropriate typedefs are provided for convenience. The defined types are called Handles and are used as return values when created and are passed as value when used. All functionality has been created with full LabVIEW compatibility in mind and it should be possible to use the SDK with most other programming languages as well. The most important handles are given in the following sections.

1.2.2 Data Handle (DataHandle, ColoredDataHandle, ComplexDataHandle, RawDataHandle)

Data acquired and used by the SDK is provided via data objects. A data object can contain

- floating point data (via DataHandle)
- complex floating point data (via ComplexDataHandle)
- ARGB32 colored data (via ColoredDataHandle)
- unprocessed RAW data (via RawDataHandle) The data objects store all information belonging to them, such as pixel data, spacing between pixels, comments attached to their data, etc. Data objects are automatically resized if necessary and can contain 1-, 2- or 3-dimensional data. The dimensionality can be read by getDataPropertyInt(), etc. Direct access to their memory is possible via getDataPtr(), etc. Data properties can be read via getDataPropertyInt(), getDataPropertyFloat(), etc. These include sizes along their first, second and third axis, physical spacing between pixels, their total range, etc.

1.2.3 OCTDeviceHandle

A handle specifying the OCT device that is used. In most cases the OCTDeviceHandle is obtained using the initDevice() function and needs to be closed after using by closeDevice(). The complete device will be initialized, the SLD will be switched on and all start-up dependent calibration will be performed. All hardware and hardware dependend actions require the OCTDeviceHandle to be passed. These include for example

- starting and stopping a measurement (startMeasurement() and stopMeasurement())
- getting properties of the device (getDevicePropertyInt() and getDevicePropertyFloat())

1.2.4 ProcessingHandle

The numerics and processing routines required in order to create A-scans, B-scans and volumes out of directly measured spectra can be accessed via the ProcessingHandle. When the ProcessingHandle is created, all required temporary memory and routines are initialized and prepared and several threads are started. In most cases the ideal way to create a processing handle is to use createProcessingForDevice() which creates optimized processing algorithms for the OCTDeviceHandle specified. If no device is available or the processing routines are to be tweaked manually createProcessing() must be used. When all required processing is done, clearProcessing() must be used to stop all processing threads and free all temporary memory. All functions whose output is dependent on the processing routines used have a ProcessingHandle parameter. These include for example

- The setProcessingParameterInt() and setProcessingFlag() functions for setting parameters that are used for processing
- The executeProcessing() function for triggering the processing of raw data

1.2.5 ProbeHandle

The probe is the hardware used for scanning the sample, usually with help of galvanometric scanners. The object referenced by ProbeHandle is responsible for creating scan patterns and holds all information and settings of the probe attached to the device. It needs to be calibrated to map suitable output voltage (for analog galvo drivers) or digital values (for digital galvo drivers) to scanning angles, inches or milimeters. In most cases this calibration data is provided by ∗.ini files and the probe is initialized by initProbe() where the probe configuration file name needs be specified as a string parameter. Probes calibrated at Thorlabs will usually come with a factory-made probe configuration file which follows the nomenclature Probe + Objective Name.ini, e.g. "ProbeLSM03.ini"

If the probe is to be hardcoded into the software one can also provide an empty string as parameter and provide the configuration manually using the setProbeParameterInt() and setProbeParameterFloat() functions. When the Probe object is no longer needed, closeProbe() must be called to free temporary memory. All actions that depend on the probe configuration require a ProbeHandle to be specified, such as:

- move galvo scanner to a specific position (moveScanner()).
- create a scan pattern (createBScanPattern()), see also ScanPatternHandle.
- set calibration parameters for a specific probe (setProbeParameterFloat() and setProbeParameterInt())

1.2.6 ScanPatternHandle

A scan pattern is used to specifiy the points on the probe to scan during data acquisition, and its information is accessible via the ScanPatternHandle. A dedicated function can be used to create a specific scan pattern, such as createBScanPattern() for a simple B-scan or createVolumePattern() for a simple volume scan. When the scan pattern is no longer needed its ressources can be freed using clearScanPattern(). The ScanPatternHandle needs to be specified to all functions that need information on the resulting scan. For example:

- creating a pattern (createBScanPattern(), createVolumePattern(), etc.)
- starting a measurement (startMeasurement())

1.2.7 Other Handles

Other Handles that are used in the Spectral Radar SDK are

- DopplerProcessingHandle: Handle to Doppler processing routines that can be used to transform complex data to Doppler phase and amplitude signals.
- SettingsHandle: Handle to an INI file that can be read and written to without explicitly taking care of parsing the file.
- ColoringHandle: Handle to processing routines that can map floating point data to color data. In general this will 32 bit color data, such as RGBA or BGRA.

1.3 First Steps

The following section describes first steps that are needed to acquire data with the Spectral Radar SDK.

1.3.1 Initializing The Device

The easiest way to initialize the device is to use the initDevice() function. It returns an appropate OCTDeviceHandle that can be used to identifiy the device:

```
OCTDeviceHandle Dev = initDevice();
// Acquire data, processing, direct hardware access...
closeDevice(Dev);
```

1.3.2 Creating Processing Routines

In most cases raw data acquired by the OCT device needs to be transformed usings a Fast Fourier transform and other pre- and postprocessing algorithms. To get a ProcessingHandle on these algorithms the most convenient way is to use the createProcessingForDevice() functionality which requires a valid OCTDeviceHandle:

```
// ...
ProcessingHandle Proc = createProcessingForDevice(Dev);
// acquire data and perform processing
clearProcessing(Proc);
// ...
```

1.3.3 Creating A Scan Pattern

In order to scan a sample and acquire B-scan OCT data one needs to specifiy a scan pattern that describes at which point to acquire data. To get the data of a simple B-Scan on can simply use createBScanPattern():

```
// ...
ProbeHandle Probe = initProbe(Dev, "Probe");
ScanPatternHandle Pattern = createBScanPattern(Probe, 2.0, 512); // get
       B-scans with 2.0mm scanning range and 512 A-scans per B-scan
// acquire data, ...
clearScanPattern(Pattern);
closeProbe(Probe);
// ...
```

1.3.4 Acquisition

The most convenient and fast way to acquire data is to acquire data asynchronously. For this one starts a measurement using startMeasurement() and retrieves the latest available getRawData(). The memory needed to store the data needs to be allocated first:

```
int i;


RawDataHandle Raw = createRawData();
DataHandle BScan = createData();
startMeasurement(Dev, Pattern, Acquisition_ASyncContinuous);

for(i=0; i<1000; ++i) // get 1000 B-scans
{
getRawData(Raw);
setProcessedDataOutput(Proc, BScan);
executeProcessing(Proc, Raw);
// data is now in BScan...
// do something with the data...
}

stopMeasurement(Dev);
clearData(BScan);
clearRawData(Raw);
```

1.4 Error Handling

Error handling is done by calling the function getError(). The function will return an ErrorCode and if the result is not NoError an error string will be provided giving details about the problem.

```
#define ERROR_STRLEN 1024;
//...
char error[ERROR_STRLEN];
OCTDeviceHandle Dev = initDevice();
if(!getError(error, ERROR_STRLEN)) // check whether the previous calls to SDK functions caused an
       error
{
printf("An error occurred: %s", error);
}
// ...
```

2 Module Index

2.1 Modules

Here is a list of all modules:

3 Data Structure Index

3.1 Data Structures

Here are the data structures with brief descriptions:

4 File Index

4.1 File List

Here is a list of all documented files with brief descriptions:

5 Module Documentation

5.1 Speckle Variance Contrast Processing

Typedefs

- typedef struct C_SpeckleVariance ∗ SpeckleVarianceHandle

  Handle used for SpeckleVariance processing.

5.1.1 Detailed Description

5.1.2 Typedef Documentation

5.1.2.1 SpeckleVarianceHandle

Handle used for SpeckleVariance processing.

5.2 Error Handling

Error handling.

Enumerations

- enum ErrorCode {
  NoError = 0x0000,
  Error = 0xE000 }

  This enum is used to describe errors that occur when operating an OCT device.
- enum LogOutputType {
  Standard,
  File,
  None }

  Specifies where to write text output by the SDK.

Functions

- SPECTRALRADAR_API ErrorCode isError (void)

  Returns error code. The error flag will not be cleared; a following call to getError thus provides detailed error information.
- SPECTRALRADAR_API ErrorCode getError (char ∗Message, int StringSize)

  Returns an error code and a message if an error occurred. The error flag will be cleared.
- SPECTRALRADAR_API void setLog (LogOutputType Type, const char ∗Filename)

  Specifies where to write text output by the SDK. The respective text output might help to debug applications or identify errors and faults.

5.2.1 Detailed Description

Error handling.

5.2.2 Enumeration Type Documentation

5.2.2.1 enum **ErrorCode**

This enum is used to describe errors that occur when operating an OCT device.

Warning

Error codes and error description texts are subject to change in future releases.

Enumerator

NoError No error occurred. This entry can be cast to FALSE.

Error Error occurred. This entry can be cast to TRUE.

5.2.2.2 enum LogOutputType

Specifies where to write text output by the SDK.

Enumerator

- ***Standard*** Write to standard output.
- ***File*** Write to text file.
- ***None*** Do not write output.

5.2.3 Function Documentation

5.2.3.1 ErrorCode getError (char ∗ *Message*, int *StringSize*)

Returns an error code and a message if an error occurred. The error flag will be cleared.

Returns

The error code (no error can be casted to FALSE, error can be casted to TRUE).

See also

ErrorCode.

This function is the ultimate criterium to establish if an error occurred or not. Under certain circumstances Spectral↩ Radar might log text lines that look like errors, but are no necessarily so. This is because the library has been conceived for very general settings, across a wide variety of hardware configurations, and messages might be generated to document a particular execution context.

Parameters

out	*Message*	Error message describing the error.
in	*StringSize*	Size of the string that was given to Message.

5.2.3.2 ErrorCode isError (void)

Returns error code. The error flag will not be cleared; a following call to getError thus provides detailed error information.

5.2.3.3 setLog (LogOutputType *Type*, const char ∗ *Filename*)

Specifies where to write text output by the SDK. The respective text output might help to debug applications or identify errors and faults.

Parameters

in	*Type*	Location where to write text output.
out	*Filename*	Full path and filename where to write output, if Type is set to File.

5.3 Data Access

Functions for accessing the information stored in data objects.

Data Structures

- struct ComplexFloat

  A standard complex data type that is used to access complex data.

Typedefs

- typedef struct C_RawData ∗ RawDataHandle

  Handle to an object holding the unprocessed raw data.
- typedef struct C_Data ∗ DataHandle

  Handle to an object holding 1-, 2- or 3-dimensional floating point data.
- typedef struct C_ColoredData ∗ ColoredDataHandle

  Handle to an object holding 1-, 2- or 3-dimensional colored data.
- typedef struct C_ComplexData ∗ ComplexDataHandle

  Handle to an object holding complex 1-, 2- or 3-dimensional complex floating point data.
- typedef struct C_ImageFieldCorrection ∗ ImageFieldHandle

  Handle to the image field description.
- typedef struct C_FileHandling ∗ OCTFileHandle

  Handle to the OCT file class.

Enumerations

- enum RawDataPropertyInt {
  RawData_Size1,
  RawData_Size2,
  RawData_Size3,
  RawData_NumberOfElements,
  RawData_SizeInBytes,
  RawData_BytesPerElement,
  RawData_LostFrames }

  Integer properties of raw data (RawDataHandle) that can be retrieved with the function getRawDataPropertyInt.
- enum DataPropertyInt {
  Data_Dimensions,
  Data_Size1,
  Data_Size2,
  Data_Size3,
  Data_NumberOfElements,
  Data_SizeInBytes,
  Data_BytesPerElement }

  Integer properties of data (DataHandle) that can be retrieved with the function getDataPropertyInt.
- enum DataPropertyFloat {
  Data_Spacing1,
  Data_Spacing2,
  Data_Spacing3,
  Data_Range1,
  Data_Range2,
  Data_Range3 }

Floating point properties of data (DataHandle), that can be retrieved with the function getDataPropertyFloat.

- enum DataAnalyzation {
  Data_Min,
  Data_Mean,
  Data_Max,
  Data_MaxDepth }

  Analysis types accepted by the functions analyzeData and computeDataProjection.
- enum AScanAnalyzation {
  Data_Noise_dB,
  Data_Noise_electrons,
  Data_PeakPos_Pixel,
  Data_PeakPos_PhysUnits,
  Data_PeakHeight_dB,
  Data_PeakWidth_6dB,
  Data_PeakWidth_20dB,
  Data_PeakWidth_40dB,
  Data_PeakPhase,
  Data_PeakRealPart,
  Data_PeakImagPart }

  Analysis types accepted by the functions analyzeAScan and analyzeComplexAScan.
- enum DataOrientation {
  DataOrientation_ZXY,
  DataOrientation_ZYX,
  DataOrientation_XZY,
  DataOrientation_XYZ,
  DataOrientation_YXZ,
  DataOrientation_YZX,
  DataOrientation_ZTX,
  DataOrientation_ZXT }

  Supported data orientations. The default orientation is the first one.

Functions

- SPECTRALRADAR_API int getDataPropertyInt (DataHandle Data, DataPropertyInt Selection)

  Returns the selected integer property of the specified data.
- SPECTRALRADAR_API double getDataPropertyFloat (DataHandle Data, DataPropertyFloat Selection)

  Returns the selected floating point property of the specified data.
- SPECTRALRADAR_API void copyData (DataHandle DataSource, DataHandle DataDestination)

  Copies the content of the specified source to the specified destination.
- SPECTRALRADAR_API void copyDataContent (DataHandle DataSource, float ∗Destination)

  Copies the data in the specified data object (DataHandle) into the specified pointer.
- SPECTRALRADAR_API float ∗ getDataPtr (DataHandle Data)

  The returned pointer points to memory owned by SpectralRadar.dll. The user should not attempt to free it.
- SPECTRALRADAR_API void reserveData (DataHandle Data, int Size1, int Size2, int Size3)

  Reserves the amount of data specified. This might improve performance if appending data to the DataHandle as no additional memory needs to be reserved then.
- SPECTRALRADAR_API void resizeData (DataHandle Data, int Size1, int Size2, int Size3)

  Resizes the respective data object. In general the data will be 1-dimensional if Size2 and Size3 are equal to 1, 2-dimensional if Size3 is equal to 1 dn 3-dimensional if all, Size1, Size2, Size3, are unequal to 1.
- SPECTRALRADAR_API void setDataRange (DataHandle Data, double range1, double range2, double range3)

  Sets the range in mm in the 3 axes represented in the RealData buffer.
- SPECTRALRADAR_API void setDataContent (DataHandle Data, float ∗NewContent)

Sets the data content of the data object. The data chunk pointed to by NewContent needs to be of the size expected by the data object, i. e. Size1∗Size2∗Size∗sizeof(float).

- SPECTRALRADAR_API DataOrientation getDataOrientation (DataHandle Data)

  Returns the data orientation of the data object.
- SPECTRALRADAR_API void setDataOrientation (DataHandle Data, DataOrientation Orientation)

  Sets the data oritentation of the data object to the given orientation.
- SPECTRALRADAR_API int getComplexDataPropertyInt (ComplexDataHandle Data, DataPropertyInt Selection)

  Returns the selected integer property of the specified data.
- SPECTRALRADAR_API double getComplexDataPropertyFloat (ComplexDataHandle Data, DataPropertyFloat Selection)

  Returns the selected floating-point property of the specified data.
- SPECTRALRADAR_API void copyComplexDataContent (ComplexDataHandle DataSource, ComplexFloat ∗Destination)

  Copies the content of the complex data to the pointer specified as destination.
- SPECTRALRADAR_API void copyComplexData (ComplexDataHandle DataSource, ComplexDataHandle DataDestination)

  Copies the contents of the specified ComplexDataHandle to the specified destination ComplexDataHandle.
- SPECTRALRADAR_API ComplexFloat ∗ getComplexDataPtr (ComplexDataHandle Data)

  The returned pointer points to memory owned by SpectralRadar.dll. The user should not attempt to free it.
- SPECTRALRADAR_API void setComplexDataContent (ComplexDataHandle Data, ComplexFloat ∗NewContent)

  Sets the data content of the ComplexDataHandle to the content specified by the pointer.
- SPECTRALRADAR_API void reserveComplexData (ComplexDataHandle Data, int Size1, int Size2, int Size3)

  Reserves the amount of data specified. This might improve performance if appending data to the ComplexDataHandle as no additional memory needs to be reserved then.
- SPECTRALRADAR_API void resizeComplexData (ComplexDataHandle Data, int Size1, int Size2, int Size3)

  Resizes the respective data object. In general the data will be 1-dimensional if Size2 and Size3 are equal to 1, 2-dimensional if Size3 is equal to 1 dn 3-dimensional if all, Size1, Size2, Size3, are unequal to 1.
- SPECTRALRADAR_API void setComplexDataRange (ComplexDataHandle Data, double range1, double range2, double range3)

  Sets the range in mm in the 3 axes represented in the RealData buffer.
- SPECTRALRADAR_API int getColoredDataPropertyInt (ColoredDataHandle ColData, DataPropertyInt Selection)

  Returns the selected integer property of the specified colored data.
- SPECTRALRADAR_API double getColoredDataPropertyFloat (ColoredDataHandle ColData, DataPropertyFloat Selection)

  Returns the selected integer property of the specified colored data.
- SPECTRALRADAR_API void copyColoredData (ColoredDataHandle ImageSource, ColoredDataHandle ImageDestionation)

  Copies the contents of the specified ColoredDataHandle to the specified destination ColoredDataHandle.
- SPECTRALRADAR_API void copyColoredDataContent (ColoredDataHandle Source, unsigned long ∗Destination)

  Copies the data in the specified colored data object (ColoredDataHandle) into the specified pointer.
- SPECTRALRADAR_API void copyColoredDataContentAligned (ColoredDataHandle ImageSource, unsigned long ∗Destination, int Stride)

  Copies the data in the specified colored data object (ColoredDataHandle) into the specified pointer.
- SPECTRALRADAR_API unsigned long ∗ getColoredDataPtr (ColoredDataHandle ColData)

  The returned pointer points to memory owned by SpectralRadar.dll. The user should not attempt to free it.
- SPECTRALRADAR_API void resizeColoredData (ColoredDataHandle ColData, int Size1, int Size2, int Size3)

  Resizes the respective colored data object. In general the data will be 1-dimensional if Size2 and Size3 are equal to 1, 2-dimensional if Size3 is equal to 1 dn 3-dimensional if all, Size1, Size2, Size3, are unequal to 1.

- SPECTRALRADAR_API void reserveColoredData (ColoredDataHandle ColData, int Size1, int Size2, int Size3)

  Reserves the amount of colored data specified. This might improve performance if appending data to the ColoredDataHandle as no additional memory needs to be reserved then.

- SPECTRALRADAR_API void setColoredDataContent (ColoredDataHandle ColData, unsigned long ∗NewContent)

  Sets the data content of the colored data object. The data chung pointed to by NewContent needs to be of the size expected by the data object, i. e. Size1∗Size2∗Size∗sizeof(unsigned long).

- SPECTRALRADAR_API void setColoredDataRange (ColoredDataHandle Data, double range1, double range2, double range3)

  Sets the range in mm in the 3 axes represented in the data object buffer.

- SPECTRALRADAR_API DataOrientation getColoredDataOrientation (ColoredDataHandle Data)

  Returns the data orientation of the colored data object.

- SPECTRALRADAR_API void setColoredDataOrientation (ColoredDataHandle Data, DataOrientation Orientation)

  Sets the data oritentation of the colored data object to the given orientation.

- SPECTRALRADAR_API void copyRawDataContent (RawDataHandle RawDataSource, void ∗DataContent)

  Copies the content of the raw data into the specified buffer.

- SPECTRALRADAR_API void copyRawData (RawDataHandle RawDataSource, RawDataHandle RawDataTarget)

  Copies raw data content and metadata into the specified target handle.

- SPECTRALRADAR_API void ∗ getRawDataPtr (RawDataHandle RawDataSource)

  Notice that raw data refers to the spectra as acquired, without processing of any kind.

- SPECTRALRADAR_API int getRawDataPropertyInt (RawDataHandle RawData, RawDataPropertyInt Property)

  Notice that raw data refers to the spectra as acquired, without processing of any kind.

- SPECTRALRADAR_API void setRawDataBytesPerPixel (RawDataHandle Raw, int BytesPerPixel)

  Sets the bytes per pixel for raw data.

- SPECTRALRADAR_API void reserveRawData (RawDataHandle Raw, int Size1, int Size2, int Size3)

  Reserves the amount of data specified. This might improve performance if appending data to the RawDataHandle as no additional memory needs to be reserved then.

- SPECTRALRADAR_API void resizeRawData (RawDataHandle Raw, int Size1, int Size2, int Size3)

  Resizes the specified raw data buffer accordingly.

- SPECTRALRADAR_API void setRawDataContent (RawDataHandle RawData, void ∗NewContent)

  Sets the content of the raw data buffer. The size of the RawDataHandle needs to be adjusted first, as otherwise not all data might be copied.

- SPECTRALRADAR_API void setScanSpectra (RawDataHandle RawData, int NumberOfScanRegions, int ∗ScanRegions)

  Notice that raw data refers to the spectra as acquired, without processing of any kind.

  .

- SPECTRALRADAR_API void setApodizationSpectra (RawDataHandle RawData, int NumberOfApoRegions, int ∗ApodizationRegions)

  Notice that raw data refers to the spectra as acquired, without processing of any kind.

- SPECTRALRADAR_API int getNumberOfScanRegions (RawDataHandle Raw)

  Returns the number of regions that have been acquired that contain scan data, i. e. spectra that are used to compute A-scans.

- SPECTRALRADAR_API int getNumberOfApodizationRegions (RawDataHandle Raw)

  Returns the number of regions in the raw data containing spectra that are supposed to be used for apodization.

- SPECTRALRADAR_API void getScanSpectra (RawDataHandle Raw, int ∗SpectraIndex)

  Returns the indices of spectra that contain scan data, i. e. spectra that are supposed to be used to compute A-scans.

- SPECTRALRADAR_API void getApodizationSpectra (RawDataHandle Raw, int ∗SpectraIndex)

  Returns the indices of spectra that contain apodization data, i. e. spectra that are supposed to be used as input for apodization.

- SPECTRALRADAR_API void determineSurface (DataHandle Volume, DataHandle Surface)

  Performs a minimal segmentation of the data, by finding a surface that is compromised of the highes signals fromo each A-scan. From the 3D input data, the output data will 2D data, where each data pixel contains the depth of the respective surface as a funciton of the x- and y-pixel position.
- SPECTRALRADAR_API void absComplexData (ComplexDataHandle ComplexData, DataHandle Abs)

  Converts the complex values from the ComplexDataHandle to its absolute values and writes them to DataHandle.
- SPECTRALRADAR_API void logAbsComplexData (ComplexDataHandle ComplexData, DataHandle dB)

  Converts the complex values from the ComplexDataHandle to its dB values and writes them to DataHandle.
- SPECTRALRADAR_API void argComplexData (ComplexDataHandle ComplexData, DataHandle Arg)

  Converts the complex values from the ComplexDataHandle to its phase angle values and writes them to DataHandle.
- SPECTRALRADAR_API void realComplexData (ComplexDataHandle ComplexData, DataHandle Real)

  Writes the real part of the complex values from the ComplexDataHandle to DataHandle.
- SPECTRALRADAR_API void imagComplexData (ComplexDataHandle ComplexData, DataHandle Imag)

  Writes the imaginary part of the complex values from the ComplexDataHandle to DataHandle.
- SPECTRALRADAR_API void crossCorrelatedProjection (DataHandle DataIn, DataHandle DataOut)

  Upon return DataOut contains an average of all B-Scans in DataIn. Right before averaging, the datasets are cross-correlated to eliminate registration errors.
- SPECTRALRADAR_API void thresholdDopplerData (DataHandle Phase, DataHandle Intensity, float intensityThreshold, float phaseTargetValue)

  At points whose Intensity does not exceed the intensityThreshold, the phase is set to the phaseTargetValue.
- SPECTRALRADAR_API void getCurrentIntensityStatistics (OCTDeviceHandle Dev, ProcessingHandle Proc, float ∗relToRefIntensity, float ∗relToProjAbsIntensity)

  Returns two statistical interpretations of the current light intensity on the sensor.

5.3.1 Detailed Description

Functions for accessing the information stored in data objects.

5.3.2 Typedef Documentation

5.3.2.1 ColoredDataHandle

Handle to an object holding 1-, 2- or 3-dimensional colored data.

Colored data handles are used to obtain processed data in a format that can readily be exported into a graphics format file, using a user selected palette. Otherwise the are the same as processed data (DataHandle).
In order to specify the desired palette and its properties, users should refer to coloring handles (ColoringHandle) and associated functions.
This structure supports reuse. That is, once created, it can be reused many times to hold different data. If passed as a parameter to the processing (e.g. throught the function setColoredDataOutput), the meta data (sizes, ranges, etc.) will be adjusted automatically each time.

5.3.2.2 ComplexDataHandle

Handle to an object holding complex 1-, 2- or 3-dimensional complex floating point data.

This structure supports reuse. That is, once created, it can be reused many times to hold different data. If passed as a parameter to the processing (e.g. throught the function setComplexDataOutput), the meta data (sizes, ranges, etc.) will be adjusted automatically each time.

5.3.2.3 DataHandle

Handle to an object holding 1-, 2- or 3-dimensional floating point data.

This structure may hold data generated by processing raw data (RawDataHandle), and also more abstract data, such as point sequences intended to determine a scan pattern (see e.g. getSizeOfScanPointsFromDataHandle or getScanPointsFromDataHandle below). The associated properties of the data (dimensionality, sizes in pixels, spacings/ranges in millimeters) are also part of this structure.
This structure supports reuse. That is, once created, it can be reused many times to hold different data. If passed as a parameter to the processing (e.g. throught the function setProcessedDataOutput), the meta data (sizes, ranges, etc.) will be adjusted automatically each time.

5.3.2.4 ImageFieldHandle

Handle to the image field description.

5.3.2.5 OCTFileHandle

Handle to the OCT file class.

5.3.2.6 RawDataHandle

Handle to an object holding the unprocessed raw data.

Raw data refers to the spectra as acquired, without processing of any kind. This structure accomodates not only the actual pixel values, but also the meta-data, such as the number of bytes per pixel, the sizes, the number of elements, or the number of frames that had been lost during the acquisition.

5.3.3 Enumeration Type Documentation

5.3.3.1 enum AScanAnalyzation

Analysis types accepted by the functions analyzeAScan and analyzeComplexAScan.

Enumerator

Data_Noise_dB Noise of the A-scan in dB. This assumes that no signal is present in the A-scan. The noise is computed by averaging all fourier channels larger than 50.

Data_Noise_electrons Noise of the A-scan in electrons. This assumes that no signal is present in the A-scan. The noise is computed by averaging all fourier channels larger than 50.

Data_PeakPos_Pixel Peak position of the highest peak in pixels. The peak position is determined by computing a parable going through the maximum value point and its surrounding pixels. The position of the maximum is used.

Data_PeakPos_PhysUnits Peak position of the highest peak in physical units. The peak position is determined by computing a parable going through the maximum value point and its surrounding pixels. The position of the maximum is used. Physical coordinates are computed by using the calibrated zSpacing property of the device. The concrete physical units of the return value depends on the calibration.

Data_PeakHeight_dB Peak height of the highest peak in dB. The peak height is determined by computing a parable going through the maximum value point and its surrounding pixels. The height of the resulting parable is returned.

Data_PeakWidth_6dB Signal width at -6dB. This is the FWHM.

Data_PeakWidth_20dB Signal width at -20dB.

Data_PeakWidth_40dB Signal width at -40dB.

Data_PeakPhase Phase of the highest peak in radians. This value is only accepted by the function analyzeComplexAScan.

Data_PeakRealPart Real part of the highest peak, expressed in \$e^-\$. This value is only accepted by the function analyzeComplexAScan.

Data_PeakImagPart Imaginary part of the highest peak, expressed in \$e^-\$. This value is only accepted by the function analyzeComplexAScan.

5.3.3.2 enum DataAnalyzation

Analysis types accepted by the functions analyzeData and computeDataProjection.

Enumerator

Data_Min Minimum of the values in the data.

Data_Mean Arithmetic mean of all values in the data.

Data_Max Maximum of the values in the data.

Data_MaxDepth The depth of the maximum of the values in the data.

5.3.3.3 enum DataOrientation

Supported data orientations. The default orientation is the first one.

See also

getDataOrientation, setDataOrientation, getColoredDataOrientation, setColoredDataOrientation.

5.3.3.4 enum DataPropertyFloat

Floating point properties of data (DataHandle), that can be retrieved with the function getDataPropertyFloat.

Enumerator

Data_Spacing1 Spacing between two subsequent data elements in direction of the first axis in physical units (millimeter).

Data_Spacing2 Spacing between two subsequent data elements in direction of the second axis in physical units (millimeter).

Data_Spacing3 Spacing between two subsequent data elements in direction of the third axis in physical units (millimeter).

Data_Range1 Total range of the data in direction of the first axis in physical units (millimeter).

Data_Range2 Total range of the data in direction of the second axis in physical units (millimeter).

Data_Range3 Total range of the data in direction of the third axis in physical units (millimeter).

5.3.3.5 enum DataPropertyInt

Integer properties of data (DataHandle) that can be retrieved with the function getDataPropertyInt.

Enumerator

Data_Dimensions Dimension of the data object. Usually 1, 2 or 3. 0 indicates empty data.

Data_Size1 Size of the first dimension. For OCT data this is usually the longitudinal axis (z)

Data_Size2 Size of the first dimension. For OCT data this is usually a transversal axis (x)

Data_Size3 Size of the first dimension. For OCT data this is usually a transversal axis (y)

Data_NumberOfElements The number of elements in the data object.

Data_SizeInBytes The size of the data object in bytes.

Data_BytesPerElement The number of bytes of a single element.

5.3.3.6 enum RawDataPropertyInt

Integer properties of raw data (RawDataHandle) that can be retrieved with the function getRawDataPropertyInt.

Enumerator

RawData_Size1 Size of the first dimension. This will be the spectral dimension, i. e. z-dimension prior to Fourier transformation.

RawData_Size2 Size of the second dimension. This is a transversal axis (x).

RawData_Size3 Size of the third dimension. This is a transversal axis (y).

RawData_NumberOfElements The number of elements in the raw data object.

RawData_SizeInBytes The size of the data object in bytes.

RawData_BytesPerElement The number of bytes of a single element, i. e. the data type of the raw data.

RawData_LostFrames The number of lost frames during data acqusition.

5.3.4 Function Documentation

5.3.4.1 void absComplexData (ComplexDataHandle *ComplexData*, DataHandle *Abs*)

Converts the complex values from the ComplexDataHandle to its absolute values and writes them to DataHandle.

5.3.4.2 void argComplexData (ComplexDataHandle *ComplexData*, DataHandle *Arg*)

Converts the complex values from the ComplexDataHandle to its phase angle values and writes them to DataHandle.

5.3.4.3 void copyColoredData (ColoredDataHandle *ImageSource*, ColoredDataHandle *ImageDestionation*)

Copies the contents of the specified ColoredDataHandle to the specified destination ColoredDataHandle.

Parameters

in	*ImageSource*	A valid (non null) colored data handle of the source (ColoredDataHandle).
in	*ImageDestionation*	A valid (non null) colored data handle of the destination (ColoredDataHandle).

5.3.4.4 void copyColoredDataContent (ColoredDataHandle *Source*, unsigned long ∗ *Destination*)

Copies the data in the specified colored data object (ColoredDataHandle) into the specified pointer.

Parameters

in	*Source*	A valid (non null) colored data handle of the source (ColoredDataHandle).
out	*Destination*	A valid (non null) pointer to an integer array, with enough space to copy the data.

In order to find out the amount of memory that has to be reserved, the size(s) of the source data has to be inquired with the function getColoredDataPropertyInt (because they are integer properties).

5.3.4.5 void copyColoredDataContentAligned (ColoredDataHandle *ImageSource*, unsigned long ∗ *Destination*, int *Stride*)

Copies the data in the specified colored data object (ColoredDataHandle) into the specified pointer.

Parameters

in	*ImageSource*	A valid (non null) colored data handle of the source (ColoredDataHandle).
out	*Destination*	A valid (non null) pointer to an integer array, with enough space to copy the data.
in	*Stride*	The total amount of bytes per row, which may contain some padding after the last pixel.

In order to find out the amount of memory that has to be reserved, the size(s) of the source data has to be inquired (they are integer properties).

5.3.4.6 void copyComplexData (ComplexDataHandle *DataSource*, ComplexDataHandle *DataDestination*)

Copies the contents of the specified ComplexDataHandle to the specified destination ComplexDataHandle.

Parameters

in	*DataSource*	A valid (non null) complex data handle of the source (ComplexDataHandle).
in	*DataDestination*	A valid (non null) complex data handle of the destination (ComplexDataHandle).

5.3.4.7 void copyComplexDataContent (ComplexDataHandle *DataSource*, ComplexFloat ∗ *Destination*)

Copies the content of the complex data to the pointer specified as destination.

Parameters

in	*DataSource*	A valid (non null) complex data handle of the source (ComplexDataHandle).
out	*Destination*	A valid (non null) pointer to a complex array, with enough space to copy the data.

In order to find out the amount of memory that has to be reserved, the size(s) of the source data has to be inquired with the function getComplexDataPropertyInt (because they are integer properties).

5.3.4.8 void copyData (DataHandle *DataSource*, DataHandle *DataDestination*)

Copies the content of the specified source to the specified destination.

Parameters

in	*DataSource*	A valid (non null) data handle of the source (DataHandle).
in	*DataDestination*	A valid (non null) data handle of the destination (DataHandle).

5.3.4.9 void copyDataContent (DataHandle *DataSource*, float ∗ *Destination*)

Copies the data in the specified data object (DataHandle) into the specified pointer.

Parameters

`in`	*DataSource*	A valid (non null) data handle of the source (DataHandle).
`out`	*Destination*	A valid (non null) pointer to float array, with enough space to copy the data.

In order to find out the amount of memory that has to be reserved, the size(s) of the source data has to be inquired with the function getDataPropertyInt (because they are integer properties).

5.3.4.10 void copyRawData (**RawDataHandle** *RawDataSource,* **RawDataHandle** *RawDataTarget*)

Copies raw data content and metadata into the specified target handle.

Parameters

`in`	*RawDataSource*	A valid (non null) raw data handle of the source (RawDataHandle).
`in`	*RawDataTarget*	A valid (non null) raw data handle of the target (RawDataHandle).

Notice that raw data refers to the spectra as acquired, without processing of any kind. The pointer is void because different cameras/sensors with different amount of bytes per pixel are supported.

5.3.4.11 void copyRawDataContent (**RawDataHandle** *RawDataSource,* void ∗ *DataContent*)

Copies the content of the raw data into the specified buffer.

Parameters

`in`	*RawDataSource*	A valid (non null) raw data handle of the source (RawDataHandle).
`out`	*DataContent*	A valid (non null) pointer to an array, with enough space to copy the data.

In order to find out the amount of memory that has to be reserved, the size(s) of the source data has to be inquired with the function getRawDataPropertyInt (because they are integer properties).
Notice that raw data refers to the spectra acquired, without processing of any kind. The pointer is void because different cameras/sensors with different amount of bytes per pixel are supported.

5.3.4.12 void crossCorrelatedProjection (**DataHandle** *DataIn,* **DataHandle** *DataOut*)

Upon return DataOut contains an average of all B-Scans in DataIn. Right before averaging, the datasets are crosscorrelated to eliminate registration errors.

5.3.4.13 void determineSurface (**DataHandle** *Volume,* **DataHandle** *Surface*)

Performs a minimal segmentation of the data, by finding a surface that is compromised of the highes signals fromo each A-scan. From the 3D input data, the output data will 2D data, where each data pixel contains the depth of the respective surface as a funciton of the x- and y-pixel position.

5.3.4.14 void getApodizationSpectra (**RawDataHandle** *Raw,* int ∗ *SpectraIndex*)

Returns the indices of spectra that contain apodization data, i. e. spectra that are supposed to be used as input for apodization.

of apodization regions which can be obtained by getNumberOfApodizationRegions() During the scanning, the light spot travels along a curve, also known as scan pattern. At each of the points of the curve, spectra are measured. Some spectra are acquired at points where an A-Scans are desired (the scan region(s)), some others where an Apodization is desired (apodization region(s)), and some others at less than interesting positions.
Notice that raw data refers to the spectra as acquired, without processing of any kind.

Parameters

in	*Raw*	A valid (non null) raw data handle (RawDataHandle).
out	*SpectraIndex*	the array of indices delimiting the apodization regions. The size of this array should be twice the number

5.3.4.15 DataOrientation getColoredDataOrientation (ColoredDataHandle *Data*)

Returns the data orientation of the colored data object.

Returns

The current orientation (DataOrientation).

Parameters

in	*Data*	A valid (non null) colored data handle (ColoredDataHandle).

5.3.4.16 int getColoredDataPropertyFloat (ColoredDataHandle *ColData,* DataPropertyFloat *Selection*)

Returns the selected integer property of the specified colored data.

Returns

The value of the desired property.

Parameters

in	*ColData*	A valid (non null) colored data handle (ColoredDataHandle).
in	*Selection*	The desired property.

5.3.4.17 int getColoredDataPropertyInt (ColoredDataHandle *ColData,* DataPropertyInt *Selection*)

Returns the selected integer property of the specified colored data.

Returns

The value of the desired property.

Parameters

in	*ColData*	A valid (non null) colored data handle (ColoredDataHandle).
in	*Selection*	The desired property.

5.3.4.18 unsigned long ∗ getColoredDataPtr (ColoredDataHandle *ColData*)

The returned pointer points to memory owned by SpectralRadar.dll. The user should not attempt to free it.

Returns a pointer to the content of the specified ColoredDataHandle.

Returns

A pointer to the memory owned by the handle.

Parameters

in	*ColData*	A valid (non null) colored data handle (ColoredDataHandle).

5.3.4.19 double getComplexDataPropertyFloat (**ComplexDataHandle** *Data*, **DataPropertyFloat** *Selection*)

Returns the selected floating-point property of the specified data.

Parameters

in	*Data*	A valid (non null) complex data handle (ComplexDataHandle).
in	*Selection*	The desired property.

Returns

The value of the desired property.

5.3.4.20 int getComplexDataPropertyInt (**ComplexDataHandle** *Data*, **DataPropertyInt** *Selection*)

Returns the selected integer property of the specified data.

Returns

The value of the desired property.

Parameters

in	*Data*	A valid (non null) complex data handle (ComplexDataHandle).
in	*Selection*	The desired property.

5.3.4.21 **ComplexFloat** ∗ getComplexDataPtr (**ComplexDataHandle** *Data*)

The returned pointer points to memory owned by SpectralRadar.dll. The user should not attempt to free it.

Returns a pointer to the data represented by the ComplexDataHandle. The data is still managed by the ComplexDataHandle object.

Returns

A pointer to the memory owned by the handle.

Parameters

`in`	*Data*	A valid (non null) complex data handle (ComplexDataHandle).

5.3.4.22 void getCurrentIntensityStatistics (OCTDeviceHandle *Dev,* ProcessingHandle *Proc,* float ∗ *relToRefIntensity,* float ∗ *relToProjAbsIntensity*)

Returns two statistical interpretations of the current light intensity on the sensor.

used in SR Service, do not remove

5.3.4.23 void DataOrientation getDataOrientation (DataHandle *Data*)

Returns the data orientation of the data object.

Parameters

`in`	*Data*	A valid (non null) data handle.

5.3.4.24 double getDataPropertyFloat (DataHandle *Data,* DataPropertyFloat *Selection*)

Returns the selected floating point property of the specified data.

Returns

The value of the desired property.

Parameters

`in`	*Data*	A valid (non null) data handle (DataHandle).
`in`	*Selection*	The desired property.

5.3.4.25 int getDataPropertyInt (DataHandle *Data,* DataPropertyInt *Selection*)

Returns the selected integer property of the specified data.

Returns

The value of the desired property.

Parameters

`in`	*Data*	A valid (non null) data handle (DataHandle).
`in`	*Selection*	The desired property.

5.3.4.26 float * getDataPtr (DataHandle *Data*)

The returned pointer points to memory owned by SpectralRadar.dll. The user should not attempt to free it.

Returns a pointer to the content of the specified data.

Returns

A pointer to the memory owned by the handle.

Parameters

`in`	*Data*	A valid (non null) data handle (DataHandle).

5.3.4.27 int getNumberOfApodizationRegions (RawDataHandle *Raw*)

Returns the number of regions in the raw data containing spectra that are supposed to be used for apodization.

Returns

The number of apodization regions (each region may contain several apodizations).During the scanning, the light spot travels along a curve, also known as scan pattern. At each of the points of the curve, spectra are measured. Some spectra are acquired at points where an A-Scans are desired (the scan region(s)), some others where an Apodization is desired (apodization region(s)), and some others at less than interesting positions.
Notice that raw data refers to the spectra as acquired, without processing of any kind.

Parameters

`in`	*Raw*	A valid (non null) raw data handle (RawDataHandle).

5.3.4.28 int getNumberOfScanRegions (RawDataHandle *Raw*)

Returns the number of regions that have been acquired that contain scan data, i. e. spectra that are used to compute A-scans.

Returns

The number of scan regions (each region may contain several scans).During the scanning, the light spot travels along a curve, also known as scan pattern. At each of the points of the curve, spectra are measured. Some spectra are acquired at points where an A-Scans are desired (the scan region(s)), some others where an Apodization is desired (apodization region(s)), and some others at less than interesting positions.
Notice that raw data refers to the spectra as acquired, without processing of any kind.

Parameters

`in`	*Raw*	A valid (non null) raw data handle (RawDataHandle).

5.3.4.29 int getRawDataPropertyInt (**RawDataHandle** *RawData,* **RawDataPropertyInt** *Property*)

Notice that raw data refers to the spectra as acquired, without processing of any kind.

Returns a raw data property.

Returns

The value of the desired property.

Parameters

`in`	*RawData*	A valid (non null) raw data handle (RawDataHandle).
`in`	*Property*	The desired property.

5.3.4.30 void ∗ getRawDataPtr (**RawDataHandle** *RawDataSource*)

Notice that raw data refers to the spectra as acquired, without processing of any kind.

Returns the pointer to the raw data content. The pointer might no longer after additional actions using the Raw←DataHandle.

Returns

A pointer to the memory owned by the handle. The pointer is void because different cameras/sensors with different amount of bytes per pixel are supported.

Parameters

`in`	*RawDataSource*	A valid (non null) raw data handle of the source (RawDataHandle).

5.3.4.31 void getScanSpectra (**RawDataHandle** *Raw,* int ∗ *SpectraIndex*)

Returns the indices of spectra that contain scan data, i. e. spectra that are supposed to be used to compute A-scans.

of scan regions which can be obtained by getNumberOfScanRegions() During the scanning, the light spot travels along a curve, also known as scan pattern. At each of the points of the curve, spectra are measured. Some spectra are acquired at points where an A-Scans are desired (the scan region(s)), some others where an Apodization is desired (apodization region(s)), and some others at less than interesting positions.
Notice that raw data refers to the spectra as acquired, without processing of any kind.

Parameters

`in`	*Raw*	A valid (non null) raw data handle (RawDataHandle).
`out`	*SpectraIndex*	the array of indices delimiting the scan regions. The size of this array should be twice the number

5.3.4.32 void imagComplexData (**ComplexDataHandle** *ComplexData,* **DataHandle** *Imag*)

Writes the imaginary part of the complex values from the ComplexDataHandle to DataHandle.

5.3.4.33 void logAbsComplexData (**ComplexDataHandle** *ComplexData,* **DataHandle** *dB*)

Converts the complex values from the ComplexDataHandle to its dB values and writes them to DataHandle.

5.3.4.34 void realComplexData (**ComplexDataHandle** *ComplexData,* **DataHandle** *Real*)

Writes the real part of the complex values from the ComplexDataHandle to DataHandle.

5.3.4.35 void reserveColoredData (**ColoredDataHandle** *ColData,* int *Size1,* int *Size2,* int *Size3*)

Reserves the amount of colored data specified. This might improve performance if appending data to the ColoredDataHandle as no additional memory needs to be reserved then.

Parameters

`in`	*ColData*	A valid (non null) colored data handle (ColoredDataHandle).
`in`	*Size1*	The desired number of data along the first axis ("z" in the default orientation).
`in`	*Size2*	The desired number of data along the second axis ("x" in the default orientation).
`in`	*Size3*	The desired number of data along the third axis ("y" in the default orientation).

5.3.4.36 void reserveComplexData (**ComplexDataHandle** *Data,* int *Size1,* int *Size2,* int *Size3*)

Reserves the amount of data specified. This might improve performance if appending data to the ComplexDataHandle as no additional memory needs to be reserved then.

Parameters

`in`	*Data*	A valid (non null) complex data handle (ComplexDataHandle).
`in`	*Size1*	The desired number of data along the first axis ("z" in the default orientation).
`in`	*Size2*	The desired number of data along the second axis ("x" in the default orientation).
`in`	*Size3*	The desired number of data along the third axis ("y" in the default orientation).

5.3.4.37 void reserveData (**DataHandle** *Data,* int *Size1,* int *Size2,* int *Size3*)

Reserves the amount of data specified. This might improve performance if appending data to the DataHandle as no additional memory needs to be reserved then.

Parameters

`in`	*Data*	A valid (non null) data handle (DataHandle).
`in`	*Size1*	The number of data along the first axis ("z" in the default orientation).
`in`	*Size2*	The number of data along the second axis ("x" in the default orientation).
`in`	*Size3*	The number of data along the third axis ("y" in the default orientation).

5.3.4.38 void reserveRawData (**RawDataHandle** *Raw,* int *Size1,* int *Size2,* int *Size3*)

Reserves the amount of data specified. This might improve performance if appending data to the RawDataHandle as no additional memory needs to be reserved then.

Parameters

`in`	*Raw*	A valid (non null) raw data handle (RawDataHandle).
`in`	*Size1*	The desired number of data along the first axis ("z" in the default orientation).
`in`	*Size2*	The desired number of data along the second axis ("x" in the default orientation).
`in`	*Size3*	The desired number of data along the third axis ("y" in the default orientation).

Notice that raw data refers to the spectra as acquired, without processing of any kind.

5.3.4.39 void resizeColoredData (**ColoredDataHandle** *ColData,* int *Size1,* int *Size2,* int *Size3*)

Resizes the respective colored data object. In general the data will be 1-dimensional if Size2 and Size3 are equal to 1, 2-dimensional if Size3 is equal to 1 dn 3-dimensional if all, Size1, Size2, Size3, are unequal to 1.

Parameters

`in`	*ColData*	A valid (non null) complex data handle (ColoredDataHandle).
`in`	*Size1*	The desired number of data along the first axis ("z" in the default orientation).
`in`	*Size2*	The desired number of data along the second axis ("x" in the default orientation).
`in`	*Size3*	The desired number of data along the third axis ("y" in the default orientation).

5.3.4.40 void resizeComplexData (**ComplexDataHandle** *Data,* int *Size1,* int *Size2,* int *Size3*)

Resizes the respective data object. In general the data will be 1-dimensional if Size2 and Size3 are equal to 1, 2-dimensional if Size3 is equal to 1 dn 3-dimensional if all, Size1, Size2, Size3, are unequal to 1.

Parameters

`in`	*Data*	A valid (non null) complex data handle (ComplexDataHandle).
`in`	*Size1*	The desired number of data along the first axis ("z" in the default orientation).
`in`	*Size2*	The desired number of data along the second axis ("x" in the default orientation).
`in`	*Size3*	The desired number of data along the third axis ("y" in the default orientation).

5.3.4.41 void resizeData (**DataHandle** *Data,* int *Size1,* int *Size2,* int *Size3*)

Resizes the respective data object. In general the data will be 1-dimensional if Size2 and Size3 are equal to 1, 2-dimensional if Size3 is equal to 1 dn 3-dimensional if all, Size1, Size2, Size3, are unequal to 1.

Parameters

`in`	*Data*	A valid (non null) data handle (DataHandle).
`in`	*Size1*	The desired number of data along the first axis ("z" in the default orientation).
`in`	*Size2*	The desired number of data along the second axis ("x" in the default orientation).
`in`	*Size3*	The desired number of data along the third axis ("y" in the default orientation).

5.3.4.42 void resizeRawData (RawDataHandle *Raw*, int *Size1*, int *Size2*, int *Size3*)

Resizes the specified raw data buffer accordingly.

Parameters

`in`	*Raw*	A valid (non null) raw data handle (RawDataHandle).
`in`	*Size1*	The desired number of data along the first axis ("z" in the default orientation).
`in`	*Size2*	The desired number of data along the second axis ("x" in the default orientation).
`in`	*Size3*	The desired number of data along the third axis ("y" in the default orientation).

Notice that raw data refers to the spectra as acquired, without processing of any kind.

5.3.4.43 void setApodizationSpectra (RawDataHandle *RawData*, int *NumberOfApoRegions*, int ∗ *ApodizationRegions*)

Notice that raw data refers to the spectra as acquired, without processing of any kind.

Sets the number of the spectra in the raw data that contain data useful as apodization spectra.

This function sets the regions where apodization spectra will be measured. The supplied array must contain an even number of indices. The even indices give the start of a region, and the odd indices give the end of the regions (actually, one point past-the-end). In other words, the index pair at position (2n,2n+1) in the array (third argument) gives the n-th region, starting at point ApodizationRegions[2n] and ending at (but not including) the point ApodizationRegions[2∗n+1]. Here 0 <= n < NumberOfApoRegions.

Parameters

`in`	*RawData*	A valid (non null) raw data handle (RawDataHandle).
`in`	*NumberOfApoRegions*	is the number of desired apodization regions.
`in`	*ApodizationRegions*	is an array containing 2∗NumberOfApoRegions elements that delimit the regions.

During the scanning, the light spot travels along a curve, also known as scan pattern. At each of the points of the curve, spectra are measured. Some spectra are acquired at points where an A-Scans are desired (the scan region(s)), some others where an Apodization is desired (apodization region(s)), and some others at less than interesting positions.

5.3.4.44 void setColoredDataContent (ColoredDataHandle *ColData*, unsigned long ∗ *NewContent*)

Sets the data content of the colored data object. The data chung pointed to by NewContent needs to be of the size expected by the data object, i. e. Size1∗Size2∗Size∗sizeof(unsigned long).

Parameters

`in`	*ColData*	A valid (non null) colored data handle (ColoredDataHandle).
`in`	*NewContent*	A valid (non null) pointer to an integer array with the source data.

The amount of data that will be copied depends on the size(s) that had previously been setup in the colored data object.

5.3.4.45 void setColoredDataOrientation (ColoredDataHandle *Data,* DataOrientation *Orientation*)

Sets the data oritentation of the colored data object to the given orientation.

Parameters

in	*Data*	A valid (non null) colored data handle (ColoredDataHandle).
in	*Orientation*	The desired orientation (DataOrientation).

5.3.4.46 void setColoredDataRange (ColoredDataHandle *Data,* double *range1,* double *range2,* double *range3*)

Sets the range in mm in the 3 axes represented in the data object buffer.

Parameters

in	*Data*	A valid (non null) colored data handle (ColoredDataHandle).
in	*range1*	The desired physical extension, in mm, along the first axis ("z" in the default orientation).
in	*range2*	The desired physical extension, in mm, along the second axis ("x" in the default orientation).
in	*range3*	The desired physical extension, in mm, along the third axis ("y" in the default orientation).

5.3.4.47 void setComplexDataContent (ComplexDataHandle *Data,* ComplexFloat ∗ *NewContent*)

Sets the data content of the ComplexDataHandle to the content specified by the pointer.

Parameters

in	*Data*	A valid (non null) complex data handle (ComplexDataHandle).
in	*NewContent*	A valid (non null) pointer to an array of complex numbers (ComplexFloat) with the desired content.

The amount of data that will be copied depends on the size(s) that had previously been setup in the complex data object.

5.3.4.48 void setComplexDataRange (ComplexDataHandle *Data,* double *range1,* double *range2,* double *range3*)

Sets the range in mm in the 3 axes represented in the RealData buffer.

Parameters

in	*Data*	A valid (non null) complex data handle (ComplexDataHandle).
in	*range1*	The desired physical extension, in mm, along the first axis ("z" in the default orientation).
in	*range2*	The desired physical extension, in mm, along the second axis ("x" in the default orientation).
in	*range3*	The desired physical extension, in mm, along the third axis ("y" in the default orientation).

5.3.4.49 void setDataContent (DataHandle *Data,* float ∗ *NewContent*)

Sets the data content of the data object. The data chunk pointed to by NewContent needs to be of the size expected by the data object, i. e. Size1∗Size2∗Size∗sizeof(float).

Parameters

in	*Data*	A valid (non null) data handle (DataHandle).
in	*NewContent*	A valid (non null) pointer to float array with the source data.

The amount of data that will be copied depends on the size(s) that had previously been setup in the data object (using resizeData to ensure that enough space has been allocated).

5.3.4.50 void setDataOrientation (DataHandle *Data,* DataOrientation *Orientation*)

Sets the data oritentation of the data object to the given orientation.

Parameters

in	*Data*	A valid (non null) data handle (DataHandle).
in	*Orientation*	The desired orientation.

5.3.4.51 void setDataRange (DataHandle *Data,* double *range1,* double *range2,* double *range3*)

Sets the range in mm in the 3 axes represented in the RealData buffer.

Parameters

in	*Data*	A valid (non null) data handle.
in	*range1*	The desired physical extension, in mm, along the first axis ("z" in the default orientation).
in	*range2*	The desired physical extension, in mm, along the second axis ("x" in the default orientation).
in	*range3*	The desired physical extension, in mm, along the third axis ("y" in the default orientation).

5.3.4.52 void setRawDataBytesPerPixel (RawDataHandle *Raw,* int *BytesPerPixel*)

Sets the bytes per pixel for raw data.

Parameters

in	*Raw*	A valid (non null) raw data handle (RawDataHandle).
in	*BytesPerPixel*	The number of bytes per pixel supported by the camera or sensor.

If the raw data are retrieved using getRawData(), this parameter is automatically set to the right value. Notice that raw data refers to the spectra as acquired, without processing of any kind.

5.3.4.53 void setRawDataContent (RawDataHandle *RawData,* void $*$ *NewContent*)

Sets the content of the raw data buffer. The size of the RawDataHandle needs to be adjusted first, as otherwise not all data might be copied.

Parameters

in	*RawData*	A valid (non null) raw data handle (RawDataHandle).
in	*NewContent*	A valid (non null) pointer to a void array with the source data.

The amount of data that will be copied depends on the size(s) that had previously been setup in the raw data object. Notice that raw data refers to the spectra as acquired, without processing of any kind.
The pointer is void because different cameras/sensors with different amount of bytes per pixel are supported.

5.3.4.54 void setScanSpectra (**RawDataHandle** *RawData,* int *NumberOfScanRegions,* int ∗ *ScanRegions*)

Notice that raw data refers to the spectra as acquired, without processing of any kind.
.

Sets the number of the spectra in the raw data that are used for creating A-scan/B-scan data.

This function sets the regions where A-Scan computation is desired. The supplied array must contain an even number of indices. The even indices give the start of a region, and the odd indices give the end of the regions (actually, one point past-the-end). In other words, the index pair at position (2n,2n+1) in the array (third argument) gives the n-th region, starting at point ScanRegions[2n] and ending at (but not including) the point ScanRegions[2∗n+1]. Here 0 <= n < NumberOfScanRegions.

Parameters

`in`	*RawData*	A valid (non null) raw data handle (RawDataHandle).
`in`	*NumberOfScanRegions*	is the number of desired scan regions.
`in`	*ScanRegions*	is an array containing 2∗NumberOfScanRegions elements that delimit the regions.

During the scanning, the light spot travels along a curve, also known as scan pattern. At each of the points of the curve, spectra are measured. Some spectra are acquired at points where an A-Scans are desired (the scan region(s)), some others where an Apodization is desired (apodization region(s)), and some others at less than interesting positions.

5.3.4.55 void thresholdDopplerData (**DataHandle** *Phase,* **DataHandle** *Intensity,* float *intensityThreshold,* float *phaseTargetValue*)

At points whose Intensity does not exceed the intensityThreshold, the phase is set to the phaseTargetValue.

5.4 Data Creation and Clearing

Functions to create and clear object containing data.

Functions

- SPECTRALRADAR_API RawDataHandle createRawData (void)

  Notice that raw data refers to the spectra as acquired, without processing of any kind.
- SPECTRALRADAR_API void clearRawData (RawDataHandle Raw)

  Notice that raw data refers to the spectra as acquired, without processing of any kind.
- SPECTRALRADAR_API DataHandle createData (void)

  Creates a 1-dimensional data object, containing floating point data.
- SPECTRALRADAR_API DataHandle createGradientData (int Size)

  Creates a 1-dimensional data object, containing floating point data with equidistant arranged values between [0, size-1] with distance 1/(size-1).
- SPECTRALRADAR_API void clearData (DataHandle Data)

  Clears the specified DataHandle object.
- SPECTRALRADAR_API ColoredDataHandle createColoredData (void)

  Creates a colored data object (ColoredDataHandle).
- SPECTRALRADAR_API void clearColoredData (ColoredDataHandle Volume)

  Clears a colored volume object.
- SPECTRALRADAR_API ComplexDataHandle createComplexData (void)

  Creates a data object holding complex data (ComplexDataHandle).
- SPECTRALRADAR_API void clearComplexData (ComplexDataHandle Data)

  Clears a data object holding complex data (ComplexDataHandle).

5.4.1 Detailed Description

Functions to create and clear object containing data.

5.4.2 Function Documentation

5.4.2.1 void clearColoredData (ColoredDataHandle *Volume*)

Clears a colored volume object.

Parameters

in	*Volume*	A colored data handle (ColoredDataHandle). If the handle is a nullptr, this function does nothing.

5.4.2.2 void clearComplexData (ComplexDataHandle *Data*)

Clears a data object holding complex data (ComplexDataHandle).

Parameters

in	*Data*	A complex data handle (ComplexDataHandle). If the handle is a nullptr, this function does nothing.

5.4.2.3 void clearData (DataHandle *Data*)

Clears the specified DataHandle object.

Parameters

in	*Data*	A data handle (DataHandle). If the handle is a nullptr, this function does nothing.

5.4.2.4 void clearRawData (RawDataHandle *Raw*)

Notice that raw data refers to the spectra as acquired, without processing of any kind.

Clears a raw data object (RawDataHandle)

Parameters

in	*Raw*	A raw data handle. If the handle is a nullptr, this function does nothing.

5.4.2.5 ColoredDataHandle createColoredData (void)

Creates a colored data object (ColoredDataHandle).

Returns

A valid colored data handle (ColoredDataHandle).

5.4.2.6 ComplexDataHandle createComplexData (void)

Creates a data object holding complex data (ComplexDataHandle).

Returns

A valid complex data handle (ComplexDataHandle).

5.4.2.7 DataHandle createData (void)

Creates a 1-dimensional data object, containing floating point data.

Returns

A valid data handle (DataHandle).

5.4.2.8 DataHandle createGradientData (int *Size*)

Creates a 1-dimensional data object, containing floating point data with equidistant arranged values between [0, size-1] with distance 1/(size-1).

Parameters

in	*Size*	Data number.

Returns

A valid data handle (DataHandle).

5.4.2.9 RawDataHandle createRawData (void)

Notice that raw data refers to the spectra as acquired, without processing of any kind.

Creates a raw data object (RawDataHandle).

Returns

A valid raw data handle.

5.5 Hardware

Functions providing direct access to OCT Hardware functionality.

Typedefs

- typedef void(__stdcall ∗ lightSourceStateCallback) (LightSourceState)

  Defines the function prototype for the light source callback(see also setLightSourceTimeoutCallback()). The argument contains the current state of the light source.
- typedef struct C_OCTDevice ∗ OCTDeviceHandle

  The OCTDeviceHandle type is used as Handle for using the SpectralRadar.

Enumerations

- enum DevicePropertyFloat {
  Device_FullWellCapacity,
  Device_zSpacing,
  Device_zRange,
  Device_SignalAmplitudeMin_dB,
  Device_SignalAmplitudeLow_dB,
  Device_SignalAmplitudeHigh_dB,
  Device_SignalAmplitudeMax_dB,
  Device_BinToElectronScaling,
  Device_Temperature,
  Device_SLD_OnTime_sec,
  Device_CenterWavelength_nm,
  Device_SpectralWidth_nm,
  Device_MaxTriggerFrequency_Hz,
  Device_LineRate_Hz }

  Floating point properties of the device that can be retrieved with the function getDevicePropertyFloat.
- enum DevicePropertyInt {
  Device_SpectrumElements,
  Device_BytesPerElement,
  Device_MaxLiveVolumeRenderingScans,
  Device_BitDepth,
  Device_NumOfCameras,
  Device_RevisionNumber }

  Integer properties of the device that can be retrieved with the function getDevicePropertyInt.
- enum DevicePropertyString {
  Device_Type,
  Device_Series,
  Device_SerialNumber,
  Device_HardwareConfig }

  String-properties of the device that can be retrieved with the function getDevicePropertyString.
- enum DeviceFlag {
  Device_On,
  Device_CameraAvailable,
  Device_SLDAvailable,
  Device_SLDStatus,
  Device_LaserDiodeStatus,
  Device_CameraShowScanPattern,
  Device_ProbeControllerAvailable,
  Device_DataIsSigned,
  Device_IsSweptSource }

Boolean properties of the device that can be retrieved with the function getDeviceFlag.

- enum ScanAxis {
ScanAxis_X = 0,
ScanAxis_Y = 1 }

  Axis selection for the function moveScanner.
- enum DeviceTriggerType {
Trigger_FreeRunning,
Trigger_TrigBoard_ExternalStart,
Trigger_External_AScan }

  Enum identifying trigger types for the OCT system.
- enum RefstageStatus {
RefStage_Status_Idle = 0,
RefStage_Status_Homing = 1,
RefStage_Status_Moving = 2,
RefStage_Status_MovingTo = 3,
RefStage_Status_Stopping = 4,
RefStage_Status_NotAvailable = 5,
RefStage_Status_Undefined = -1 }

  Defines the status of the motorized reference stage.
- enum RefstageSpeed {
RefStage_Speed_Slow = 0,
RefStage_Speed_Fast = 1,
RefStage_Speed_VerySlow = 2,
RefStage_Speed_VeryFast = 3 }

  Defines the velocity of movement for the motorized reference stage.
- enum RefstageWaitForMovement {
RefStage_Movement_Wait = 0,
RefStage_Movement_Continue = 1 }

  Defines the behaviour whether the the function should wait until the movement of the motorized reference stage has stopped to return.
- enum RefstageMovementDirection {
RefStage_MoveShorter = 0,
RefStage_MoveLonger = 1 }

  Defines the direction of movement for the motorized reference stage. Please note that not in all systems a motorized reference stage is present.
- enum LightSourceState {
Activating,
On,
Off }

  Values that define the state of the light source.
- enum WaitForCompletion {
Wait = 0,
Continue = 1 }

  Defines the behaviour whether a function should wait for the operation to complete or return immediately.

Functions

- SPECTRALRADAR_API OCTDeviceHandle initDevice (void)

  Initializes the installed device.
- SPECTRALRADAR_API int getDevicePropertyInt (OCTDeviceHandle Dev, DevicePropertyInt Selection)

  Returns properties of the device belonging to the specfied OCTDeviceHandle.
- SPECTRALRADAR_API const char ∗ getDevicePropertyString (OCTDeviceHandle Dev, DevicePropertyString Selection)

  Returns properties of the device belonging to the specfied OCTDeviceHandle.

- SPECTRALRADAR_API double getDevicePropertyFloat (OCTDeviceHandle Dev, DevicePropertyFloat Selection)

  Returns properties of the device belonging to the specfied OCTDeviceHandle.
- SPECTRALRADAR_API BOOL getDeviceFlag (OCTDeviceHandle Dev, DeviceFlag Selection)

  Returns properties of the device belonging to the specified OCTDeviceHandle.
- SPECTRALRADAR_API void setDeviceFlag (OCTDeviceHandle Dev, DeviceFlag Selection, BOOL Value)

  Sets the selcted flag of the device belonging to the specified OCTDeviceHandle.
- SPECTRALRADAR_API void closeDevice (OCTDeviceHandle Dev)

  Closes the device opened previously with initDevice.
- SPECTRALRADAR_API void moveScanner (OCTDeviceHandle Dev, ProbeHandle Probe, ScanAxis Axis, double Position_mm)

  Manually moves the scanner to a given position.
- SPECTRALRADAR_API void moveScannerToApoPosition (OCTDeviceHandle Dev, ProbeHandle Probe)

  Moves the scanner to the apodization position.
- SPECTRALRADAR_API int getNumberOfDevicePresetCategories (OCTDeviceHandle Dev)

  If the hardware supports multiple presets, the funciton returns the number of categories in which presets can be set.
- SPECTRALRADAR_API const char ∗ getDevicePresetCategoryName (OCTDeviceHandle Dev, int Category)

  Gets a descriptor/name for the respective preset category.
- SPECTRALRADAR_API int getDevicePresetCategoryIndex (OCTDeviceHandle Dev, const char ∗Name)

  Gets the index of a preset category from the name of the category.
- SPECTRALRADAR_API void setDevicePreset (OCTDeviceHandle Dev, int Category, ProbeHandle Probe, ProcessingHandle Proc, int Preset)

  Sets the preset of the device. Using presets the sensitivity and acquisition speed of the device can be influenced.
- SPECTRALRADAR_API int getDevicePreset (OCTDeviceHandle Dev, int Category)

  Gets the currently used device preset.
- SPECTRALRADAR_API const char ∗ getDevicePresetDescription (OCTDeviceHandle Dev, int Category, int Preset)

  Returns a description of the selected device preset. Using the description more information about sensitivity and acquisition speed of the respective set can be found.
- SPECTRALRADAR_API int getNumberOfDevicePresets (OCTDeviceHandle Dev, int Category)

  Returns the number of available device presets.
- SPECTRALRADAR_API void setRequiredSLDOnTime_s (int Time_s)

  Sets the time the SLD needs to be switched on before any measurement can be started. Default is 3 seconds.
- SPECTRALRADAR_API void resetCamera (void)

  Resets the spectrometer camera.
- SPECTRALRADAR_API BOOL isDeviceAvailable (void)

  Returns whethter any supported Base-Unit is available.
- SPECTRALRADAR_API double QuantumEfficiency (OCTDeviceHandle Dev, double CenterWavelength_nm, double PowerIntoSpectrometer_W, DataHandle Spectrum_e)

  Calculates the quantum efficiency from the processed input spectrum in the Data instance.
- SPECTRALRADAR_API BOOL isRefstageAvailable (OCTDeviceHandle Dev)

  Returns whether a motorized reference stage is available or not for the specified device. Please note that a motorized reference stage is not included in all systems.
- SPECTRALRADAR_API RefstageStatus getRefstageStatus (OCTDeviceHandle Dev)

  Returns the current status of the reference stage, e.g. if it is moving.
- SPECTRALRADAR_API double getRefstageLength_mm (OCTDeviceHandle Dev, ProbeHandle Probe)

  Returns the total length in mm of the reference stage.
- SPECTRALRADAR_API double getRefstagePosition_mm (OCTDeviceHandle Dev, ProbeHandle Probe)

  Returns the current position in mm of the reference stage.
- SPECTRALRADAR_API void homeRefstage (OCTDeviceHandle Dev, RefstageWaitForMovement WaitFor←Moving)

  Homes the reference stage to calibrate the zero position.

- SPECTRALRADAR_API void moveRefstageToPosition_mm (OCTDeviceHandle Dev, ProbeHandle Probe, double Pos_mm, RefstageSpeed Speed, RefstageWaitForMovement WaitForMoving)

  Moves the reference stage to the specified position in mm.
- SPECTRALRADAR_API void moveRefstage_mm (OCTDeviceHandle Dev, ProbeHandle Probe, double Length_mm, RefstageMovementDirection Direction, RefstageSpeed Speed, RefstageWaitForMovement WaitForMoving)

  Moves the reference stage with the specified length in mm.
- SPECTRALRADAR_API void startRefstageMovement (OCTDeviceHandle Dev, RefstageMovementDirection Direction, RefstageSpeed Speed)

  Starts the movement of the reference stage with the chosen speed. Please note that the movement does not stop until stopRefstageMovement is called.
- SPECTRALRADAR_API void stopRefstageMovement (OCTDeviceHandle Dev)

  Stops the movement of the reference stage.
- SPECTRALRADAR_API void setRefstageSpeed (OCTDeviceHandle Dev, RefstageSpeed Speed)

  Sets the velocity of the movement of the reference stage.
- SPECTRALRADAR_API void setRefstageStatusCallback (OCTDeviceHandle Dev, cbRefstageStatusChanged Callback)

  Registers the callback to get notified if the reference stage status changed.
- SPECTRALRADAR_API void setRefstagePosChangedCallback (OCTDeviceHandle Dev, cbRefstagePositionChanged Callback)

  Registers the callback to get notified if the reference stage position changed.
- SPECTRALRADAR_API double getRefstageMinPosition_mm (OCTDeviceHandle Dev, ProbeHandle Probe)

  Returns the minimal position in mm the reference stage can move to.
- SPECTRALRADAR_API double getRefstageMaxPosition_mm (OCTDeviceHandle Dev, ProbeHandle Probe)

  Returns the maximal position in mm the reference stage can move to.
- SPECTRALRADAR_API void setLightSourceTimeoutCallback (OCTDeviceHandle Dev, lightSourceStateCallback Callback)

  Sets a callback function that will be invoked by the SDK whenever the state of the lightsource of the device changes.
- SPECTRALRADAR_API void setLightSourceTimeout_s (OCTDeviceHandle Dev, double Timeout)

  Sets a the timeout in seconds, after which the OCT lightsource will be turned off if no scanning is performed.
- SPECTRALRADAR_API double getLightSourceTimeout_s (OCTDeviceHandle Dev)

  Gets a the timeout in seconds, after which the OCT lightsource will be turned off if no scanning is performed.
- SPECTRALRADAR_API void updateAfterPresetChange (OCTDeviceHandle Dev, ProbeHandle Probe, ProcessingHandle Proc, int CameraIndex)

  Updates the processing handle after preset change. Please use setDevicePreset first for the first camera (with index 0) and this function to update the corresponding ProcessingHandle for the second camera (with index 1).

5.5.1 Detailed Description

Functions providing direct access to OCT Hardware functionality.

5.5.2 Typedef Documentation

5.5.2.1 typedef void(__stdcall∗ lightSourceStateCallback) (LightSourceState)

Defines the function prototype for the light source callback(see also setLightSourceTimeoutCallback()). The argument contains the current state of the light source.

Parameters

LightSourceState	Current state of the light source

5.5.2.2 OCTDeviceHandle

The OCTDeviceHandle type is used as Handle for using the SpectralRadar.

5.5.3 Enumeration Type Documentation

5.5.3.1 enum DeviceFlag

Boolean properties of the device that can be retrieved with the function getDeviceFlag.

Enumerator

Device_On The type name of the device.

Device_CameraAvailable Specifies if there is a video camera available.

Device_SLDAvailable Specifies if there is a SLD available.

Device_SLDStatus Status of the SLD, either on (true) or off (false)

Device_LaserDiodeStatus Status of the laser diode, either on (true) or off (false)

Warning

Not all devices are equiped

Device_CameraShowScanPattern Parameter for the overlay of the video camera which shows the scan pattern in red.

Device_ProbeControllerAvailable Gives information whether a probe controller (with buttons) is available.

Device_DataIsSigned Flag indicating if the data is signed.

Device_IsSweptSource Flag indicating whether system is a swept source (or spectral domain) system.

5.5.3.2 enum DevicePropertyFloat

Floating point properties of the device that can be retrieved with the function getDevicePropertyFloat.

Enumerator

Device_FullWellCapacity The full well capacity of the device.

Device_zSpacing The spacing between two pixels in an A-scan.

Device_zRange The maximum measurement range for an A-scan.

Device_SignalAmplitudeMin_dB The minimum expected dB value for final data.

Device_SignalAmplitudeLow_dB The typical low dB value for final data.

Device_SignalAmplitudeHigh_dB The typical high dB value for final data.

Device_SignalAmplitudeMax_dB The maximum expected dB value for final data.

Device_BinToElectronScaling Scaling factor between binary raw data and electrons/photons.

Device_Temperature Internal device temperature in degrees C.

Device_SLD_OnTime_sec Absolute power-on time of the SLD since first start in seconds.

Device_CenterWavelength_nm The center wavelength of the device.

Device_SpectralWidth_nm The approximate spectral width of the spectrometer.

Device_MaxTriggerFrequency_Hz Maximal valid trigger frequency depending on the chosen camera preset.

5.5.3.3 enum DevicePropertyInt

Integer properties of the device that can be retrieved with the function getDevicePropertyInt.

Enumerator

Device_SpectrumElements The number of pixels provided by the spectrometer.

Device_BytesPerElement The number of bytes one element of the spectrum occupies.

Device_MaxLiveVolumeRenderingScans The maximum number of scans per dimension in the live volume rendering mode.

Device_BitDepth Bit depth of the DAQ.

Device_NumOfCameras Number of spectrometer cameras.

Device_RevisionNumber Revision number of the device.

5.5.3.4 enum DevicePropertyString

String-properties of the device that can be retrieved with the function getDevicePropertyString.

Enumerator

Device_Type The type name of the device.

Device_Series The series of the device.

Device_SerialNumber Serial number of the device.

Device_HardwareConfig Hardware Config of the currently used device.

5.5.3.5 enum DeviceTriggerType

Enum identifying trigger types for the OCT system.

Warning

Not all trigger types are available for all different systems. To check whether the specified trogger mode is available or not please use isTriggerModeAvailable

Enumerator

Trigger_FreeRunning Standard mode.

Trigger_TrigBoard_ExternalStart Used to trigger the start of an acquisition. Additional hardware is needed.

Trigger_External_AScan Mode to trigger the acquisition of each A-scan. An external trigger signal is needed. Please see the software manual for detailed information.

5.5.3.6 enum LightSourceState

Values that define the state of the light source.

5.5.3.7 enum RefstageMovementDirection

Defines the direction of movement for the motorized reference stage. Please note that not in all systems a motorized reference stage is present.

Enumerator

RefStage_MoveShorter Shortens reference arm length.

RefStage_MoveLonger Extends reference arm length.

5.5.3.8 enum RefstageSpeed

Defines the velocity of movement for the motorized reference stage.

Enumerator

RefStage_Speed_Slow Slow speed (∼0.4mm/s)

RefStage_Speed_Fast Fast speed (∼1.8mm/s)

RefStage_Speed_VerySlow Very slow speed.

RefStage_Speed_VeryFast Very fast speed (∼13mm/s)

5.5.3.9 enum RefstageStatus

Defines the status of the motorized reference stage.

Enumerator

RefStage_Status_Idle The reference stage is not busy and available for a task.

RefStage_Status_Homing The reference stage is in its homing process. Please wait until this process is finished.

RefStage_Status_Moving The reference stage is moving, you can stop this movement with stopRefstageMovement.

RefStage_Status_MovingTo The reference stage it moving to a certain position. Please wait until this process is finished.

RefStage_Status_Stopping The reference stage is in the stopping process after stopRefstageMovement was called. Please wait until this process is finished.

RefStage_Status_NotAvailable The reference stage is not available any more.

RefStage_Status_Undefined The status of the reference stage is not defined.

5.5.3.10 enum RefstageWaitForMovement

Defines the behaviour whether the the function should wait until the movement of the motorized reference stage has stopped to return.

Enumerator

RefStage_Movement_Wait Function waits until the movement has stopped before it returns.

RefStage_Movement_Continue The movement of the motorized reference stage will be started and runs in another thread. The function returns while the reference stage is still moving.

5.5.3.11 enum ScanAxis

Axis selection for the function moveScanner.

Enumerator

ScanAxis_X	X-Axis of the scanner.
ScanAxis_Y	Y-Axis of the scanner.

5.5.3.12 enum WaitForCompletion

Defines the behaviour whether a function should wait for the operation to complete or return immediately.

5.5.4 Function Documentation

5.5.4.1 void closeDevice (OCTDeviceHandle *Dev*)

Closes the device opened previously with initDevice.

Parameters

`in`	*Dev*	An OCT device handle (OCTDeviceHandle). If the handle is a nullptr, this function does nothing. In most cases, this handle will have been previously generated with the function initDevice.

5.5.4.2 BOOL getDeviceFlag (OCTDeviceHandle *Dev,* DeviceFlag *Selection*)

Returns properties of the device belonging to the specified OCTDeviceHandle.

Parameters

`in`	*Selection*	The desired flag.

Returns

The value of the desired flag.

Parameters

`in`	*Dev*	A valid (non null) OCT device handle (OCTDeviceHandle), previously generated with the function initDevice.

5.5.4.3 int getDevicePreset (OCTDeviceHandle *Dev,* int *Category*)

Gets the currently used device preset.

Returns

The current device preset index. Different devices support different preset categories (gain, speed, etc). When getting or setting a preset, the right category must be provided. To get the number of supported categories, use the function getNumberOfDevicePresetCategories. To get a name (i.e. a description of the category), use the function getDevicePresetCategoryName. To get the index of a supported category, provided you know the name, use the function getDevicePresetCategoryIndex (this is the index need when getting or setting a preset of a given category).
A description of the device preset associated with a particular index can be obtained by invoking the function getDevicePresetDescription. The total number of presets for the active device can be retrieved with the function getNumberOfDevicePresets.

Parameters

in	*Dev*	A valid (non null) OCT device handle (OCTDeviceHandle), previously generated with the function initDevice.
in	*Category*	An index describing the preset category in the range between 0 and the number of preset categories minus 1, given by getNumberOfDevicePresetCategories

5.5.4.4 const char ∗ getDevicePresetCategoryIndex (OCTDeviceHandle *Dev,* const char ∗ *Name*)

Gets the index of a preset category from the name of the category.

Parameters

in	*Dev*	A valid (non null) OCT device handle (OCTDeviceHandle), previously generated with the function initDevice.
in	*Name*	The name of the device preset category.

Returns

An index describing the preset category in the range between 0 and the number of preset categories minus 1, given by getNumberOfDevicePresetCategories.

Different devices support different preset categories (gain, speed, etc). When getting or setting a preset, the right category must be provided. To get the number of supported categories, use the function getNumberOfDevicePresetCategories. To get a name (i.e. a description of the category), use the function getDevicePresetCategoryName. To get the index of a supported category, provided you know the name, use the function getDevicePresetCategoryIndex (this is the index need when getting or setting a preset of a given category).

5.5.4.5 const char ∗ getDevicePresetCategoryName (OCTDeviceHandle *Dev,* int *Category*)

Gets a descriptor/name for the respective preset category.

Parameters

in	*Dev*	A valid (non null) OCT device handle (OCTDeviceHandle), previously generated with the function initDevice.
in	*Category*	An index describing the preset category in the range between 0 and the number of preset categories minus 1, given by getNumberOfDevicePresetCategories

Returns

The name of the requested device preset category.

Different devices support different preset categories (gain, speed, etc). When getting or setting a preset, the right category must be provided. To get the number of supported categories, use the function getNumberOfDevicePresetCategories. To get a name (i.e. a description of the category), use the function getDevicePresetCategoryName. To get the index of a supported category, provided you know the name, use the function getDevicePresetCategoryIndex (this is the index need when getting or setting a preset of a given category).

5.5.4.6 const char ∗ getDevicePresetDescription (OCTDeviceHandle *Dev,* int *Category,* int *Preset*)

Returns a description of the selected device preset. Using the description more information about sensitivity and acquisition speed of the respective set can be found.

Returns

A text describing the preset (speed, sensitivity). This pointer refers to memory owned by SpectralRadar.dll. The er should not attempt to free it. Different devices support different preset categories (gain, speed, etc). When getting or setting a preset, the right category must be provided. To get the number of supported categories, use the function getNumberOfDevicePresetCategories. To get a name (i.e. a description of the category), use the function getDevicePresetCategoryName. To get the index of a supported category, provided you know the name, use the function getDevicePresetCategoryIndex (this is the index need when getting or setting a preset of a given category).
The current device preset can be obtained by invoking the function getDevicePreset. The total number of presets for the active device can be retrieved with the function getNumberOfDevicePresets.

Parameters

in	*Dev*	A valid (non null) OCT device handle (OCTDeviceHandle), previously generated with the function initDevice.
in	*Category*	An index describing the preset category in the range between 0 and the number of preset categories minus 1, given by getNumberOfDevicePresetCategories.
in	*Preset*	The index of the preset.

5.5.4.7 double getDevicePropertyFloat (OCTDeviceHandle *Dev,* DevicePropertyFloat *Selection*)

Returns properties of the device belonging to the specfied OCTDeviceHandle.

Parameters

in	*Selection*	The desired property.

Returns

The value of the desired property.

Parameters

in	*Dev*	A valid (non null) OCT device handle (OCTDeviceHandle), previously generated with the function initDevice.

5.5.4.8 int getDevicePropertyInt (**OCTDeviceHandle** *Dev,* **DevicePropertyInt** *Selection*)

Returns properties of the device belonging to the specfied OCTDeviceHandle.

Parameters

`in`	*Selection*	The desired property.

Returns

The value of the desired property.

Parameters

`in`	*Dev*	A valid (non null) OCT device handle (OCTDeviceHandle), previously generated with the function initDevice.

5.5.4.9 const char ∗ getDevicePropertyString (**OCTDeviceHandle** *Dev,* **DevicePropertyString** *Selection*)

Returns properties of the device belonging to the specfied OCTDeviceHandle.

Parameters

`in`	*Selection*	The desired property.

Returns

The value of the desired property. This memory pointed belongs to SpectralRadar.dll and the user should not attempt to free it.

Parameters

`in`	*Dev*	A valid (non null) OCT device handle (OCTDeviceHandle), previously generated with the function initDevice.

5.5.4.10 double getLightSourceTimeout_s (**OCTDeviceHandle** *Dev*)

Gets a the timeout in seconds, after which the OCT lightsource will be turned off if no scanning is performed.

Parameters

Dev	the OCTDeviceHandle that was initially provided by initDevice.

Returns

Time in seconds after which the lightsource will be turned off.

5.5.4.11 int getNumberOfDevicePresetCategories (OCTDeviceHandle *Dev*)

If the hardware supports multiple presets, the funciton returns the number of categories in which presets can be set.

Parameters

in	*Dev*	A valid (non null) OCT device handle (OCTDeviceHandle), previously generated with the function initDevice.

Different devices support different preset categories (gain, speed, etc). When getting or setting a preset, the right category must be provided. To get the number of supported categories, use the function getNumberOfDevicePresetCategories. To get a name (i.e. a description of the category), use the function getDevicePresetCategoryName. To get the index of a supported category, provided you know the name, use the function getDevicePresetCategoryIndex (this is the index need when getting or setting a preset of a given category).

5.5.4.12 int getNumberOfDevicePresets (OCTDeviceHandle *Dev*, int *Category*)

Returns the number of available device presets.

Returns

The number of presets supported by the OCT device. Different devices support different preset categories (gain, speed, etc). When getting or setting a preset, the right category must be provided. To get the number of supported categories, use the function getNumberOfDevicePresetCategories. To get a name (i.e. a description of the category), use the function getDevicePresetCategoryName. To get the index of a supported category, provided you know the name, use the function getDevicePresetCategoryIndex (this is the index need when getting or setting a preset of a given category).
The current device preset can be obtained by invoking the function getDevicePreset. A description of the device preset associated with a particular index can be obtained by invoking the function getDevicePresetDescription.

Parameters

in	*Dev*	A valid (non null) OCT device handle (OCTDeviceHandle), previously generated with the function initDevice.
in	*Category*	An index describing the preset category in the range between 0 and the number of preset categories minus 1, given by getNumberOfDevicePresetCategories.

5.5.4.13 double getRefstageLength_mm (OCTDeviceHandle *Dev*, ProbeHandle *Probe*)

Returns the total length in mm of the reference stage.

Parameters

Dev	the OCTDeviceHandle that was initially provided by initDevice.
Probe	the ProbeHandle that was initially provided by initProbe.

5.5.4.14 double getRefstageMaxPosition_mm (OCTDeviceHandle *Dev*, ProbeHandle *Probe*)

Returns the maximal position in mm the reference stage can move to.

Parameters

Dev	the OCTDeviceHandle that was initially provided by initDevice.
Probe	the ProbeHandle that is currently.

5.5.4.15 double getRefstageMinPosition_mm (OCTDeviceHandle *Dev,* ProbeHandle *Probe*)

Returns the minimal position in mm the reference stage can move to.

Parameters

Dev	the OCTDeviceHandle that was initially provided by initDevice.
Probe	the ProbeHandle that is currently.

5.5.4.16 double getRefstagePosition_mm (OCTDeviceHandle *Dev,* ProbeHandle *Probe*)

Returns the current position in mm of the reference stage.

Parameters

Dev	the OCTDeviceHandle that was initially provided by initDevice.
Probe	the ProbeHandle that was initially provided by initProbe.

5.5.4.17 RefstageStatus getRefstageStatus (OCTDeviceHandle *Dev*)

Returns the current status of the reference stage, e.g. if it is moving.

Parameters

Dev	the OCTDeviceHandle that was initially provided by initDevice.

5.5.4.18 void homeRefstage (OCTDeviceHandle *Dev,* RefstageWaitForMovement *WaitForMoving*)

Homes the reference stage to calibrate the zero position.

Parameters

Dev	the OCTDeviceHandle that was initially provided by initDevice.
WaitForMoving	specifies whether to wait for the end of the homing process before returning from the function or not.

5.5.4.19 OCTDeviceHandle initDevice (void)

Initializes the installed device.

Returns

Handle to the initialized OCT device (OCTDeviceHandle).

This function attempts to discover the hardware specified in the file SpectralRadar.ini. The components of the hardware are represented on the software side by plugins. The discovering process will log its activity. As some of the messages may appear to be error messages to the untrained eye, it is recommended to invoke the function getError to check if this function actually succeeded.

5.5.4.20 BOOL isDeviceAvailable (void)

Returns whethter any supported Base-Unit is available.

This function attemps to communicate with the device, and returns TRUE if a minimum of working functionality can be guaranted, FALSE otherwise. This function can be invoked as many times as desired (e.g. in a polling strategy) without side effects.

5.5.4.21 SPECTRALRADAR_API BOOL isRefstageAvailable (OCTDeviceHandle *Dev*)

Returns whether a motorized reference stage is available or not for the specified device. Please note that a motorized reference stage is not included in all systems.

Parameters

Dev	the OCTDeviceHandle that was initially provided by initDevice.

5.5.4.22 void moveRefstage_mm (OCTDeviceHandle *Dev,* ProbeHandle *Probe,* double *Length_mm,* RefstageMovementDirection *Direction,* RefstageSpeed *Speed,* RefstageWaitForMovement *WaitForMoving*)

Moves the reference stage with the specified length in mm.

Parameters

Dev	the OCTDeviceHandle that was initially provided by initDevice.
Probe	the ProbeHandle that was initially provided by initProbe.
Length_mm	gives the desired length in mm relative to the current position
Direction	is the specified direction of the movement with RefstageMovementDirection.
Speed	is the velocity of the reference stage movement specified with RefstageSpeed
WaitForMoving	defines whether the function should wait until the movement of the reference stage has stopped or not until it returns

5.5.4.23 void moveRefstageToPosition_mm (OCTDeviceHandle *Dev,* ProbeHandle *Probe,* double *pos_mm,* RefstageSpeed *Speed,* RefstageWaitForMovement *WaitForMoving*)

Moves the reference stage to the specified position in mm.

Parameters

Dev	the OCTDeviceHandle that was initially provided by initDevice.
Probe	the ProbeHandle that was initially provided by initProbe.

Parameters

pos_mm	gives the desired position in mm
Speed	is the velocity of the reference stage movement specified with RefstageSpeed
WaitForMoving	defines whether the function should wait until the movement of the reference stage has stopped or not until it returns

5.5.4.24 void moveScanner (**OCTDeviceHandle** *Dev,* **ProbeHandle** *Probe,* **ScanAxis** *Axis,* double *Position_mm*)

Manually moves the scanner to a given position.

Parameters

in	*Dev*	A valid (non null) OCT device handle (OCTDeviceHandle), previously generated with the function initDevice.
in	*Probe*	A handle to the probe (ProbeHandle), whose galvo position is to be set.
in	*Axis*	the axis in which you want to set the position manually
in	*Position_mm*	the actual position in mm you want to move the galvo to.

5.5.4.25 void moveScannerToApoPosition (**OCTDeviceHandle** *Dev,* **ProbeHandle** *Probe*)

Moves the scanner to the apodization position.

Parameters

in	*Dev*	A valid (non null) OCT device handle (OCTDeviceHandle), previously generated with the function initDevice.
in	*Probe*	A handle to the probe (ProbeHandle); whose galvo position is to be set.

5.5.4.26 double QuantumEfficiency (**OCTDeviceHandle** *Dev,* double *CenterWavelength_nm,* double *PowerIntoSpectrometer_W,* **DataHandle** *Spectrum_e*)

Calculates the quantum efficiency from the processed input spectrum in the Data instance.

5.5.4.27 void resetCamera (void)

Resets the spectrometer camera.

5.5.4.28 void setDeviceFlag (**OCTDeviceHandle** *Dev,* **DeviceFlag** *Selection,* **BOOL** *Value*)

Sets the selcted flag of the device belonging to the specified OCTDeviceHandle.

Parameters

in	*Selection*	The desired flag.
in	*Value*	The value of the desired flag.
in	*Dev*	A valid (non null) OCT device handle (OCTDeviceHandle), previously generated with the function initDevice.

5.5.4.29 void setDevicePreset (OCTDeviceHandle *Dev*, int *Category*, ProbeHandle *Probe*, ProcessingHandle *Proc*, int *Preset*)

Sets the preset of the device. Using presets the sensitivity and acquisition speed of the device can be influenced.

Parameters

in	*Dev*	A valid (non null) OCT device handle (OCTDeviceHandle), previously generated with the function initDevice.
in	*Category*	An index describing the preset category in the range between 0 and the number of preset categories minus 1, given by getNumberOfDevicePresetCategories
in	*Probe*	A handle to the probe (ProbeHandle); whose galvo position is to be set.
in	*Proc*	A valid (non null) processing handle.
in	*Preset*	The index of the preset.

Different devices support different preset categories (gain, speed, etc). When getting or setting a preset, the right category must be provided. To get the number of supported categories, use the function getNumberOfDevicePresetCategories. To get a name (i.e. a description of the category), use the function getDevicePresetCategoryName. To get the index of a supported category, provided you know the name, use the function getDevicePresetCategoryIndex (this is the index need when getting or setting a preset of a given category).

5.5.4.30 void setLightSourceTimeout_s (OCTDeviceHandle *Dev*, double *Timeout*)

Sets a the timeout in seconds, after which the OCT lightsource will be turned off if no scanning is performed.

Parameters

Dev	the OCTDeviceHandle that was initially provided by initDevice.
Timeout	Time in seconds after which the lightsource will be turned off.

5.5.4.31 void setLightSourceTimeoutCallback (OCTDeviceHandle *Dev*, lightSourceStateCallback *Callback*)

Sets a callback function that will be invoked by the SDK whenever the state of the lightsource of the device changes.

Parameters

Dev	the OCTDeviceHandle that was initially provided by initDevice.
Callback	the lightSourceStateCallback that will be called when state of the lightsource changes

5.5.4.32 void setRefstagePosChangedCallback (OCTDeviceHandle *Dev*, cbRefstagePositionChanged *Callback*)

Registers the callback to get notified if the reference stage position changed.

Parameters

Dev	the OCTDeviceHandle that was initially provided by initDevice.
Callback	to register.

5.5.4.33 void setRefstageSpeed (OCTDeviceHandle *Dev,* RefstageSpeed *Speed*)

Sets the velocity of the movement of the reference stage.

Parameters

Dev	the OCTDeviceHandle that was initially provided by initDevice.
Speed	the chosen velocity of the movement.

5.5.4.34 void setRefstageStatusCallback (OCTDeviceHandle *Dev,* cbRefstageStatusChanged *Callback*)

Registers the callback to get notified if the reference stage status changed.

Parameters

Dev	the OCTDeviceHandle that was initially provided by initDevice.
Callback	to register.

5.5.4.35 void setRequiredSLDOnTime_s (int *Time_s*)

Sets the time the SLD needs to be switched on before any measurement can be started. Default is 3 seconds.

Parameters

in	*Time↩ _s*	Minimum required on time in seconds.

5.5.4.36 void startRefstageMovement (OCTDeviceHandle *Dev,* RefstageMovementDirection *Direction,* RefstageSpeed *Speed*)

Starts the movement of the reference stage with the chosen speed. Please note that the movement does not stop until stopRefstageMovement is called.

Parameters

Dev	the OCTDeviceHandle that was initially provided by initDevice.
Direction	is the specified direction of the movement with RefstageMovementDirection.
Speed	is the velocity of the reference stage movement specified with RefstageSpeed.

5.5.4.37 void stopRefstageMovement (OCTDeviceHandle *Dev*)

Stops the movement of the reference stage.

Parameters

Dev	the OCTDeviceHandle that was initially provided by initDevice.

5.5.4.38 void updateAfterPresetChange (OCTDeviceHandle *Dev*, ProbeHandle *Probe*, ProcessingHandle *Proc*, int *CameraIndex*)

Updates the processing handle after preset change. Please use setDevicePreset first for the first camera (with index 0) and this function to update the corresponding ProcessingHandle for the second camera (with index 1).

Parameters

in	*Dev*	A valid (non null) OCT device handle (OCTDeviceHandle), previously generated with the function initDevice.
in	*Probe*	A handle to the probe (ProbeHandle); whose galvo position is to be set.
in	*Proc*	A valid (non null) processing handle.
in	*CameraIndex*	The index of the camera. The function setDevicePreset updates the ProcessingHandle for the first camera (with index 0) automatically.

5.6 Internal Values

Functions for access to all kinds of Digital-to-Analog and Analog-to-Digital on the device.

Functions

- SPECTRALRADAR_API int getNumberOfInternalDeviceValues (OCTDeviceHandle Dev)

  Returns the number of Analog-to-Digital Converter present in the device.
- SPECTRALRADAR_API void getInternalDeviceValueName (OCTDeviceHandle Dev, int Index, char ∗Name, int NameStringSize, char ∗Unit, int UnitStringSize)

  Returns names and unit for the specified Analog-to-Digital Converter.
- SPECTRALRADAR_API double getInternalDeviceValueByName (OCTDeviceHandle Dev, const char ∗Name)

  Returns the value of the specified Analog-to-Digital Converter (ADC);.
- SPECTRALRADAR_API double getInternalDeviceValueByIndex (OCTDeviceHandle Dev, int Index)

  Returns the value of the selected ADC.
- SPECTRALRADAR_API void setInternalDeviceValueByIndex (OCTDeviceHandle Dev, int Index, double Value)

  Sets the value of the selected ADC.

5.6.1 Detailed Description

Functions for access to all kinds of Digital-to-Analog and Analog-to-Digital on the device.

5.6.2 Function Documentation

5.6.2.1 double getInternalDeviceValueByIndex (**OCTDeviceHandle** *Dev,* int *Index*)

Returns the value of the selected ADC.

Parameters

in	*Dev*	A valid (non null) OCT device handle (OCTDeviceHandle), previously generated with the function initDevice.
in	*Index*	The index of the internal device value.

Returns

The internal device value.

The index is a running integer number, starting with 0, smaller than the number specified by getNumberOfInternalDeviceValues.

5.6.2.2 double getInternalDeviceValueByName (**OCTDeviceHandle** *Dev,* const char ∗ *Name*)

Returns the value of the specified Analog-to-Digital Converter (ADC);.

Parameters

in	*Dev*	A valid (non null) OCT device handle (OCTDeviceHandle), previously generated with the function initDevice.
in	*Name*	Name of the internal device value.

Returns

The internal device value.

The ADC is specified by the name returned by getInternalDeviceValueName.

5.6.2.3 void getInternalDeviceValueName (OCTDeviceHandle *Dev,* int *Index,* char ∗ *Name,* int *NameStringSize,* char ∗ *Unit,* int *UnitStringSize*)

Returns names and unit for the specified Analog-to-Digital Converter.

Parameters

in	*Dev*	A valid (non null) OCT device handle (OCTDeviceHandle), previously generated with the function initDevice.
in	*Index*	The index of the internal value whose name and unit are sought.
out	*Name*	Name of the internal device value. If this pointer is null, it will not be used. If it is non-null, it must point to a memory are at least as large as `NameStringSize` bytes.
in	*NameStringSize*	The maximal number of bytes that will be copied onto the array holding the name.
out	*Unit*	Unit of the internal device value. If this pointer is null, it will not be used. If it is non-null, it must point to a memory are at least as large as `UnitStringSize` bytes.
in	*UnitStringSize*	The maximal number of bytes that will be copied onto the array holding the unit.

The index is a running number, starting with 0, smaller than the number specified by getNumberOfInternalDeviceValues.

5.6.2.4 int getNumberOfInternalDeviceValues (OCTDeviceHandle *Dev*)

Returns the number of Analog-to-Digital Converter present in the device.

Returns

The number of Analog-to-Digital Converter present in the device.

Parameters

in	*Dev*	A valid (non null) OCT device handle (OCTDeviceHandle), previously generated with the function initDevice.

5.6.2.5 void setInternalDeviceValueByIndex (OCTDeviceHandle *Dev,* int *Index,* double *Value*)

Sets the value of the selected ADC.

Parameters

in	*Dev*	A valid (non null) OCT device handle (OCTDeviceHandle), previously generated with the function initDevice.
in	*Index*	The index of the internal device value.
in	*Value*	The internal device value.

The index is running number, starting with 0, smaller than the number specified by getNumberOfInternalDeviceValues.

5.7 Pattern Factory/Probe

Functions setting up a probe that can be used to create scan patterns.

Typedefs

- typedef struct C_Probe ∗ ProbeHandle

  Handle for controlling the galvo scanner.
- typedef void(__stdcall ∗ cbProbeMessageReceived) (int)

  The prototype for callback functions registered for probe button events. As of the creation time of this document, only the OCTH probe is equipped with buttons.

Enumerations

- enum ProbeParameterFloat {
  Probe_FactorX,
  Probe_OffsetX,
  Probe_FactorY,
  Probe_OffsetY,
  Probe_FlybackTime_Sec,
  Probe_ExpansionTime_Sec,
  Probe_RotationTime_Sec,
  Probe_ExpectedScanRate_Hz,
  Probe_CameraScalingX,
  Probe_CameraOffsetX,
  Probe_CameraScalingY,
  Probe_CameraOffsetY,
  Probe_CameraAngle,
  Probe_RangeMaxX,
  Probe_RangeMaxY,
  Probe_MaximumSlope_XY,
  Probe_SpeckleSize,
  Probe_ApoVoltageX,
  Probe_ApoVoltageY,
  Probe_ReferenceStageOffset,
  Probe_FiberOpticalPathLength_mm,
  Probe_ProbeOpticalPathLength_mm,
  Probe_ObjectiveOpticalPathLength_mm,
  Probe_ObjectiveFocalLength_mm }

  Parameters describing the behaviour of the Probe, such as calibration factors and scan parameters.
- enum ProbeParameterString {
  Probe_Name,
  Probe_SerialNumber,
  Probe_Description,
  Probe_Objective }

  Parameters describing the composition of the probe. These properties refer to a probe that has already been created and for which a valid ProbeHandle has been obtained.
- enum ProbeParameterInt {
  Probe_ApodizationCycles,
  Probe_Oversampling,
  Probe_Oversampling_SlowAxis,
  Probe_SpeckleReduction }

  Parameters describing the behaviour of the Probe, such as calibration factors and scan parameters.

- enum ProbeFlag {
  Probe_CameraInverted_X,
  Probe_CameraInverted_Y,
  Probe_HasMEMSScanner }

  Boolean parameters describing the behaviour of the Probe.
- enum ObjectivePropertyString {
  Objective_DisplayName,
  Objective_Mount }

  Properties of the objective mounted to the scanner such as the name.
- enum ObjectivePropertyInt {
  Objective_RangeMaxX_mm,
  Objective_RangeMaxY_mm }

  Properties of the objective mounted to the scanner such as valid scan range in mm.
- enum ObjectivePropertyFloat {
  Objective_FocalLength_mm,
  Objective_OpticalPathLength }

  Properties of the objective mounted to the scanner such as the focal length of the lens.
- enum ProbeScanRangeShape {
  Probe_ScanRange_Rectangular,
  Probe_ScanRange_Round }

  The shape of the maximal valid scan range.

Functions

- SPECTRALRADAR_API ProbeHandle initProbe (OCTDeviceHandle Dev, const char ∗ProbeFile)

  Initializes a probe specified by ProbeFile.
- SPECTRALRADAR_API ProbeHandle initDefaultProbe (OCTDeviceHandle Dev, const char ∗Type, const char ∗Objective)

  Creates a standard probe using standard parameters for the specified probe type.
- SPECTRALRADAR_API ProbeHandle initProbeFromOCTFile (OCTDeviceHandle Dev, OCTFileHandle File)

  Creates a probe using the parameters from the specified OCT file.
- SPECTRALRADAR_API void saveProbe (ProbeHandle Probe, const char ∗ProbeFile)

  Saves the current properties of the ProbeHandle to a specified INI file to be reloaded using the initProbe() function.
- SPECTRALRADAR_API void setProbeParameterInt (ProbeHandle Probe, ProbeParameterInt Selection, int Value)

  Sets integer parameter of the specified probe.
- SPECTRALRADAR_API void setProbeParameterFloat (ProbeHandle Probe, ProbeParameterFloat Selection, double Value)

  Sets floating point parameters of the specified probe.
- SPECTRALRADAR_API int getProbeParameterInt (ProbeHandle Probe, ProbeParameterInt Selection)

  Gets integer parameters of the specified probe.
- SPECTRALRADAR_API double getProbeParameterFloat (ProbeHandle Probe, ProbeParameterFloat Selection)

  Gets floating point parameters of the specified probe.
- SPECTRALRADAR_API BOOL getProbeFlag (ProbeHandle Probe, ProbeFlag Selection)

  Returns the selected boolean value of the specified probe.
- SPECTRALRADAR_API void setProbeParameterString (ProbeHandle Probe, ProbeParameterString Selection, const char ∗Value)

  Sets a string property of the specified probe.
- SPECTRALRADAR_API const char ∗ getProbeParameterString (ProbeHandle Probe, ProbeParameterString Selection)

  Gets the desired string property of the specified probe.

- SPECTRALRADAR_API const char ∗ getProbeType (ProbeHandle Probe)

  Gets the type of the specified probe.
- SPECTRALRADAR_API void setProbeType (ProbeHandle Probe, const char ∗Type)

  Sets the type of the specified probe.
- SPECTRALRADAR_API void closeProbe (ProbeHandle Probe)

  Closes the probe and frees all memory associated with it.
- SPECTRALRADAR_API void CameraPixelToPosition (ProbeHandle Probe, ColoredDataHandle Image, int PixelX, int PixelY, double ∗PosX, double ∗PosY)

  Computes the physical position of a camera pixel of the video camera in the probe. It assumes a properly calibrated device.
- SPECTRALRADAR_API void PositionToCameraPixel (ProbeHandle Probe, ColoredDataHandle Image, double PosX, double PosY, int ∗PixelX, int ∗PixelY)

  Computes the pixel of the video camera corresponding to a physical position. It needs to be assured that the device is properly calibrated.
- SPECTRALRADAR_API void visualizeScanPatternOnDevice (OCTDeviceHandle Dev, ProbeHandle Probe, ScanPatternHandle Pattern, BOOL ShowRawPattern)

  Visualizes the scan pattern on top of the camera image; if appropriate hardware is used for visualization.
- SPECTRALRADAR_API void visualizeScanPatternOnImage (ProbeHandle Probe, ScanPatternHandle ScanPattern, ColoredDataHandle VideoImage)

  Visualizes the scan pattern on top of the camera image; scan pattern data is written into the image.
- SPECTRALRADAR_API int getNumberOfProbeConfigs ()

  Returns the number of available probe configuration files.
- SPECTRALRADAR_API void getProbeConfigName (int Index, char ∗ProbeName, int StringSize)

  Returns the name of the specified probe configuration file.
- SPECTRALRADAR_API int getNumberOfAvailableProbes (void)

  Returns the number of the available probe types.
- SPECTRALRADAR_API void getAvailableProbe (int Index, char ∗ProbeName, int StringSize)

  Returns the name of the desired probe type.
- SPECTRALRADAR_API void getProbeDisplayName (const char ∗ProbeName, char ∗DisplayName, int StringSize)

  Returns the display name for the probe name specified.
- SPECTRALRADAR_API void getObjectiveDisplayName (const char ∗ObjectiveName, char ∗DisplayName, int StringSize)

  Returns the display name for the objective name specified.
- SPECTRALRADAR_API int getNumberOfCompatibleObjectives (const char ∗ProbeName)

  Returns the number of objectives compatible with the specified objective mount.
- SPECTRALRADAR_API void getCompatibleObjective (int Index, const char ∗ProbeName, char ∗Objective, int StringSize)

  Returns the name of the specified objective for the selected probe type.
- SPECTRALRADAR_API ProbeScanRangeShape getProbeMaxScanRangeShape (ProbeHandle Probe)

  Returns the shape of the valid scan range for the ProbeHandle. All possible scan range are defined in ProbeScanRangeShape.
- SPECTRALRADAR_API void setProbeMaxScanRangeShape (ProbeHandle Probe, ProbeScanRangeShape Shape)

  Sets the `Shape` of the valid scan range for the ProbeHandle. All possible scan-range shapes are defined in ProbeScanRangeShape.
- SPECTRALRADAR_API int getObjectivePropertyInt (const char ∗Objective, ObjectivePropertyInt Selection)

  Returns the selected ObjectivePropertyInt for the chosen objective.
- SPECTRALRADAR_API double getObjectivePropertyFloat (const char ∗Objective, ObjectivePropertyFloat Selection)

  Returns the selected ObjectivePropertyFloat for the chosen objective.
- SPECTRALRADAR_API const char ∗ getObjectivePropertyString (const char ∗Objective, ObjectivePropertyString Selection)

Returns the selected ObjectivePropertyString for the chosen objective. Warning: The returned const char∗ will only be valid until the next call to getObjectivePropertyString.

- SPECTRALRADAR_API void addProbeButtonCallback (OCTDeviceHandle Dev, cbProbeMessageReceived Callback)

  Registers a callback function to notify when a button on the probe has been pressed. The int parameter passed to the callback function will contain the pressed button's ID. Caution: Since the callbacks will not be called in separate threads but in the order of addition, make sure that the callback function returns as soon as possible.
- SPECTRALRADAR_API void removeProbeButtonCallback (OCTDeviceHandle Dev, cbProbeMessageReceived Callback)

  Removes a previously registered probe button callback function.

5.7.1 Detailed Description

Functions setting up a probe that can be used to create scan patterns.

5.7.2 Typedef Documentation

5.7.2.1 cbProbeMessageReceived

The prototype for callback functions registered for probe button events. As of the creation time of this document, only the OCTH probe is equipped with buttons.

Parameters

int	Zero-based ID of the pressed button

5.7.2.2 ProbeHandle

Handle for controlling the galvo scanner.

5.7.3 Enumeration Type Documentation

5.7.3.1 enum ObjectivePropertyFloat

Properties of the objective mounted to the scanner such as the focal length of the lens.

Enumerator

Objective_FocalLength_mm Focal length in mm of the specidifed objective.

Objective_OpticalPathLength Optical path length, in millimeter (without counting the focal length, multiplied by the equivalent refractive index).

5.7.3.2 enum ObjectivePropertyInt

Properties of the objective mounted to the scanner such as valid scan range in mm.

Enumerator

Objective_RangeMaxX_mm The maximum range in mm of the x-direction for the specified objective.

Objective_RangeMaxY_mm The maximum range in mm of the y-direction for the specified objective.

5.7.3.3 enum ObjectivePropertyString

Properties of the objective mounted to the scanner such as the name.

Enumerator

Objective_DisplayName Human-readable name of the objective to display in calibration process, or as device info.

Objective_Mount The mount specification is used to find the compatible probes and objectives (to be found in .ordf and .prdf files).

5.7.3.4 enum ProbeFlag

Boolean parameters describing the behaviour of the Probe.

Enumerator

Probe_CameraInverted_X Bool if the scan pattern in the video camera image is flipped around x-axis or not.

Probe_CameraInverted_Y Bool if the scan pattern in the video camera image is flipped around y-axis or not.

Probe_HasMEMSScanner Boolean if the probe type uses a MEMS mirror or not, e.g. a handheld probe.

5.7.3.5 enum ProbeParameterFloat

Parameters describing the behaviour of the Probe, such as calibration factors and scan parameters.

Computation of physical position and raw values for the scanner is done by PhyscialPosition = Factor $*$ RawValue + Offset

Enumerator

Probe_FactorX Factor for the x axis.

Probe_OffsetX Offset for the x axis.

Probe_FactorY Factor for the y axis.

Probe_OffsetY Offset for the y axis.

Probe_FlybackTime_Sec Flyback time of the system. This time is usually needed to get from an apodization position to scan position and vice versa.

Probe_ExpansionTime_Sec The scanning range is extended by a number of A-scans equivalent to the expansion time.

Probe_RotationTime_Sec The scan pattern is usually shifted by a number of A-scans equivalent to the rotation time.

Probe_ExpectedScanRate_Hz The expected scan rate.

Warning

> In general the expected scan rate is set during initialization of the probe with respect to the attached device. In most cases it should not be altered manually.

Probe_CameraScalingX The px/mm ratio in X direction for the BScan overlay on the video image.

Probe_CameraOffsetX The BScan overlay X offset in pixels.

Probe_CameraScalingY The px/mm ratio in Y direction for the BScan overlay on the video image.

Probe_CameraOffsetY The BScan overlay Y offset in pixels.

Probe_CameraAngle Corrective rotation angle for the BScan overlay.

Probe_RangeMaxX Maximum scan range in X direction.

Probe_RangeMaxY Maximum scan range in Y direction.

Probe_MaximumSlope_XY Maximum galvo slope (accounting for the distortion capabilities of different galvo types)

Probe_SpeckleSize Speckle size to be used for scan pattern computation if speckle reduction is switched on.

Probe_ApoVoltageX X-voltage used to acquire the apodization spectrum.

Probe_ApoVoltageY Y-voltage used to acquire the apodization spectrum.

Probe_ReferenceStageOffset Offset for reference stage marking the zero delay line.

Probe_FiberOpticalPathLength_mm Optical path length, in millimeter (fiber length up to the scanner, multiplied by the refractive index)

Probe_ProbeOpticalPathLength_mm Optical path length, in millimeter (from scanner input to objective mount, multiplied by the refractive index)

Probe_ObjectiveOpticalPathLength_mm Optical path length, in millimeter (without couning the focal length, multiplied by the equivalent refractive index)

Probe_ObjectiveFocalLength_mm Optical focal length, in millimeter.

5.7.3.6 enum ProbeParameterInt

Parameters describing the behaviour of the Probe, such as calibration factors and scan parameters.

Enumerator

Probe_ApodizationCycles The number of cycles used for apodization.

Probe_Oversampling A factor used as oversampling.

Probe_Oversampling_SlowAxis A factor used as oversampling of the slow scanner axis.

Probe_SpeckleReduction Number of speckles that are scanned over for averaging. Requires Oversampling >= SpeckleReduction.

5.7.3.7 enum ProbeParameterString

Parameters describing the composition of the probe. These properties refer to a probe that has already been created and for which a valid ProbeHandle has been obtained.

Enumerator

Probe_Name The filename. Just Probe.ini, or some other name.

Probe_SerialNumber Serial number of the probe.

Probe_Description Name of the probe. From this name it is possible to find out the probe definition file. A version suffix (e.g. "_V2") might be part of it. The termination ".prdf" is not part of the name.

Probe_Objective Objective from the probe. From this string it is possible to find out the objective definition file. A version suffix (e.g. "_V2") might be part of it. The termination ".odf" is not part of the name.

5.7.3.8 enum **ProbeScanRangeShape**

The shape of the maximal valid scan range.

Enumerator

Probe_ScanRange_Rectangular The shape of the valid scan range for the specified objective.

Probe_ScanRange_Round The maximum range in mm of the y-direction for the specified objective.

5.7.4 Function Documentation

5.7.4.1 void addProbeButtonCallback (**OCTDeviceHandle** *Dev,* **cbProbeMessageReceived** *Callback*)

Registers a callback function to notify when a button on the probe has been pressed. The int parameter passed to the callback function will contain the pressed button's ID. Caution: Since the callbacks will not be called in separate threads but in the order of addition, make sure that the callback function returns as soon as possible.

5.7.4.2 void CameraPixelToPosition (**ProbeHandle** *Probe,* **ColoredDataHandle** *Image,* int *PixelX,* int *PixelY,* double ∗ *PosX,* double ∗ *PosY*)

Computes the physical position of a camera pixel of the video camera in the probe. It assumes a properly calibrated device.

Parameters

in	*Probe*	A valid (non null) handle of a probe (ProbeHandle), previously generated by one of the functions initProbe, initDefaultProbe, or initProbeFromOCTFile.
in	*Image*	A valid (non null) handle of colored data.
in	*PixelX*	The x-pixel coordinate.
in	*PixelY*	The y-pixel coordinate.
out	*PosX*	The x coordinate. If this pointer happens to be null, it will not be used.
out	*PosY*	The y coordinate. If this pointer happens to be null, it will not be used.

5.7.4.3 void closeProbe (**ProbeHandle** *Probe*)

Closes the probe and frees all memory associated with it.

Parameters

in	*Probe*	A handle of a probe (ProbeHandle). If the handle is a nullptr, this function does nothing. In most cases this handle will have been previously generated by one of the functions initProbe, initDefaultProbe, or initProbeFromOCTFile.

5.7.4.4 void getAvailableProbe (int *Index,* char ∗ *ProbeName,* int *StringSize*)

Returns the name of the desired probe type.

Parameters

in	*Index*	Selects one specific probe type from all available ones.
out	*ProbeName*	The desired string with the name of the probe type, e.g. standard, user-customizable or compact handheld. This string is essentially the name of the corresponding .prdf file, except that a version number and the termination should be added.
in	*StringSize*	The length of the returned char∗.

5.7.4.5 void getCompatibleObjective (int *Index,* const char ∗ *ProbeName,* char ∗ *Objective,* int *StringSize*)

Returns the name of the specified objective for the selected probe type.

Parameters

in	*Index*	Selects one specific objective from all available objective for the specified probe type.
in	*ProbeName*	The name of the probe, as retrieved with the function getAvailableProbe.
out	*Objective*	Return value for the name of the objective file. This string is essentially the name of the corresponding .odf file, except that a version number and the termination will be added.
in	*StringSize*	The length of the returned char∗.

5.7.4.6 int getNumberOfAvailableProbes (void)

Returns the number of the available probe types.

Returns

The number of the available probe types.

5.7.4.7 int getNumberOfCompatibleObjectives (const char ∗ *ProbeName*)

Returns the number of objectives compatible with the specified objective mount.

Parameters

in	*ProbeName*	The name of the probe, as retrieved with the function getAvailableProbe.

Returns

The number of objectives compatible with the specified probe name.

5.7.4.8 int getNumberOfProbeConfigs ( )

Returns the number of available probe configuration files.

5.7.4.9 void getObjectiveDisplayName (const char ∗ *ObjectiveName,* char ∗ *DisplayName,* int *StringSize*)

Returns the display name for the objective name specified.

Parameters

in	*ObjectiveName*	Name of the objective. This string is essentially the name of the corresponding .odf file, except that a version number and the termination should be added.
out	*DisplayName*	The string to be shown in OCTImage software.
in	*StringSize*	The length of the returned char∗.

5.7.4.10 double getObjectivePropertyFloat (const char ∗ *Objective,* ObjectivePropertyFloat *Selection*)

Returns the selected ObjectivePropertyFloat for the chosen objective.

Parameters

Objective	Specifies the name of the objective.
Selection	Specifies the ObjectivePropertyFloat property.

5.7.4.11 int getObjectivePropertyInt (const char ∗ *Objective,* ObjectivePropertyInt *Selection*)

Returns the selected ObjectivePropertyInt for the chosen objective.

Parameters

Objective	Specifies the name of the objective.
Selection	Specifies the ObjectivePropertyInt property.

5.7.4.12 const char ∗ getObjectivePropertyString (const char ∗ *Objective,* ObjectivePropertyString *Selection*)

Returns the selected ObjectivePropertyString for the chosen objective. Warning: The returned const char∗ will only be valid until the next call to getObjectivePropertyString.

5.7.4.13 void getProbeConfigName (int *Index,* char ∗ *ProbeName,* int *StringSize*)

Returns the name of the specified probe configuration file.

Parameters

Index	Selects one specific configuration file from all available probe configuration files.
ProbeName	Return value for the name of the probe configuration file.
StringSize	The length of the returned char∗.

5.7.4.14 void getProbeDisplayName (const char ∗ *ProbeName,* char ∗ *DisplayName,* int *StringSize*)

Returns the display name for the probe name specified.

Parameters

in	*ProbeName*	Name of the probe. This string is essentially the name of the corresponding .prdf file, except that a version number and the termination should be added.

Parameters

out	*DisplayName*	The string to be shown in OCTImage software.
in	*StringSize*	The length of the returned char∗.

5.7.4.15 BOOL getProbeFlag (ProbeHandle *Probe,* ProbeFlag *Selection*)

Returns the selected boolean value of the specified probe.

Parameters

in	*Probe*	A valid (non null) handle of a probe (ProbeHandle), previously generated by one of the functions initProbe, initDefaultProbe, or initProbeFromOCTFile.
in	*Selection*	The desired flag.

Returns

The current value of the flag.

5.7.4.16 ProbeScanRangeShape getProbeMaxScanRangeShape (ProbeHandle *Probe*)

Returns the shape of the valid scan range for the ProbeHandle. All possible scan range are defined in ProbeScanRangeShape.

Parameters

in	*Probe*	Specified ProbeHandle.

5.7.4.17 double getProbeParameterFloat (ProbeHandle *Probe,* ProbeParameterFloat *Selection*)

Gets floating point parameters of the specified probe.

Parameters

in	*Probe*	A valid (non null) handle of a probe (ProbeHandle), previously generated by one of the functions initProbe, initDefaultProbe, or initProbeFromOCTFile.
in	*Selection*	The desired parameter.

Returns

The current value of the parameter.

5.7.4.18 int getProbeParameterInt (ProbeHandle *Probe,* ProbeParameterInt *Selection*)

Gets integer parameters of the specified probe.

Parameters

in	*Probe*	A valid (non null) handle of a probe (ProbeHandle), previously generated by one of the functions initProbe, initDefaultProbe, or initProbeFromOCTFile.
in	*Selection*	The desired parameter.

Returns

The current value of the parameter.

5.7.4.19 const char ∗ getProbeParameterString (**ProbeHandle** *Probe,* **ProbeParameterString** *Selection*)

Gets the desired string property of the specified probe.

Parameters

in	*Probe*	A valid (non null) handle of a probe (ProbeHandle), previously generated by one of the functions initProbe, initDefaultProbe, or initProbeFromOCTFile.
in	*Selection*	The desired parameter.

Returns

The current value of the parameter. The pointer referes to memory owned by SpectralRadar.dll. The user should not attempt to free it.

5.7.4.20 const char ∗ getProbeType (**ProbeHandle** *Probe*)

Gets the type of the specified probe.

Returns

The current type name (one of `Standard_OCTG`, `UserCustomizable_OCTP`, `Handheld_OCTH`).

Parameters

in	*Probe*	A valid (non null) handle of a probe (ProbeHandle), previously generated by one of the functions initProbe, initDefaultProbe, or initProbeFromOCTFile.

5.7.4.21 **ProbeHandle** initDefaultProbe (**OCTDeviceHandle** *Dev,* const char ∗ *Type,* const char ∗ *Objective*)

Creates a standard probe using standard parameters for the specified probe type.

Parameters

in	*Dev*	A valid (non null) OCT device handle (OCTDeviceHandle), previously generated with the function initDevice.
in	*Type*	A zero terminated string with the probe type name (one of `Standard_OCTG`, `UserCustomizable_OCTP`, `Handheld_OCTH`).
in	*Objective*	A zero terminated string with the objective name (e.g. `"LSM03"`).

Returns

A valid probe handle (ProbeHandle).

5.7.4.22 ProbeHandle initProbe (OCTDeviceHandle *Dev*, const char ∗ *ProbeFile*)

Initializes a probe specified by ProbeFile.

Parameters

in	*Dev*	The OCTDeviceHandle that was initially provided by initDevice. Can be NULL in case no device is initialized or available.
in	*ProbeFile*	The filename of the .ini. If the path is not given, it will be assumed that this file is in the configuration directory (typically C:\Program Files\Thorlabs\SpectralRadar\Config). To indicate that file is in the current working directory, prepend a "∼\\" before the name. If a termination ".ini" is not there, it will be appended.

Returns

A valid probe handle (ProbeHandle).

In older systems up until a manufacturing date of May 2011 either "Handheld" or "Microscope" are used. An according .ini file (i. e. "Handheld.ini" or "Microscope.ini) will be loaded from the config path of the SpectralRadar installation containing all necessary information. With systems manufactured after May 2011 "Probe" should be used.

5.7.4.23 ProbeHandle initProbeFromOCTFile (OCTDeviceHandle *Dev*, OCTFileHandle *File*)

Creates a probe using the parameters from the specified OCT file.

Parameters

in	*Dev*	A valid (non null) OCT device handle (OCTDeviceHandle), previously generated with the function initDevice.
in	*File*	A valid (non null) handle of an OCT file.

Returns

A valid probe handle (ProbeHandle).

5.7.4.24 void PositionToCameraPixel (ProbeHandle *Probe*, ColoredDataHandle *Image*, double *PosX*, double *PosY*, int ∗ *PixelX*, int ∗ *PixelY*)

Computes the pixel of the video camera corresponding to a physical position. It needs to be assured that the device is properly calibrated.

Parameters

in	*Probe*	A valid (non null) handle of a probe (ProbeHandle), previously generated by one of the functions initProbe, initDefaultProbe, or initProbeFromOCTFile.
in	*Image*	A valid (non null) handle of colored data.

Parameters

in	*PosX*	The x coordinate.
in	*PosY*	The y coordinate.
out	*PixelX*	The x-pixel coordinate. If this pointer happens to be null, it will not be used.
out	*PixelY*	The y-pixel coordinate. If this pointer happens to be null, it will not be used.

5.7.4.25 void removeProbeButtonCallback (OCTDeviceHandle *Dev,* cbProbeMessageReceived *Callback*)

Removes a previously registered probe button callback function.

5.7.4.26 void saveProbe (ProbeHandle *Probe,* const char ∗ *ProbeFile*)

Saves the current properties of the ProbeHandle to a specified INI file to be reloaded using the initProbe() function.

Parameters

in	*Probe*	A valid (non null) handle of a probe (ProbeHandle), previously generated by one of the functions initProbe, initDefaultProbe, or initProbeFromOCTFile.
in	*ProbeFile*	The filename of the .ini. If the path is not given, it will be assumed that this file should go in the configuration directory (typically C:\Program Files\Thorlabs\SpectralRadar\Config). To indicate that file is in the current working directory, prepend a "∼\\" before the name. If a termination ".ini" is not there, it will be appended.

5.7.4.27 ProbeScanRangeShape setProbeMaxScanRangeShape (ProbeHandle *Probe,* ProbeScanRangeShape *Shape*)

Sets the `Shape` of the valid scan range for the ProbeHandle. All possible scan-range shapes are defined in ProbeScanRangeShape.

Parameters

in	*Probe*	Specified ProbeHandle.
in	*Shape*	the desired shape, which should be in the range defined by ProbeScanRangeShape.

5.7.4.28 void setProbeParameterFloat (ProbeHandle *Probe,* ProbeParameterFloat *Selection,* double *Value*)

Sets floating point parameters of the specified probe.

Parameters

in	*Probe*	A valid (non null) handle of a probe (ProbeHandle), previously generated by one of the functions initProbe, initDefaultProbe, or initProbeFromOCTFile.
in	*Selection*	The desired parameter.
in	*Value*	The new value for the parameter.

5.7.4.29 void setProbeParameterInt (ProbeHandle *Probe,* ProbeParameterInt *Selection,* int *Value*)

Sets integer parameter of the specified probe.

Parameters

in	*Probe*	A valid (non null) handle of a probe (ProbeHandle), previously generated by one of the functions initProbe, initDefaultProbe, or initProbeFromOCTFile.
in	*Selection*	The desired parameter.
in	*Value*	The new value for the parameter.

5.7.4.30 void setProbeParameterString (ProbeHandle *Probe,* ProbeParameterString *Selection,* const char ∗ *Value*)

Sets a string property of the specified probe.

Parameters

in	*Probe*	A valid (non null) handle of a probe (ProbeHandle), previously generated by one of the functions initProbe, initDefaultProbe, or initProbeFromOCTFile.
in	*Selection*	The desired parameter.
in	*Value*	The desired value for the parameter.

5.7.4.31 void setProbeType (ProbeHandle *Probe,* const char ∗ *Type*)

Sets the type of the specified probe.

Parameters

in	*Probe*	A valid (non null) handle of a probe (ProbeHandle), previously generated by one of the functions initProbe, initDefaultProbe, or initProbeFromOCTFile.
in	*Type*	A zero terminated string describing the probe type (one of `Standard_OCTG`, `UserCustomizable_OCTP`, `Handheld_OCTH`).

5.7.4.32 void visualizeScanPatternOnDevice (OCTDeviceHandle *Dev,* ProbeHandle *Probe,* ScanPatternHandle *Pattern,* BOOL *ShowRawPattern*)

Visualizes the scan pattern on top of the camera image; if appropriate hardware is used for visualization.

Parameters

in	*Dev*	A valid (non null) OCT device handle (OCTDeviceHandle), previously generated with the function initDevice.
in	*Probe*	A valid (non null) handle of a probe (ProbeHandle), previously generated by one of the functions initProbe, initDefaultProbe, or initProbeFromOCTFile.
in	*Pattern*	A valid (non null) handle of a scan pattern.
in	*ShowRawPattern*	Indicates whether the scan should shown (TRUE) or hidden (FALSE).

5.7.4.33 void visualizeScanPatternOnImage (ProbeHandle *Probe,* ScanPatternHandle *ScanPattern,* ColoredDataHandle *VideoImage*)

Visualizes the scan pattern on top of the camera image; scan pattern data is written into the image.

Parameters

in	*Probe*	A valid (non null) handle of a probe (ProbeHandle), previously generated by one of the functions initProbe, initDefaultProbe, or initProbeFromOCTFile.
in	*ScanPattern*	A valid (non null) handle of a scan pattern.
in	*VideoImage*	A valid (non null) handle of colored data.

5.8 Scan Pattern

Functions that describe the movement of the Scanner during measurement.

Typedefs

- typedef struct C_ScanPattern ∗ ScanPatternHandle

  Handle for creating, manipulating, and discarding a scan pattern.

Enumerations

- enum ScanPatternAcquisitionOrder {
  ScanPattern_AcqOrderFrameByFrame,
  ScanPattern_AcqOrderAll }

  Parameters describing the behaviour of the scan pattern.
- enum ScanPatternApodizationType {
  ScanPattern_ApoOneForAll,
  ScanPattern_ApoEachBScan }

  Parameters describing how often the apodization spectra will be acquired. If you want to create a scan pattern without an apodization please use (setProbeParameterInt) and (Probe_ApodizationCycles) to set the size of apodization to zero.
- enum InflationMethod { Inflation_NormalDirection }

  Describes how to use a 2D freeform scan pattern to create a 3D scan pattern.
- enum ScanPointsDataFormat {
  ScanPoints_DataFormat_TXT,
  ScanPoints_DataFormat_RAWandSRM }

  Selects format with the functions loadScanPointsFromFile or saveScanPointsToFile to import or export data points.
- enum ScanPatternPropertyInt {
  ScanPattern_SizeTotal,
  ScanPattern_Cycles,
  ScanPattern_SizeCycle,
  ScanPattern_SizePreparationCycle,
  ScanPattern_SizeImagingCycle }

  Enum identifying different properties of typ int of the specified scan pattern.
- enum ScanPatternPropertyFloat {
  ScanPattern_RangeX,
  ScanPattern_RangeY,
  ScanPattern_CenterX,
  ScanPattern_CenterY,
  ScanPattern_Angle,
  ScanPattern_MeanLength_mm }

  Enum identifying different floating-type properties of the specified scan pattern.

Functions

- SPECTRALRADAR_API ScanPatternHandle createNoScanPattern (ProbeHandle Probe, int AScans, int NumberOfScans)

  Creates a simple scan pattern that does not move the galvo. Use this pattern for point scans and/or non-scanning probes. The pattern will however use a specified amount of trigger signals. For continuous acquisition use NumberOfScans set to 1.
- SPECTRALRADAR_API ScanPatternHandle createBScanPattern (ProbeHandle Probe, double Range_mm, int AScans)

Creates a horizontal rectilinear-segment B-scan pattern that moves the galvo over a specified range.

- SPECTRALRADAR_API ScanPatternHandle createBScanPatternManual (ProbeHandle Probe, double StartX_mm, double StartY_mm, double StopX_mm, double StopY_mm, int AScans)

  Creates a B-scan pattern specified by start and end points.

- SPECTRALRADAR_API ScanPatternHandle createIdealBScanPattern (ProbeHandle Probe, double Range_mm, int AScans)

  Creates an ideal B-scan pattern assuming scanners with infinite speed. No correction factors are taken into account. This is only used for internal purposes and not as a scan pattern designed to be output to the galvo drivers.

- SPECTRALRADAR_API ScanPatternHandle createCirclePattern (ProbeHandle Probe, double Radius_mm, int AScans)

  Creates a circle scan pattern.

- SPECTRALRADAR_API ScanPatternHandle createVolumePattern (ProbeHandle Probe, double Range↩X_mm, int SizeX, double RangeY_mm, int SizeY, ScanPatternApodizationType ApoType, ScanPattern↩AcquisitionOrder AcqOrder)

  Creates a simple volume pattern.

- SPECTRALRADAR_API ScanPatternHandle createVolumePatternEx (ProbeHandle Probe, double Range↩X_mm, int SizeX, double RangeY_mm, int SizeY, double CenterX_mm, double CenterY_mm, double Angle↩_rad, ScanPatternApodizationType ApoType, ScanPatternAcquisitionOrder AcqOrder)

  Creates a simple volume pattern.

- SPECTRALRADAR_API void updateScanPattern (ScanPatternHandle Pattern)

  Updates the specfied pattern (ScanPatternHandle) and computes the full look-up-table.

- SPECTRALRADAR_API void rotateScanPattern (ScanPatternHandle Pattern, double Angle_rad)

  Rotates the specfied pattern (ScanPatternHandle), counter-clockwise. The rotation is relative to current angle, not to the horizontal. That is, after multiple invokations of this function the final rotation is the addition of all rotations.

- SPECTRALRADAR_API void rotateScanPatternEx (ScanPatternHandle Pattern, double Angle_rad, int Index)

  Counter-clockwise rotates the scan `Index` (0-based, i.e. zero for the first, one for the second, and so on) of the specfied volume scan pattern (ScanPatternHandle). The rotation is relative to current angle, not to the horizontal. That is, after multiple invokations of this function the final rotation is the addition of all rotations.

- SPECTRALRADAR_API void shiftScanPattern (ScanPatternHandle Pattern, double ShiftX_mm, double ShiftY_mm)

  Shifts the specified pattern (ScanPatternHandle). The shift is relative to current position, not to (0,0). That is, after multiple invokations of this function the final shift is the addition of all shifts.

- SPECTRALRADAR_API void shiftScanPatternEx (ScanPatternHandle Pattern, double ShiftX_mm, double ShiftY_mm, BOOL ShiftApo, int Index)

  Shifts the scan `Index` (0-based, i.e. zero for the first, one for the second, and so on) of the specified volume pattern (ScanPatternHandle). The shift is relative to current position, not to (0,0). That is, after multiple invokations of this function the final shift is the addition of all shifts.

- SPECTRALRADAR_API void zoomScanPattern (ScanPatternHandle Pattern, double Factor)

  Zooms the specified pattern (ScanPatternHandle) around the optical center that coincides with the center of the camera image and the physical coordinates (0 mm,0 mm). The apodization position will not be modified.

- SPECTRALRADAR_API int getScanPatternLUTSize (ScanPatternHandle Pattern)

  Returns the number of points in the specified scan pattern (ScanPatternHandle), including apodization and flyback.

- SPECTRALRADAR_API void getScanPatternLUT (ScanPatternHandle Pattern, double ∗VoltX, double ∗VoltY)

  Returns the voltages that will be applied to reach the positions to be scanned, in the specified scan pattern (Scan↩PatternHandle).

- SPECTRALRADAR_API int getScanPointsSize (ScanPatternHandle Pattern)

  Returns the number of points in the specified scan pattern (ScanPatternHandle), including apodization and flyback.

- SPECTRALRADAR_API void getScanPoints (ScanPatternHandle Pattern, double ∗PosX_mm, double ∗PosY_mm)

  Returns the position coordinates (in mm) of the points that in the specified scan pattern (ScanPatternHandle).

- SPECTRALRADAR_API void clearScanPattern (ScanPatternHandle Pattern)

  Clears the specified scan pattern (ScanPatternHandle).

- SPECTRALRADAR_API ScanPatternHandle createFreeformScanPattern2D (ProbeHandle Probe, double ∗PosX_mm, double ∗PosY_mm, int Size, int AScans, InterpolationMethod InterpolationMethod, BOOL CloseScanPattern)

  Creates a B-scan scan pattern of arbitrary form with equidistant sampled scan points.
- SPECTRALRADAR_API ScanPatternHandle createFreeformScanPattern2DFromLUT (ProbeHandle Probe, double ∗PosX_mm, double ∗PosY_mm, int Size, BOOL ClosedScanPattern)

  Creates a B-scan scan pattern of arbitrary form with the specified scan points. The voltages array is taken as-is, so care must be taken to use sensible values with regard to the capabilities of the utilized scanner system and to the resolution of the system resp. the desired resolution of your scan pattern.
- SPECTRALRADAR_API ScanPatternHandle createFreeformScanPattern3DFromLUT (ProbeHandle Probe, double ∗PosX_mm, double ∗PosY_mm, int AScansPerBScan, int NumberOfBScans, BOOL ClosedScan←Pattern, ScanPatternApodizationType ApoType, ScanPatternAcquisitionOrder AcqOrder)

  Creates a volume scan pattern of arbitrary form with the specified scan voltages. The voltages array is taken as-is, so care must be taken to use sensible values with regard to the capabilities of the utilized scanner system and to the resolution of the system resp. the desired resolution of your scan pattern. With this function the definition of each single scan point is required. In order to create a scan pattern specifying only the end coordinates, please consider createFreeformScanPattern3D.
- SPECTRALRADAR_API ScanPatternHandle createFreeformScanPattern3D (ProbeHandle Probe, double ∗PosX_mm, double ∗PosY_mm, int ∗ScanIndices, int Size, int NumberOfAScansPerBScan, Interpolation←Method InterpolationMethod, BOOL CloseScanPattern, ScanPatternApodizationType ApoType, Scan←PatternAcquisitionOrder AcqOrder)

  Creates a volume scan pattern of arbitrary form with equidistant sampled scan points.
- SPECTRALRADAR_API void saveScanPointsToFile (double ∗ScanPosX_mm, double ∗ScanPosY_mm, int ∗ScanIndices, int Size, const char ∗Filename, ScanPointsDataFormat DataFormat)

  Saves the scan points and scan indices to a file with the specified ScanPointsDataFormat.
- SPECTRALRADAR_API int getSizeOfScanPointsFromFile (const char ∗Filename, ScanPointsDataFormat DataFormat)

  Returns the number of scan points in the specified file.
- SPECTRALRADAR_API void loadScanPointsFromFile (double ∗ScanPosX_mm, double ∗ScanPosY_mm, int ∗ScanIndices, int Size, const char ∗Filename, ScanPointsDataFormat DataFormat)

  Copies the scan points and scan indices from the file to the provided arrays.
- SPECTRALRADAR_API int getSizeOfScanPointsFromDataHandle (DataHandle ScanPoints)

  Returns the size of the scan points and scan indices in the DataHandle.
- SPECTRALRADAR_API void getScanPointsFromDataHandle (DataHandle ScanPoints, double ∗PosX_mm, double ∗PosY_mm, int ∗ScanIndices, int Length)

  Copies the scan points and scan indices from the DataHandle to the provided arrays.
- SPECTRALRADAR_API DataHandle createDataHandleFromScanPoints (double ∗PosX_mm, double ∗Pos←Y_mm, int ∗ScanIndices, int Length)

  Creates a DataHandle from the specified scan points and corresponding indices.
- SPECTRALRADAR_API int getScanPatternPropertyInt (ScanPatternHandle ScanPattern, ScanPattern←PropertyInt Property)

  Returns the specified property of the scan pattern.
- SPECTRALRADAR_API double getScanPatternPropertyFloat (ScanPatternHandle Pattern, ScanPattern←PropertyFloat Selection)

  Returns the specified property of the scan pattern.
- SPECTRALRADAR_API double expectedAcquisitionTime_s (ScanPatternHandle ScanPattern, OCT←DeviceHandle Dev)

  Returns the expected acquisition time of the scan pattern. Please.
- SPECTRALRADAR_API ScanPatternAcquisitionOrder getScanPatternAcqOrder (ScanPatternHandle ScanPattern)

  Returns the acquisition order of the scan pattern. See definition of ScanPatternAcquisitionOrder for detailed information.
- SPECTRALRADAR_API BOOL isAcqTypeForScanPatternAvailable (ScanPatternHandle ScanPattern, AcquisitionType AcqType)

  Returns whether the acquisition type is available for the scan pattern.

5.8.1 Detailed Description

Functions that describe the movement of the Scanner during measurement.

5.8.2 Typedef Documentation

5.8.2.1 ScanPatternHandle

Handle for creating, manipulating, and discarding a scan pattern.

A scan pattern can be created with one of the functions createNoScanPattern, createAScanPattern, createBScanPattern, createBScanPatternManual, createIdealBScanPattern, createCirclePattern, createVolumePattern, createFreeformScanPattern2D, createFreeformScanPattern2DFromLUT, createFreeformScanPattern3DFromLUT, or createFreeformScanPattern3D.

5.8.3 Enumeration Type Documentation

5.8.3.1 enum InflationMethod

Describes how to use a 2D freeform scan pattern to create a 3D scan pattern.

Enumerator

Inflation_NormalDirection Inflates the points to the outer normal direction.

5.8.3.2 enum ScanPatternAcquisitionOrder

Parameters describing the behaviour of the scan pattern.

Enumerator

ScanPattern_AcqOrderFrameByFrame The scan pattern will be acquired slice by slice which means that the function getRawData() needs to be called more than once to get the data for the whole scan pattern

ScanPattern_AcqOrderAll The scan patten will be acquired in one piece.

5.8.3.3 enum ScanPatternApodizationType

Parameters describing how often the apodization spectra will be acquired. If you want to create a scan pattern without an apodization please use (setProbeParameterInt) and (Probe_ApodizationCycles) to set the size of apodization to zero.

Enumerator

ScanPattern_ApoOneForAll The volume scan pattern will be acquired with one apodization for the whole pattern.

ScanPattern_ApoEachBScan The volume scan pattern will be acquired with one apodization before each B-scan which results in a slightly better image quality but longer acquisition time.

5.8.3.4 enum ScanPatternPropertyFloat

Enum identifying different floating-type properties of the specified scan pattern.

Enumerator

ScanPattern_RangeX The range of the scan pattern in mm for the x-direction.

ScanPattern_RangeY The range of the scan pattern in mm for the y-direction.

ScanPattern_CenterX the current x-center position in mm

ScanPattern_CenterY the current y-center position in mm

ScanPattern_Angle the current scan pattern angle in radians

ScanPattern_MeanLength_mm the mean of the B-scan lengths of the scan pattern in mm

5.8.3.5 enum ScanPatternPropertyInt

Enum identifying different properties of typ int of the specified scan pattern.

Enumerator

ScanPattern_SizeTotal Total count of trigger pulses needed for acquisition of the scan pattern once. The acquisition will start again after finishing for continuous acquisition mode.

ScanPattern_Cycles Count of cycles for the scan pattern.

ScanPattern_SizeCycle Count of trigger pulses needed to acquire one cycle, e.g. one B-scan in a volume scan.

ScanPattern_SizePreparationCycle Count of trigger pulses needed before the scanning of the sample starts. The OCT beam needs to be positioned and the apodization scans used for processing need to be acquired. The flyback time is the time used to reach the position of apodization and start of scan pattern.

ScanPattern_SizeImagingCycle Count of trigger pulses to acquire the sample depending on averaging and size-x of the scan pattern.

5.8.3.6 enum ScanPointsDataFormat

Selects format with the functions loadScanPointsFromFile or saveScanPointsToFile to import or export data points.

Enumerator

ScanPoints_DataFormat_TXT Data format txt.

ScanPoints_DataFormat_RAWandSRM Data format raw/srm pair.

5.8.4 Function Documentation

5.8.4.1 void clearScanPattern (ScanPatternHandle *Pattern*)

Clears the specified scan pattern (ScanPatternHandle).

Parameters

in	*Pattern*	A handle of a scan pattern (ScanPatternHandle). If the handle is a nullptr, this function does nothing.

5.8.4.2 ScanPatternHandle createBScanPattern (ProbeHandle *Probe,* double *Range_mm,* int *AScans*)

Creates a horizontal rectilinear-segment B-scan pattern that moves the galvo over a specified range.

Parameters

in	*Probe*	A valid (non null) handle of a probe (ProbeHandle), previously generated by one of the functions initProbe, initDefaultProbe, or initProbeFromOCTFile.
in	*Range_mm*	The extension of the horizontal segment, expressed in mm, centered at (0,0).
in	*AScans*	The number of A-Scans that will be measured along the segment.

Returns

A valid (non null) handle to a scan pattern.

If a different center position is desired, one of the functions shiftScanPattern(), shiftScanPatternEx() should be invoked afterwards, passing the scan pattern handle returned by this function.
If a different orientation is desired (i.e other than horizontal), one of the functions rotateScanPattern(ScanPatternHandle(), rotateScanPatternEx(ScanPatternHandle() should be invoked afterwards, passing the scan pattern handle returned by this function.

5.8.4.3 ScanPatternHandle createBScanPatternManual (ProbeHandle *Probe,* double *StartX_mm,* double *StartY_mm,* double *StopX_mm,* double *StopY_mm,* int *AScans*)

Creates a B-scan pattern specified by start and end points.

Parameters

in	*Probe*	A valid (non null) handle of a probe (ProbeHandle), previously generated by one of the functions initProbe, initDefaultProbe, or initProbeFromOCTFile.
in	*StartX_mm*	The x-coordinate of the start point, in mm.
in	*StartY_mm*	The y-coordinate of the start point, in mm.
in	*StopX_mm*	The x-coordinate of the stop point, in mm.
in	*StopY_mm*	The y-coordinate of the stop point, in mm.
in	*AScans*	The number of A-Scans that will be measured along the segment.

Returns

A valid (non null) handle to a scan pattern.

5.8.4.4 ScanPatternHandle createCirclePattern (ProbeHandle *Probe,* double *Radius_mm,* int *AScans*)

Creates a circle scan pattern.

Parameters

in	*Probe*	A valid (non null) handle of a probe (ProbeHandle), previously generated by one of the functions initProbe, initDefaultProbe, or initProbeFromOCTFile.
in	*Radius_mm*	The radius of the circle pattern.
in	*AScans*	The number of A-Scans that will be measured along the segment.

Warning

Circle patterns cannot be rotated properly.

Returns

A valid (non null) handle to a scan pattern.

5.8.4.5 DataHandle createDataHandleFromScanPoints (double ∗ *PosX_mm,* double ∗ *PosY_mm,* int ∗ *ScanIndices,* int *Length*)

Creates a DataHandle from the specified scan points and corresponding indices.

Parameters

in	*PosX_mm*	A pointer to the array of X-coords of the scan pattern, with length `Size`
in	*PosY_mm*	A pointer to the array of Y-coords of the scan pattern, with length `Size`
in	*ScanIndices*	The array specifies the assignment of each point to its B-scan, with length `Size`. The entries need to go from 0 to number of B-scans - 1) The number of B-scans is defined with the entries of `ScanIndices`. To save scan points for a 2D-pattern set all entries to zero.
in	*Length*	The length of the arrays `FreeFromCoordsX`, `FreeformCoordsY`, and `ScanIndices`.

Returns

A DataHandle containing the scan points and indices.

5.8.4.6 ScanPatternHandle createFreeformScanPattern2D (ProbeHandle *Probe,* double ∗ *PosX_mm,* double ∗ *PosY_mm,* int *Size,* int *AScans,* InterpolationMethod *InterpolationMethod,* BOOL *CloseScanPattern*)

Creates a B-scan scan pattern of arbitrary form with equidistant sampled scan points.

Parameters

in	*Probe*	A valid (non null) handle of a probe (ProbeHandle), previously generated by one of the functions initProbe, or initDefaultProbe.
in	*PosX_mm*	A pointer to the double array of x-positions (in mm) of the scan pattern with length `Size`
in	*PosY_mm*	A pointer to the double array of y-positions (in mm) of the scan pattern with length `Size`
in	*Size*	The length of the arrays `PosX_mm` and `PosY_mm`.
in	*AScans*	The number of A-scans in the scan pattern that will be created. The number of A-scans should be greater than Size.

Parameters

in	*InterpolationMethod*	The interpolation method used to fill up the specified points by `PosX_mm` and `PosY_mm` to create a pattern with evenly spaced sampled points.
in	*CloseScanPattern*	Specifies whether the scan pattern should be closed (TRUE) or not (FALSE). Closing the scan pattern will lead to the same start and end point of each B-scan.

5.8.4.7 ScanPatternHandle createFreeformScanPattern2DFromLUT (ProbeHandle *Probe,* double * *PosX_mm,* double * *PosY_mm,* int *Size,* BOOL *ClosedScanPattern*)

Creates a B-scan scan pattern of arbitrary form with the specified scan points. The voltages array is taken as-is, so care must be taken to use sensible values with regard to the capabilities of the utilized scanner system and to the resolution of the system resp. the desired resolution of your scan pattern.

Parameters

in	*Probe*	A valid (non null) handle of a probe (ProbeHandle), previously generated by one of the functions initProbe, or initDefaultProbe.
in	*PosX_mm*	A pointer to the double array of X-positions (in mm) of the scan pattern with length `Size`
in	*PosY_mm*	A pointer to the double array of Y-positions (in mm) of the scan pattern with length `Size`
in	*Size*	The length of the arrays PositionsX and PositionsY.
in	*ClosedScanPattern*	Specifies whether the scan pattern should be closed (TRUE) or not (FALSE). Closing the scan pattern will lead to the same start and end point of each B-scan.

With this function the definition of every single scan point is required. In order to create a scan pattern specifying only some "edge" points of the pattern, please consider createFreeformScanPattern2D.

5.8.4.8 ScanPatternHandle createFreeformScanPattern3D (ProbeHandle *Probe,* double * *PosX_mm,* double * *PosY_mm,* int * *ScanIndices,* int *Size,* int *NumberOfAScansPerBScan,* InterpolationMethod *InterpolationMethod,* BOOL *CloseScanPattern,* ScanPatternApodizationType *ApoType,* ScanPatternAcquisitionOrder *AcqOrder*)

Creates a volume scan pattern of arbitrary form with equidistant sampled scan points.

Parameters

in	*Probe*	A valid (non null) handle of a probe (ProbeHandle), previously generated by one of the functions initProbe, or initDefaultProbe.
in	*PosX_mm*	The pointer to the array of x-positions of the scan pattern with length Size.
in	*PosY_mm*	The pointer to the array of y-positions of the scan pattern with length Size.
in	*ScanIndices*	The array specifies the assignment of each point to its B-scan. It needs to have the length Size. The entries need to go from 0 to number of (B-scans - 1). The number of B-scans is defined with the entries of `ScanIndices`. For example, if the minimum entry is 0 (cannot be negative!) and the maximum entry is 2, there will be three B-scans in the pattern.
in	*Size*	The length of the arrays `PosX_mm`, `PosY_mm`, and `ScanIndices`.
in	*NumberOfAScansPerBScan*	The number of A-scans in each B-scan of the created scan pattern. The number of B-scans will be defined with the entries in the `ScanIndices`.

Parameters

in	*InterpolationMethod*	The interpolation method used to fill up the specified points by `PositionsX` and `PositionsY` to create a pattern with evenly-spaced sampled points.
in	*CloseScanPattern*	Specifies whether the scan pattern should be closed or not. Closing the scan pattern means that each B-scan starts and stops at the same point.
in	*ApoType*	The specified method used for apodization in a volume pattern. Please see ScanPatternApodizationType for more information.
in	*AcqOrder*	The specified method used for the acquisition order in a volume pattern. Please see ScanPatternAcquisitionOrder for more information.

Returns

A scan pattern handle containing the created 3D-freeform scan pattern.

5.8.4.9 ScanPatternHandle createFreeformScanPattern3DFromLUT (ProbeHandle *Probe,* double ∗ *PosX_mm,* double ∗ *PosY_mm,* int *AScansPerBScan,* int *NumberOfBScans,* BOOL *ClosedScanPattern,* ScanPatternApodizationType *ApoType,* ScanPatternAcquisitionOrder *AcqOrder*)

Creates a volume scan pattern of arbitrary form with the specified scan voltages. The voltages array is taken as-is, so care must be taken to use sensible values with regard to the capabilities of the utilized scanner system and to the resolution of the system resp. the desired resolution of your scan pattern. With this function the definition of each single scan point is required. In order to create a scan pattern specifying only the end coordinates, please consider createFreeformScanPattern3D.

Parameters

in	*Probe*	A valid (non null) handle of a probe (ProbeHandle), previously generated by one of the functions initProbe, or initDefaultProbe.
in	*PosX_mm*	A pointer to the array of X-positions (in mm) of the scan pattern whose length is the product of `AScansPerBScan` and `NumberOfBScans`.
in	*PosY_mm*	A pointer to the array of Y-positions (in mm) of the scan pattern whose length is the product of `AScansPerBScan` and `NumberOfBScans`.
in	*AScansPerBScan*	The desired number of A-scans in each B-scan of the volume pattern. All B-scans will have the same size.
in	*NumberOfBScans*	The desired number of B-scans in the volume pattern.
in	*ClosedScanPattern*	Specifies whether the scan pattern should be closed or not. Closing the scan pattern will lead to the same start and end point of each B-scan.
in	*ApoType*	The specified method used for apodization in a volume pattern.

See also

ScanPatternApodizationType.

Parameters

in	*AcqOrder*	The specified method used for the acquisition order in a volume pattern.

See also

ScanPatternAcquisitionOrder.

Returns

A scan pattern handle containing the created 3D-freeform scan pattern.

5.8.4.10 ScanPatternHandle createIdealBScanPattern (ProbeHandle *Probe*, double *Range_mm*, int *AScans*)

Creates an ideal B-scan pattern assuming scanners with infinite speed. No correction factors are taken into account. This is only used for internal purposes and not as a scan pattern designed to be output to the galvo drivers.

Parameters

in	*Probe*	A valid (non null) handle of a probe (ProbeHandle), previously generated by one of the functions initProbe, initDefaultProbe, or initProbeFromOCTFile.
in	*Range_mm*	The extension of the segment, expressed in mm, centered at the current position.
in	*AScans*	The number of A-Scans that will be measured along the segment.

Returns

A valid (non null) handle to a scan pattern.

5.8.4.11 ScanPatternHandle createNoScanPattern (ProbeHandle *Probe*, int *AScans*, int *NumberOfScans*)

Creates a simple scan pattern that does not move the galvo. Use this pattern for point scans and/or non-scanning probes. The pattern will however use a specified amount of trigger signals. For continuous acquisition use Number←OfScans set to 1.

Parameters

in	*Probe*	A valid (non null) handle of a probe (ProbeHandle), previously generated by one of the functions initProbe, initDefaultProbe, or initProbeFromOCTFile.
in	*AScans*	The number of A-Scans that will be measured in each part of this ScanPatternHandle.
in	*NumberOfScans*	The number of parts in this ScanPatternHandle. It should be "1" for continuous acquisition.

Returns

A valid (non null) handle to a scan pattern.

5.8.4.12 ScanPatternHandle createVolumePattern (ProbeHandle *Probe*, double *RangeX_mm*, int *SizeX*, double *RangeY_mm*, int *SizeY*, ScanPatternApodizationType *ApoType*, ScanPatternAcquisitionOrder *AcqOrder*)

Creates a simple volume pattern.

Parameters

in	*Probe*	A valid (non null) handle of a probe (ProbeHandle), previously generated by one of the functions initProbe, initDefaultProbe, or initProbeFromOCTFile.
in	*RangeX_mm*	The extension of the volume along the x-axis, expressed in mm, centered at the current position.
in	*SizeX*	The number of planes that cross the x-axis..
in	*RangeY_mm*	The extension of the volume along the y-axis, expressed in mm, centered at the current position.
in	*SizeY*	The number of planes that cross the y-axis.
in	*ApoType*	The apodization type decides whether one apodization suffices for the whole set of measurements in the volume, or one apodization will be measured for each B-Scan (each segment).
in	*AcqOrder*	Dictates the acquisition strategy, as explained below, which reflects the way the user wants to retrieve the acquired data.

Returns

A valid (non null) handle to a scan pattern.

A volume scan pattern is actually a stack of B-scan patterns. At creation time the stack fills a parallelepiped volume in space but the shape can be subsequently modified if the individual slices are rotated, translated, or both (see explanation of functions rotateScanPatternEx(), shiftScanPatternEx() for more information). Notice that the individual B-scans (the slices) will always contain the segment of the laser beam that iluminates the sample (rotations and translations cannot change that).
This functions creates a parallelepiped volume scan pattern and in the default orientation (first axis is the depth "z", second axis is "x", third axis is "y") the slices will be accomodated along the "y" axis. Hence, the number of slices in the stack is given by the parameter `SizeY`. Afterwards, this parameter may be retrieved by invoking the function getScanPatternPropertyInt() with the argument ScanPattern_Cycles.
Depending on the setting for ScanPatternApodizationType, there will be either one apodization for the entire volume (ScanPattern_ApoOneForAll) or a single apodization for each B-scan (ScanPattern_ApoEachBScan).
The volume pattern with ScanPatternAcquisitionOrder set to ScanPattern_AcqOrderAll consists of a single uninterrupted scan and all data is acquired in a single measurement. The complete volume will be returned in one raw data (RawDataHandle) by calling getRawData().
Otherwise (i.e. if ScanPatternAcquisitionOrder is set to ScanPattern_AcqOrderFrameByFrame) the scan pattern consists of individual B-Scan measurements that get retrieved separately through separate invokations of get↩RawData(). In other words: The structure of the final dataset will be identical to the former case, but the stack will be returned slice-by-slice by calling getRawData(), once for each slice.
Notice that raw data refers to the spectra as acquired, without processing of any kind.

5.8.4.13 ScanPatternHandle createVolumePatternEx (ProbeHandle *Probe*, double *RangeX_mm*, int *SizeX*, double *RangeY_mm*, int *SizeY*, double *CenterX_mm*, double *CenterY_mm*, double *Angle_rad*, ScanPatternApodizationType *ApoType*, ScanPatternAcquisitionOrder *AcqOrder*)

Creates a simple volume pattern.

Parameters

in	*Probe*	A valid (non null) handle of a probe (ProbeHandle), previously generated by one of the functions initProbe, initDefaultProbe, or initProbeFromOCTFile.
in	*RangeX_mm*	The extension of the volume along the x-axis, expressed in mm, centered at the current position.
in	*SizeX*	The number of planes that cross the x-axis.

Parameters

in	*RangeY_mm*	The extension of the volume along the y-axis, expressed in mm, centered at the current position.
in	*SizeY*	The number of planes that cross the y-axis.
in	*CenterX_mm*	Center of the volume pattern
in	*CenterY_mm*	Center of the volume pattern
in	*Angle_rad*	Rotation in radians of the entire scan pattern
in	*ApoType*	The apodization type decides whether one apodization suffices for the whole set of measurements in the volume, or one apodization will be measured for each B-Scan (each segment).
in	*AcqOrder*	Dictates the acquisition strategy, as explained below, which reflects the way the user wants to retrieve the acquired data.

Returns

A valid (non null) handle to a scan pattern.

A volume scan pattern is actually a stack of B-scan patterns. At creation time the stack fills a parallelepiped volume in space but the shape can be subsequently modified if the individual slices are rotated, translated, or both (see explanation of functions rotateScanPatternEx(), shiftScanPatternEx() for more information). Notice that the individual B-scans (the slices) will always contain the segment of the laser beam that iluminates the sample (rotations and translations cannot change that).
This functions creates a parallelepiped volume scan pattern and in the default orientation (first axis is the depth "z", second axis is "x", third axis is "y") the slices will be accomodated along the "y" axis. Hence, the number of slices in the stack is given by the parameter `SizeY`. Afterwards, this parameter may be retrieved by invoking the function getScanPatternPropertyInt() with the argument ScanPattern_Cycles.
Depending on the setting for ScanPatternApodizationType, there will be either one apodization for the entire volume (ScanPattern_ApoOneForAll) or a single apodization for each B-scan (ScanPattern_ApoEachBScan).
The volume pattern with ScanPatternAcquisitionOrder set to ScanPattern_AcqOrderAll consists of a single uninterrupted scan and all data is acquired in a single measurement. The complete volume will be returned in one raw data (RawDataHandle) by calling getRawData().
Otherwise (i.e. if ScanPatternAcquisitionOrder is set to ScanPattern_AcqOrderFrameByFrame) the scan pattern consists of individual B-Scan measurements that get retrieved separately through separate invokations of get↩RawData(). In other words: The structure of the final dataset will be identical to the former case, but the stack will be returned slice-by-slice by calling getRawData(), once for each slice.
Notice that raw data refers to the spectra as acquired, without processing of any kind.

5.8.4.14 double expectedAcquisitionTime_s (**ScanPatternHandle** *ScanPattern,* **OCTDeviceHandle** *Dev*)

Returns the expected acquisition time of the scan pattern. Please.

5.8.4.15 **ScanPatternAcquisitionOrder** getScanPatternAcqOrder (**ScanPatternHandle** *ScanPattern*)

Returns the acquisition order of the scan pattern. See definition of ScanPatternAcquisitionOrder for detailed information.

5.8.4.16 void getScanPatternLUT (**ScanPatternHandle** *Pattern,* double ∗ *VoltsX,* double ∗ *VoltsY*)

Returns the voltages that will be applied to reach the positions to be scanned, in the specified scan pattern (Scan↩PatternHandle).

Parameters

in	*Pattern*	A valid (non null) handle of a scan pattern.
out	*VoltsX*	A pointer to the array in which the voltage for the X-positions will be written. If a nullptr is passed, nothing will be written. Otherwise it should have space for at least the size returned by getScanPatternLUTSize().
out	*VoltsY*	A pointer to the array in which the voltage for the Y-positions will be written. If a nullptr is passed, nothing will be written. Otherwise it should have space for at least the size returned by getScanPatternLUTSize().

The look-up-table mentioned here is a table with the voltages that will be sent to the galvos. It is computed afore-hand.

5.8.4.17 int getScanPatternLUTSize (**ScanPatternHandle** *Pattern*)

Returns the number of points in the specified scan pattern (ScanPatternHandle), including apodization and flyback.

Parameters

in	*Pattern*	A valid (non null) handle of a scan pattern.

Returns

The size of the look-up-table.

The look-up-table mentioned here is a table with the voltages that will be sent to the galvos. It is computed afore-hand.

5.8.4.18 double getScanPatternPropertyFloat (**ScanPatternHandle** *Pattern,* **ScanPatternPropertyFloat** *Selection*)

Returns the specified property of the scan pattern.

5.8.4.19 int getScanPatternPropertyInt (**ScanPatternHandle** *ScanPattern,* **ScanPatternPropertyInt** *Property*)

Returns the specified property of the scan pattern.

5.8.4.20 int getScanPoints (**ScanPatternHandle** *Pattern,* double ∗ *PosX_mm,* double ∗ *PosY_mm*)

Returns the position coordinates (in mm) of the points that in the specified scan pattern (ScanPatternHandle).

Parameters

in	*Pattern*	A valid (non null) handle of a scan pattern.
out	*PosX_mm*	A pointer to the array in which the X-positions (in mm) will be written. If a nullptr is passed, nothing will be written. Otherwise it should have space for at least the size returned by getScanPointsSize().
out	*PosY_mm*	A pointer to the array in which the Y-positions (in mm) will be written. If a nullptr is passed, nothing will be written. Otherwise it should have space for at least the size returned by getScanPointsSize().

5.8.4.21 void getScanPointsFromDataHandle (DataHandle *ScanPoints,* double ∗ *PosX_mm,* double ∗ *PosY_mm,* int ∗ *ScanIndices,* int *Length*)

Copies the scan points and scan indices from the DataHandle to the provided arrays.

Parameters

in	*ScanPoints*	The created DataHandle containing the provided points and scan indices.
out	*PosX_mm*	The pointer to the array of X-coords of the scan pattern, with length `Size`
out	*PosY_mm*	The pointer to the array of Y-coords of the scan pattern, with length `Size`
out	*ScanIndices*	The array specifies the assignment of each point to its B-scan, with length `Size`. The entries will go from 0 to number of B-scans - 1). The number of B-scans is defined with the entries of `ScanIndices`. To save scan points for a 2D-pattern set all entries to zero.
in	*Length*	The length of the arrays `FreeformCoordsX`, `FreeformCoordsY`, and `ScanIndices`.

5.8.4.22 int getScanPointsSize (ScanPatternHandle *Pattern*)

Returns the number of points in the specified scan pattern (ScanPatternHandle), including apodization and flyback.

Parameters

in	*Pattern*	A valid (non null) handle of a scan pattern.

Returns

The number of points in the scan pattern, including apodization and flyback.

5.8.4.23 int getSizeOfScanPointsFromDataHandle (DataHandle *ScanPoints*)

Returns the size of the scan points and scan indices in the DataHandle.

Parameters

in	*ScanPoints*	The DataHandle containing the provided points and scan indices.

Returns

The number of scan points.

Notice that in this case a data structure is used to hold data other than spectra or A-scans.

5.8.4.24 int getSizeOfScanPointsFromFile (const char ∗ *Filename,* ScanPointsDataFormat *DataFormat*)

Returns the number of scan points in the specified file.

Parameters

`in`	*Filename*	(including path) of the file that contains the scan points and indices.
`in`	*DataFormat*	The desired ScanPointsDataFormat.

Returns

The number of scan points in the give file.

5.8.4.25 BOOL isAcqTypeForScanPatternAvailable (ScanPatternHandle *ScanPattern,* AcquisitionType *AcqType*)

Returns whether the acquisition type is available for the scan pattern.

5.8.4.26 void loadScanPointsFromFile (double ∗ *ScanPosX_mm,* double ∗ *ScanPosY_mm,* int ∗ *ScanIndices,* int *Size,* const char ∗ *Filename,* ScanPointsDataFormat *DataFormat*)

Copies the scan points and scan indices from the file to the provided arrays.

Parameters

`out`	*ScanPosX_mm*	The pointer to the double array of x-positions of the scan pattern with length Size in mm
`out`	*ScanPosY_mm*	The pointer to the double array of y-positions of the scan pattern with length Size in mm
`out`	*ScanIndices*	The array specifies the assignment of each point to its B-scan. It has the length Size with entries from 0 to number of (B-scans - 1). The number of B-scans is defined with the entries of `ScanIndices`. To save scan points for a 2D-pattern set all entries to zero.
`in`	*Size*	The length of the arrays PositionsX, PositionsY and ScanIndices.
`in`	*Filename*	Path and name of the file containing the scan points and indices.
`in`	*DataFormat*	The selected ScanPointsDataFormat.

5.8.4.27 void rotateScanPattern (ScanPatternHandle *Pattern,* double *Angle_rad*)

Rotates the specfied pattern (ScanPatternHandle), counter-clockwise. The rotation is relative to current angle, not to the horizontal. That is, after multiple invokations of this function the final rotation is the addition of all rotations.

Parameters

`in`	*Pattern*	A valid (non null) handle of a scan pattern.
`in`	*Angle_rad*	The angle (expressed in radians) of the rotation.

5.8.4.28 void rotateScanPatternEx (ScanPatternHandle *Pattern,* double *Angle_rad,* int *Index*)

Counter-clockwise rotates the scan `Index` (0-based, i.e. zero for the first, one for the second, and so on) of the specfied volume scan pattern (ScanPatternHandle). The rotation is relative to current angle, not to the horizontal. That is, after multiple invokations of this function the final rotation is the addition of all rotations.

Parameters

in	*Pattern*	A valid (non null) handle of a volume scan pattern.
in	*Angle_rad*	The angle (expressed in radians) of the counter-clockwise rotation.
in	*Index*	The slice of the stack that should be rotated.

This function is specific of volume scan patterns, although only a slice of it will be rotated. A volume scan pattern is actually a stack of B-scan patterns. In the default orientation (first axis is the depth "z", second axis is "x", third axis is "y"), the slices will be accomodated along the "y" axis. The number of slices in the stack may be retrieved by invoking the function getScanPatternPropertyInt() with the argument ScanPattern_Cycles.

5.8.4.29 void saveScanPointsToFile (double ∗ *ScanPosX_mm,* double ∗ *ScanPosY_mm,* int ∗ *ScanIndices,* int *Size,* const char ∗ *Filename,* ScanPointsDataFormat *DataFormat*)

Saves the scan points and scan indices to a file with the specified ScanPointsDataFormat.

Parameters

in	*ScanPosX_mm*	The pointer to the double array of x-positions of the scan pattern with length Size in mm
in	*ScanPosY_mm*	The pointer to the double array of y-positions of the scan pattern with length Size in mm
in	*ScanIndices*	The array specifies the assignment of each point to its B-scan. It needs to have the length Size with entries from 0 to number of (B-scans - 1). The number of B-scans is defined with the entries of `ScanIndices`. To save scan points for a 2D-pattern set all entries to zero.
in	*Size*	The length of the arrays PositionsX, PositionsY and ScanIndices.
in	*Filename*	Path and name of the file containing the scan points and indices.
in	*DataFormat*	The specified ScanPointsDataFormat.

5.8.4.30 void shiftScanPattern (ScanPatternHandle *Pattern,* double *ShiftX,* double *ShiftY*)

Shifts the specified pattern (ScanPatternHandle). The shift is relative to current position, not to (0,0). That is, after multiple invokations of this function the final shift is the addition of all shifts.

Parameters

in	*Pattern*	A valid (non null) handle of a scan pattern.
in	*ShiftX*	The relative shift in the x-axis direction, expressed in mm.
in	*ShiftY*	The relative shift in the y-axis direction, expressed in mm.

5.8.4.31 void shiftScanPatternEx (ScanPatternHandle *Pattern,* double *ShiftX_mm,* double *ShiftY_mm,* BOOL *ShiftApo,* int *Index*)

Shifts the scan `Index` (0-based, i.e. zero for the first, one for the second, and so on) of the specified volume pattern (ScanPatternHandle). The shift is relative to current position, not to (0,0). That is, after multiple invokations of this function the final shift is the addition of all shifts.

Parameters

in	*Pattern*	A valid (non null) handle of a scan pattern.

Parameters

in	*ShiftX_mm*	The relative shift in the x-axis direction, expressed in mm.
in	*ShiftY_mm*	The relative shift in the y-axis direction, expressed in mm.
in	*ShiftApo*	TRUE if the apodization should also be shifted. FALSE otherwise.
in	*Index*	The slice of the stack that should be shifted.

This function is specific of volume scan patterns, although only a slice of it will be shifted. A volume scan pattern is actually a stack of B-scan patterns. In the default orientation (first axis is the depth "z", second axis is "x", third axis is "y"), the slices will be accomodated along the "y" axis. The number of slices in the stack may be retrieved by invoking the function getScanPatternPropertyInt() with the argument ScanPattern_Cycles.

5.8.4.32 void updateScanPattern (ScanPatternHandle *Pattern*)

Updates the specfied pattern (ScanPatternHandle) and computes the full look-up-table.

Parameters

in	*Pattern*	A valid (non null) handle of a scan pattern.

5.8.4.33 void zoomScanPattern (ScanPatternHandle *Pattern,* double *Factor*)

Zooms the specified pattern (ScanPatternHandle) around the optical center that coincides with the center of the camera image and the physical coordinates (0 mm,0 mm). The apodization position will not be modified.

Parameters

in	*Pattern*	A valid (non null) handle of a scan pattern.
in	*Factor*	The zoom factor.

5.9 Mathematical manipulations

Functions for pure mathematical manipulations (i.e. no physics involved).

Enumerations

- enum InterpolationMethod {
  Interpolation_Linear,
  Interpolation_Spline }

  Selects the interpolation method.
- enum BoundaryCondition {
  BoundaryCondition_Standard,
  BoundaryCondition_Natural,
  BoundaryCondition_Periodic }

  Selects the boundary conditions for the interpolation.

Functions

- SPECTRALRADAR_API void interpolatePoints2D (double ∗OrigPosX, double ∗OrigPosY, int Size, double ∗InterpPosX, double ∗InterpPosY, int NewSize, InterpolationMethod InterpolationMet, BoundaryCondition BoundaryCond)

  Interpolates the imaginary curve defined by the given sequence of points with the specified InterpolationMethod. The coordinates are abstract and this funcion has no sideffects that could affect any physical property. The original and the interpolated coordinates have a meaning for the user, but no consequence for SpectralRadar.
- SPECTRALRADAR_API void inflatePoints (double ∗PosX, double ∗PosY, int Size, double ∗InflatedPosX, double ∗InflatedPosY, int NumberOfInflationLines, double RangeOfInflation, InflationMethod Method)

  Inflates the provided curve in space with the specified InflationMethod. It can be used to create scan patterns of arbitrary forms with createFreeformScanPattern3DFromLUT if the used positions correspond to coordinates of the valid scan field in mm.
- SPECTRALRADAR_API void polynomialFitAndEval1D (int Size, const float ∗OrigPosX, const float ∗OrigY, int DegreePolynom, int EvalSize, const float ∗EvalPosX, float ∗EvalY)

  Computes the polynomial fit of the given 1D data.
- SPECTRALRADAR_API float calcParabolaMaximum (float x0, float y0, float yLeft, float yRight, float ∗peak↩ Height)

  Computes the x-position of the highest peak of the parabola given by the point x0, y0, yLeft, yRight. y0 needs to be the point with the highest value.

5.9.1 Detailed Description

Functions for pure mathematical manipulations (i.e. no physics involved).

5.9.2 Enumeration Type Documentation

5.9.2.1 enum BoundaryCondition

Selects the boundary conditions for the interpolation.

Enumerator

BoundaryCondition_Standard Matches the slope of the interpolated function at starting/end point to the following/previous points.

BoundaryCondition_Natural Natural boundary considtions used for interpolation which means the interpolated spline will turn into a straight line at the start/end.

BoundaryCondition_Periodic Peridoc boundary conditions used for interpolation which means that the interpolated function will interpret the points as a closed loop and use therefore the points from the start/end for interpolation of the end/start.

5.9.2.2 enum InterpolationMethod

Selects the interpolation method.

Enumerator

Interpolation_Linear	Linear interpolation.
Interpolation_Spline	Cubic B-Spline interpolation.

5.9.3 Function Documentation

5.9.3.1 float calcParabolaMaximum (float *x0,* float *y0,* float *yLeft,* float *yRight,* float ∗ *peakHeight*)

Computes the x-position of the highest peak of the parabola given by the point x0, y0, yLeft, yRight. y0 needs to be the point with the highest value.

Parameters

x0	The x-position of the point with the highest value y0.
y0	The value of x0.
yLeft	The y-value from the point left to x0. The distance (x0, xLeft) is assumed to be 1.
yRight	The y-value from the point right to x0. The distance (x0, xRight) is assumed to be 1.
peakHeight	The y-value of th highest peak of the parabloa will be written to this parameter.

5.9.3.2 void inflatePoints (double ∗ *PosX,* double ∗ *PosY,* int *Size,* double ∗ *InflatedPosX,* double ∗ *InflatedPosY,* int *NumberOfInflationLines,* double *RangeOfInflation,* InflationMethod *Method*)

Inflates the provided curve in space with the specified InflationMethod. It can be used to create scan patterns of arbitrary forms with createFreeformScanPattern3DFromLUT if the used positions correspond to coordinates of the valid scan field in mm.

Parameters

`in`	*PosX*	The pointer to the double array of x-positions of the scan pattern with length `Size`.
`in`	*PosY*	The pointer to the double array of y-positions of the scan pattern with length `Size`.
`in`	*Size*	The length of the arrays PositionsX, PositionsY and ScanIndices.
`out`	*InflatedPosX*	The pointer to the double array of x-positions of the scan pattern with length Size ∗ NumberOfInflationLines
`out`	*InflatedPosY*	The pointer to the double array of y-positions of the scan pattern with length Size ∗ NumberOfInflationLines
`in`	*NumberOfInflationLines*	The number of inflation lines. Please note that the length of the arrays InflatedPointsX and InflatedPointsY need to match.
`in`	*RangeOfInflation*	The range of inflation which results in the width of the created data object.
`in`	*Method*	The specified InflationMethod.

5.9.3.3 void interpolatePoints2D (double ∗ *OrigPosX,* double ∗ *OrigPosY,* int *Size,* double ∗ *InterpPosX,* double ∗ *InterpPosY,* int *NewSize,* InterpolationMethod *InterpolationMet,* BoundaryCondition *BoundaryCond*)

Interpolates the imaginary curve defined by the given sequence of points with the specified InterpolationMethod. The coordinates are abstract and this funcion has no sideffects that could affect any physical property. The original and the interpolated coordinates have a meaning for the user, but no consequence for SpectralRadar.

Parameters

`in`	*OrigPosX*	A pointer to the array of x-coords with length `Size`.
`in`	*OrigPosY*	A pointer to the array of y-coords with length `Size`.
`in`	*Size*	The length of the arrays `PositionsX`, `PositionsY` and `ScanIndices`.
`out`	*InterpPosX*	A pointer to the array of x-coords whose length should be `NewSize`.
`out`	*InterpPosY*	A pointer to the array of y-coords whose length should be `NewSize`.
`in`	*NewSize*	The number of interpolated points.
`in`	*InterpolationMet*	The desired InterpolationMethod.
`in`	*BoundaryCond*	The desired BoundaryCondition.

5.9.3.4 void polynomialFitAndEval1D (int *Size,* const float ∗ *OrigPosX,* const float ∗ *OrigY,* int *DegreePolynom,* int *EvalSize,* const float ∗ *EvalPosX,* float ∗ *EvalY*)

Computes the polynomial fit of the given 1D data.

Parameters

Size	The size of the arrays OrigPosX and OrigY
OrigPosX	The x-positions of the OrigY of the given data.
OrigY	The y-values to the belonging OrigPosX of the given data.
DegreePolynom	The degree of the polynomial for the fit.
EvalSize	The size of the array EvalPosX.
EvalPosX	The x-positions for evaluation the polynomial fit.
EvalY	The resulting y-values belonging to the given positions EvalPosX.

5.10 Acquisition

Functions for acquisition.

Enumerations

- enum AcquisitionType {
  Acquisition_AsyncContinuous,
  Acquisition_AsyncFinite,
  Acquisition_Sync }

  Determines the kind of acquisition process. The type of acquisition process affects e.g. whether consecutive B-scans are acquired or if it is possible to lose some data.

Functions

- SPECTRALRADAR_API size_t projectMemoryRequirement (OCTDeviceHandle Handle, ScanPatternHandle Pattern, AcquisitionType type)

  Returns the size of the required memory, e.g. for a raw data object, in bytes to acquire the scan pattern once.
- SPECTRALRADAR_API void startMeasurement (OCTDeviceHandle Dev, ScanPatternHandle Pattern, AcquisitionType Type)

  starts a continuous measurement BScans.
- SPECTRALRADAR_API void getRawData (OCTDeviceHandle Dev, RawDataHandle RawData)

  Acquires data and stores the data unprocessed.
- SPECTRALRADAR_API void getRawDataEx (OCTDeviceHandle Dev, RawDataHandle RawData, int CameraIdx)

  Acquires data with the specific camera given with camera index and stores the data unprocessed.
- SPECTRALRADAR_API void stopMeasurement (OCTDeviceHandle Dev)

  stops the current measurement.
- SPECTRALRADAR_API void measureSpectra (OCTDeviceHandle Dev, int NumberOfSpectra, RawDataHandle Raw)

  Acquires the desired number of spectra (raw data without processing) without moving galvo scanners.
- SPECTRALRADAR_API void measureSpectraEx (OCTDeviceHandle Dev, int NumberOfSpectra, RawDataHandle Raw, int CameraIndex)

  Acquires the desired number of spectra (raw data without processing) without moving galvo scanners, for the desired camera.

5.10.1 Detailed Description

Functions for acquisition.

5.10.2 Enumeration Type Documentation

5.10.2.1 enum AcquisitionType

Determines the kind of acquisition process. The type of acquisition process affects e.g. whether consecutive B-scans are acquired or if it is possible to lose some data.

Enumerator

Acquisition_AsyncContinuous Specifies an asynchronous infinite/continuous measurement. With this acquisition type an infinite loop to acquire the specified scan pattern will be started and stopped with the call of stopMeasurement. Several buffers will be created internally to hold the data of the specified scan pattern several times. With this acquisiton mode it is possible to lose data if the acquisition is faster than the copying from the framegrabber with getRawData. If you lose data you will always lose a whole frame, e.g. a whole B-scan. The acquisiton thread runs independently from the thread for grabbing the data to acquire the data as fast as possible. To get the information whether the data of a whole scan pattern got lost please use getRawDataPropertyInt with RawData_LostFrames when grabbing the data.

Acquisition_AsyncFinite Specifies an asynchronous finite measurement. With this acquisitions type enough memory is created internally to hold the data for the whole scan pattern once. Therefore it is guaranteed to grab all the data and not losing frames. Please note that it is possible to acquire the scan pattern once only with this acquisition mode.

Acquisition_Sync Specfies a synchronous measurement. With this acquisition mode the acquisition of the specified scan pattern will be started with the call of getRawData. You can interpret this acquisition type as a software trigger to start the measurement. To start the data acquisition externally please see the chapter in the software manual about external triggering.

5.10.3 Function Documentation

5.10.3.1 void getRawData (OCTDeviceHandle *Dev*, RawDataHandle *RawData*)

Acquires data and stores the data unprocessed.

In case of a synchronic measurement, this function will trigger the data acquisition. Otherwise it will return the latest acquired data buffer. In any case, this function will block until a data buffer is available (asynchronic measurements may satistfy this requirement immediately if a previously acquired buffer has not already been consumed).
This function is equivalent to

```
getRawDataEx(Dev, RawData, 0);
```

. In other words, in systems with more than just one camera, this function retrieves the raw data of the first camera. Notice that raw data refers to the spectra as acquired, without processing of any kind.

Parameters

in	*Dev*	A valid (non null) OCT device handle (OCTDeviceHandle), previously generated with the function initDevice.
in	*RawData*	A valid (non null) raw data handle (RawDataHandle).

5.10.3.2 void getRawDataEx (OCTDeviceHandle *Dev*, RawDataHandle *RawData*, int *CameraIdx*)

Acquires data with the specific camera given with camera index and stores the data unprocessed.

In case of a synchronic measurement, this function will trigger the data acquisition. Otherwise it will return the latest acquired data buffer. In any case, this function will block until a data buffer is available (asynchronic measurements may satistfy this requirement immediately if a previously acquired buffer has not already been consumed).
In systems with more than one camera, the hardware connections ensure that all cameras measure simultaneoulsy. That is, they have a common trigger. The master camera (index 0) will actually trigger the measurement of all slaves. for this reason, this function should be invoked first for the master (index 0) and only afterwards for the slaves (index greater than 0). If a slave triggers first, it will wait for the master (that is, this function call will block the

current execution thread). If the master triggers first, the buffer for the slave will be ready for pick up by the time the slave retrieves (without blocking).
Notice that raw data refers to the spectra as acquired, without processing of any kind.

Warning

> {Unless the program divides the acquistion in different threads, this function should be invoked first for the master camera (`CameraIdx` = 0) and only then for the slaves. Otherwise it will block for ever.}

Parameters

`in`	*Dev*	A valid (non null) OCT device handle (OCTDeviceHandle), previously generated with the function initDevice.
`in`	*RawData*	A valid (non null) raw data handle (RawDataHandle).
`in`	*CameraIdx*	The camera index (0-based, i.e. zero for the first = master, one for the second, and so on).

5.10.3.3 void measureSpectra (**OCTDeviceHandle** *Dev,* int *NumberOfSpectra,* **RawDataHandle** *Raw*)

Acquires the desired number of spectra (raw data without processing) without moving galvo scanners.

Parameters

`in`	*Dev*	A valid (non null) OCT device handle (OCTDeviceHandle), previously generated with the function initDevice.
`in`	*NumberOfSpectra*	The desired number of spectra.
`out`	*Raw*	A valid (non null) handle of raw data (RawDataHandle), where the acquired spectra will be stored. The meta data (dimensions, sizes, bytes per pixel, etc.) will be adjusted automatically.

This procedure assumes that there is no any ongoing measurement process (started with the function start←Measurement). The indicated number of measurements will be carried out. The user should not stop the measurement (this function will block till the whole data is ready).
If the hardware contains more than one camera, all cameras will be triggered, because the hardware has been setup to do so. This function will return raw data only for the first camera (the master). The raw data for the slaves, acquired simultaneously, will be available for retrieval any time afterwards (the function measureSpectraEx should be used).
This function blocks till the desired number of spectra get written in the indicated buffer (`Raw`).
Notice that raw data refers to the spectra as acquired, without processing of any kind.

5.10.3.4 void measureSpectraEx (**OCTDeviceHandle** *Dev,* int *NumberOfSpectra,* **RawDataHandle** *Raw,* int *CameraIndex*)

Acquires the desired number of spectra (raw data without processing) without moving galvo scanners, for the desired camera.

Parameters

`in`	*Dev*	A valid (non null) OCT device handle (OCTDeviceHandle), previously generated with the function initDevice.
`in`	*NumberOfSpectra*	The desired number of spectra.
`out`	*Raw*	A valid (non null) handle of raw data (RawDataHandle), where the acquired spectra will be stored. The meta data (dimensions, sizes, bytes per pixel, etc.) will be adjusted automatically.

Parameters

in	*CameraIndex*	The camera index (0-based, i.e. zero for the first = master, one for the second, and so on).

Warning

> {Unless the program divides the acquistion in different threads, this function should be invoked first for the master camera (`CameraIdx` = 0) and only then for the slaves. Otherwise it will block for ever.}

This procedure assumes that there is no any ongoing measurement process (started with the function startMeasurement). The indicated number of measurements will be carried out. The user should not stop the measurement (this function will block till the whole data is ready).
If the hardware contains more than one camera, all cameras will be triggered together with the first one (the master), because the hardware has been setup to do so. If `CameraIdx` is different from zero, i.e. a slave is meant, this function will retrieve the spectra measured together with the master. If those data happen to be already consumed, this function will block until the master triggers. Notice that in a single thread programming model, the program would stop execution for ever. For this reason, it is strongly adviced to invoke this function first for the master (`CameraIdx` = 0) and only then for the slaves.
This function will retrieve raw data only for the selected camera. The user must invoke this function for each camera separately, but in judicious order, as explained before.
This function blocks till the desired number of spectra get written in the indicated buffer (`Raw`).
Notice that raw data refers to the spectra as acquired, without processing of any kind.

5.10.3.5 size_t projectMemoryRequirement (OCTDeviceHandle *Handle,* ScanPatternHandle *Pattern,* AcquisitionType *type*)

Returns the size of the required memory, e.g. for a raw data object, in bytes to acquire the scan pattern once.

5.10.3.6 void startMeasurement (OCTDeviceHandle *Dev,* ScanPatternHandle *Pattern,* AcquisitionType *Type*)

starts a continuous measurement BScans.

Parameters

in	*Dev*	A valid (non null) OCT device handle (OCTDeviceHandle), previously generated with the function initDevice.
in	*Pattern*	A valid (non null) scan pattern handle (ScanPatternHandle).
in	*Type*	This parameter (AcquisitionType) decides whether the acquisition proceeds asynchronic (continuous or finite) or synchronic.

Scanning proceeds according to the specified scan pattern handle. In order to retrieve the acquired data, refer to the getRawData() function. To stop the measuring process, invoke stopMeasurement().
Synchronic measurements get triggered when the user invokes function that retrieves the data. Asynchronic measurements proceed in background, and the retrieving function returns the last available buffer that has been filled with fresh data. Asynchronic measurements can acquire a pre-specified number of buffers (finite) or continue indefinetely (continuous). If it is not possible to retrieve acquired data for a while, intermediate buffers might be skipped.

5.10.3.7 void stopMeasurement (OCTDeviceHandle *Dev*)

stops the current measurement.

Parameters

in	*Dev*	A valid (non null) OCT device handle (OCTDeviceHandle), previously generated with the function initDevice.

5.11 Processing

Standard Processing Routines.

Typedefs

- typedef struct C_Processing ∗ ProcessingHandle

  Handle for a processing routine.

Enumerations

- enum Processing_FFTType {
  Processing_StandardFFT,
  Processing_StandardNDFT,
  Processing_iFFT,
  Processing_NFFT1,
  Processing_NFFT2,
  Processing_NFFT3,
  Processing_NFFT4 }

  defindes the algorithm used for dechirping the input signal and Fourier transformation
- enum DispersionCorrectionType {
  Dispersion_None,
  Dispersion_QuadraticCoeff,
  Dispersion_Preset,
  Dispersion_Manual }

  To select the dispersion correction algorithm.
- enum ApodizationWindow {
  Apodization_Hann = 0,
  Apodization_Hamming = 1,
  Apodization_Gauss = 2,
  Apodization_TaperedCosine = 3,
  Apodization_Blackman = 4,
  Apodization_BlackmanHarris = 5,
  Apodization_LightSourceBased = 6,
  Apodization_Unknown = 999 }

  To select the apodization window function.
- enum ProcessingParameterInt {
  Processing_SpectrumAveraging,
  Processing_AScanAveraging,
  Processing_BScanAveraging,
  Processing_ZeroPadding,
  Processing_NumberOfThreads,
  Processing_FourierAveraging }

  Parameters that set the behavious of the processing algorithms.
- enum ProcessingParameterFloat {
  Processing_ApodizationDamping,
  Processing_MinElectrons,
  Processing_FFTOversampling,
  Processing_MaxSensorValue }

  Parameters that set the behaviour of the processing algorithms.

- enum CalibrationData {
  Calibration_OffsetErrors,
  Calibration_ApodizationSpectrum,
  Calibration_ApodizationVector,
  Calibration_Dispersion,
  Calibration_Chirp,
  Calibration_ExtendedAdjust,
  Calibration_FixedPattern }

  Data describing the calibration of the processing routines.
- enum ProcessingFlag {
  Processing_UseOffsetErrors,
  Processing_RemoveDCSpectrum,
  Processing_RemoveAdvancedDCSpectrum,
  Processing_UseApodization,
  Processing_UseScanForApodization,
  Processing_UseUndersamplingFilter,
  Processing_UseDispersionCompensation,
  Processing_UseDechirp,
  Processing_UseExtendedAdjust,
  Processing_FullRangeOutput,
  Processing_FilterDC,
  Processing_UseAutocorrCompensation,
  Processing_UseDEFR,
  Processing_OnlyWindowing,
  Processing_RemoveFixedPattern,
  Processing_CalculateSaturation }

  Flags that set the behaviour of the processing algorithms.
- enum ProcessingAveragingAlgorithm {
  Processing_Averaging_Min,
  Processing_Averaging_Mean,
  Processing_Averaging_Median,
  Processing_Averaging_Norm2,
  Processing_Averaging_Max,
  Processing_Averaging_Fourier_Min,
  Processing_Averaging_Fourier_Norm4,
  Processing_Averaging_Fourier_Max,
  Processing_Averaging_StandardDeviationAbs,
  Processing_Averaging_PhaseMatched }

  This sets the averaging algorithm to be used for processing.
- enum ApodizationWindowParameter {
  ApodizationWindowParameter_Sigma,
  ApodizationWindowParameter_Ratio,
  ApodizationWindowParameter_Frequency }

  Sets certain parameters that are used by the window functions to be applied during apodization.

Functions

- SPECTRALRADAR_API ProcessingHandle createProcessing (int SpectrumSize, int BytesPerRawPixel, B↩OOL Signed, float ScalingFactor, float MinElectrons, Processing_FFTType Type, float FFTOversampling)

  Creates processing routines with the desired properties.
- SPECTRALRADAR_API ProcessingHandle createProcessingForDevice (OCTDeviceHandle Dev)

  Creates processing routines for the specified device (OCTDeviceHandle).
- SPECTRALRADAR_API ProcessingHandle createProcessingForDeviceEx (OCTDeviceHandle Dev, int CameraIndex)

  Creates processing routines for the specified device (OCTDeviceHandle) with camera index.

- SPECTRALRADAR_API ProcessingHandle createProcessingForOCTFile (OCTFileHandle File)

  Creates processing routines for the specified OCT file (OCTFileHandle), such that the processing conditions are exactly the same as those when the file had been saved.
- SPECTRALRADAR_API ProcessingHandle createProcessingForOCTFileEx (OCTFileHandle File, const int CameraIndex)

  Creates processing routines for the specified OCT file (OCTFileHandle), such that the processing conditions are exactly the same as those when the file had been saved.
- SPECTRALRADAR_API int getInputSize (ProcessingHandle Proc)

  Returns the expected input size (pixels per spectrum) of the processing algorithms.
- SPECTRALRADAR_API int getAScanSize (ProcessingHandle Proc)

  Returns the number of pixels in an A-Scan that can be obtained (computed) with the given processing routines.
- SPECTRALRADAR_API void setApodizationWindow (ProcessingHandle Proc, ApodizationWindow Window)

  Sets the windowing function that will be used for apodization (this apodization has nothing to do with the reference spectra measured without a sample!). The selected windowing function will be used in all subsequent processings right before the fast Fourier transformation.
- SPECTRALRADAR_API ApodizationWindow getApodizationWindow (ProcessingHandle Proc)

  Returns the current windowing function that is being used for apodization, ApodizationWindow (this apodization is not the reference spectrum measured without a sample!).
- SPECTRALRADAR_API void setApodizationWindowParameter (ProcessingHandle Proc, Apodization↩ WindowParameter Selection, double Value)

  Sets the apodization window parameter, such as window width or ratio between constant and cosine part. Notice that this apodization is unrelated to the reference spectrum measured without a sample!.
- SPECTRALRADAR_API double getApodizationWindowParameter (ProcessingHandle Proc, Apodization↩ WindowParameter Selection)

  Gets the apodization window parameter, such as window width or ratio between constant and cosine part. Notice that this apodization is unrelated to the reference spectrum measured without a sample!.
- SPECTRALRADAR_API void getCurrentApodizationEdgeChannels (ProcessingHandle Proc, int ∗LeftPix, int ∗RightPix)

  Returns the pixel positions of the left/right edge channels of the current apodization. Here apodization refers to the reference spectra measured without sample.
- SPECTRALRADAR_API void setProcessingDechirpAlgorithm (ProcessingHandle Proc, Processing_FFT↩ Type Type, float Oversampling)

  Sets the algorithm to be used for dechirping the input spectra.
- SPECTRALRADAR_API void setProcessingParameterInt (ProcessingHandle Proc, ProcessingParameterInt Selection, int Value)

  Sets the specified integer value processing parameter.
- SPECTRALRADAR_API int getProcessingParameterInt (ProcessingHandle Proc, ProcessingParameterInt Selection)

  Returns the specified integer value processing parameter.
- SPECTRALRADAR_API void setProcessingParameterFloat (ProcessingHandle Proc, Processing↩ ParameterFloat Selection, double Value)

  Sets the specified floating point processing parameter.
- SPECTRALRADAR_API double getProcessingParameterFloat (ProcessingHandle Proc, Processing↩ ParameterFloat Selection)

  Gets the specified floating point processing parameter.
- SPECTRALRADAR_API void setProcessingFlag (ProcessingHandle Proc, ProcessingFlag Flag, BOOL Value)

  Sets the specified processing flag.
- SPECTRALRADAR_API BOOL getProcessingFlag (ProcessingHandle Proc, ProcessingFlag Flag)

  Returns TRUE if the specified processing flag is set, FALSE otherwise.
- SPECTRALRADAR_API void setProcessingAveragingAlgorithm (ProcessingHandle Proc, Processing↩ AveragingAlgorithm Algorithm)

  Sets the algorithm that will be used for averaging during the processing.

- SPECTRALRADAR_API void setCalibration (ProcessingHandle Proc, CalibrationData Selection, DataHandle Data)

  Sets the calibration data.
- SPECTRALRADAR_API void getCalibration (ProcessingHandle Proc, CalibrationData Selection, DataHandle Data)

  Retrieves the desired calibration vector.
- SPECTRALRADAR_API void measureCalibration (OCTDeviceHandle Dev, ProcessingHandle Proc, CalibrationData Selection)

  Measures the specified calibration parameters and uses them in subsequent processing.
- SPECTRALRADAR_API void measureCalibrationEx (OCTDeviceHandle Dev, ProcessingHandle Proc, CalibrationData Selection, int CameraIndex)

  Measures the specified calibration parameters and uses them in subsequent processing with specified camera index.
- SPECTRALRADAR_API void measureApodizationSpectra (OCTDeviceHandle Dev, ProbeHandle Probe, ProcessingHandle Proc)

  Measures the apodization spectra in the defined apodization position and size and uses them in subsequent processing.
- SPECTRALRADAR_API void saveCalibrationDefault (ProcessingHandle Proc, CalibrationData Selection)

  Saves the selected calibration in its default path. This same default path will be used by SpectralRadar in subsequent executions to retrieve the calibration data.
- SPECTRALRADAR_API void saveCalibrationDefaultEx (ProcessingHandle Proc, CalibrationData Selection, int CameraIndex)

  Saves the selected calibration in its default path, for the selected camera. This same default path will be used by SpectralRadar in.
- SPECTRALRADAR_API void saveCalibration (ProcessingHandle Proc, CalibrationData Selection, const char ∗Path)

  Saves the selected calibration in the specified path.
- SPECTRALRADAR_API void loadCalibration (ProcessingHandle Proc, CalibrationData Selection, const char ∗Path)

  Will load a specified calibration file and its content will be used for subsequent processing.
- SPECTRALRADAR_API void setSpectrumOutput (ProcessingHandle Proc, DataHandle Spectrum)

  Sets the location for the resulting spectral data.
- SPECTRALRADAR_API void setOffsetCorrectedSpectrumOutput (ProcessingHandle Proc, DataHandle OffsetCorrectedSpectrum)

  Sets the location for the resulting offset corrected spectral data.
- SPECTRALRADAR_API void setDCCorrectedSpectrumOutput (ProcessingHandle Proc, DataHandle DCCorrectedSpectrum)

  Sets the location for the resulting DC removed spectral data.
- SPECTRALRADAR_API void setApodizedSpectrumOutput (ProcessingHandle Proc, DataHandle ApodizedSpectrum)

  Sets the location for the resulting apodized spectral data.
- SPECTRALRADAR_API void setComplexDataOutput (ProcessingHandle Proc, ComplexDataHandle ComplexScan)

  Sets the pointer to the resulting complex scans that will be written after subsequent processing executions.
- SPECTRALRADAR_API void setProcessedDataOutput (ProcessingHandle Proc, DataHandle Scan)

  Sets the pointer to the resulting scans that will be written after subsequent processing executions.
- SPECTRALRADAR_API void setColoredDataOutput (ProcessingHandle Proc, ColoredDataHandle Scan, ColoringHandle Color)

  Sets the pointer to the resulting colored scans that will be written after subsequent processing executions.
- SPECTRALRADAR_API void setTransposedColoredDataOutput (ProcessingHandle Proc, ColoredDataHandle Scan, ColoringHandle Color)

  Sets the pointer to the resulting colored scans that will be written after subsequent processing executions. The orientation of the colored data will be transposed in such a way that the first axis (normally z-axis) will be the x-axis (the depth of each individual A-scan) and the second axis (normally x-axis) will be the z-axis.

- SPECTRALRADAR_API void executeProcessing (ProcessingHandle Proc, RawDataHandle RawData)

  Executes the processing. The results will be stored as requested through the functions setProcessedDataOutput(), setComplexDataOutput(), setColoredDataOutput() (including coloring properties) and similar ones. In all cases, sizes and ranges will be adjusted automatically to the right values.
- SPECTRALRADAR_API void clearProcessing (ProcessingHandle Proc)

  Clears the processing instance and frees all temporary memory that was associated with it. Processing threads will be stopped.
- SPECTRALRADAR_API void computeDispersion (DataHandle Spectrum1, DataHandle Spectrum2, DataHandle Chirp, DataHandle Disp)

  Computes the dispersion and chirp of the two provided spectra, where both spectra need to have been subjected to same dispersion mismatch. Both spectra need to have been acquired for different path length differences.
- SPECTRALRADAR_API void computeDispersionByCoeff (double Quadratic, DataHandle Chirp, DataHandle Disp)

  Computes dispersion by a quadratic approximation specified by the quadratic factor.
- SPECTRALRADAR_API void computeDispersionByImage (DataHandle LinearKSpectra, DataHandle Chirp, DataHandle Disp)

  Guesses the dispersion based on the spectral data specified. The spectral data needs to be linearized in wavenumber before using this function.
- SPECTRALRADAR_API int getNumberOfDispersionPresets (ProcessingHandle Proc)

  Gets the number of dispersion presets.
- SPECTRALRADAR_API const char ∗ getDispersionPresetName (ProcessingHandle Proc, int Index)

  Gets the name of the dispersion preset specified with index.
- SPECTRALRADAR_API void setDispersionPresetByName (ProcessingHandle Proc, const char ∗Name)

  Sets the dispersion preset specified with name.
- SPECTRALRADAR_API void setDispersionPresetByIndex (ProcessingHandle Proc, int Index)

  Sets the dispersion preset specified with index.
- SPECTRALRADAR_API void setDispersionPresets (ProcessingHandle Proc, ProbeHandle Probe)

  Sets the dispersion presets for the probe.
- SPECTRALRADAR_API Processing_FFTType getProcessing_FFTType (ProcessingHandle Proc)

  Retrieve the active FFT Type.
- SPECTRALRADAR_API void setDispersionCorrectionType (ProcessingHandle Proc, DispersionCorrectionType Type)

  Sets the active dispersion correction type.
- SPECTRALRADAR_API DispersionCorrectionType getDispersionCorrectionType (ProcessingHandle Proc)

  Sets the active dispersion correction type.
- SPECTRALRADAR_API void setDispersionQuadraticCoeff (ProcessingHandle Proc, double Coeff)

  Sets the coefficient for the quadratic correction of the dispersion.
- SPECTRALRADAR_API double getDispersionQuadraticCoeff (ProcessingHandle Proc)

  Sets the coefficient for the quadratic correction of the dispersion.
- SPECTRALRADAR_API const char ∗ getCurrentDispersionPresetName (ProcessingHandle Proc)

  Gets the name of the active dispersion preset.
- SPECTRALRADAR_API void computeLinearKRawData (ComplexDataHandle ComplexDataAfterFFT, DataHandle LinearKData)

  Computes the linear k raw data of the complex data after FFT by an inverse Fourier transform.
- SPECTRALRADAR_API void linearizeSpectralData (DataHandle SpectraIn, DataHandle SpectraOut, DataHandle Chirp)

  Linearizes the spectral data using the given chirp vector.

5.11.1 Detailed Description

Standard Processing Routines.

5.11.2 Typedef Documentation

5.11.2.1 ProcessingHandle

Handle for a processing routine.

The purpose of the processing routines is to compute A-Scans (light intensity as a function of depth) from spectra (light intensity as a function of wavelength). The former is typically stored in different types of data (DataHandle, ComplexDataHandle, ColoredDataHandle) whereas the latter is raw data (RawDataHandle).
A handle of processing routines can be obtained with one of the functions createProcessing, createProcessing←ForDevice, createProcessingForDeviceEx or createProcessingForOCTFile.

5.11.3 Enumeration Type Documentation

5.11.3.1 enum ApodizationWindow

To select the apodization window function.

Enumerator

- ***Apodization_Hann*** Hann window function.
- ***Apodization_Hamming*** Hamming window function.
- ***Apodization_Gauss*** Gaussian window function.
- ***Apodization_TaperedCosine*** Tapered cosine window function.
- ***Apodization_Blackman*** Blackman window function.
- ***Apodization_BlackmanHarris*** 4-Term Blackman-Harris window function
- ***Apodization_LightSourceBased*** The apodizatin function is determined, based on the shape of the light source at hand.

  Warning

  {This feature is still experimental.}
- ***Apodization_Unknown*** Unknown apodization window.

5.11.3.2 enum ApodizationWindowParameter

Sets certain parameters that are used by the window functions to be applied during apodization.

Enumerator

- ***ApodizationWindowParameter_Sigma*** Sets the width of a Gaussian apodization window.
- ***ApodizationWindowParameter_Ratio*** Sets the ratio of the constant to the cosine part when using a tapered cosine window.
- ***ApodizationWindowParameter_Frequency*** Sets the corner frequency of the filter applied when using a light-source based apodization.

  Warning

  {Light source based apodization is still experimental and might contatin bugs or decrease performance of the OCT system.}

5.11.3.3 enum CalibrationData

Data describing the calibration of the processing routines.

Enumerator

Calibration_OffsetErrors Calibration vector used as offset.

Calibration_ApodizationSpectrum Calibration data used as reference spectrum.

Calibration_ApodizationVector Calibration data used as apodization multiplicators.

Calibration_Dispersion Calibration data used to compensate for dispersion.

Calibration_Chirp Calibration data used for dechirping spectral data.

Calibration_ExtendedAdjust Calibration data used as extended adjust.

Calibration_FixedPattern Calibration data used as fixed scan pattern data.

5.11.3.4 enum DispersionCorrectionType

To select the dispersion correction algorithm.

Enumerator

Dispersion_None No software dispersion correction is used.

Dispersion_QuadraticCoeff Quadratic dispersion correction is used with the specified factor in setDispersionQuadraticCoeff.

Dispersion_Preset The specified dispersion preset from setDispersionPresets is used. For more information please see the documentation of setDispersionPresets.

Dispersion_Manual No software dispersion correction is used.

5.11.3.5 enum Processing_FFTType

defindes the algorithm used for dechirping the input signal and Fourier transformation

Enumerator

Processing_StandardFFT FFT with no dehchirp algorithm applied.

Processing_StandardNDFT Full matrix multiplication ("filter bank"). Mathematical precise dechirp, but rather slow.

Processing_iFFT Linear interpolation prior to FFT.

Processing_NFFT1 NFFT algorithm with parameter m=1.

Processing_NFFT2 NFFT algorithm with parameter m=2.

Processing_NFFT3 NFFT algorithm with parameter m=3.

Processing_NFFT4 NFFT algorithm with parameter m=4.

5.11.3.6 enum ProcessingAveragingAlgorithm

This sets the averaging algorithm to be used for processing.

Warning

{This features is still experimental and might contain bugs.}

Enumerator

Processing_Averaging_Mean Default.

5.11.3.7 enum ProcessingFlag

Flags that set the behaviour of the processing algorithms.

Enumerator

Processing_UseOffsetErrors Flag identifying whether to apply offset error removal. This flag is activated by default.

Processing_RemoveDCSpectrum Flag sets whether the DC spectrum as measured is to be removed from the spectral data. This flag is activated by default.

Processing_RemoveAdvancedDCSpectrum Flag sets whether the DC spectrum to be removed is rescaled by the respective spectrum intensity it is applied to. This flag is activated by default.

Processing_UseApodization Flag identifying whether to apply apodization. This flag is activated by default.

Processing_UseScanForApodization Flag to determine whether the acquired data is to be averaged in order to compute an apodization spectrum. This flag is deactivated by default.

Processing_UseUndersamplingFilter Flag to activate or deactivate a filter removing undersampled signals from the A-scan. This flag is deactivated by default.

Processing_UseDispersionCompensation Flag activating or deactivating dispersion compensation. This flag is deactivated by default.

Processing_UseDechirp Flag identifying whether to apply dechirp. This flag is activated by default.

Processing_UseExtendedAdjust Flag identifying whether to use extended adjust. This flag is deactivated by default.

Processing_FullRangeOutput Flag identifying whether to use full range output. This flag is deactivated by default.

Processing_FilterDC Experimental: Flag for an experimental lateral DC filtering algorithm. This flag is deactivated by default.

Processing_UseAutocorrCompensation Flag activating or deactivating autocorrelation compensation. This flag is deactivated by default.

Processing_UseDEFR Exprtimental: Toggles dispersion encoded full range processing mode, eliminating folding of the signal at the top. This flag is deactivated by default.

Processing_OnlyWindowing Flag deactivating deconvolution in apodization processing, using windowing only. This flag is deactivated by default.

Processing_RemoveFixedPattern Flag for removal of fixed pattern noise, used for swept source OCT systems. This flag is deactivated by default.

Processing_CalculateSaturation Flag to calculate sensor saturation, used in swept source OCT systems. This flag is deactivated by default.

5.11.3.8 enum ProcessingParameterFloat

Parameters that set the behaviour of the processing algorithms.

Enumerator

Processing_ApodizationDamping Sets how much influence newly acquired apodizations have compared to older ones.

Processing_MinElectrons Determines the minimum signal intensity on the edge channels of the spectra.

Warning

{Setting this value may seriously reduce performance of the system.}

Processing_MaxSensorValue Largest (absolute) value that the processing will expect for raw samples.

5.11.3.9 enum ProcessingParameterInt

Parameters that set the behavious of the processing algorithms.

Enumerator

Processing_SpectrumAveraging Identifyer for averaging of several subsequent spectra prior to Fourier transform.

Processing_AScanAveraging Identifyer for averaging the absolute values of several subsequent A-scan after Fourier transform.

Processing_BScanAveraging Averaging of subsequent B-scans.

Processing_ZeroPadding Identifier for zero padding prior to Fourier transformation.

Processing_NumberOfThreads The maximum number of threads to used by processing. A value of 0 indicates automatic selection, equal to the number of cores in the host PC.

Processing_FourierAveraging Averaging of fourier spectra.

5.11.4 Function Documentation

5.11.4.1 void clearProcessing (ProcessingHandle *Proc*)

Clears the processing instance and frees all temporary memory that was associated with it. Processing threads will be stopped.

Parameters

in	*Proc*	A handle of the processing routines (ProcessingHandle). If the handle is a nullptr, this function does nothing. In most cases this handle will have been previously obtained through one of the functions createProcessing, createProcessingForDevice, createProcessingForDeviceEx or createProcessingForOCTFile.

5.11.4.2 void computeDispersion (DataHandle *Spectrum1,* DataHandle *Spectrum2,* DataHandle *Chirp,* DataHandle *Disp*)

Computes the dispersion and chirp of the two provided spectra, where both spectra need to have been subjected to same dispersion mismatch. Both spectra need to have been acquired for different path length differences.

Parameters

in	*Spectrum1*	A valid (non null) handle of data (DataHandle) with an apodized spectrum, with the functions setApodizedSpectrumOutput() followed by executeProcessing(), measuring a test reflector positioned at a a distance different from the one used for the second parameter.
in	*Spectrum2*	A valid (non null) handle of data (DataHandle) with an apodized spectrum, with the functions setApodizedSpectrumOutput() followed by executeProcessing(), measuring a test reflector positioned at a a distance different from the one used for the first parameter.
out	*Chirp*	A valid (non null) handle of data (DataHandle) where the calculated chirp curve will be written.
out	*Disp*	A valid (non null) handle of data (DataHandle) where the calculated dispersion curve will be written.

For a detailed explanation, do please refer to the documentation of setDispersionPresets.

5.11.4.3 void computeDispersionByCoeff (double *Quadratic*, DataHandle *Chirp*, DataHandle *Disp*)

Computes dispersion by a quadratic approximation specified by the quadratic factor.

Parameters

in	*Quadratic*	The leading coefficient of the second order polynomia that will define the dispersion curve.
in	*Chirp*	A valid (non null) handle of data (DataHandle) where a valid chirp curve has been stored.
out	*Disp*	A valid (non null) handle of data (DataHandle) where the calculated dispersion curve will be written.

For a detailed explanation, do please refer to the documentation of setDispersionPresets.

5.11.4.4 void computeDispersionByImage (DataHandle *LinearKSpectra*, DataHandle *Chirp*, DataHandle *Disp*)

Guesses the dispersion based on the spectral data specified. The spectral data needs to be linearized in wavenumber before using this function.

Parameters

in	*LinearKSpectra*	A valid (non null) handle of data (DataHandle) where the input spectra is stored. The spectral data needs to be linearized in wavenumber (not wavelength) before using this function.
in	*Chirp*	A valid (non null) handle of data (DataHandle) where a valid chirp curve has been stored.
out	*Disp*	A valid (non null) handle of data (DataHandle) where the calculated dispersion curve will be written.

For a detailed explanation, do please refer to the documentation of setDispersionPresets.

5.11.4.5 void computeLinearKRawData (ComplexDataHandle *ComplexDataAfterFFT*, DataHandle *LinearKData*)

Computes the linear k raw data of the complex data after FFT by an inverse Fourier transform.

5.11.4.6 ProcessingHandle createProcessing (int *SpectrumSize*, int *BytesPerRawPixel*, BOOL *Signed*, float *ScalingFactor*, float *MinElectrons*, Processing_FFTType *Type*, float *FFTOversampling*)

Creates processing routines with the desired properties.

Parameters

in	*SpectrumSize*	The number of pixels in each spectrum.
in	*BytesPerRawPixel*	The number of bytes in each pixel (e.g. two for a 12-bit resolution). Currently, 1, 2, and 4-bytes per pixel are supported. 1 and 2-bytes per pixel assume an integer representation, whereas 4-bytes per pixel assumes a single precision floating point representation.
in	*Signed*	Indicates whether the value of each pixel is signed or not. This parameter is ignored in case of floating point representations.
in	*ScalingFactor*	A multiplicative constant to transform digital levels into the number of electrons actually freed.

Parameters

in	*MinElectrons*	A threshold. This value is used to identify the portions of the measured spectra (close to the edges) where the signal-to-noise ratio is too poor for any practical purposes. After the `ScalingFactor` has been applied to the digitized data (i.e. a spectrum has been measured), this threshold can be used to identify the portions near the edges that can be regarded as "near zero".
in	*Type*	Specifies the FFT algorithm (Processing_FFTType) that will combine the dechirping with the Fourier transform.
in	*FFTOversampling*	In case the selected FFT algorithm bases on oversampling, this parameter gives the factor.

Returns

A handle of the newly created processing routines (ProcessingHandle).

5.11.4.7 ProcessingHandle createProcessingForDevice (OCTDeviceHandle *Dev*)

Creates processing routines for the specified device (OCTDeviceHandle).

Parameters

in	*Dev*	A valid (non null) OCT device handle (OCTDeviceHandle), previously generated with the function initDevice.

Returns

A handle of the newly created processing routines (ProcessingHandle).

In systems containing several cameras, there should be one set of processing routines for each camera. The reason is that each camera has its own calibration, and the calibration is an integral part of the computations. This function creates and returns a handle only for the first camera. Thus, this function is intended for systems contining a single camera. In case of systems containing several cameras, the function createProcessingForDeviceEx should be used instead.

5.11.4.8 ProcessingHandle createProcessingForDeviceEx (OCTDeviceHandle *Dev,* int *CameraIndex*)

Creates processing routines for the specified device (OCTDeviceHandle) with camera index.

Parameters

in	*Dev*	A valid (non null) OCT device handle (OCTDeviceHandle), previously generated with the function initDevice.
in	*CameraIndex*	The camera index (0-based, i.e. zero for the first, one for the second, and so on).

Returns

A handle of the newly created processing routines (ProcessingHandle).

In systems containing several cameras, there should be one set of processing routines for each camera. The reason is that each camera has its own calibration, and the calibration is an integral part of the computations. This function creates and returns a handle only for the first camera. Thus, this function is intented for systems contining more than one camera. In case the second parameter (`CameraIndex`) is zero, this function is equivalent to createProcessingForDevice.

5.11.4.9 ProcessingHandle createProcessingForOCTFile (OCTFileHandle *File*)

Creates processing routines for the specified OCT file (OCTFileHandle), such that the processing conditions are exactly the same as those when the file had been saved.

Parameters

`in`	*File*	A valid (non null) OCT file handle (OCTFileHandle).

Returns

A handle of the newly created processing routines (ProcessingHandle).

5.11.4.10 ProcessingHandle createProcessingForOCTFileEx (OCTFileHandle *File*, const int *CameraIndex*)

Creates processing routines for the specified OCT file (OCTFileHandle), such that the processing conditions are exactly the same as those when the file had been saved.

Parameters

`in`	*File*	A valid (non null) OCT file handle (OCTFileHandle).
`in`	*CameraIndex*	The detector index (first camera has zero index).

Returns

A handle of the newly created processing routines (ProcessingHandle).

For systems with one camera, this function fall backs to createProcessingForOCTFile.

5.11.4.11 void executeProcessing (ProcessingHandle *Proc*, RawDataHandle *RawData*)

Executes the processing. The results will be stored as requested through the functions setProcessedDataOutput(), setComplexDataOutput(), setColoredDataOutput() (including coloring properties) and similar ones. In all cases, sizes and ranges will be adjusted automatically to the right values.

Parameters

`in`	*Proc*	A valid (non null) handle of the processing routines (ProcessingHandle), previously obtained through one of the functions createProcessing, createProcessingForDevice, createProcessingForDeviceEx or createProcessingForOCTFile.
`in`	*RawData*	A valid (non null) handle of raw data (RawDataHandle) with fresh measured data (e.g. acquired with the getRawData() function.

5.11.4.12 ApodizationWindow getApodizationWindow (ProcessingHandle *Proc*)

Returns the current windowing function that is being used for apodization, ApodizationWindow (this apodization is not the reference spectrum measured without a sample!).

Parameters

in	*Proc*	A valid (non null) handle of the processing routines (ProcessingHandle), previously obtained through one of the functions createProcessing, createProcessingForDevice, createProcessingForDeviceEx or createProcessingForOCTFile.

Returns

The current windowing function that is being used for apodization (ApodizationWindow) right before Fourier transformations.

5.11.4.13 double getApodizationWindowParameter (ProcessingHandle *Proc,* ApodizationWindowParameter *Selection*)

Gets the apodization window parameter, such as window width or ratio between constant and cosine part. Notice that this apodization is unrelated to the reference spectrum measured without a sample!.

Parameters

in	*Proc*	A valid (non null) handle of the processing routines (ProcessingHandle), previously obtained through one of the functions createProcessing, createProcessingForDevice, createProcessingForDeviceEx or createProcessingForOCTFile.
in	*Selection*	The desired parameter whose value shall be be retrieved (ApodizationWindowParameter).

Returns

The current value of the parameter.

5.11.4.14 int getAScanSize (ProcessingHandle *Proc*)

Returns the number of pixels in an A-Scan that can be obtained (computed) with the given processing routines.

Parameters

in	*Proc*	A valid (non null) handle of the processing routines (ProcessingHandle), previously obtained through one of the functions createProcessing, createProcessingForDevice, createProcessingForDeviceEx or createProcessingForOCTFile.

Returns

The number of pixels in an A-Scan that can be obtained (computed) with the given processing routines.

The returned number is identical to the number of rows in a finished B-Scan, that can also be retrieved (after the processing has been executed) by invoking one of the functions getDataPropertyInt, getComplexDataPropertyInt, or getColoredDataPropertyInt, passing the enumeration item Data_Size1 as the second parameter, and passing the respective data object (DataHandle, ComplexDataHandle, ColoredDataHandle) as the first parameter.

5.11.4.15 void getCalibration (ProcessingHandle *Proc*, CalibrationData *Selection*, DataHandle *Data*)

Retrieves the desired calibration vector.

Parameters

in	*Proc*	A valid (non null) handle of the processing routines (ProcessingHandle), previously obtained through one of the functions createProcessing, createProcessingForDevice, createProcessingForDeviceEx or createProcessingForOCTFile.
in	*Selection*	Indicates the calibration that will be set (CalibrationData).
out	*Data*	A valid handle (DataHandle) of the calibration data that will be retrieved. `Data` will be automatically resized for the data to fit in the structure.

5.11.4.16 SPECTRALRADAR_API void getCurrentApodizationEdgeChannels (ProcessingHandle *Proc*, int ∗ *LeftPix*, int ∗ *RightPix*)

Returns the pixel positions of the left/right edge channels of the current apodization. Here apodization refers to the reference spectra measured without sample.

Parameters

in	*Proc*	A valid (non null) handle of the processing routines (ProcessingHandle), previously obtained through one of the functions createProcessing, createProcessingForDevice, createProcessingForDeviceEx or createProcessingForOCTFile.
out	*LeftPix*	The address to store the position of the last pixel position, starting from the left, at which the intensity is too low for reliable computations. If a nullptr is given, nothing will be written on it.
out	*RightPix*	The address to store the position of the last pixel position, starting from the right, at which the intensity is too low for reliable computations. If a nullptr is given, nothing will be written on it.

The apodization spectra (i.e. the spectra measured without a sample) have regions, at their left and right edges, where the signal to noise ratio is too low for practical purposes. This function returns the position of the last pixel position (or channel) at which the measured intensity is insufficient for reliable computations.
Notice that the camera is upside down. Hence the right-most pixel refers to the shortest measured wavelength, and the left-most pixel refers to the longest measured wavelength.
The second and third pointers are addresses in memory managed by the user, not by SpectralRadar.

5.11.4.17 const char ∗ getCurrentDispersionPresetName (ProcessingHandle *Proc*)

Gets the name of the active dispersion preset.

Parameters

in	*Proc*	A valid (non null) handle of the processing routines (ProcessingHandle), previously obtained through one of the functions createProcessing, createProcessingForDevice, createProcessingForDeviceEx or createProcessingForOCTFile.

Returns

A zero terminated string with the name of the active dispersion preset.

5.11.4.18 DispersionCorrectionType getDispersionCorrectionType (ProcessingHandle *Proc*)

Sets the active dispersion correction type.

Parameters

in	*Proc*	A valid (non null) handle of the processing routines (ProcessingHandle), previously obtained through one of the functions createProcessing, createProcessingForDevice, createProcessingForDeviceEx or createProcessingForOCTFile.

Returns

The currently active dispersion correction algorithm (DispersionCorrectionType).

5.11.4.19 const char ∗ getDispersionPresetName (ProcessingHandle *Proc,* int *Index*)

Gets the name of the dispersion preset specified with index.

Parameters

in	*Proc*	A valid (non null) handle of the processing routines (ProcessingHandle), previously obtained through one of the functions createProcessing, createProcessingForDevice, createProcessingForDeviceEx or createProcessingForOCTFile.
in	*Index*	The index of the desired dispersion preset.

Returns

A zero terminated string with the name of the dispersion preset associated with the given index.

For a detailed explanation, do please refer to the documentation of setDispersionPresets.

5.11.4.20 double getDispersionQuadraticCoeff (ProcessingHandle *Proc*)

Sets the coefficient for the quadratic correction of the dispersion.

Parameters

in	*Proc*	A valid (non null) handle of the processing routines (ProcessingHandle), previously obtained through one of the functions createProcessing, createProcessingForDevice, createProcessingForDeviceEx or createProcessingForOCTFile.

Returns

The coefficient currently used for the quadratic correction of the dispersion.

5.11.4.21 int getInputSize (ProcessingHandle *Proc*)

Returns the expected input size (pixels per spectrum) of the processing algorithms.

Parameters

in	*Proc*	A valid (non null) handle of the processing routines (ProcessingHandle), previously obtained through one of the functions createProcessing, createProcessingForDevice, createProcessingForDeviceEx or createProcessingForOCTFile.

Returns

The number of pixels per spectrum.

This function is provided for convenience as processing routines can be used independently of the device.

5.11.4.22 int getNumberOfDispersionPresets (**ProcessingHandle** *Proc*)

Gets the number of dispersion presets.

Parameters

in	*Proc*	A valid (non null) handle of the processing routines (ProcessingHandle), previously obtained through one of the functions createProcessing, createProcessingForDevice, createProcessingForDeviceEx or createProcessingForOCTFile.

Returns

The number of dispersion presets.

For a detailed explanation, do please refer to the documentation of setDispersionPresets.

5.11.4.23 **Processing_FFTType** getProcessing_FFTType (**ProcessingHandle** *Proc*)

Retrieve the active FFT Type.

Parameters

in	*Proc*	A valid (non null) handle of the processing routines (ProcessingHandle), previously obtained through one of the functions createProcessing, createProcessingForDevice, createProcessingForDeviceEx or createProcessingForOCTFile.

Returns

the current FFT algorithm type (Processing_FFTType) that will combine the dechirping with the Fourier transform.

5.11.4.24 **BOOL** getProcessingFlag (**ProcessingHandle** *Proc,* **ProcessingFlag** *Flag*)

Returns TRUE if the specified processing flag is set, FALSE otherwise.

Parameters

in	*Proc*	A valid (non null) handle of the processing routines (ProcessingHandle), previously obtained through one of the functions createProcessing, createProcessingForDevice, createProcessingForDeviceEx or createProcessingForOCTFile.
in	*Flag*	The flag whose value will be retrieved.

Returns

The current value of the flag.

5.11.4.25 double getProcessingParameterFloat (**ProcessingHandle** *Proc,* **ProcessingParameterFloat** *Selection*)

Gets the specified floating point processing parameter.

Parameters

in	*Proc*	A valid (non null) handle of the processing routines (ProcessingHandle), previously obtained through one of the functions createProcessing, createProcessingForDevice, createProcessingForDeviceEx or createProcessingForOCTFile.
in	*Selection*	The floating point parameter whose value will be retrieved.

Returns

The current value of the floating point parameter.

5.11.4.26 int getProcessingParameterInt (**ProcessingHandle** *Proc,* **ProcessingParameterInt** *Selection*)

Returns the specified integer value processing parameter.

Parameters

in	*Proc*	A valid (non null) handle of the processing routines (ProcessingHandle), previously obtained through one of the functions createProcessing, createProcessingForDevice, createProcessingForDeviceEx or createProcessingForOCTFile.
in	*Selection*	The parameter whose value will be retrieved.

Returns

The current value of the integer parameter.

5.11.4.27 void linearizeSpectralData (**DataHandle** *SpectraIn,* **DataHandle** *SpectraOut,* **DataHandle** *Chirp*)

Linearizes the spectral data using the given chirp vector.

5.11.4.28 void loadCalibration (**ProcessingHandle** *Proc,* **CalibrationData** *Selection,* const char ∗ *Path*)

Will load a specified calibration file and its content will be used for subsequent processing.

Parameters

in	*Proc*	A valid (non null) handle of the processing routines (ProcessingHandle), previously obtained through one of the functions createProcessing, createProcessingForDevice, createProcessingForDeviceEx or createProcessingForOCTFile.
in	*Selection*	Indicates the calibration that will be saved (CalibrationData).
in	*Path*	A zero terminated string specifying the filename, including full path.

5.11.4.29 void measureApodizationSpectra (OCTDeviceHandle *Dev*, ProbeHandle *Probe*, ProcessingHandle *Proc*)

Measures the apodization spectra in the defined apodization position and size and uses them in subsequent processing.

Parameters

in	*Dev*	A valid (non null) OCT device handle (OCTDeviceHandle), previously generated with the function initDevice.
in	*Probe*	A valid (non null) probe handle (ProbeHandle), previously generated with the function initProbe.
in	*Proc*	A valid (non null) handle of the processing routines (ProcessingHandle), previously obtained through one of the functions createProcessing, createProcessingForDevice, createProcessingForDeviceEx or

If the hardware contains more than one camera, all cameras will be triggered, because the hardware has been setup to do so. This function will return raw data only for the first camera (the master). The raw data for the slaves, acquired simultaneously, will be available for retrieval any time afterwards (the function measureCalibrationEx should be used).

5.11.4.30 void measureCalibration (OCTDeviceHandle *Dev*, ProcessingHandle *Proc*, CalibrationData *Selection*)

Measures the specified calibration parameters and uses them in subsequent processing.

Parameters

in	*Dev*	A valid (non null) OCT device handle (OCTDeviceHandle), previously generated with the function initDevice.
in	*Proc*	A valid (non null) handle of the processing routines (ProcessingHandle), previously obtained through one of the functions createProcessing, createProcessingForDevice, createProcessingForDeviceEx or createProcessingForOCTFile.
in	*Selection*	Indicates the calibration that will be measured (CalibrationData).

If the hardware contains more than one camera, all cameras will be triggered, because the hardware has been setup to do so. This function will return raw data only for the first camera (the master). The raw data for the slaves, acquired simultaneously, will be available for retrieval any time afterwards (the function measureCalibrationEx should be used).
Using the parameters Calibration_ApodizationSpectrum or Calibration_ApodizationVector will acquire the apodization spectra without moving the mirrors to the apodization position. To acquire the spectra used for the processing in the apodization position use measureApodizationSpectra. Please note that the apodization spectra will not be acquired in thespecified apodization position from the ProbeHandle.

5.11.4.31 void measureCalibrationEx (OCTDeviceHandle *Dev,* ProcessingHandle *Proc,* CalibrationData *Selection,* int *CameraIndex*)

Measures the specified calibration parameters and uses them in subsequent processing with specified camera index.

Parameters

`in`	*Dev*	A valid (non null) OCT device handle (OCTDeviceHandle), previously generated with the function initDevice.
`in`	*Proc*	A valid (non null) handle of the processing routines (ProcessingHandle), previously obtained through one of the functions createProcessing, createProcessingForDevice, createProcessingForDeviceEx or createProcessingForOCTFile.
`in`	*Selection*	Indicates the calibration that will be measured (CalibrationData).
`in`	*CameraIndex*	The camera index (0-based, i.e. zero for the first = master, one for the second, and so on).

Warning

> {Unless the program divides the acquistion in different threads, this function should be invoked first for the master camera (`CameraIdx` = 0) and only then for the slaves. Otherwise it will block for ever.}

If the hardware contains more than one camera, all cameras will be triggered together with the first one (the master), because the hardware has been setup to do so. If `CameraIdx` is different from zero, i.e. a slave is meant, this function will retrieve the spectra measured together with the master. If those data happen to be already consumed, this function will block until the master triggers. Notice that in a single thread programming model, the program would stop execution for ever. For this reason, it is strongly adviced to invoke this function first for the master (`CameraIdx` = 0) and only then for the slaves.
Using the parameters Calibration_ApodizationSpectrum or Calibration_ApodizationVector will acquire the apodization spectra without moving the mirrors to the apodization position. To acquire the spectra used for the processing in the apodization position use measureApodizationSpectra.

5.11.4.32 void saveCalibration (ProcessingHandle *Proc,* CalibrationData *Selection,* const char ∗ *Path*)

Saves the selected calibration in the specified path.

Warning

> This will override your default calibration of the device if you specifiy the default path.

Parameters

`in`	*Proc*	A valid (non null) handle of the processing routines (ProcessingHandle), previously obtained through one of the functions createProcessing, createProcessingForDevice, createProcessingForDeviceEx or createProcessingForOCTFile.
`in`	*Selection*	Indicates the calibration that will be saved (CalibrationData).
`in`	*Path*	A zero terminated string specifying the filename, including full path.

5.11.4.33 void saveCalibrationDefault (ProcessingHandle *Proc,* CalibrationData *Selection*)

Saves the selected calibration in its default path. This same default path will be used by SpectralRadar in subsequent executions to retrieve the calibration data.

Warning

This will override your default calibration of the device.

Parameters

in	*Proc*	A valid (non null) handle of the processing routines (ProcessingHandle), previously obtained through one of the functions createProcessing, createProcessingForDevice, createProcessingForDeviceEx or createProcessingForOCTFile.
in	*Selection*	Indicates the calibration that will be saved (CalibrationData).

In systems with more than one camera, this function will only save calibration data pertaining to the first camera. For the other cameras use function saveCalibrationDefaultEx.

5.11.4.34 void saveCalibrationDefaultEx (ProcessingHandle *Proc*, CalibrationData *Selection*, int *CameraIndex*)

Saves the selected calibration in its default path, for the selected camera. This same default path will be used by SpectralRadar in.

subsequent executions to retrieve the calibration data.

Warning

This will override your default calibration of the device.

Parameters

in	*Proc*	A valid (non null) handle of the processing routines (ProcessingHandle), previously obtained through one of the functions createProcessing, createProcessingForDevice, createProcessingForDeviceEx or createProcessingForOCTFile.
in	*Selection*	Indicates the calibration that will be saved (CalibrationData).
in	*CameraIndex*	The camera index (0-based, i.e. zero for the first, one for the second, and so on).

This function will only save calibration data pertaining to the selected camera. To save the calibration of all cameras, multiple invokations are needed. the order plays no role.

5.11.4.35 void setApodizationWindow (ProcessingHandle *Proc*, ApodizationWindow *Window*)

Sets the windowing function that will be used for apodization (this apodization has nothing to do with the reference spectra measured without a sample!). The selected windowing function will be used in all subsequent processings right before the fast Fourier transformation.

Parameters

in	*Proc*	A valid (non null) handle of the processing routines (ProcessingHandle), previously obtained through one of the functions createProcessing, createProcessingForDevice, createProcessingForDeviceEx or createProcessingForOCTFile.
in	*Window*	The desired apodization window to be used for apodizations right before Fourier transformations.

The selection of a windowing function is a balance between the acceptable width of the main lobe (that is, how many

"frequency bins" does it take the response to reach half maximum power) and the attenuation of the side lobes (that is, what level of artifacts caused by the spectral leakage can be tolerated). As such, it depends on the paticular experiment. The default selection (Hann windowing) cannot be expected to fit everyone's needs.
If this function is not explicitly called, a Hann window will be assumed (Apodization_Hann).

5.11.4.36 void setApodizationWindowParameter (**ProcessingHandle** *Proc,* **ApodizationWindowParameter** *Selection,* double *Value*)

Sets the apodization window parameter, such as window width or ratio between constant and cosine part. Notice that this apodization is unrelated to the reference spectrum measured without a sample!.

Parameters

in	*Proc*	A valid (non null) handle of the processing routines (ProcessingHandle), previously obtained through one of the functions createProcessing, createProcessingForDevice, createProcessingForDeviceEx or createProcessingForOCTFile.
in	*Selection*	The desired parameter whose value will be changed (ApodizationWindowParameter).
in	*Value*	The desired value for the parameter.

5.11.4.37 void setApodizedSpectrumOutput (**ProcessingHandle** *Proc,* **DataHandle** *ApodizedSpectrum*)

Sets the location for the resulting apodized spectral data.

Parameters

in	*Proc*	A valid (non null) handle of the processing routines (ProcessingHandle), previously obtained through one of the functions createProcessing, createProcessingForDevice, createProcessingForDeviceEx or createProcessingForOCTFile.
in	*ApodizedSpectrum*	A valid (non null) data handle (DataHandle). Suitable sizes and ranges will be automatically set during the processing (executeProcessing).

5.11.4.38 void setCalibration (**ProcessingHandle** *Proc,* **CalibrationData** *Selection,* **DataHandle** *Data*)

Sets the calibration data.

Parameters

in	*Proc*	A valid (non null) handle of the processing routines (ProcessingHandle), previously obtained through one of the functions createProcessing, createProcessingForDevice, createProcessingForDeviceEx or createProcessingForOCTFile.
in	*Selection*	Indicates the calibration that will be set (CalibrationData).
in	*Data*	A valid handle (DataHandle) of the calibration data that will be set.

5.11.4.39 void setColoredDataOutput (**ProcessingHandle** *Proc,* **ColoredDataHandle** *Scan,* **ColoringHandle** *Color*)

Sets the pointer to the resulting colored scans that will be written after subsequent processing executions.

After the next completion of the function executeProcessing(), this data object will contain the colored amplitude of the scans.
If set to nullptr no colored data will be written in the subsequent processing executions.

Parameters

in	*Proc*	A valid (non null) handle of the processing routines (ProcessingHandle), previously obtained through one of the functions createProcessing, createProcessingForDevice, createProcessingForDeviceEx or createProcessingForOCTFile.
in	*Scan*	A valid (non null) colored data handle (ColoredDataHandle). Suitable sizes and ranges will be automatically set during the processing (executeProcessing).
in	*Color*	A valid (non null) coloring handle (ColoringHandle) as created, for example, with the functions createColoring32Bit() or createCustomColoring32Bit().

5.11.4.40 void setComplexDataOutput (ProcessingHandle *Proc,* ComplexDataHandle *ComplexScan*)

Sets the pointer to the resulting complex scans that will be written after subsequent processing executions.

Parameters

in	*Proc*	A valid (non null) handle of the processing routines (ProcessingHandle), previously obtained through one of the functions createProcessing, createProcessingForDevice, createProcessingForDeviceEx or createProcessingForOCTFile.
in	*ComplexScan*	A valid (non null) complex data handle (ComplexDataHandle). Suitable sizes and ranges will be automatically set during the processing (executeProcessing).

After the next completion of the function executeProcessing(), this complex data object will contain the real and imaginary parts of the scans.
If set to a nullptr, no complex data result will be written in the subsequent processing executions.

5.11.4.41 void setDCCorrectedSpectrumOutput (ProcessingHandle *Proc,* DataHandle *DCCorrectedSpectrum*)

Sets the location for the resulting DC removed spectral data.

Parameters

in	*Proc*	A valid (non null) handle of the processing routines (ProcessingHandle), previously obtained through one of the functions createProcessing, createProcessingForDevice, createProcessingForDeviceEx or createProcessingForOCTFile.
in	*DCCorrectedSpectrum*	A valid (non null) data handle (DataHandle). Suitable sizes and ranges will be automatically set during the processing (executeProcessing).

5.11.4.42 void setDispersionCorrectionType (ProcessingHandle *Proc,* DispersionCorrectionType *Type*)

Sets the active dispersion correction type.

Parameters

in	*Proc*	A valid (non null) handle of the processing routines (ProcessingHandle), previously obtained through one of the functions createProcessing, createProcessingForDevice, createProcessingForDeviceEx or createProcessingForOCTFile.
in	*Type*	The specification of the dispersion correction algorithm (DispersionCorrectionType).

5.11.4.43 void setDispersionPresetByIndex (ProcessingHandle *Proc,* int *Index*)

Sets the dispersion preset specified with index.

Parameters

in	*Proc*	A valid (non null) handle of the processing routines (ProcessingHandle), previously obtained through one of the functions createProcessing, createProcessingForDevice, createProcessingForDeviceEx or createProcessingForOCTFile.
in	*Index*	An index specifying the desired dispersion preset that has to be set.

For a detailed explanation, do please refer to the documentation of setDispersionPresets.

5.11.4.44 void setDispersionPresetByName (ProcessingHandle *Proc,* const char ∗ *Name*)

Sets the dispersion preset specified with name.

Parameters

in	*Proc*	A valid (non null) handle of the processing routines (ProcessingHandle), previously obtained through one of the functions createProcessing, createProcessingForDevice, createProcessingForDeviceEx or createProcessingForOCTFile.
in	*Name*	A zero terminated string with the name of the desired dispersion preset that has to be set.

For a detailed explanation, do please refer to the documentation of setDispersionPresets.

5.11.4.45 void setDispersionPresets (ProcessingHandle *Proc,* ProbeHandle *Probe*)

Sets the dispersion presets for the probe.

Hence it is suggested to use image quality as of criterion to set the coefficients, because this criterion usually works quite well. The quadratic coefficient can be easily found by using the ThorImage OCT software, and using the built-in quadratic slider and simultaneously looking for image quality and axial sharpness. Usually, the quadratic parameter alone gives rather good quality and dispersion correction and only for very broadband sources and strong dispersion higher coefficients are required.
To set higher coefficients, either this SDK is required or an entry "Dispersion_NameOfPreset" has to be added to the respective probe.ini file being used (see Settings Dialog of ThorImage OCT). This file is default located in C:\Program Files\Thorlabs\SpectralRadar\Config (or equivalent, if the software has been installed to another location).
In probe.ini, the relevant entry looks like:
Dispersion_Probe = 10.0, 2.0, -1.0
and this particular example would set three dispersion factors, the quadratic one being 10, the third degree 2, and fourth degree -1. Again, unfortunately, there is no option to set these automatically. The user has to experiment with different parameters and iteratively optimize for a sharp signal. After this entry has been added to the file probe.ini, the Preset dispersions menu will contain a new entry on the next software start of ThorImageOCT.
Of course, the presets added to probe.ini can also be used by the functions in this SDK. Each line should give a different preset-name, and after this function is invoked, all of them will be available through indeces (setDispersionPresetByIndex, getNumberOfDispersionPresets) or names (getDispersionPresetName, setDispersionPresetByName).

Parameters

in	*Proc*	A valid (non null) handle of the processing routines (ProcessingHandle), previously obtained through one of the functions createProcessing, createProcessingForDevice, createProcessingForDeviceEx or createProcessingForOCTFile.
in	*Probe*	A valid (non null) probe handle (ProbeHandle), previously generated with the function initProbe.

Unfortunately no really good and easy method to predict the dispersion coefficient(s) is offered here. The coefficients (currently) used in the software do not correspond to physically meaningful parameters, but are rather given in arbitrary units.

5.11.4.46 void setDispersionQuadraticCoeff (ProcessingHandle *Proc,* double *Coeff*)

Sets the coefficient for the quadratic correction of the dispersion.

Parameters

in	*Proc*	A valid (non null) handle of the processing routines (ProcessingHandle), previously obtained through one of the functions createProcessing, createProcessingForDevice, createProcessingForDeviceEx or createProcessingForOCTFile.
in	*Coeff*	The desired coefficient.

5.11.4.47 void setOffsetCorrectedSpectrumOutput (ProcessingHandle *Proc,* DataHandle *OffsetCorrectedSpectrum*)

Sets the location for the resulting offset corrected spectral data.

Parameters

in	*Proc*	A valid (non null) handle of the processing routines (ProcessingHandle), previously obtained through one of the functions createProcessing, createProcessingForDevice, createProcessingForDeviceEx or createProcessingForOCTFile.
in	*OffsetCorrectedSpectrum*	A valid (non null) data handle (DataHandle). Suitable sizes and ranges will be automatically set during the processing (executeProcessing).

5.11.4.48 void setProcessedDataOutput (ProcessingHandle *Proc,* DataHandle *Scan*)

Sets the pointer to the resulting scans that will be written after subsequent processing executions.

Parameters

in	*Proc*	A valid (non null) handle of the processing routines (ProcessingHandle), previously obtained through one of the functions createProcessing, createProcessingForDevice, createProcessingForDeviceEx or createProcessingForOCTFile.
in	*Scan*	A valid (non null) data handle (DataHandle). Suitable sizes and ranges will be automatically set during the processing (executeProcessing).

After the next completion of the function executeProcessing(), this data object will contain the amplitude (in dB) of

the scans.
If set to nullptr no processed floating point data in dB will be written in the subsequent processing executions.

5.11.4.49 void setProcessingAveragingAlgorithm (ProcessingHandle *Proc*, ProcessingAveragingAlgorithm *Algorithm*)

Sets the algorithm that will be used for averaging during the processing.

Parameters

in	*Proc*	A valid (non null) handle of the processing routines (ProcessingHandle), previously obtained through one of the functions createProcessing, createProcessingForDevice, createProcessingForDeviceEx or createProcessingForOCTFile.
in	*Algorithm*	The averaging algorithm (ProcessingAveragingAlgorithm). If this function is not explicetely invoked, the value Processing_Averaging_Mean can be assumed.

5.11.4.50 void setProcessingDechirpAlgorithm (ProcessingHandle *Proc*, Processing_FFTType *Type*, float *Oversampling*)

Sets the algorithm to be used for dechirping the input spectra.

Parameters

in	*Proc*	A valid (non null) handle of the processing routines (ProcessingHandle), previously obtained through one of the functions createProcessing, createProcessingForDevice, createProcessingForDeviceEx or createProcessingForOCTFile.
in	*Type*	Specifies the FFT algorithm (Processing_FFTType) that will combine the dechirping with the Fourier transform.
in	*Oversampling*	In case the selected FFT algorithm bases on oversampling, this parameter gives the factor.

5.11.4.51 void setProcessingFlag (ProcessingHandle *Proc*, ProcessingFlag *Flag*, BOOL *Value*)

Sets the specified processing flag.

Parameters

in	*Proc*	A valid (non null) handle of the processing routines (ProcessingHandle), previously obtained through one of the functions createProcessing, createProcessingForDevice, createProcessingForDeviceEx or createProcessingForOCTFile.
in	*Flag*	The flag whose value will be modified.
in	*Value*	The desired value for the flag.

5.11.4.52 void setProcessingParameterFloat (ProcessingHandle *Proc*, ProcessingParameterFloat *Selection*, double *Value*)

Sets the specified floating point processing parameter.

Parameters

in	*Proc*	A valid (non null) handle of the processing routines (ProcessingHandle), previously obtained through one of the functions createProcessing, createProcessingForDevice, createProcessingForDeviceEx or createProcessingForOCTFile.
in	*Selection*	The floating point parameter whose value will be modified.
in	*Value*	The desired value for the floating point parameter.

5.11.4.53 void setProcessingParameterInt (ProcessingHandle *Proc,* ProcessingParameterInt *Selection,* int *Value*)

Sets the specified integer value processing parameter.

Parameters

in	*Proc*	A valid (non null) handle of the processing routines (ProcessingHandle), previously obtained through one of the functions createProcessing, createProcessingForDevice, createProcessingForDeviceEx or createProcessingForOCTFile.
in	*Selection*	The parameter whose value will be modified.
in	*Value*	The desired value for the integer parameter.

5.11.4.54 void setSpectrumOutput (ProcessingHandle *Proc,* DataHandle *Spectrum*)

Sets the location for the resulting spectral data.

Parameters

in	*Proc*	A valid (non null) handle of the processing routines (ProcessingHandle), previously obtained through one of the functions createProcessing, createProcessingForDevice, createProcessingForDeviceEx or createProcessingForOCTFile.
in	*Spectrum*	A valid (non null) data handle (DataHandle). Suitable sizes and ranges will be automatically set during the processing (executeProcessing).

5.11.4.55 void setTransposedColoredDataOutput (ProcessingHandle *Proc,* ColoredDataHandle *Scan,* ColoringHandle *Color*)

Sets the pointer to the resulting colored scans that will be written after subsequent processing executions. The orientation of the colored data will be transposed in such a way that the first axis (normally z-axis) will be the x-axis (the depth of each individual A-scan) and the second axis (normally x-axis) will be the z-axis.

After the next completion of the function executeProcessing(), this data object will contain the transposed colored amplitude of the scans.
If set to nullptr no colored data will be written in the subsequent processing executions.

Parameters

in	*Proc*	A valid (non null) handle of the processing routines (ProcessingHandle), previously obtained through one of the functions createProcessing, createProcessingForDevice, createProcessingForDeviceEx or createProcessingForOCTFile.
in	*Scan*	A valid (non null) colored data handle (ColoredDataHandle). Suitable sizes and ranges will be automatically set during the processing (executeProcessing).
in	*Color*	A valid (non null) coloring handle (ColoringHandle) as created, for example, with the functions createColoring32Bit() or createCustomColoring32Bit().

5.12 Export and Import

Functionality to store data to disk and load it from there.

Enumerations

- enum DataExportFormat {
DataExport_SRM,
DataExport_RAW,
DataExport_CSV,
DataExport_TXT,
DataExport_TableTXT,
DataExport_Fits,
DataExport_VFF,
DataExport_VTK,
DataExport_TIFF }

  Export format for any data represented by a DataHandle.
- enum ComplexDataExportFormat { ComplexDataExport_RAW }

  Export format for complex data.
- enum ColoredDataExportFormat {
ColoredDataExport_SRM,
ColoredDataExport_RAW,
ColoredDataExport_BMP,
ColoredDataExport_PNG,
ColoredDataExport_JPG,
ColoredDataExport_PDF,
ColoredDataExport_TIFF }

  Export format for images (ColoredDataHandle).
- enum DataImportFormat { DataImport_SRM }

  Supported import format to load data from disk.
- enum RawDataExportFormat {
RawDataExport_RAW,
RawDataExport_SRR }

  Supported raw data export formats to store data to disk.
- enum RawDataImportFormat { RawDataImport_SRR }

  Supported raw data import formats to load data from disk.

Functions

- SPECTRALRADAR_API void exportData (DataHandle Data, DataExportFormat Format, const char ∗FileName)

  Exports data (DataHandle) to a file. The number of dimensions is handled automatically upon analysis of the first argument.
- SPECTRALRADAR_API void exportDataAsImage (DataHandle Data, ColoringHandle Color, ColoredDataExportFormat Format, Direction SliceNormalDirection, const char ∗FileName, int ExportOptionMask)

  Exports 2-dimensional and 3-dimensional data (DataHandle) as image data (such as BMP, PNG, JPEG, ...).
- SPECTRALRADAR_API void exportComplexData (ComplexDataHandle Data, ComplexDataExportFormat Format, const char ∗FileName)

  Exports 1-, 2- and 3-dimensional complex data (ComplexDataHandle)
- SPECTRALRADAR_API void exportColoredData (ColoredDataHandle Data, ColoredDataExportFormat Format, Direction SliceNormalDirection, const char ∗FileName, int ExportOptionMask)

  Exports colored data (ColoredDataHandle).

- SPECTRALRADAR_API void importColoredData (ColoredDataHandle ColoredData, DataImportFormat Format, const char ∗FileName)

  Imports colored data (ColoredDataHandle) with the specified format and copied it into a data object (ColoredDataHandle)
- SPECTRALRADAR_API void importData (DataHandle Data, DataImportFormat Format, const char ∗FileName)

  Imports data with the specified format and copies it into a data object (DataHandle).
- SPECTRALRADAR_API void exportRawData (RawDataHandle Raw, RawDataExportFormat Format, const char ∗FileName)

  Exports the specified data to disk.
- SPECTRALRADAR_API void importRawData (RawDataHandle Raw, RawDataImportFormat Format, const char ∗FileName)

  Imports the specified data from disk.

ExportOptions

Specifies additional export options to be used with functions such as exportDataAsImage(). Multiple options can be combined by bit-wise or ("|"). Different options can be used for different export format. If an option is not supported by an export format, it is ignored.

- const int ExportOption_None = 0x00000000
- const int ExportOption_DrawScaleBar = 0x00000001

  Draw scale bar on exported image.
- const int ExportOption_DrawMarkers = 0x00000002

  Draw markers on exported image.
- const int ExportOption_UsePhysicalAspectRatio = 0x00000004

  Honor physical aspect ratio when exporting data (width and height of each pixel will have the same physical dimensions).
- const int ExportOption_Flip_X_Axis = 0x00000008

  Flip X-axis.
- const int ExportOption_Flip_Y_Axis = 0x00000010

  Flip Y-axis.
- const int ExportOption_Flip_Z_Axis = 0x00000020

  Flip Z-axis.

5.12.1 Detailed Description

Functionality to store data to disk and load it from there.

5.12.2 Enumeration Type Documentation

5.12.2.1 enum ColoredDataExportFormat

Export format for images (ColoredDataHandle).

Enumerator

ColoredDataExport_SRM Spectral Radar Metaformat, containing no data but all additinal parameters, such as spacing, size, etc.

ColoredDataExport_RAW RAW data format containing the data of the object as binary, 32-bit unsigned integer values, little endian. The concrete format of the data depends on the colored data object (Colored↩ DataHandle). In most cases it will be RGB32 or RGBA32.

ColoredDataExport_BMP BMP - Bitmap image format.

ColoredDataExport_PNG PNG image format.

ColoredDataExport_JPG JPG/JPEG image format.

ColoredDataExport_PDF PDF image format.

ColoredDataExport_TIFF TIFF image format.

5.12.2.2 enum ComplexDataExportFormat

Export format for complex data.

Enumerator

ComplexDataExport_RAW RAW data format containg binary data.

5.12.2.3 enum DataExportFormat

Export format for any data represented by a DataHandle.

Enumerator

DataExport_SRM Spectral Radar Metaformat, containing no data but many additinal parameters, such as spacing, size, etc.

DataExport_RAW RAW data format containing the data of the object as binary, single precision floating point values, little endian.

DataExport_CSV CSV (Comma Seperated Values) is a text file having all values stored, comma seperated and human readable.

DataExport_TXT TXT is a text file having all values stored space seperated and human readable.

DataExport_TableTXT TableTXT is a human readable text-file in a table like format, having the physical 1- and 2-axis as first two columns and the data value as third. Currently only works for 1D- and 2D-Data.

DataExport_Fits FITS Data format.

DataExport_VFF VFF data format.

DataExport_VTK VTK data format.

DataExport_TIFF TIFF Data format as 32-bit floating point numbers.

5.12.2.4 enum DataImportFormat

Supported import format to load data from disk.

Enumerator

DataImport_SRM Spectral Radar Metaformat, containing no data but all additinal parameters, such as spacing, size, etc. It is searched for an appropriate file with same name but different extension containg the according data.

5.12.2.5 enum RawDataExportFormat

Supported raw data export formats to store data to disk.

Enumerator

RawDataExport_RAW Single precision floating point raw data.

RawDataExport_SRR Spectral Radar raw data format, specified additional information such as apodization scans, scan range, etc.

5.12.2.6 enum RawDataImportFormat

Supported raw data import formats to load data from disk.

Enumerator

RawDataImport_SRR Spectral Radar raw data-format, specified additional information such as apodization scans, scan range, etc.

5.12.3 Function Documentation

5.12.3.1 void exportColoredData (ColoredDataHandle *Data,* ColoredDataExportFormat *Format,* Direction *SliceNormalDirection,* const char ∗ *FileName,* int *ExportOptionMask*)

Exports colored data (ColoredDataHandle).

Parameters

in	*Data*	A valid (non null) colored-data handle of the data (ColoredDataHandle). These data may be multi-dimensional.
in	*Format*	The desired data-format, to be selected among those in ColoredDataExportFormat.
in	*SliceNormalDirection*	Specifies the direction normal to the generated pictures (to be chosen among those elements in Direction).
in	*FileName*	A zero-terminated string specifying the full pathname to the file to be written. Notice that backslashes should be escaped with an additional backslash.
in	*ExportOptionMask*	An OR-ed combination of the flags ExportOption_None, ExportOption_DrawScaleBar, ExportOption_DrawMarkers, and ExportOption_UsePhysicalAspectRatio.

5.12.3.2 void exportComplexData (ComplexDataHandle *Data,* ComplexDataExportFormat *Format,* const char ∗ *FileName*)

Exports 1-, 2- and 3-dimensional complex data (ComplexDataHandle)

Parameters

in	*Data*	A valid (non null) complex-data handle of the data (ComplexDataHandle). These data may be multi-dimensional.
in	*Format*	The desired data-format, to be selected among those in ComplexDataExportFormat.
in	*FileName*	A zero-terminated string specifying the full pathname to the file to be written. Notice that backslashes should be escaped with an additional backslash.

5.12.3.3 void exportData (**DataHandle** *Data,* **DataExportFormat** *Format,* const char ∗ *FileName*)

Exports data (DataHandle) to a file. The number of dimensions is handled automatically upon analysis of the first argument.

Parameters

`in`	*Data*	A valid (non null) data handle of the data (DataHandle). These data may be multi-dimensional.
`in`	*Format*	The desired data-format, to be selected among those in DataExportFormat.
`in`	*FileName*	A zero-terminated string specifying the full pathname to the file to be written. Notice that backslashes should be escaped with an additional backslash.

5.12.3.4 void exportDataAsImage (**DataHandle** *Data,* **ColoringHandle** *Color,* **ColoredDataExportFormat** *Format,* **Direction** *SliceNormalDirection,* const char ∗ *FileName,* int *ExportOptionMask*)

Exports 2-dimensional and 3-dimensional data (DataHandle) as image data (such as BMP, PNG, JPEG, ...).

Parameters

`in`	*Data*	A valid (non null) data handle of the data (DataHandle). These data may be multi-dimensional.
`in`	*Color*	A valid (non null) coloring handle (ColoringHandle) as created, for example, with the functions createColoring32Bit() or createCustomColoring32Bit().
`in`	*Format*	The desired data-format, to be selected among those in ColoredDataExportFormat.
`in`	*SliceNormalDirection*	Specifies the direction normal to the generated pictures (to be chosen among those elements in Direction).
`in`	*FileName*	A zero-terminated string specifying the full pathname to the file to be written. Notice that backslashes should be escaped with an additional backslash.
`in`	*ExportOptionMask*	An OR-ed combination of the flags ExportOption_None, ExportOption_DrawScaleBar, ExportOption_DrawMarkers, and ExportOption_UsePhysicalAspectRatio.

5.12.3.5 void exportRawData (**RawDataHandle** *Raw,* **RawDataExportFormat** *Format,* const char ∗ *FileName*)

Exports the specified data to disk.

Parameters

`in`	*Raw*	A valid (non null) raw-data handle of the data (RawDataHandle).
`in`	*Format*	The desired data-format to be stored in the file (the supported ones are the items of RawDataExportFormat).
`in`	*FileName*	A zero-terminated string specifying the full pathname to the file to be written. Notice that backslashes should be escaped with an additional backslash.

Notice that raw data refers to the spectra as acquired, without processing of any kind.

5.12.3.6 void importColoredData (ColoredDataHandle *ColoredData,* DataImportFormat *Format,* const char ∗ *FileName*)

Imports colored data (ColoredDataHandle) with the specified format and copied it into a data object (ColoredDataHandle)

Parameters

out	*ColoredData*	A valid (non null) colored-data handle of the data (ColoredDataHandle). These data may be multi-dimensional.
in	*Format*	The data-format stored in the file (the supported ones are the items of DataImportFormat).
in	*FileName*	A zero-terminated string specifying the full pathname to the file to be written. Notice that backslashes should be escaped with an additional backslash.

5.12.3.7 void importData (DataHandle *Data,* DataImportFormat *Format,* const char ∗ *FileName*)

Imports data with the specified format and copies it into a data object (DataHandle).

Parameters

out	*Data*	A valid (non null) data handle of the data (DataHandle). These data may be multi-dimensional.
in	*Format*	The data-format stored in the file (the supported ones are the items of DataImportFormat).
in	*FileName*	A zero-terminated string specifying the full pathname to the file to be written. Notice that backslashes should be escaped with an additional backslash.

5.12.3.8 void importRawData (RawDataHandle *Raw,* RawDataImportFormat *Format,* const char ∗ *FileName*)

Imports the specified data from disk.

Parameters

out	*Raw*	A valid (non null) raw-data handle of the data (RawDataHandle).
in	*Format*	The data-format stored in the file (the supported ones are the items of RawDataImportFormat).
in	*FileName*	A zero-terminated string specifying the full pathname to the file to be written. Notice that backslashes should be escaped with an additional backslash.

Notice that raw data refers to the spectra as acquired, without processing of any kind.

5.12.4 Variable Documentation

5.12.4.1 const int ExportOption_DrawMarkers = 0x00000002

Draw markers on exported image.

5.12.4.2 const int ExportOption_DrawScaleBar = 0x00000001

Draw scale bar on exported image.

5.12.4.3 const int ExportOption_Flip_X_Axis = 0x00000008

Flip X-axis.

5.12.4.4 const int ExportOption_Flip_Y_Axis = 0x00000010

Flip Y-axis.

5.12.4.5 const int ExportOption_Flip_Z_Axis = 0x00000020

Flip Z-axis.

5.12.4.6 const int ExportOption_None = 0x00000000

For default or no specific export options.

5.12.4.7 const int ExportOption_UsePhysicalAspectRatio = 0x00000004

Honor physical aspect ratio when exporting data (width and height of each pixel will have the same physical dimensions).

5.13 Volume

Functionality to store and access volume data.

Enumerations

- enum Direction {
  Direction_1,
  Direction_2,
  Direction_3 }

  Specifies a direction. In the default orientation, the first orientation is the Z-axis (parallel to the illumination-ray during the measurement), the second is the X-axis, and the third is the Y-axis.
- enum Plane2D {
  Plane2D_12,
  Plane2D_23,
  Plane2D_13 }

  Planes for slices of the volume data.

Functions

- SPECTRALRADAR_API void appendRawData (RawDataHandle Raw, RawDataHandle DataToAppend, Direction Dir)

  Appends the new raw data to the old raw data perpendicular to the specified direction.
- SPECTRALRADAR_API void getRawDataSliceAtIndex (RawDataHandle Raw, RawDataHandle Slice, Direction SliceNormalDirection, int Index)

  Returns a slice of raw data perpendicular to the specified direction at the specified index.
- SPECTRALRADAR_API double analyzeData (DataHandle Data, DataAnalyzation Selection)

  Analyzes the given data, extracts the selected feature, and returns the computed value.
- SPECTRALRADAR_API double analyzeAScan (DataHandle Data, AScanAnalyzation Selection)

  Analyzes the given A-scan data, extracts the selected feature, and returns the computed value.
- SPECTRALRADAR_API void transposeData (DataHandle DataIn, DataHandle DataOut)

  Transposes the given data and writes the result to DataOut. First and second axes will be swaped.
- SPECTRALRADAR_API void transposeDataInplace (DataHandle Data)

  Transposes the given Data. First and second axes will be swaped.
- SPECTRALRADAR_API void transposeAndScaleData (DataHandle DataIn, DataHandle DataOut, float Min, float Max)

  Transposes the given data and writes the result to DataOut. First and second axes will be swaped, and the range of the entries will be scaled in such a way, that the range [Min,Max] will be mapped onto the range [0,1].
- SPECTRALRADAR_API void normalizeData (DataHandle Data, float Min, float Max)

  Scales the given data in such a way, that the range [Min, Max] is mapped onto the range [0,1].
- SPECTRALRADAR_API void getDataSliceAtPos (DataHandle Data, DataHandle Slice, Direction Slice↩NormalDirection, double Pos_mm)

  Returns a slice of data perpendicular to the specified direction at the specified position.
- SPECTRALRADAR_API void getComplexDataSlicePos (ComplexDataHandle Data, ComplexDataHandle Slice, Direction SliceNormalDirection, double Pos_mm)

  Returns a slice of complex data perpendicular to the specified direction at the specified position.
- SPECTRALRADAR_API void getColoredDataSlicePos (ColoredDataHandle Data, ColoredDataHandle Slice, Direction SliceNormalDirection, double Pos_mm)

  Returns a slice of colored data perpendicular to the specified direction at the specified position.
- SPECTRALRADAR_API void getDataSliceAtIndex (DataHandle Data, DataHandle Slice, Direction Slice↩NormalDirection, int Index)

Returns a slice of data perpendicular to the specified direction at the specified index.

- SPECTRALRADAR_API void getComplexDataSliceIndex (ComplexDataHandle Data, ComplexDataHandle Slice, Direction SliceNormalDirection, int Index)

    Returns a slice of complex data perpendicular to the specified direction at the specified index.
- SPECTRALRADAR_API void getColoredDataSliceIndex (ColoredDataHandle Data, ColoredDataHandle Slice, Direction SliceNormalDirection, int Index)

    Returns a slice of colored data perpendicular to the specified direction at the specified index.
- SPECTRALRADAR_API void computeDataProjection (DataHandle Data, DataHandle Slice, Direction ProjectionDirection, DataAnalyzation Selection)

    Returns a single slice of data, in which each pixel value is the feature extracted through an analysis along the specified direction.
- SPECTRALRADAR_API void appendData (DataHandle Data, DataHandle DataToAppend, Direction Dir)

    Appends the new data to the provided data, perpendicular to the specified direction.
- SPECTRALRADAR_API void appendComplexData (ComplexDataHandle Data, ComplexDataHandle Data←ToAppend, Direction Dir)

    Appends the new data to the provided data, perpendicular to the specified direction.
- SPECTRALRADAR_API void appendColoredData (ColoredDataHandle Data, ColoredDataHandle DataTo←Append, Direction Dir)

    Appends the new data to the provided data, perpendicular to the specified direction.
- SPECTRALRADAR_API void cropData (DataHandle Data, Direction Dir, int IndexMax, int IndexMin)

    Crops the data along the desired direction at the given indices. Upon return the data will only contain those slices whose indices where in the interval [IndexMin, IndexMax), counted along the cropping direction.
- SPECTRALRADAR_API void cropComplexData (ComplexDataHandle Data, Direction Dir, int IndexMax, int IndexMin)

    Crops the complex data along the desired direction at the given indices. Upon return the data will only contain those slices whose indices where in the interval [IndexMin, IndexMax), counted along the cropping direction.
- SPECTRALRADAR_API void cropColoredData (ColoredDataHandle Data, Direction Dir, int IndexMax, int IndexMin)

    Crops the colored data along the desired direction at the given indices. Upon return the data will only contain those slices whose indices where in the interval [IndexMin, IndexMax), counted along the cropping direction.
- SPECTRALRADAR_API void separateData (DataHandle Data1, DataHandle Data2, int SeparationIndex, Direction Dir)

    Separates the data at the given index at specific separation direction. The first part of the separated data will remain in Data1, the second separated in Data2.
- SPECTRALRADAR_API void separateComplexData (ComplexDataHandle Data1, ComplexDataHandle Data2, int SeparationIndex, Direction Dir)

    Separates the data at the given index at specific separation direction. The first part of the separated data will remain in Data1, the second separated in Data2.
- SPECTRALRADAR_API void separateColoredData (ColoredDataHandle Data1, ColoredDataHandle Data2, int SeparationIndex, Direction Dir)

    Separates the data at the given index at specific separation direction. The first part of the separated data will remain in Data1, the second separated in Data2.
- SPECTRALRADAR_API void flipData (DataHandle Data, Direction FlippingDir)

    Mirrors the data across a plane perpendicular to the given direction.
- SPECTRALRADAR_API void flipComplexData (ComplexDataHandle Data, Direction FlippingDir)

    Mirrors the data across a plane perpendicular to the given direction.
- SPECTRALRADAR_API void flipColoredData (ColoredDataHandle Data, Direction FlippingDir)

    Mirrors the data across a plane perpendicular to the given direction.
- SPECTRALRADAR_API ImageFieldHandle createImageField (void)

    Creates an object holding image field data.
- SPECTRALRADAR_API ImageFieldHandle createImageFieldFromProbe (ProbeHandle Probe)

    Creates an object holding image field data from the specified Probe Handle.
- SPECTRALRADAR_API void clearImageField (ImageFieldHandle ImageField)

Frees an object holding image field data.

- SPECTRALRADAR_API void saveImageField (ImageFieldHandle ImageField, const char ∗Path)

  Saves data containing image field data.
- SPECTRALRADAR_API void loadImageField (ImageFieldHandle ImageField, const char ∗Path)

  Loads data containing image field data.
- SPECTRALRADAR_API void determineImageField (ImageFieldHandle ImageField, ScanPatternHandle Pattern, DataHandle Surface)

  Determines the image field correction for the given surface data, previously measured with the given scan pattern.
- SPECTRALRADAR_API void determineImageFieldWithMask (ImageFieldHandle ImageField, ScanPatternHandle Pattern, DataHandle Surface, DataHandle Mask)

  Determines the image field correction for the given surface data, previously measured with the given scan pattern. The positive entries of the mask determine the points that actually enter in the computation.
- SPECTRALRADAR_API void correctImageField (ImageFieldHandle ImageField, ScanPatternHandle Pattern, DataHandle Data)

  Applies the image field correction to the given B-Scan or volume data.
- SPECTRALRADAR_API void correctImageFieldComplex (ImageFieldHandle ImageField, ScanPatternHandle Pattern, ComplexDataHandle Data)

  Applies the image field correction to the complex B-Scan or volume complex data.
- SPECTRALRADAR_API void correctSurface (ImageFieldHandle ImageField, ScanPatternHandle Pattern, DataHandle Surface)

  Applies the image field correction to the given Surface. Surface must contain depth values as a function of x/y coordinates.
- SPECTRALRADAR_API void setImageFieldInProbe (ImageFieldHandle ImageField, ProbeHandle Probe)

  Sets the specified image field to the specified Probe handle. Notice that no probe file will be automatically saved.
- SPECTRALRADAR_API double analyzeComplexAScan (ComplexDataHandle AScanIn, AScanAnalyzation Selection)

  Analyzes the given complex A-scan data, extracts the selected feature, and returns the computed value.

5.13.1 Detailed Description

Functionality to store and access volume data.

5.13.2 Enumeration Type Documentation

5.13.2.1 enum Direction

Specifies a direction. In the default orientation, the first orientation is the Z-axis (parallel to the illumination-ray during the measurement), the second is the X-axis, and the third is the Y-axis.

Enumerator

Direction_1 The 1-axis direction.

Direction_2 The 2-axis direction.

Direction_3 The 3-axis direction.

5.13.2.2 enum Plane2D

Planes for slices of the volume data.

Enumerator

Plane2D_12 The 12 (XZ) plane, orthogonal to the 3 (Y) axis.

Plane2D_23 The 23 (XY) plane, orthogonal to the 3 (Z) axis.

Plane2D_13 The 13 (ZY) plane, orthogonal to the 2 (X) axis.

5.13.3 Function Documentation

5.13.3.1 double analyzeAScan (DataHandle *Data,* AScanAnalyzation *Selection*)

Analyzes the given A-scan data, extracts the selected feature, and returns the computed value.

Parameters

in	*Data*	A valid (non null) data handle of the A-scan (DataHandle).
in	*Selection*	The desired feature that should be computed (AScanAnalyzation).

Returns

The computed feature.

If the given data is multi-dimensional, only the first A-scan will be analyzed.

5.13.3.2 double analyzeComplexAScan (ComplexDataHandle *AScanIn,* AScanAnalyzation *Selection*)

Analyzes the given complex A-scan data, extracts the selected feature, and returns the computed value.

Parameters

in	*AScanIn*	A valid (non null) complex data handle of the A-scan (ComplexDataHandle).
in	*Selection*	The desired feature that should be computed (AScanAnalyzation).

Returns

The computed feature.

If the given data is multi-dimensional, only the first A-scan will be analyzed.

5.13.3.3 double analyzeData (DataHandle *Data,* DataAnalyzation *Selection*)

Analyzes the given data, extracts the selected feature, and returns the computed value.

Parameters

in	*Data*	A valid (non null) data handle of the data (DataHandle). These data may be multi-dimensional.
in	*Selection*	The desired feature that should be computed (DataAnalyzation).

Returns

The value of the desired feature.

5.13.3.4 void appendColoredData (ColoredDataHandle *Data,* ColoredDataHandle *DataToAppend,* Direction *Dir*)

Appends the new data to the provided data, perpendicular to the specified direction.

Parameters

in,out	*Data*	A valid (non null) colored data handle of the existing data (ColoredDataHandle), that will be expanded.
in	*DataToAppend*	A valid (non null) colored data handle of the new data (ColoredDataHandle). These data will not be modified.
in	*Dir*	The physical direction (Direction) along which the existing data will be expanded in order to accomodate the new data. Currently the Direction_1 (usually the Z-axis) is not supported and should not be specified.

Appending data implies expanding the number of data and also their physical range. These expansions are carried out automatically before the function returns.

5.13.3.5 void appendComplexData (ComplexDataHandle *Data,* ComplexDataHandle *DataToAppend,* Direction *Dir*)

Appends the new data to the provided data, perpendicular to the specified direction.

Parameters

in,out	*Data*	A valid (non null) complex data handle of the existing data (ComplexDataHandle), that will be expanded.
in	*DataToAppend*	A valid (non null) complex data handle of the new data (ComplexDataHandle). These data will not be modified.
in	*Dir*	The physical direction (Direction) along which the existing data will be expanded in order to accomodate the new data. Currently the Direction_1 (usually the Z-axis) is not supported and should not be specified.

Appending data implies expanding the number of data and also their physical range. These expansions are carried out automatically before the function returns.

5.13.3.6 void appendData (DataHandle *Data,* DataHandle *DataToAppend,* Direction *Dir*)

Appends the new data to the provided data, perpendicular to the specified direction.

Parameters

in,out	*Data*	A valid (non null) data handle of the existing data (DataHandle), that will be expanded.
in	*DataToAppend*	A valid (non null) data handle of the new data (DataHandle). These data will not be modified.
in	*Dir*	The physical direction (Direction) along which the existing data will be expanded in order to accomodate the new data. Currently the Direction_1 (usually the Z-axis) is not supported and should not be specified.

Appending data implies expanding the number of data and also their physical range. These expansions are carried out automatically before the function returns.

5.13.3.7 void appendRawData (RawDataHandle *Raw,* RawDataHandle *DataToAppend,* Direction *Dir*)

Appends the new raw data to the old raw data perpendicular to the specified direction.

Parameters

in,out	*Raw*	A valid (non null) raw-data handle of the existing data (RawDataHandle), that will be expanded.
in	*DataToAppend*	A valid (non null) raw-data handle of the new data (RawDataHandle). These raw-data will not be modified.
in	*Dir*	The physical direction (Direction) in which the new data will be appended. Currently the Direction_1 (usually the Z-axis) is not supported and should not be specified.

Appending data implies expanding the number of data and also their physical range. These expansions are carried out automatically before the function returns.
Notice that raw data refers to the spectra as acquired, without processing of any kind.

5.13.3.8 void clearImageField (ImageFieldHandle *ImageField*)

Frees an object holding image field data.

Parameters

in	*ImageField*	A handle of the image field (ImageFieldHandle). If the handle is a nullptr, this function does nothing.

5.13.3.9 void computeDataProjection (DataHandle *Data,* DataHandle *Slice,* Direction *ProjectionDirection,* DataAnalyzation *Selection*)

Returns a single slice of data, in which each pixel value is the feature extracted through an analysis along the specified direction.

Parameters

in	*Data*	A valid (non null) data handle of the existing, three-dimensional data (DataHandle). These data will not be modified.
out	*Slice*	A valid (non null) data handle (DataHandle), where the data of the slice will be written. Geometrically, this slice is situated perpendicular to the specified Direction.
in	*ProjectionDirection*	The physical direction (Direction) along which the provided data will be analyzed.
in	*Selection*	The desired feature that should be extracted from the data along the specified direction.

5.13.3.10 void correctImageField (ImageFieldHandle *ImageField,* ScanPatternHandle *Pattern,* DataHandle *Data*)

Applies the image field correction to the given B-Scan or volume data.

Parameters

in	*ImageField*	A valid (non null) handle of the image field (ImageFieldHandle), previously created with one of the functions createImageField or createImageFieldFromProbe. Besides, the image field has already been determined with the help of the function determineImageField or determineImageFieldWithMask.

Parameters

`in`	*Pattern*	A valid (non null) handle of a scan pattern (ScanPatternHandle). This scan pattern should be the one used to acquire the `Data` (third parameter), because the correction depends on the measurement coordinates. The scan pattern enables the conversion between index coordinates (i,j) and physical coordinates (in millimeter). Hence it should be a scan pattern that covers the coordinates of the `Data` (third parameter).
`in,out`	*Data*	A valid (non null) handle of data (DataHandle) pointing to data measured (acquired and processed) in a B-scan or in a volume scan.

5.13.3.11 void correctImageFieldComplex (**ImageFieldHandle** *ImageField,* **ScanPatternHandle** *Pattern,* **ComplexDataHandle** *Data*)

Applies the image field correction to the complex B-Scan or volume complex data.

Parameters

`in`	*ImageField*	A valid (non null) handle of the image field (ImageFieldHandle), previously created with one of the functions createImageField or createImageFieldFromProbe. Besides, the image field has already been determined with the help of the function determineImageField or determineImageFieldWithMask.
`in`	*Pattern*	A valid (non null) handle of a scan pattern (ScanPatternHandle). This scan pattern should be the one used to acquire the `Data` (third parameter), because the correction depends on the measurement coordinates. The scan pattern enables the conversion between index coordinates (i,j) and physical coordinates (in millimeter). Hence it should be a scan pattern that covers the coordinates of the `Data` (third parameter).
`in,out`	*Data*	A valid (non null) handle of complex data (ComplexDataHandle) pointing to data measured (acquired and processed) in a B-scan or in a volume scan.

5.13.3.12 void correctSurface (**ImageFieldHandle** *ImageField,* **ScanPatternHandle** *Pattern,* **DataHandle** *Surface*)

Applies the image field correction to the given Surface. Surface must contain depth values as a function of x/y coordinates.

Parameters

`in`	*ImageField*	A valid (non null) handle of the image field (ImageFieldHandle), previously created with one of the functions createImageField or createImageFieldFromProbe. Besides, the image field has already been determined with the help of the function determineImageField or determineImageFieldWithMask.
`in`	*Pattern*	A valid (non null) handle of a scan pattern (ScanPatternHandle). The scan pattern enables the conversion between index coordinates (i,j) and physical coordinates (in millimeter). Hence it should be a scan pattern that covers the coordinates of the `Surface` (third parameter).
`in,out`	*Surface*	A 2D data array, in a DataHandle structure, whose entries are the depth of the surface at each (x,y) coordinate, expressed in millimeter. This surface will be corrected. Notice that, unlike scans, the first coordinate is the x-axis and the second coordinate is the y-axis.

5.13.3.13 ImageFieldHandle createImageField (void)

Creates an object holding image field data.

Returns

A valid handle of the newly created image field (ImageFieldHandle).

5.13.3.14 ImageFieldHandle createImageFieldFromProbe (ProbeHandle *Probe*)

Creates an object holding image field data from the specified Probe Handle.

Parameters

in	*Probe*	A valid (non null) probe handle (ProbeHandle), previously generated with the function initProbe.

Returns

A valid handle of the newly created image field (ImageFieldHandle).

5.13.3.15 void cropColoredData (ColoredDataHandle *Data,* Direction *Dir,* int *IndexMax,* int *IndexMin*)

Crops the colored data along the desired direction at the given indices. Upon return the data will only contain those slices whose indices where in the interval [IndexMin, IndexMax), counted along the cropping direction.

Parameters

in,out	*Data*	A valid (non null) colored data handle of the data (ColoredDataHandle). These data will be cropped.
in	*Dir*	The physical direction (Direction) along which the existing data will be cropped.
in	*IndexMax*	One-past-the-last slice that will be kept. This index is zero-based.
in	*IndexMin*	The first slice that will be kept. This index is zero-based.

5.13.3.16 void cropComplexData (ComplexDataHandle *Data,* Direction *Dir,* int *IndexMax,* int *IndexMin*)

Crops the complex data along the desired direction at the given indices. Upon return the data will only contain those slices whose indices where in the interval [IndexMin, IndexMax), counted along the cropping direction.

Parameters

in,out	*Data*	A valid (non null) complex data handle of the data (ComplexDataHandle). These data will be cropped.
in	*Dir*	The physical direction (Direction) along which the existing data will be cropped.
in	*IndexMax*	One-past-the-last slice that will be kept. This index is zero-based.
in	*IndexMin*	The first slice that will be kept. This index is zero-based.

5.13.3.17 void cropData (DataHandle *Data,* Direction *Dir,* int *IndexMax,* int *IndexMin*)

Crops the data along the desired direction at the given indices. Upon return the data will only contain those slices whose indices where in the interval [IndexMin, IndexMax), counted along the cropping direction.

Parameters

`in,out`	*Data*	A valid (non null) data handle of the data (DataHandle). These data will be cropped.
`in`	*Dir*	The physical direction (Direction) along which the existing data will be cropped.
`in`	*IndexMax*	One-past-the-last slice that will be kept. This index is zero-based.
`in`	*IndexMin*	The first slice that will be kept. This index is zero-based.

5.13.3.18 void determineImageField (ImageFieldHandle *ImageField,* ScanPatternHandle *Pattern,* DataHandle *Surface*)

Determines the image field correction for the given surface data, previously measured with the given scan pattern.

Parameters

`out`	*ImageField*	A valid (non null) handle of the image field (ImageFieldHandle), previously created with one of the functions createImageField or createImageFieldFromProbe.
`in`	*Pattern*	A valid (non null) handle of a volume scan pattern (ScanPatternHandle), created with one of the functions createVolumePattern, createFreeformScanPattern3DFromLUT, or createFreeformScanPattern3D. The scan pattern should uniformly cover the whole field of view.

Warning

> If the scan pattern is non uniform, or fails to cover some areas, the resulting image field corrections will be impaired. The scan pattern enables the conversion between index coordinates (i,j) and physical coordinates (in millimeter). Hence it should be a scan pattern that covers the coordinates of the `Surface` (third parameter).

Parameters

`in`	*Surface*	A 2D data array, in a DataHandle structure, whose entries are the depth of the surface at each (x,y) coordinate, expressed in millimeter. The surface can be calculated from a volume scan using the function determineSurface. Notice that, unlike scans, the first coordinate is the x-axis and the second coordinate is the y-axis.

The purpose of the image field is to compensate the deformations introduced by the optical elements (e.g. lenses). To that end, a measurement of the substrate surface is carried out, and the geometric correction is computed. The default calibration of an instrument needs not be re-computed, unless a new objective is installed, or the objective is the same but the desired reference surface is non planar (the user must supply the desired surface, which should actually be measured).

5.13.3.19 void determineImageFieldWithMask (ImageFieldHandle *ImageField,* ScanPatternHandle *Pattern,* DataHandle *Surface,* DataHandle *Mask*)

Determines the image field correction for the given surface data, previously measured with the given scan pattern. The positive entries of the mask determine the points that actually enter in the computation.

Parameters

`out`	*ImageField*	A valid (non null) handle of the image field (ImageFieldHandle), previously created with one of the functions createImageField or createImageFieldFromProbe.
`in`	*Pattern*	A valid (non null) handle of a volume scan pattern (ScanPatternHandle), created with one of the functions createVolumePattern, createFreeformScanPattern3DFromLUT, or createFreeformScanPattern3D. The scan pattern should uniformly cover the whole field of view.

Warning

> If the scan pattern is non uniform, or fails to cover some areas, the resulting image field corrections will be impaired. The scan pattern enables the conversion between index coordinates (i,j) and physical coordinates (in millimeter). Hence it should be a scan pattern that covers the coordinates of the `Surface` (third parameter).

Parameters

`in`	*Surface*	A 2D data array, in a DataHandle structure, whose entries are the depth of the surface at each (x,y) coordinate, expressed in millimeter. The surface can be calculated from a volume scan using the function determineSurface. Notice that, unlike scans, the first coordinate is the x-axis and the second coordinate is the y-axis.
`in`	*Mask*	A 2D array, in stored a DataHandle structure, indicating which points of the Surface should be taken into account (positive entries in `Mask`). Negative entries in `Mask` identify points of the `Surface` which should not be considered in the computation. Notice that the entries are single-precision floating-point numbers. In case a 3D data structure is passed, only the first slice will be used (index zero along the third Direction).

The purpose of the image field is to compensate the deformations introduced by the optical elements (e.g. lenses). To that end, a measurement of the substrate surface is carried out, and the geometric correction is computed. The default calibration of an instrument needs not be re-computed, unless a new objective is installed, or the objective is the same but the desired reference surface is non planar (the user must supply the desired surface, which should actually be measured).
This function checks that the first two dimensions (in pixels) of `Surface` and `Mask` match each other.

5.13.3.20 void flipColoredData (**ColoredDataHandle** *Data,* **Direction** *FlippingDir*)

Mirrors the data across a plane perpendicular to the given direction.

Parameters

`in,out`	*Data*	A valid (non null) complex data handle of the data (ComplexDataHandle). These data will be flipped.
`in`	*FlippingDir*	The physical direction (Direction) along which the existing data will be flipped.

5.13.3.21 void flipComplexData (**ComplexDataHandle** *Data,* **Direction** *FlippingDir*)

Mirrors the data across a plane perpendicular to the given direction.

Parameters

`in,out`	*Data*	A valid (non null) complex data handle of the data (ComplexDataHandle). These data will be flipped.
`in`	*FlippingDir*	The physical direction (Direction) along which the existing data will be flipped.

5.13.3.22 void flipData (DataHandle *Data*, Direction *FlippingDir*)

Mirrors the data across a plane perpendicular to the given direction.

Parameters

in,out	*Data*	A valid (non null) data handle of the data (DataHandle). These data will be flipped.
in	*FlippingDir*	The physical direction (Direction) along which the existing data will be flipped.

5.13.3.23 void getColoredDataSliceIndex (ColoredDataHandle *Data*, ColoredDataHandle *Slice*, Direction *SliceNormalDirection*, int *Index*)

Returns a slice of colored data perpendicular to the specified direction at the specified index.

Parameters

in	*Data*	A valid (non null) colored data handle of the existing, three-dimensional data (ColoredDataHandle). These data will not be modified.
out	*Slice*	A valid (non null) complex data handle (ColoredDataHandle), where the data of the slice will be written.
in	*SliceNormalDirection*	The physical direction (Direction) in which the existing data will be sliced. Currently only the Direction_3 (usually the Y-axis) is supported.
in	*Index*	The index of the desired slice along the direction `Dir` (zero-based, that is, the first slice is 0). The total number of slices can be obtained with the function getDataPropertyInt, and specifying the property Data_Size2 or Data_Size3 (depending on the `Dir` specified).

The colored data that will be sliced (`data`) should be three-dimensional, e.g., a sequence of B-scans. A slice is one of the B-scans, perpendicular to the specified direction `Dir`.

5.13.3.24 void getColoredDataSlicePos (ColoredDataHandle *Data*, ColoredDataHandle *Slice*, Direction *SliceNormalDirection*, double *Pos_mm*)

Returns a slice of colored data perpendicular to the specified direction at the specified position.

Parameters

in	*Data*	A valid (non null) colored data handle of the existing, three-dimensional data (ColoredDataHandle). These data will not be modified.
out	*Slice*	A valid (non null) complex data handle (ComplexDataHandle), where the data of the slice will be written.
in	*SliceNormalDirection*	The physical direction (Direction) in which the existing data will be sliced. Currently only the Direction_3 (usually the Y-axis) is supported.
in	*Pos_mm*	The position of the desired slice along the direction `Dir`, expressed in millimeter. The total range of positions can be inquired with the function getDataPropertyFloat, specifying the property Data_Range2 or Data_Range3 (depending on the `Dir` specified). If the scan pattern has not been manipulated (e.g. shifted), the center position is 0 mm.

The colored data that will be sliced (`Data`) should be three-dimensional, e.g., a sequence of B-scans. A slice is one of the B-scans, perpendicular to the specified direction `Dir`. If a position intermediate between two measured

B-scans is given, this function will pick the closest; no interpolation will take place.

5.13.3.25 void getComplexDataSliceIndex (ComplexDataHandle *Data,* ComplexDataHandle *Slice,* Direction *SliceNormalDirection,* int *Index*)

Returns a slice of complex data perpendicular to the specified direction at the specified index.

Parameters

`in`	*Data*	A valid (non null) complex data handle of the existing, three-dimensional data (ComplexDataHandle). These data will not be modified.
`out`	*Slice*	A valid (non null) complex data handle (ComplexDataHandle), where the data of the slice will be written.
`in`	*SliceNormalDirection*	The physical direction (Direction) in which the existing data will be sliced. Currently only the Direction_3 (usually the Y-axis) is supported.
`in`	*Index*	The index of the desired slice along the direction `Dir` (zero-based, that is, the first slice is 0). The total number of slices can be obtained with the function getDataPropertyInt, and specifying the property Data_Size2 or Data_Size3 (depending on the `Dir` specified).

The complex data that will be sliced (`data`) should be three-dimensional, e.g., a sequence of B-scans. A slice is one of the B-scans, perpendicular to the specified direction `Dir`.

5.13.3.26 void getComplexDataSlicePos (ComplexDataHandle *Data,* ComplexDataHandle *Slice,* Direction *SliceNormalDirection,* double *Pos_mm*)

Returns a slice of complex data perpendicular to the specified direction at the specified position.

Parameters

`in`	*Data*	A valid (non null) complex data handle of the existing, three-dimensional data (ComplexDataHandle). These data will not be modified.
`out`	*Slice*	A valid (non null) complex data handle (ComplexDataHandle), where the data of the slice will be written.
`in`	*SliceNormalDirection*	The physical direction (Direction) in which the existing data will be sliced. Currently only the Direction_3 (usually the Y-axis) is supported.
`in`	*Pos_mm*	The position of the desired slice along the direction `Dir`, expressed in millimeter. The total range of positions can be inquired with the function getDataPropertyFloat, specifying the property Data_Range2 or Data_Range3 (depending on the `Dir` specified). If the scan pattern has not been manipulated (e.g. shifted), the center position is 0 mm.

The complex data that will be sliced (`Data`) should be three-dimensional, e.g., a sequence of B-scans. A slice is one of the B-scans, perpendicular to the specified direction `Dir`. If a position intermediate between two measured B-scans is given, this function will pick the closest; no interpolation will take place.

5.13.3.27 void getDataSliceAtIndex (DataHandle *Data,* DataHandle *Slice,* Direction *SliceNormalDirection,* int *Index*)

Returns a slice of data perpendicular to the specified direction at the specified index.

Parameters

in	*Data*	A valid (non null) data handle of the existing, three-dimensional data (DataHandle). These data will not be modified.
out	*Slice*	A valid (non null) data handle (DataHandle), where the data of the slice will be written.
in	*SliceNormalDirection*	The physical direction (Direction) in which the existing data will be sliced. Currently only the Direction_3 (usually the Y-axis) is supported.
in	*Index*	The index of the desired slice along the direction `Dir` (zero-based, that is, the first slice is 0). The total number of slices can be obtained with the function getDataPropertyInt, and specifying the property Data_Size2 or Data_Size3 (depending on the `Dir` specified).

The data that will be sliced (`data`) should be three-dimensional, e.g., a sequence of B-scans. A slice is one of the B-scans, perpendicular to the specified direction `Dir`.

5.13.3.28 void getDataSliceAtPos (DataHandle *Data,* DataHandle *Slice,* Direction *SliceNormalDirection,* double *Pos_mm*)

Returns a slice of data perpendicular to the specified direction at the specified position.

Parameters

in	*Data*	A valid (non null) data handle of the existing, three-dimensional data (DataHandle). These data will not be modified.
out	*Slice*	A valid (non null) data handle (DataHandle), where the data of the slice will be written.
in	*SliceNormalDirection*	The physical direction (Direction) in which the existing data will be sliced. Currently only the Direction_3 (usually the Y-axis) is supported.
in	*Pos_mm*	The position of the desired slice along the direction `Dir`, expressed in millimeter. The total range of positions can be inquired with the function getDataPropertyFloat, specifying the property Data_Range2 or Data_Range3 (depending on the `Dir` specified). If the scan pattern has not been manipulated (e.g. shifted), the center position is 0 mm.

The data that will be sliced (`Data`) should be three-dimensional, e.g., a sequence of B-scans. A slice is one of the B-scans, perpendicular to the specified direction `Dir`. If a position intermediate between two measured B-scans is given, this function will pick the closest; no interpolation will take place.

5.13.3.29 void getRawDataSliceAtIndex (RawDataHandle *Raw,* RawDataHandle *Slice,* Direction *SliceNormalDirection,* int *Index*)

Returns a slice of raw data perpendicular to the specified direction at the specified index.

Parameters

in	*Raw*	A valid (non null) raw-data handle of the existing, three-dimensional raw data (RawDataHandle). These data will not be modified.
out	*Slice*	A valid (non null) raw-data handle (RawDataHandle), where the raw data of the slice will be written.
in	*SliceNormalDirection*	The physical direction (Direction) in which the existing data will be sliced. Currently only the Direction_3 (usually the Y-axis) is supported.
in	*Index*	The desired slice number in the direction `Dir`.

The raw data that will be sliced (`Raw`) should be three-dimensional, that is, a sequence of B-scans. A slice is one of the B-scans, perpendicular to the third direction (usually the Y-axis).
Notice that raw data refers to the spectra as acquired, without processing of any kind.

5.13.3.30 void loadImageField (**ImageFieldHandle** *ImageField,* const char ∗ *Path*)

Loads data containing image field data.

Parameters

`out`	*ImageField*	A valid (non null) handle of the image field (ImageFieldHandle), previously created with one of the functions createImageField or createImageFieldFromProbe.
`in`	*Path*	Filename (including path), where the data will be read from.

5.13.3.31 void normalizeData (**DataHandle** *Data,* float *Min,* float *Max*)

Scales the given data in such a way, that the range [Min, Max] is mapped onto the range [0,1].

Parameters

`in,out`	*Data*	A valid (non null) data handle of the data (DataHandle). These data will be scaled.
`in`	*Min*	The lower bound of the data that will be mapped to 0 in `DataOut`.
`in`	*Max*	The upper bound of the data that will be mapped to 1 in `DataOut`.

5.13.3.32 void saveImageField (**ImageFieldHandle** *ImageField,* const char ∗ *Path*)

Saves data containing image field data.

Parameters

`in`	*ImageField*	A valid (non null) handle of the image field (ImageFieldHandle), previously created with one of the functions createImageField or createImageFieldFromProbe.
`in`	*Path*	Filename (including path), where the data will be saved. If the file exists, it will be (merciless) overwritten.

5.13.3.33 void separateColoredData (**ColoredDataHandle** *Data1,* **ColoredDataHandle** *Data2,* int *SeparationIndex,* **Direction** *Dir*)

Separates the data at the given index at specific separation direction. The first part of the separated data will remain in Data1, the second separated in Data2.

Parameters

`in,out`	*Data1*	A valid (non null) colored data handle of the data (ColoredDataHandle). Upon return, only the first part will remain in this container.
`out`	*Data2*	A valid (non null) colored data handle to the second part of the data (ColoredDataHandle).
`in`	*SeparationIndex*	The first slice of the second part, or one-past-the-last slice kept in the first part.
`in`	*Dir*	The physical direction (Direction) along which the separation will take place.

5.13.3.34 void separateComplexData (ComplexDataHandle *Data1*, ComplexDataHandle *Data2*, int *SeparationIndex*, Direction *Dir*)

Separates the data at the given index at specific separation direction. The first part of the separated data will remain in Data1, the second separated in Data2.

Parameters

`in,out`	*Data1*	A valid (non null) complex data handle of the data (ComplexDataHandle). Upon return, only the first part will remain in this container.
`out`	*Data2*	A valid (non null) complex data handle to the second part of the data (ComplexDataHandle).
`in`	*SeparationIndex*	The first slice of the second part, or one-past-the-last slice kept in the first part.
`in`	*Dir*	The physical direction (Direction) along which the separation will take place.

5.13.3.35 void separateData (DataHandle *Data1*, DataHandle *Data2*, int *SeparationIndex*, Direction *Dir*)

Separates the data at the given index at specific separation direction. The first part of the separated data will remain in Data1, the second separated in Data2.

Parameters

`in,out`	*Data1*	A valid (non null) data handle of the data (DataHandle). Upon return, only the first part will remain in this container.
`out`	*Data2*	A valid (non null) data handle to the second part of the data (DataHandle).
`in`	*SeparationIndex*	The first slice of the second part, or one-past-the-last slice kept in the first part.
`in`	*Dir*	The physical direction (Direction) along which the separation will take place.

5.13.3.36 void setImageFieldInProbe (ImageFieldHandle *ImageField*, ProbeHandle *Probe*)

Sets the specified image field to the specified Probe handle. Notice that no probe file will be automatically saved.

Parameters

`in`	*ImageField*	A valid (non null) handle of the image field (ImageFieldHandle), previously created with one of the functions createImageField or createImageFieldFromProbe. Besides, the image field has already been determined with the help of the function determineImageField or determineImageFieldWithMask.
`in`	*Probe*	A valid (non null) probe handle (ProbeHandle), previously generated with the function initProbe.

5.13.3.37 void transposeAndScaleData (DataHandle *DataIn*, DataHandle *DataOut*, float *Min*, float *Max*)

Transposes the given data and writes the result to DataOut. First and second axes will be swaped, and the range of the entries will be scaled in such a way, that the range [Min,Max] will be mapped onto the range [0,1].

Parameters

`in`	*DataIn*	A valid (non null) data handle of the input data (DataHandle). These data should be multi-dimensional. These data will not be modified.

Parameters

`out`	*DataOut*	A valid (non null) data handle of the output data (DataHandle). These data will be a scaled and transposed copy of the input data, that is, the first and the second axes will be swaped (usually Z- and X-axes).
`in`	*Min*	The lower bound of the data that will be mapped to 0 in `DataOut`.
`in`	*Max*	The upper bound of the data that will be mapped to 1 in `DataOut`.

5.13.3.38 void transposeData (DataHandle *DataIn,* DataHandle *DataOut*)

Transposes the given data and writes the result to DataOut. First and second axes will be swaped.

Parameters

`in`	*DataIn*	A valid (non null) data handle of the input data (DataHandle). These data should be multi-dimensional. These data will not be modified.
`out`	*DataOut*	A valid (non null) data handle of the output data (DataHandle). These data will be a copy of the input data, except that the first and the second axes will be swaped (usually Z- and X-axes).

5.13.3.39 void transposeDataInplace (DataHandle *Data*)

Transposes the given Data. First and second axes will be swaped.

Parameters

`in,out`	*Data*	A valid (non null) data handle of the data (DataHandle). These data will be modified: the first and the second axes will be swaped (usually Z- and X-axes).

5.14 ProbeCalibration

Functionality to perform the probe calibration. Please use the ThorImageOCT software to perform probe calibrations, if necessary.

Functionality to perform the probe calibration. Please use the ThorImageOCT software to perform probe calibrations, if necessary.

Warning

ThorImageOCT uses these functions to calibrate the galvo, assuming a very specific sequence of actions and conditions, as explained in the ThorImageOCT. For these functions to properly work, the user need to re-create the same sequence of actions and conditions.
The galvo offset/factor / Draw & Scan overlay calibration assumes a sample with a triangular dot pattern with a fixed edge length which must be aligned parallel to the video image egdes.

5.15 Doppler

Doppler Processing Routines.

Typedefs

- typedef struct C_DopplerProcessing ∗ DopplerProcessingHandle

  Handle used for Doppler processing.

Enumerations

- enum DopplerPropertyInt {
  Doppler_Averaging_1,
  Doppler_Averaging_2,
  Doppler_Stride_1,
  Doppler_Stride_2 }

  Values that determine the behaviour of the Doppler processing routines.
- enum DopplerPropertyFloat {
  Doppler_RefractiveIndex,
  Doppler_ScanRate_Hz,
  Doppler_CenterWavelength_nm,
  Doppler_DopplerAngle_Deg }

  Values that determine the behaviour of the Doppler processing routines.
- enum DopplerFlag { Doppler_VelocityScaling }

  Flats that determine the behaviour of the Doppler processing routines.

Functions

- SPECTRALRADAR_API DopplerProcessingHandle createDopplerProcessing (void)

  Returns a handle for the use of Doppler-computation routines.
- SPECTRALRADAR_API DopplerProcessingHandle createDopplerProcessingForFile (OCTFileHandle File)

  Returns a handle for the use of Doppler-computation routines. The handle is created based on a saved OCT file.
- SPECTRALRADAR_API int getDopplerPropertyInt (DopplerProcessingHandle Handle, DopplerPropertyInt Property)

  Gets the value of the given Doppler processing property.
- SPECTRALRADAR_API void setDopplerPropertyInt (DopplerProcessingHandle Handle, DopplerPropertyInt Property, int Value)

  Sets the value of the given Doppler processing property.
- SPECTRALRADAR_API double getDopplerPropertyFloat (DopplerProcessingHandle Doppler, DopplerPropertyFloat Property)

  Gets the value of the given Doppler processing property.
- SPECTRALRADAR_API void setDopplerPropertyFloat (DopplerProcessingHandle Handle, DopplerPropertyFloat Property, float Value)

  Sets the value of the given Doppler processing property.
- SPECTRALRADAR_API BOOL getDopplerFlag (DopplerProcessingHandle Handle, DopplerFlag Flag)

  Gets the given Doppler processing flag.
- SPECTRALRADAR_API void setDopplerFlag (DopplerProcessingHandle Handle, DopplerFlag Flag, BOOL OnOff)

  Sets the given Doppler processing flag.
- SPECTRALRADAR_API void setDopplerAmplitudeOutput (DopplerProcessingHandle Handle, DataHandle AmpOut)

Sets the location of the resulting Doppler amplitude output.

- SPECTRALRADAR_API void setDopplerPhaseOutput (DopplerProcessingHandle Handle, DataHandle PhasesOut)

  Sets the location of the resulting Doppler phase output.
- SPECTRALRADAR_API void executeDopplerProcessing (DopplerProcessingHandle Handle, ComplexDataHandle Input)

  Executes the Doppler processing of the input data and returns phases and amplitudes.
- SPECTRALRADAR_API void dopplerVelocityToPhase (DopplerProcessingHandle Doppler, DataHandle InOut)

  Scales flow velocities computed by Doppler OCT back to original phase differencees.
- SPECTRALRADAR_API void clearDopplerProcessing (DopplerProcessingHandle Handle)

  Closes the Doppler processing routines and frees the memory that has been allocated for these to work properly.
- SPECTRALRADAR_API void getDopplerOutputSize (DopplerProcessingHandle Handle, int Size1In, int Size2In, int ∗Size1Out, int ∗Size2Out)

  Returns the final size of the Doppler output if executeDopplerProcessing is executed using data of the specified input size.

5.15.1 Detailed Description

Doppler Processing Routines.

5.15.2 Typedef Documentation

5.15.2.1 DopplerProcessingHandle

Handle used for Doppler processing.

5.15.3 Enumeration Type Documentation

5.15.3.1 enum DopplerFlag

Flats that determine the behaviour of the Doppler processing routines.

Enumerator

Doppler_VelocityScaling Averaging along the first axis, usually the longitudinal axis (z)

5.15.3.2 enum DopplerPropertyFloat

Values that determine the behaviour of the Doppler processing routines.

Enumerator

Doppler_RefractiveIndex Averaging along the first axis, usually the longitudinal axis (z)

Doppler_ScanRate_Hz Scan Rate (in Hz) that was used to acquire the Doppler data to be processed. This is only required for computing the actual velocity scaling.

Doppler_CenterWavelength_nm Center Wavelength (in nanometers) that was used to acquire the Doppler data to be processed. This is only required for computing the actual velocity scaling.

Doppler_DopplerAngle_Deg Angle of the Doppler detection beam to the normal. This is only required for computing the actual velocity scaling.

5.15.3.3 enum DopplerPropertyInt

Values that determine the behaviour of the Doppler processing routines.

Enumerator

Doppler_Averaging_1 Averaging along the first axis, usually the longitudinal axis (z)

Doppler_Averaging_2 Averaging along the first axis, usually the first transversal axis (x)

Doppler_Stride_1 Step size for calculating the doppler processing in the longitudinal axis (z). Stride needs to be smaller or equal to Doppler_Averaging_1 and larger or equal to 1.

Doppler_Stride_2 Step size for calculating the doppler processing in the transversal axis (x). Stride needs to be smaller or equal to Doppler_Averaging_2 and larger or equal to 1.

5.15.4 Function Documentation

5.15.4.1 void clearDopplerProcessing (DopplerProcessingHandle *Handle*)

Closes the Doppler processing routines and frees the memory that has been allocated for these to work properly.

Parameters

in	*Handle*	A handle of Doppler processing routines (DopplerProcessingHandle). If the handle is a nullptr, this function does nothing. In most cases this handle will have been previously obtained with the function createDopplerProcessing.

5.15.4.2 DopplerProcessingHandle createDopplerProcessing (void)

Returns a handle for the use of Doppler-computation routines.

Returns

DopplerProcessingHandle to the created Doppler routines.

5.15.4.3 DopplerProcessingHandle createDopplerProcessingForFile (OCTFileHandle *File*)

Returns a handle for the use of Doppler-computation routines. The handle is created based on a saved OCT file.

Returns

DopplerProcessingHandle to the created Doppler routines.

5.15.4.4 void dopplerVelocityToPhase (DopplerProcessingHandle *Handle,* DataHandle *InOut*)

Scales flow velocities computed by Doppler OCT back to original phase differencees.

Parameters

in	*Handle*	A valid (non null) handle of Doppler processing routines (DopplerProcessingHandle), obtained with the function createDopplerProcessing.
in,out	*InOut*	A handle of data representing first velocity data that will then be modified to conttain velocity data.

This requires the Doppler scan rate, Doppler angle and center velocity of the Doppler object to be set correctly.

5.15.4.5 void executeDopplerProcessing (DopplerProcessingHandle *Handle,* ComplexDataHandle *Input*)

Executes the Doppler processing of the input data and returns phases and amplitudes.

Parameters

in	*Handle*	A valid (non null) handle of Doppler processing routines (DopplerProcessingHandle), obtained with the function createDopplerProcessing.
in	*Input*	A valid (non null) handle of complex data (ComplexDataHandle). These data should have previously obtained by invoking the functions createComplexData, setComplexDataOutput and executeProcessing.

Doppler procesing takes place after the standard processing. It takes as input complex data computed by the standard processing, and during execution it writes amplitudes and phases, provided either 8or both) of the function setDopplerAmplitudeOutput or setDopplerPhaseOutput have previously been invoked.

5.15.4.6 void getDopplerFlag (DopplerProcessingHandle *Handle,* DopplerFlag *Flag*)

Gets the given Doppler processing flag.

Parameters

in	*Handle*	A valid (non null) handle of Doppler processing routines (DopplerProcessingHandle), obtained with the function createDopplerProcessing.
in	*Flag*	The desired boolean flag (DopplerFlag).

Returns

The boolen value of the selected flag.

5.15.4.7 void getDopplerOutputSize (DopplerProcessingHandle *Handle,* int *Size1In,* int *Size2In,* int ∗ *Size1Out,* int ∗ *Size2Out*)

Returns the final size of the Doppler output if executeDopplerProcessing is executed using data of the specified input size.

Parameters

in	*Handle*	A valid (non null) handle of Doppler processing routines (DopplerProcessingHandle), obtained with the function createDopplerProcessing.
in	*Size1In*	The value of the Data_Size1 property (DataPropertyInt) of the complex-data that will be used as input. In the default orientation, this is the number of pixels along the z-axis.
in	*Size2In*	The value of the Data_Size2 property (DataPropertyInt) of the complex-data that will be used as input. In the default orientation, this is the number of pixels along the x-axis.
out	*Size1Out*	The value of the Data_Size1 property (DataPropertyInt) of the amplitude/phase data that will result upon invokation of the function executeDopplerProcessing. In the default orientation, this is the number of pixels along the z-axis.
out	*Size2Out*	The value of the Data_Size2 property (DataPropertyInt) of the amplitude/phase data that will result upon invokation of the function executeDopplerProcessing. In the default orientation, this is the number of pixels along the x-axis.

5.15.4.8 double getDopplerPropertyFloat (**DopplerProcessingHandle** *Handle,* **DopplerPropertyFloat** *Property*)

Gets the value of the given Doppler processing property.

Parameters

in	*Handle*	A valid (non null) handle of Doppler processing routines (DopplerProcessingHandle), obtained with the function createDopplerProcessing.
in	*Property*	The desired floating-point property (DopplerPropertyFloat).

Returns

The value of the desired property.

5.15.4.9 int getDopplerPropertyInt (**DopplerProcessingHandle** *Handle,* **DopplerPropertyInt** *Property*)

Gets the value of the given Doppler processing property.

Parameters

in	*Handle*	A valid (non null) handle of Doppler processing routines (DopplerProcessingHandle), obtained with the function createDopplerProcessing.
in	*Property*	The desired integer property (DopplerPropertyInt).

Returns

The value of the desired property.

5.15.4.10 void setDopplerAmplitudeOutput (**DopplerProcessingHandle** *Handle,* **DataHandle** *AmpOut*)

Sets the location of the resulting Doppler amplitude output.

Parameters

in	*Handle*	A valid (non null) handle of Doppler processing routines (DopplerProcessingHandle), obtained with the function createDopplerProcessing.
in	*AmpOut*	A valid (non null) handle of data (DataHandle), where the resulting amplitudes of the Doppler computation will be written. The right number of dimensions, sizes, and ranges will be automatically adjusted by the function executeDopplerProcessing.

5.15.4.11 void setDopplerFlag (**DopplerProcessingHandle** *Handle,* **DopplerFlag** *Flag,* **BOOL** *OnOff*)

Sets the given Doppler processing flag.

Parameters

in	*Handle*	A valid (non null) handle of Doppler processing routines (DopplerProcessingHandle), obtained with the function createDopplerProcessing.
in	*Flag*	The selected boolean flag (DopplerFlag).
in	*OnOff*	The desired boolean value for the selected flag.

5.15.4.12 void setDopplerPhaseOutput (**DopplerProcessingHandle** *Handle,* **DataHandle** *PhasesOut*)

Sets the location of the resulting Doppler phase output.

Parameters

in	*Handle*	A valid (non null) handle of Doppler processing routines (DopplerProcessingHandle), obtained with the function createDopplerProcessing.
in	*PhasesOut*	A valid (non null) handle of data (DataHandle), where the resulting phases of the Doppler computation will be written. The right number of dimensions, sizes, and ranges will be automatically adjusted by the function executeDopplerProcessing.

5.15.4.13 void setDopplerPropertyFloat (**DopplerProcessingHandle** *Handle,* **DopplerPropertyFloat** *Property,* float *Value*)

Sets the value of the given Doppler processing property.

Parameters

in	*Handle*	A valid (non null) handle of Doppler processing routines (DopplerProcessingHandle), obtained with the function createDopplerProcessing.
in	*Property*	The selected floating-point property (DopplerPropertyFloat).
in	*Value*	The desired value for the selected property.

5.15.4.14 void setDopplerPropertyInt (**DopplerProcessingHandle** *Handle,* **DopplerPropertyInt** *Property,* int *Value*)

Sets the value of the given Doppler processing property.

Parameters

in	*Handle*	A valid (non null) handle of Doppler processing routines (DopplerProcessingHandle), obtained with the function createDopplerProcessing.
in	*Property*	The selected integer property (DopplerPropertyInt).
in	*Value*	The desired value for the selected property.

5.16 Service

Service functions for additional analyzing of OCT functionality.

Functions

- SPECTRALRADAR_API void calcContrast (DataHandle ApodizedSpectrum, DataHandle Contrast)
  Computes the contrast for the specified (apodized) spectrum.

5.16.1 Detailed Description

Service functions for additional analyzing of OCT functionality.

5.16.2 Function Documentation

5.16.2.1 void calcContrast (DataHandle *ApodizedSpectrum*, DataHandle *Contrast*)

Computes the contrast for the specified (apodized) spectrum.

Parameters

in	*ApodizedSpectrum*	The spectrum after offset substraction and apodization. This spectrum can be obtained using the functions setApodizedSpectrumOutput and executeProcessing in sucession.
out	*Contrast*	A valid (non null) data handle (DataHandle). Its dimensions will be automatically be adjusted.

The contrast is a measure of the amount of information in the interference pattern as a fraction of the total signal. The computed values are expressed as percentage of the measured amplitudes, for each camera pixel.

5.17 Settings

Direct access to INI files and settings.

Typedefs

- typedef struct C_Settings ∗ SettingsHandle

  Handle for saving settings on disk.

Functions

- SPECTRALRADAR_API SettingsHandle initSettingsFile (const char ∗Path)

  Loads a settings file (usually ∗.ini); and prepares its properties to be read.
- SPECTRALRADAR_API int getSettingsEntryInt (SettingsHandle SettingsFile, const char ∗Node, int Default↩ Value)

  Gets an integer number from the specified ini file (see SettingsHandle and initSettingsFile);.
- SPECTRALRADAR_API double getSettingsEntryFloat (SettingsHandle SettingsFile, const char ∗Node, double DefaultValue)

  Gets an floating point number from the specified ini file (see SettingsHandle and initSettingsFile);.
- SPECTRALRADAR_API void getSettingsEntryFloatArray (SettingsHandle SettingsFile, const char ∗Node, const double ∗DefaultValues, double ∗Values, int ∗Size)

  Gets an array of floating point numbers from the specified ini file (see SettingsHandle and initSettingsFile);.
- SPECTRALRADAR_API const char ∗ getSettingsEntryString (SettingsHandle SettingsFile, const char ∗Node, const char ∗Default)

  Gets a string from the specified ini file (see SettingsHandle and initSettingsFile);. The resulting const char∗ ptr will be valid until the settings file is closed by closeSettingsFile).
- SPECTRALRADAR_API void setSettingsEntryInt (SettingsHandle SettingsFile, const char ∗Node, int Value)

  Sets an integer entry in the specified ini file (see SettingsHandle and initSettingsFile);.
- SPECTRALRADAR_API void setSettingsEntryFloat (SettingsHandle SettingsFile, const char ∗Node, double Value)

  Sets a floating point entry in the specified ini file (see SettingsHandle and initSettingsFile);.
- SPECTRALRADAR_API void setSettingsEntryString (SettingsHandle SettingsFile, const char ∗Node, const char ∗Value)

  Sets a string in the specified ini file (see SettingsHandle and initSettingsFile);.
- SPECTRALRADAR_API void saveSettings (SettingsHandle SettingsFile)

  Saves the changes to the specified Settings file.
- SPECTRALRADAR_API void closeSettingsFile (SettingsHandle Handle)

  Closes the specified ini file and stores the set entries (.

5.17.1 Detailed Description

Direct access to INI files and settings.

5.17.2 Typedef Documentation

5.17.2.1 SettingsHandle

Handle for saving settings on disk.

5.17.3 Function Documentation

5.17.3.1 void closeSettingsFile (SettingsHandle *Handle*)

Closes the specified ini file and stores the set entries (.

See also

SettingsHandle, initSettingsFile).

Parameters

in	*Handle*	A handle of settings (SettingsHandle). If the handle is a nullptr, this function does nothing. In most cases this handle will have been previously obtained with the function initSettingsFile.

5.17.3.2 double getSettingsEntryFloat (SettingsHandle *SettingsFile,* const char ∗ *Node,* double *DefaultValue*)

Gets an floating point number from the specified ini file (see SettingsHandle and initSettingsFile);.

5.17.3.3 void getSettingsEntryFloatArray (SettingsHandle *SettingsFile,* const char ∗ *Node,* const double ∗ *DefaultValues,* double ∗ *Values,* int ∗ *Size*)

Gets an array of floating point numbers from the specified ini file (see SettingsHandle and initSettingsFile);.

5.17.3.4 int getSettingsEntryInt (SettingsHandle *SettingsFile,* const char ∗ *Node,* int *DefaultValue*)

Gets an integer number from the specified ini file (see SettingsHandle and initSettingsFile);.

5.17.3.5 const char ∗ getSettingsEntryString (SettingsHandle *SettingsFile,* const char ∗ *Node,* const char ∗ *Default*)

Gets a string from the specified ini file (see SettingsHandle and initSettingsFile);. The resulting const char∗ ptr will be valid until the settings file is closed by closeSettingsFile).

5.17.3.6 SettingsHandle initSettingsFile (const char ∗ *Path*)

Loads a settings file (usually ∗.ini); and prepares its properties to be read.

5.17.3.7 void saveSettings (SettingsHandle *SettingsFile*)

Saves the changes to the specified Settings file.

5.17.3.8 void setSettingsEntryFloat (SettingsHandle *SettingsFile,* const char ∗ *Node,* double *Value*)

Sets a floating point entry in the specified ini file (see SettingsHandle and initSettingsFile);.

5.17.3.9 void setSettingsEntryInt (SettingsHandle *SettingsFile,* const char ∗ *Node,* int *Value*)

Sets an integer entry in the specified ini file (see SettingsHandle and initSettingsFile);.

5.17.3.10 void setSettingsEntryString (SettingsHandle *SettingsFile,* const char ∗ *Node,* const char ∗ *Value*)

Sets a string in the specified ini file (see SettingsHandle and initSettingsFile);.

5.18 Coloring

Functions used for coloring of floating point data.

Typedefs

- typedef struct C_Coloring32Bit ∗ ColoringHandle

  Handle for routines that color avaible scans for displaying.

Enumerations

- enum ColorScheme {
  ColorScheme_BlackAndWhite = 0,
  ColorScheme_Inverted = 1,
  ColorScheme_Color = 2,
  ColorScheme_BlackAndOrange = 3,
  ColorScheme_BlackAndRed = 4,
  ColorScheme_BlackRedAndYellow = 5,
  ColorScheme_DopplerPhase = 6,
  ColorScheme_BlueAndBlack = 7,
  ColorScheme_PolarizationRetardation = 8,
  ColorScheme_GreenBlueAndBlack = 9,
  ColorScheme_BlackAndRedYellow = 10,
  ColorScheme_TransparentAndWhite = 11,
  ColorScheme_GreenBlueWhiteRedYellow = 12,
  ColorScheme_BlueGreenBlackYellowRed = 13,
  ColorScheme_RedGreenBlue = 14,
  ColorScheme_GreenBlueRed = 15,
  ColorScheme_BlueRedGreen = 16,
  ColorScheme_GreenBlueRedGreen = 17,
  ColorScheme_BlueRedGreenBlue = 18,
  ColorScheme_Inverse_RedGreenBlue = 19,
  ColorScheme_Inverse_GreenBlueRed = 20,
  ColorScheme_Inverse_BlueRedGreen = 21,
  ColorScheme_Inverse_GreenBlueRedGreen = 22,
  ColorScheme_Inverse_BlueRedGreenBlue = 23,
  ColorScheme_RedYellowGreenBlueRed = 24,
  ColorScheme_RedGreenBlueRed = 25,
  ColorScheme_Inverse_RedGreenBlueRed = 26,
  ColorScheme_RedYellowBlue = 27,
  ColorScheme_Inverse_RedYellowBlue = 28,
  ColorScheme_DEM_Normal = 29,
  ColorScheme_Inverse_DEM_Normal = 30,
  ColorScheme_DEM_Blind = 31,
  ColorScheme_Inverse_DEM_Blind = 32,
  ColorScheme_WhiteBlackWhite = 33,
  ColorScheme_BlackWhiteBlack = 34 }

  selects the ColorScheme of the data to transform real data to colored data.

- enum ColoringByteOrder {
  Coloring_RGBA = 0,
  Coloring_BGRA = 1,
  Coloring_ARGB = 2 }

  Selects the byte order of the coloring to be applied.

- enum ColorEnhancement {
  ColorEnhancement_None = 0,
  ColorEnhancement_Sine = 1,
  ColorEnhancement_Parable = 2,
  ColorEnhancement_Cubic = 3,
  ColorEnhancement_Sqrt = 4 }

  Selects the byte order of the coloring to be applied.

Functions

- SPECTRALRADAR_API ColoringHandle createColoring32Bit (ColorScheme Color, ColoringByteOrder ByteOrder)

  Creates processing that can be used to color given floating point B-scans to 32 bit colored images.
- SPECTRALRADAR_API ColoringHandle createCustomColoring32Bit (int LUTSize, unsigned long ∗LUT)

  Create custom coloring using the specified color look-up-table.
- SPECTRALRADAR_API void setColoringBoundaries (ColoringHandle Colorng, float Min_dB, float Max_dB)

  Sets the boundaries in dB which are used by the coloring algorithm to map colors to floating point values in dB.
- SPECTRALRADAR_API void setColoringEnhancement (ColoringHandle Coloring, ColorEnhancement Enhancement)

  Selects a function for non-linear coloring to enhance (subjective) image impression.
- SPECTRALRADAR_API void colorizeData (ColoringHandle Coloring, DataHandle Data, ColoredDataHandle ColoredData, BOOL Transpose)

  Colors a given data object (DataHandle) into a given colored object (ColoredDataHandle).
- SPECTRALRADAR_API void colorizeDopplerData (ColoringHandle AmpColoring, ColoringHandle Phase←Coloring, DataHandle AmpData, DataHandle PhaseData, ColoredDataHandle Output, double MinSignal_dB, BOOL Transpose)

  Colors a two given data object (DataHandle) using overlay and intensity to represent phase and amplitude data. Used for Doppler imaging.
- SPECTRALRADAR_API void colorizeDopplerDataEx (ColoringHandle AmpColoring, ColoringHandle PhaseColoring[2], DataHandle AmpData, DataHandle PhaseData, ColoredDataHandle Output, double MinSignal_dB, BOOL Transpose)

  Colors a two given data object (DataHandle) using overlay and intensity to represent phase and amplitude data. Used for Doppler imaging. In the extended version, two ColoringHandles can be specified, two provide different coloring for increasing and decreasing phase, for example.
- SPECTRALRADAR_API void clearColoring (ColoringHandle Handle)

  Clears the coloring previously created by createColoring32Bit.

5.18.1 Detailed Description

Functions used for coloring of floating point data.

5.18.2 Typedef Documentation

5.18.2.1 ColoringHandle

Handle for routines that color avaible scans for displaying.

5.18.3 Enumeration Type Documentation

5.18.3.1 enum ColorEnhancement

Selects the byte order of the coloring to be applied.

Enumerator

ColorEnhancement_None Use no color enhancement.

ColorEnhancement_Sine Apply a sine function as enhancement.

ColorEnhancement_Parable Apply a parable as enhancement.

ColorEnhancement_Cubic Apply a cubic function as enhancement.

ColorEnhancement_Sqrt Aplly a sqrt function as enhancement.

5.18.3.2 enum ColoringByteOrder

Selects the byte order of the coloring to be applied.

Enumerator

Coloring_RGBA Byte order RGBA.

Coloring_BGRA Byte order BGRA.

Coloring_ARGB Byte order ARGB.

5.18.3.3 enum ColorScheme

selects the ColorScheme of the data to transform real data to colored data.

Enumerator

ColorScheme_BlackAndWhite Black and white (monochrome) coloring.

ColorScheme_Inverted Black and white inverted (monochrome inverted) coloring.

ColorScheme_Color colored

ColorScheme_BlackAndOrange orange and black coloring

ColorScheme_BlackAndRed red and black coloring

ColorScheme_BlackRedAndYellow black, red and yellow coloring

ColorScheme_DopplerPhase Doppler phase data coloring. Red and blue allways colored in a range from -pi to +pi. Setting the boundaries for this color scheme is only allowed inbetween +pi and -pi

ColorScheme_BlueAndBlack blue and black coloring

ColorScheme_PolarizationRetardation colorful colorscheme

ColorScheme_GreenBlueAndBlack Green, blue and black is used as one half of a Doppler color scheme.

ColorScheme_BlackAndRedYellow Black, red, and yellow is used as one half of a Doppler color scheme.

ColorScheme_TransparentAndWhite Transparent and white coloring for overlay and 3D volume rendering purposes.

ColorScheme_GreenBlueWhiteRedYellow Green, blue, White, Red, and Yellow for polarization sensitive measurements.

ColorScheme_BlueGreenBlackYellowRed Blue, green, black, yellow, and red for polarization sensitive measurements.

ColorScheme_RedGreenBlue Red, green, and blue for polarization sensitive measurements.

ColorScheme_GreenBlueRed Green, blue and red for polarization sensitive measurements.

ColorScheme_BlueRedGreen Blue, red, and green for polarization sensitive measurements.

ColorScheme_GreenBlueRedGreen Green, blue and red for polarization sensitive measurements.

ColorScheme_BlueRedGreenBlue Blue, red, and green for polarization sensitive measurements.

ColorScheme_Inverse_RedGreenBlue Red, green, and blue for polarization sensitive measurements.

ColorScheme_Inverse_GreenBlueRed Green, blue and red for polarization sensitive measurements.

ColorScheme_Inverse_BlueRedGreen Blue, red, and green for polarization sensitive measurements.

ColorScheme_Inverse_GreenBlueRedGreen Green, blue and red for polarization sensitive measurements.

ColorScheme_Inverse_BlueRedGreenBlue Blue, red, and green for polarization sensitive measurements.

ColorScheme_RedYellowGreenBlueRed Red, yellow, green, blue, and red for polarization sensitive measurements.

ColorScheme_RedGreenBlueRed Red, green, blue, and red for polarization sensitive measurements.

ColorScheme_Inverse_RedGreenBlueRed Red, green, blue, and red for polarization sensitive measurements.

ColorScheme_RedYellowBlue Red, yellow, and blue.

ColorScheme_Inverse_RedYellowBlue Red, yellow, and blue.

ColorScheme_DEM_Normal DEM.

ColorScheme_DEM_Blind DEM.

5.18.4 Function Documentation

5.18.4.1 void clearColoring (**ColoringHandle** *Handle*)

Clears the coloring previously created by createColoring32Bit.

Parameters

`in`	*Handle*	A handle of a coloring (ColoringHandle). If the handle is a nullptr, this function does nothing. In most cases this handle will have been previously obtained with the function createColoring32Bit.

5.18.4.2 void colorizeData (**ColoringHandle** *Coloring*, **DataHandle** *Data*, **ColoredDataHandle** *ColoredData*, **BOOL** *Transpose*)

Colors a given data object (DataHandle) into a given colored object (ColoredDataHandle).

5.18.4.3 oid colorizeDopplerData (**ColoringHandle** *AmpColoring*, **ColoringHandle** *PhaseColoring*, **DataHandle** *AmpData*, **DataHandle** *PhaseData*, **ColoredDataHandle** *Output*, double *MinSignal_dB*, **BOOL** *Transpose*)

Colors a two given data object (DataHandle) using overlay and intensity to represent phase and amplitude data. Used for Doppler imaging.

5.18.4.4 oid colorizeDopplerDataEx (**ColoringHandle** *AmpColoring*, **ColoringHandle** *PhaseColoring[2]*, **DataHandle** *AmpData*, **DataHandle** *PhaseData*, **ColoredDataHandle** *Output*, double *MinSignal_dB*, **BOOL** *Transpose*)

Colors a two given data object (DataHandle) using overlay and intensity to represent phase and amplitude data. Used for Doppler imaging. In the extended version, two ColoringHandles can be specified, two provide different coloring for increasing and decreasing phase, for example.

5.18.4.5 ColoringHandle createColoring32Bit (ColorScheme *Color,* ColoringByteOrder *ByteOrder*)

Creates processing that can be used to color given floating point B-scans to 32 bit colored images.

Parameters

Color	The color-table to be used
ByteOrder	The byte order the coloring is supposed to use.

Returns

The handle (ColoringHandle) to the coloring algorithm.

5.18.4.6 ColoringHandle createCustomColoring32Bit (int *LUTSize,* unsigned long ∗ *LUT*)

Create custom coloring using the specified color look-up-table.

5.18.4.7 void setColoringBoundaries (ColoringHandle *Colorng,* float *Min_dB,* float *Max_dB*)

Sets the boundaries in dB which are used by the coloring algorithm to map colors to floating point values in dB.

5.18.4.8 void setColoringEnhancement (ColoringHandle *Coloring,* ColorEnhancement *Enhancement*)

Selects a function for non-linear coloring to enhance (subjective) image impression.

5.19 Camera

Functions for acquiring camera video images.

Functions

- SPECTRALRADAR_API void getMaxCameraImageSize (OCTDeviceHandle Dev, int ∗SizeX, int ∗SizeY)

    Returns the maximum possible camera image size for the current device.
- SPECTRALRADAR_API void getCameraImage (OCTDeviceHandle Dev, ColoredDataHandle Image)

    Gets a camera image.

5.19.1 Detailed Description

Functions for acquiring camera video images.

5.19.2 Function Documentation

5.19.2.1 void getCameraImage (**OCTDeviceHandle** *Dev,* **ColoredDataHandle** *Image*)

Gets a camera image.

5.19.2.2 void getMaxCameraImageSize (**OCTDeviceHandle** *Dev,* int ∗ *SizeX,* int ∗ *SizeY*)

Returns the maximum possible camera image size for the current device.

5.20 Helper function

Functions for chores common to many categories and scenarios.

Typedefs

- typedef struct C_VisualCalibration ∗ VisualCalibrationHandle

  Handle to the visual galvo calibration class.

Functions

- SPECTRALRADAR_API unsigned long InterpretReferenceIntensity (float intensity)

  interprets the reference intensity and gives a color code that reflects its state.
- SPECTRALRADAR_API void getConfigPath (char ∗Path, int StrSize)

  Returns the path that hold the config files.
- SPECTRALRADAR_API void getPluginPath (char ∗Path, int StrSize)

  Returns the path that hold the plugins.
- SPECTRALRADAR_API void getInstallationPath (char ∗Path, int StrSize)

  Returns the installation path.
- SPECTRALRADAR_API double getReferenceIntensity (ProcessingHandle Proc)

  Returns an absolute value that indicates the refernce intensity that was present when the currently used apodization was determined.
- SPECTRALRADAR_API double getRelativeReferenceIntensity (OCTDeviceHandle Dev, ProcessingHandle Proc)

  Returns a value larger than 0.0 and smaller than 1.0 that indicates the reference intensity (relative to saturation) that was present when the currently used apodization was determined.
- SPECTRALRADAR_API double getRelativeSaturation (ProcessingHandle Proc)

  Returns a value larger than 0.0 and smaller than 1.0 that indicates the saturation of the sensor that was present during the last processing cycle.

5.20.1 Detailed Description

Functions for chores common to many categories and scenarios.

5.20.2 Typedef Documentation

5.20.2.1 VisualCalibrationHandle

Handle to the visual galvo calibration class.

5.20.3 Function Documentation

5.20.3.1 void getConfigPath (char ∗ *Path,* int *StrSize*)

Returns the path that hold the config files.

5.20.3.2 void getInstallationPath (char * *Path,* int *StrSize*)

Returns the installation path.

5.20.3.3 void getPluginPath (char * *Path,* int *StrSize*)

Returns the path that hold the plugins.

5.20.3.4 double getReferenceIntensity (ProcessingHandle *Proc*)

Returns an absolute value that indicates the refernce intensity that was present when the currently used apodization was determined.

5.20.3.5 double double getRelativeReferenceIntensity (OCTDeviceHandle *Dev,* ProcessingHandle *Proc*)

Returns a value larger than 0.0 and smaller than 1.0 that indicates the reference intensity (relative to saturation) that was present when the currently used apodization was determined.

5.20.3.6 double double getRelativeSaturation (ProcessingHandle *Proc*)

Returns a value larger than 0.0 and smaller than 1.0 that indicates the saturation of the sensor that was present during the last processing cycle.

5.20.3.7 unsigned long InterpretReferenceIntensity (float *intensity*)

interprets the reference intensity and gives a color code that reflects its state.

Possible colors include:

- red = 0x00FF0000 (bad intensity);
- orange = 0x00FF7700 (okay intensity);
- green = 0x0000FF00 (good intensity);

Parameters

intensity	the current reference intensity as a value between 0.0 and 1.0

Returns

the color code reflecting the state of the refernce intensity

5.21 Buffer

Functions for acquiring camera video images.

Typedefs

- typedef struct C_Buffer ∗ BufferHandle

  The BufferHandle identifies a data buffer.

Functions

- SPECTRALRADAR_API BufferHandle createMemoryBuffer (void)

  Creates a buffer holding data and colored data.
- SPECTRALRADAR_API void appendToBuffer (BufferHandle, DataHandle, ColoredDataHandle)

  Appends specified data and colored data to the requested buffer.
- SPECTRALRADAR_API void purgeBuffer (BufferHandle)

  Discards all data.
- SPECTRALRADAR_API int getBufferSize (BufferHandle)

  Returns the currently avaiable data sets in the buffer.
- SPECTRALRADAR_API int getBufferFirstIndex (BufferHandle)

  Returns the index of the first data sets available in the buffer.
- SPECTRALRADAR_API int getBufferLastIndex (BufferHandle)

  Returns the index of one past the last data sets available in the buffer.
- SPECTRALRADAR_API DataHandle getBufferData (BufferHandle, int Index)

  Returns the data in the buffer.
- SPECTRALRADAR_API ColoredDataHandle getColoredBufferData (BufferHandle, int Index)

  Returns the colored data in the buffer.
- SPECTRALRADAR_API void clearBuffer (BufferHandle BufferHandle)

  Clears the buffer and frees all data and colored data objects in it.

5.21.1 Detailed Description

Functions for acquiring camera video images.

5.21.2 Typedef Documentation

5.21.2.1 BufferHandle

The BufferHandle identifies a data buffer.

5.21.3 Function Documentation

5.21.3.1 void appendToBuffer (BufferHandle , DataHandle , ColoredDataHandle)

Appends specified data and colored data to the requested buffer.

If insufficient memory is availalbe the oldest items in the buffer will be freed automatically.

5.21.3.2 void clearBuffer (BufferHandle *BufferHandle*)

Clears the buffer and frees all data and colored data objects in it.

Parameters

`in`	*BufferHandle*	A handle of a buffer (BufferHandle). If the handle is a nullptr, this function does nothing.

5.21.3.3 BufferHandle createMemoryBuffer (void)

Creates a buffer holding data and colored data.

5.21.3.4 DataHandle getBufferData (BufferHandle , int *Index*)

Returns the data in the buffer.

5.21.3.5 int getBufferFirstIndex (BufferHandle)

Returns the index of the first data sets available in the buffer.

5.21.3.6 int getBufferLastIndex (BufferHandle)

Returns the index of one past the last data sets available in the buffer.

5.21.3.7 int getBufferSize (BufferHandle)

Returns the currently avaiable data sets in the buffer.

5.21.3.8 ColoredDataHandle getColoredBufferData (BufferHandle , int *Index*)

Returns the colored data in the buffer.

5.21.3.9 void purgeBuffer (BufferHandle)

Discards all data.

5.22 Output Values (digital or analog)

Functions to inquire, setup and generate output values. Whether this functionality is supported, and to what extent, depends on the hardware.

Functions

- SPECTRALRADAR_API int getNumberOfOutputDeviceValues (OCTDeviceHandle Dev)

  Returns the number of output values.
- SPECTRALRADAR_API void getOutputDeviceValueName (OCTDeviceHandle Dev, int Index, char ∗Name, int NameStringSize, char ∗Unit, int UnitStringSize)

  Returns names and units of the requested output values.
- SPECTRALRADAR_API BOOL doesOutputDeviceValueExist (OCTDeviceHandle Dev, const char ∗Name)

  Returns whether the requested output device values exists or not.
- SPECTRALRADAR_API void setOutputDeviceValueByName (OCTDeviceHandle Dev, const char ∗Name, double value)

  Sets the specified output value.
- SPECTRALRADAR_API void setOutputValueByIndex (OCTDeviceHandle Dev, int Index, double Value)

  Sets the specified output value.
- SPECTRALRADAR_API void getOutputDeviceValueRangeByName (OCTDeviceHandle Dev, const char ∗Name, double ∗Min, double ∗Max)

  Gives the range of the specified output value.
- SPECTRALRADAR_API void getOutputValueRangeByIndex (OCTDeviceHandle Dev, int Index, double ∗Min, double ∗Max)

  Gives the range of the specified output value.

5.22.1 Detailed Description

Functions to inquire, setup and generate output values. Whether this functionality is supported, and to what extent, depends on the hardware.

5.22.2 Function Documentation

5.22.2.1 BOOL doesOutputDeviceValueExist (OCTDeviceHandle *Dev,* const char ∗ *Name*)

Returns whether the requested output device values exists or not.

5.22.2.2 int getNumberOfOutputDeviceValues (OCTDeviceHandle *Dev*)

Returns the number of output values.

5.22.2.3 void getOutputDeviceValueName (OCTDeviceHandle *Dev,* int *Index,* char ∗ *Name,* int *NameStringSize,* char ∗ *Unit,* int *UnitStringSize*)

Returns names and units of the requested output values.

5.22.2.4 void getOutputDeviceValueRangeByName (**OCTDeviceHandle** *Dev,* const char ∗ *Name,* double ∗ *Min,* double ∗ *Max*)

Gives the range of the specified output value.

5.22.2.5 void getOutputValueRangeByIndex (**OCTDeviceHandle** *Dev,* int *Index,* double ∗ *Min,* double ∗ *Max*)

Gives the range of the specified output value.

5.22.2.6 void setOutputDeviceValueByName (**OCTDeviceHandle** *Dev,* const char ∗ *Name,* double *value*)

Sets the specified output value.

5.22.2.7 void setOutputValueByIndex (**OCTDeviceHandle** *Dev,* int *Index,* double *Value*)

Sets the specified output value.

5.23 File Handling

Typedefs

- typedef struct C_MarkerList ∗ MarkerListHandle

  Handle to the marker list class.

Enumerations

- enum OCTFileFormat {
  FileFormat_OCITY,
  FileFormat_IMG,
  FileFormat_SDR,
  FileFormat_SRM,
  FileFormat_TIFF32 }

  Enum identifying possible file formats.
- enum DataObjectType {
  DataObjectType_Real,
  DataObjectType_Colored,
  DataObjectType_Complex,
  DataObjectType_Raw,
  DataObjectType_Binary,
  DataObjectType_Text,
  DataObjectType_Unknown = 999 }

  Enum identifying.
- enum FileMetadataFloat {
  FileMetadata_RefractiveIndex,
  FileMetadata_RangeX,
  FileMetadata_RangeY,
  FileMetadata_RangeZ,
  FileMetadata_CenterX,
  FileMetadata_CenterY,
  FileMetadata_Angle,
  FileMetadata_BinToElectronScaling,
  FileMetadata_CentralWavelength_nm,
  FileMetadata_SourceBandwidth_nm,
  FileMetadata_MinElectrons,
  FileMetadata_QuadraticDispersionCorrectionFactor,
  FileMetadata_SpeckleVarianceThreshold,
  FileMetadata_ScanTime_Sec,
  FileMetadata_ReferenceIntensity,
  FileMetadata_ScanPause_Sec,
  FileMetadata_Zoom,
  FileMetadata_MinPointDistance,
  FileMetadata_MaxPointDistance,
  FileMetadata_FFTOversampling,
  FileMetadata_FullWellCapacity,
  FileMetadata_Saturation,
  FileMetadata_CameraLineRate_Hz,
  FileMetadata_PMDCorrectionAngle_rad,
  FileMetadata_OpticalAxisOffset_rad }

  Enum identifying file metadata fields of floating point type.

- enum FileMetadataInt {
FileMetadata_ProcessState,
FileMetadata_SizeX,
FileMetadata_SizeY,
FileMetadata_SizeZ,
FileMetadata_Oversampling,
FileMetadata_IntensityAveragedSpectra,
FileMetadata_IntensityAveragedAScans,
FileMetadata_IntensityAveragedBScans,
FileMetadata_DopplerAverageX,
FileMetadata_DopplerAverageZ,
FileMetadata_ApoWindow,
FileMetadata_DeviceBitDepth,
FileMetadata_SpectrometerElements,
FileMetadata_ExperimentNumber,
FileMetadata_DeviceBytesPerPixel,
FileMetadata_SpeckleAveragingFastAxis,
FileMetadata_SpeckleAveragingSlowAxis,
FileMetadata_Processing_FFTType,
FileMetadata_NumOfCameras,
FileMetadata_SelectedCamera,
FileMetadata_ApodizationType,
FileMetadata_AcquisitionOrder,
FileMetadata_DOPUFilter,
FileMetadata_DOPUAverageZ,
FileMetadata_DOPUAverageX,
FileMetadata_DOPUAverageY,
FileMetadata_PolarizationAverageZ,
FileMetadata_PolarizationAverageX,
FileMetadata_PolarizationAverageY }

  Enum identifying file metadata fields of integral type.

- enum FileMetadataString {
FileMetadata_DeviceSeries,
FileMetadata_DeviceName,
FileMetadata_Serial,
FileMetadata_Comment,
FileMetadata_CustomInfo,
FileMetadata_AcquisitionMode,
FileMetadata_Study,
FileMetadata_DispersionPreset,
FileMetadata_ProbeName,
FileMetadata_FreeformScanPatternInterpolation,
FileMetadata_HardwareConfig,
FileMetadata_OrigVersion,
FileMetadata_LastModVersion }

  Enum identifying file metadata fields of character string type.

- enum FileMetadataFlag {

FileMetadata_OffsetApplied,
FileMetadata_DCSubtracted,
FileMetadata_ApoApplied,
FileMetadata_DechirpApplied,
FileMetadata_UndersamplingFilterApplied,
FileMetadata_DispersionCompensationApplied,
FileMetadata_QuadraticDispersionCorrectionUsed,
FileMetadata_ImageFieldCorrectionApplied,
FileMetadata_ScanLineShown,
FileMetadata_AutoCorrCompensationUsed,
FileMetadata_BScanCrossCorrelation,
FileMetadata_DCSubtractedAdvanced,
FileMetadata_OnlyWindowing,
FileMetadata_RawDataIsSigned,
FileMetadata_FreeformScanPatternIsActive,
FileMetadata_FreeformScanPatternCloseLoop,
FileMetadata_IsSweptSource }

Enum identifying file metadata fields of bool type.

Functions

- SPECTRALRADAR_API const char ∗ DataObjectName_SpectralData (int index)

  Returns the filename of the spectral-data object with the specified index.
- SPECTRALRADAR_API OCTFileHandle createOCTFile (OCTFileFormat format)

  Creates a handle to an OCT file of the given format.
- SPECTRALRADAR_API void clearOCTFile (OCTFileHandle Handle)

  Clears the given OCT file handle and frees its resources.
- SPECTRALRADAR_API int getFileDataObjectCount (OCTFileHandle Handle)

  Returns the number of data objects in the OCT file. This number will vary depending on the file's format and contents (Files with the .oct extension may contain multiple OCT data objects depending on their internal structure).
- SPECTRALRADAR_API void loadFile (OCTFileHandle Handle, const char ∗Filename)

  Loads the actual OCT data file from a file system. The file must have the format given in createOCTFile().
- SPECTRALRADAR_API void saveFile (OCTFileHandle Handle, const char ∗Filename)

  Saves the OCT data file in the given fully qualified path name.
- SPECTRALRADAR_API void saveChangesToFile (OCTFileHandle Handle)

  Saves the OCT data file in the file previously opened with loadFile(). Only changes will be saved.
- SPECTRALRADAR_API void copyFileMetadata (OCTFileHandle SrcHandle, OCTFileHandle DstHandle)

  Copies metadata from one OCT file to another.
- SPECTRALRADAR_API double getFileMetadataFloat (OCTFileHandle Handle, FileMetadataFloat Floatfield)

  Returns the value of the given file metadata field as a floating point number if found.
- SPECTRALRADAR_API void setFileMetadataFloat (OCTFileHandle Handle, FileMetadataFloat Floatfield, double Value)

  Sets the value of the given file metadata field as a floating point number.
- SPECTRALRADAR_API int getFileMetadataInt (OCTFileHandle Handle, FileMetadataInt Intfield)

  Returns the value of the given file metadata field as an integer if found.
- SPECTRALRADAR_API void setFileMetadataInt (OCTFileHandle Handle, FileMetadataInt Intfield, int Value)

  Sets the value of the given file metadata field as an integer.
- SPECTRALRADAR_API const char ∗ getFileMetadataString (OCTFileHandle Handle, FileMetadataString Stringfield)

  Returns the value of the given file metadata field as a string if found.
- SPECTRALRADAR_API void setFileMetadataString (OCTFileHandle Handle, FileMetadataString Stringfield, const char ∗Content)

  Sets the value of the given file metadata field as a string.

- SPECTRALRADAR_API BOOL getFileMetadataFlag (OCTFileHandle Handle, FileMetadataFlag Boolfield)

  Gets the boolean value of the given file metadata field.
- SPECTRALRADAR_API void setFileMetadataFlag (OCTFileHandle Handle, FileMetadataFlag Boolfield, BOOL Value)

  Sets the boolean value of the given file metadata field.
- SPECTRALRADAR_API void saveFileMetadata (OCTFileHandle Handle, OCTDeviceHandle Dev, ProcessingHandle Proc, ProbeHandle Probe, ScanPatternHandle Pattern)

  Saves meta information from the given device, processing, probe and scan pattern instances in the metadata block of the given file handle. This information will be available in files of type FileFormat_OCITY; mileage on other formats may vary according to their description.
- SPECTRALRADAR_API void saveFileMetadataDoppler (OCTFileHandle Handle, DopplerProcessingHandle DopplerProc)

  Saves meta information from the given DopplerProcessingHandle. A corresponding DopplerProcessingHandle can then be recreated using createDopplerProcessingForFile.
- SPECTRALRADAR_API void saveFileMetadataSpeckle (OCTFileHandle Handle, SpeckleVarianceHandle SpeckleVarianceProc)

  Saves meta information from the given SpeckleVarianceHandle. A corresponding SpeckleVarianceHandle can then be recreated using initSpeckleVarianceForFile.
- SPECTRALRADAR_API void loadCalibrationFromFile (OCTFileHandle Handle, ProcessingHandle Proc)

  Loads Chirp, Offset, and Apodization vectors from the given OCT file into the given processing object.
- SPECTRALRADAR_API void loadCalibrationFromFileEx (OCTFileHandle Handle, ProcessingHandle Proc, const int CameraIndex)

  Loads Chirp, Offset, and Apodization vectors from the given OCT file into the given processing object.
- SPECTRALRADAR_API void saveCalibrationToFile (OCTFileHandle Handle, ProcessingHandle Proc)

  Saves Chirp, Offset, and Apodization vectors from the given processing object into the given OCT file.
- SPECTRALRADAR_API void saveCalibrationToFileEx (OCTFileHandle Handle, ProcessingHandle Proc, int CameraIndex)

  Saves Chirp, Offset, and Apodization vectors from the given processing object into the given OCT file.
- SPECTRALRADAR_API void getFileRealData (OCTFileHandle Handle, DataHandle Data, int Index)

  Retrieves a RealData object from the OCT file at the given index with 0 <= index < getFileDataObjectCount(OCTFileHandle handle). Users must ensure that the data handle is properly prepared and destroyed.
- SPECTRALRADAR_API void getFileColoredData (OCTFileHandle Handle, ColoredDataHandle Data, size_t Index)

  Retrieves a ColoredData object from the OCT file at the given index with 0 <= index < getFileDataObjectCount(OCTFileHandle handle). Users must ensure that the data handle is properly prepared and destroyed.
- SPECTRALRADAR_API void getFileComplexData (OCTFileHandle Handle, ComplexDataHandle Data, size_t Index)

  Retrieves a ComplexData object from the OCT file at the given index with 0 <= index < getFileDataObjectCount(OCTFileHandle handle). Users must ensure that the data handle is properly prepared and destroyed.
- SPECTRALRADAR_API void getFileRawData (OCTFileHandle Handle, RawDataHandle Data, size_t Index)

  Retrieves a RawData object from the OCT file at the given index with 0 <= index < getFileDataObjectCount(OCTFileHandle handle). Users must ensure that the data handle is properly prepared and destroyed.
- SPECTRALRADAR_API void getFile (OCTFileHandle Handle, size_t Index, const char *FilenameOnDisk)

  Retrieves a data object of arbitrary type from the OCT file at the given index with 0 <= index < getFileDataObjectCount(OCTFileHandle handle) and stores it at the given fully qualified path.
- SPECTRALRADAR_API int findFileDataObject (OCTFileHandle Handle, const char *Search)

  Searches for a data object the name of which contains the given string and returns its index, -1 if not found.
- SPECTRALRADAR_API BOOL containsFileDataObject (OCTFileHandle Handle, const char *Search)

  Searches for a data object the name of which contains the given string and returns TRUE if at least one data object name matches.
- SPECTRALRADAR_API BOOL containsFileRawData (OCTFileHandle Handle)

  Returns TRUE if the file contains raw data objects.
- SPECTRALRADAR_API void addFileRealData (OCTFileHandle Handle, DataHandle Data, const char *DataObjectName)

Adds a RealData object to the OCT file; dataObjectName will be its name inside the OCT file if applicable. The object that the DataHandle refers to must live until after saveFile() has been called.

- SPECTRALRADAR_API void addFileColoredData (OCTFileHandle Handle, ColoredDataHandle Data, const char ∗DataObjectName)

  Adds a ColoredData object to the OCT file; dataObjectName will be its name inside the OCT file if applicable. The object that the ColoredDataHandle refers to must live until after saveFile() has been called.

- SPECTRALRADAR_API void addFileComplexData (OCTFileHandle Handle, ComplexDataHandle Data, const char ∗DataObjectName)

  Adds a ComplexData object to the OCT file; dataObjectName will be its name inside the OCT file if applicable. The object that the ComplexDataHandle refers to must live until after saveFile() has been called.

- SPECTRALRADAR_API void addFileRawData (OCTFileHandle Handle, RawDataHandle Data, const char ∗DataObjectName)

  Adds raw Data object to the OCT file; DataObjectName will be its name inside the OCT file if applicable. The object that the RawDataHandle refers to must live until after saveFile has been called.

- SPECTRALRADAR_API void addFileText (OCTFileHandle Handle, const char ∗FilenameOnDisk, const char ∗DataObjectName)

  Adds a text object read from FilenameOnDisk to the OCT file; DataObjectName will be its name inside the OCT file if applicable. The file identified by filenameOnDisk must exist until after saveFile() has been called.

- SPECTRALRADAR_API DataObjectType getFileDataObjectType (OCTFileHandle Handle, int Index)

  Returns the type of the data object at the given Index in the OCT file.

- SPECTRALRADAR_API void getFileDataObjectName (OCTFileHandle Handle, int Index, char ∗Filename, int Length)

  Returns the name of the data object at the given Index in the OCT file.

- SPECTRALRADAR_API int getFileDataSizeX (OCTFileHandle Handle, size_t Index)

  Returns the pixel count in X of the data object at the given Index in the OCT file.

- SPECTRALRADAR_API int getFileDataSizeY (OCTFileHandle Handle, size_t Index)

  Returns the pixel count in Y of the data object at the given Index in the OCT file.

- SPECTRALRADAR_API int getFileDataSizeZ (OCTFileHandle Handle, size_t Index)

  Returns the pixel count in Z of the data object at the given Index in the OCT file.

- SPECTRALRADAR_API float getFileDataRangeX (OCTFileHandle Handle, size_t Index)

  Returns the range (usually in mm) in X of the data object at the given Index in the OCT file.

- SPECTRALRADAR_API float getFileDataRangeY (OCTFileHandle Handle, size_t Index)

  Returns the range (usually in mm) in Y of the data object at the given Index in the OCT file.

- SPECTRALRADAR_API float getFileDataRangeZ (OCTFileHandle Handle, size_t Index)

  Returns the range (usually in mm) in Z of the data object at the given Index in the OCT file.

- SPECTRALRADAR_API void copyMarkerListFromRealData (OCTFileHandle Handle, DataHandle Data)

  coordinates, so re-use is possible.

- SPECTRALRADAR_API void copyMarkerListToRealData (OCTFileHandle Handle, DataHandle Data)

  coordinates, so re-use is possible.

- SPECTRALRADAR_API void addFileMetadataPreset (OCTFileHandle Handle, const char ∗Category, const char ∗PresetDescription)

  Adds one of the presets set during acquisition for the OCTFileHandle.

- SPECTRALRADAR_API int getFileMetadataNumberOfPresets (OCTFileHandle Handle)

  Gets the number of presets that were set during the acquisition.

- SPECTRALRADAR_API const char ∗ getFileMetadataPresetCategory (OCTFileHandle Handle, int Index)

  Gets the preset category belonging to the preset with given Index.

- SPECTRALRADAR_API const char ∗ getFileMetadataPresetDescription (OCTFileHandle Handle, int Index)

  Gets the preset description belonging to the preset with given Index.

5.23.1 Detailed Description

5.23.2 Typedef Documentation

5.23.2.1 MarkerListHandle

Handle to the marker list class.

5.23.3 Enumeration Type Documentation

5.23.3.1 enum DataObjectType

Enum identifying.

5.23.3.2 enum FileMetadataFlag

Enum identifying file metadata fields of bool type.

Enumerator

FileMetadata_OffsetApplied This field is the flag that can be accessed with the functions getProcessingFlag / setProcessingFlag and the constant Processing_UseOffsetErrors.

FileMetadata_DCSubtracted This field is the flag that can be accessed with the functions getProcessingFlag / setProcessingFlag and the constant Processing_RemoveDCSpectrum.

FileMetadata_UndersamplingFilterApplied This field is the flag that can be accessed with the functions getProcessingFlag / setProcessingFlag and the constant Processing_UseUndersamplingFilter.

FileMetadata_AutoCorrCompensationUsed This field is the flag that can be accessed with the functions getProcessingFlag / setProcessingFlag and the constant Processing_UseAutocorrCompensation.

FileMetadata_DCSubtractedAdvanced This field is the flag that can be accessed with the functions getProcessingFlag / setProcessingFlag and the constant Processing_RemoveAdvancedDCSpectrum.

FileMetadata_OnlyWindowing This field is the flag that can be accessed with the functions getProcessingFlag / setProcessingFlag and the constant Processing_OnlyWindowing.

FileMetadata_RawDataIsSigned This field is the flag that can be retrieved with the function getDevicePropertyInt and the constant Device_DataIsSigned.

5.23.3.3 enum FileMetadataFloat

Enum identifying file metadata fields of floating point type.

Enumerator

FileMetadata_RefractiveIndex The refractive index applied to the whole image.

FileMetadata_RangeX The FOV in axial direction (x) in mm.

FileMetadata_RangeY The FOV in axial direction (y) in mm.

FileMetadata_RangeZ The FOV in longitudinal axis (z) in mm.

FileMetadata_CenterX The center of the scan pattern in axial direction (x) in mm.

FileMetadata_CenterY The center of the scan pattern in axial direction (y) in mm.

FileMetadata_Angle The angle betwenn the scanner and the video camera image.

FileMetadata_BinToElectronScaling Ratio between the binary value from the camera to the count of electrons.

FileMetadata_CentralWavelength_nm Central wavelength of the device.

FileMetadata_SourceBandwidth_nm Bandwidth of the light source.

FileMetadata_MinElectrons Electron cut-off parameter used for processing.

FileMetadata_QuadraticDispersionCorrectionFactor Quadratic dispersion factor used for dispersion correction.

FileMetadata_SpeckleVarianceThreshold Threshold for speckle variance mode.

FileMetadata_ScanTime_Sec Time needed for data acqusition. The processing and saving time is not included.

FileMetadata_ReferenceIntensity Value for the reference intensity.

FileMetadata_ScanPause_Sec Scan pause in between scans.

FileMetadata_Zoom Zooms the scan pattern.

FileMetadata_MinPointDistance Minimum distance between two points of the scan pattern used for freeform scan patterns.

FileMetadata_MaxPointDistance Maximum distance between two points of the scan pattern used for freeform scan patterns.

FileMetadata_FFTOversampling FFT oversampling use for processing and chirp correction.

FileMetadata_PMDCorrectionAngle_rad Polarization mode correction. This angle (expresssed in radians) is used to compute a phasor ($\exp(i\alpha)$), that will be applied to the complex reflectivities vector associated with camera 0.

FileMetadata_OpticalAxisOffset_rad In birefringent samples, this offset allows referring the angle of the fast axis to an axis in the sample holder.

5.23.3.4 enum FileMetadataInt

Enum identifying file metadata fields of integral type.

Enumerator

FileMetadata_ProcessState Contains the specifif data format.

FileMetadata_SizeX Number of pixels in x.

FileMetadata_SizeY Number of pixels in y.

FileMetadata_SizeZ Number of pixels in z.

FileMetadata_Oversampling Oversampling parameter.

FileMetadata_IntensityAveragedSpectra Spectrum averaging.

FileMetadata_IntensityAveragedAScans A-scan averaging.

FileMetadata_IntensityAveragedBScans B-scan averaging.

FileMetadata_DopplerAverageX Averaging for doppler processing in x-direction.

FileMetadata_DopplerAverageZ Averaging for doppler processing in z-direction.

FileMetadata_ApoWindow Type of window used for apodization.

FileMetadata_DeviceBitDepth Bits per pixel of the camera.

FileMetadata_SpectrometerElements Number of elements of the spectrometer.

FileMetadata_ExperimentNumber Serial number of the dataset.

FileMetadata_DeviceBytesPerPixel Bytes per pixel of the camera.

FileMetadata_SpeckleAveragingFastAxis Averaging parameter of the fast scan axis in speckle variance mode.

FileMetadata_SpeckleAveragingSlowAxis Averaging parameter of the slow scan axis in speckle variance mode.

FileMetadata_Processing_FFTType FFT algorithm used.

FileMetadata_NumOfCameras Number of cameras, or sensors, stored in the file. In case of legacy files, this property takes the default value "1".

FileMetadata_SelectedCamera In devices with more than one camera, some modi need to know which camera is active, because they do not support work with multiple cameras. In case of legacy files, this property takes the default value "0".

FileMetadata_DOPUFilter DOPU filter specification. See PolarizationDOPUFilterType.

FileMetadata_DOPUAverageZ Number of pixels for DOPU averaging in the z-direction.

FileMetadata_DOPUAverageX Number of pixels for DOPU averaging in the x-direction.

FileMetadata_DOPUAverageY Number of pixels for DOPU averaging in the y-direction.

FileMetadata_PolarizationAverageZ Number of pixels for averaging along the z axis.

FileMetadata_PolarizationAverageX Number of pixels for averaging along the x axis.

FileMetadata_PolarizationAverageY Number of pixels for averaging along the y axis.

5.23.3.5 enum FileMetadataString

Enum identifying file metadata fields of character string type.

Enumerator

FileMetadata_DeviceSeries Order of the axis, e.g. ZXY. FileMetadata_AxisOrder,

FileMetadata_DeviceName Name of the OCT device.

FileMetadata_Serial Serial number of the OCT device.

FileMetadata_Comment Comment of the OCT data file.

FileMetadata_CustomInfo Additional, custom info.

FileMetadata_AcquisitionMode Acquisition mode of the OCT data file.

FileMetadata_Study Study of the OCT data file.

FileMetadata_DispersionPreset Dispersion Preset of the OCT data file.

FileMetadata_ProbeName Name of the probe.

5.23.3.6 enum OCTFileFormat

Enum identifying possible file formats.

5.23.4 Function Documentation

5.23.4.1 void addFileColoredData (OCTFileHandle *Handle*, ColoredDataHandle *Data*, const char ∗ *DataObjectName*)

Adds a ColoredData object to the OCT file; dataObjectName will be its name inside the OCT file if applicable. The object that the ColoredDataHandle refers to must live until after saveFile() has been called.

Parameters

in	*Handle*	A valid (non null) handle of OCTFile (OCTFileHandle), obtained with the function createOCTFile.
in	*Data*	A valid (non null) handle to the ColoredData object (ColoredDataHandle) to add.
in	*DataObjectName*	Name that will be assigned to the object in the OCT file.

5.23.4.2 void addFileComplexData (OCTFileHandle *Handle,* ComplexDataHandle *Data,* const char ∗ *DataObjectName*)

Adds a ComplexData object to the OCT file; dataObjectName will be its name inside the OCT file if applicable. The object that the ComplexDataHandle refers to must live until after saveFile() has been called.

Parameters

in	*Handle*	A valid (non null) handle of OCTFile (OCTFileHandle), obtained with the function createOCTFile.
in	*Data*	A valid (non null) handle to the ComplexData object (ComplexDataHandle) to add.
in	*DataObjectName*	Name that will be assigned to the object in the OCT file.

5.23.4.3 void addFileMetadataPreset (OCTFileHandle *Handle,* const char ∗ *Category,* const char ∗ *PresetDescription*)

Adds one of the presets set during acquisition for the OCTFileHandle.

Parameters

in	*Handle*	A valid (non null) handle of OCTFile (OCTFileHandle), obtained with the function createOCTFile.
in	*Category*	Name of the category of the added preset.
in	*PresetDescription*	Description for the added preset.

5.23.4.4 void addFileRawData (OCTFileHandle *Handle,* RawDataHandle *Data,* const char ∗ *DataObjectName*)

Adds raw *Data* object to the OCT file; *DataObjectName* will be its name inside the OCT file if applicable. The object that the RawDataHandle refers to must live until after saveFile has been called.

Parameters

in	*Handle*	A valid (non null) handle of OCTFile (OCTFileHandle), obtained with the function createOCTFile.
in	*Data*	A valid (non null) raw data handle of the existing data (RawDataHandle), previously obtained with the function createRawData. It is assumed that these data have already been filled in with an appropiate data acquisition procedure.
in	*DataObjectName*	Name that will be assigned to the object in the OCT file. Notice that raw data refers to the spectra as acquired, without processing of any kind.

5.23.4.5 void addFileRealData (OCTFileHandle *Handle,* DataHandle *Data,* const char ∗ *DataObjectName*)

Adds a RealData object to the OCT file; dataObjectName will be its name inside the OCT file if applicable. The object that the DataHandle refers to must live until after saveFile() has been called.

Parameters

in	*Handle*	A valid (non null) handle of OCTFile (OCTFileHandle), obtained with the function createOCTFile.
in	*Data*	A valid (non null) handle to the RealData object (DataHandle) to add.
in	*DataObjectName*	Name that will be assigned to the object in the OCT file.

5.23.4.6 void addFileText (OCTFileHandle *Handle*, const char ∗ *FilenameOnDisk*, const char ∗ *DataObjectName*)

Adds a text object read from *FilenameOnDisk* to the OCT file; *DataObjectName* will be its name inside the OCT file if applicable. The file identified by filenameOnDisk must exist until after saveFile() has been called.

Parameters

in	*Handle*	A valid (non null) handle of OCTFile (OCTFileHandle), obtained with the function createOCTFile.
in	*FilenameOnDisk*	Filename from which text file will be read.
in	*DataObjectName*	Name that will be assigned to the object in the OCT file.

5.23.4.7 void clearOCTFile (OCTFileHandle *Handle*)

Clears the given OCT file handle and frees its resources.

Parameters

in	*Handle*	A valid (non null) handle of OCTFile (OCTFileHandle), obtained with the function createOCTFile.

5.23.4.8 BOOL containsFileDataObject (OCTFileHandle *Handle*, const char ∗ *Search*)

Searches for a data object the name of which contains the given string and returns TRUE if at least one data object name matches.

Parameters

in	*Handle*	A valid (non null) handle of OCTFile (OCTFileHandle), obtained with the function createOCTFile.
in	*Search*	Data object name to find in OCT file.

5.23.4.9 BOOL containsFileRawData (OCTFileHandle *Handle*)

Returns TRUE if the file contains raw data objects.

Parameters

in	*Handle*	A valid (non null) handle of OCTFile (OCTFileHandle), obtained with the function createOCTFile. Notice that raw data refers to the spectra as acquired, without processing of any kind.

5.23.4.10 void copyFileMetadata (OCTFileHandle *SrcHandle*, OCTFileHandle *DstHandle*)

Copies metadata from one OCT file to another.

Parameters

in	*SrcHandle*	A valid (non null) handle of OCTFile (OCTFileHandle), obtained with the function createOCTFile. This is the source and will not be altered by this function in any way.

Parameters

out	*DstHandle*	A valid (non null) handle of OCTFile (OCTFileHandle), obtained with the function createOCTFile. This is the destination and will be filled in using the information in the source.

5.23.4.11 void copyMarkerListFromRealData (OCTFileHandle *Handle*, DataHandle *Data*)

coordinates, so re-use is possible.

Copies the marker list from the given data handle into the metadata block of the given OCT file handle.

Markers are a visual help, that can be created or manipulated by ThorImage-OCT. Markers are always expressed in physical

Parameters

in	*Handle*	A valid (non null) handle of OCTFile (OCTFileHandle), obtained with the function createOCTFile.
in	*Data*	A valid (non null) data handle of the existing data (DataHandle), previously obtained with the function createData. It is assumed that this structure has already been filled with processed data. If no markers are present, this function does nothing.

5.23.4.12 void copyMarkerListToRealData (OCTFileHandle *Handle*, DataHandle *Data*)

coordinates, so re-use is possible.

Copies the marker list from the metadata block of the given file handle to the given data handle.

Markers are a visual help, that can be created or manipulated by ThorImage-OCT. Markers are always expressed in physical

Parameters

in	*Handle*	A valid (non null) handle of OCTFile (OCTFileHandle), obtained with the function createOCTFile.
out	*Data*	A valid (non null) data handle of the existing data (DataHandle), previously obtained with the function createData. If no markers are present, this function does nothing.

5.23.4.13 OCTFileHandle createOCTFile (OCTFileFormat *format*)

Creates a handle to an OCT file of the given format.

5.23.4.14 const char ∗ DataObjectName_SpectralData (int *index*)

Returns the filename of the spectral-data object with the specified index.

Parameters

index	Index of spectral-data object to return

Returns

Filename of the specified data object

5.23.4.15 int findFileDataObject (OCTFileHandle *Handle*, const char ∗ *Search*)

Searches for a data object the name of which contains the given string and returns its index, -1 if not found.

Parameters

in	*Handle*	A valid (non null) handle of OCTFile (OCTFileHandle), obtained with the function createOCTFile.
in	*Search*	Data object name to find in OCT file.

5.23.4.16 void getFile (OCTFileHandle *Handle*, size_t *Index*, const char ∗ *FilenameOnDisk*)

Retrieves a data object of arbitrary type from the OCT file at the given index with 0 <= index < getFileDataObjectCount(OCTFileHandle handle) and stores it at the given fully qualified path.

Parameters

in	*Handle*	A valid (non null) handle of OCTFile (OCTFileHandle), obtained with the function createOCTFile.
in	*Index*	Index of the file inside the OCT file, e.g. returned by findFileDataObject.
in	*FilenameOnDisk*	Filename to which requested file will be written.

5.23.4.17 void getFileColoredData (OCTFileHandle *Handle*, ColoredDataHandle *Data*, size_t *Index*)

Retrieves a ColoredData object from the OCT file at the given index with 0 <= index < getFileDataObjectCount(OCTFileHandle handle). Users must ensure that the data handle is properly prepared and destroyed.

Parameters

in	*Handle*	A valid (non null) handle of OCTFile (OCTFileHandle), obtained with the function createOCTFile.
out	*Data*	A valid (non null) colored data handle of the existing data (ColoredDataHandle), previously obtained with the function createColoredData. It will be filled in with the data read from the OCT file at the given *Index*.
in	*Index*	Index of the data inside the OCT file, e.g. returned by findFileDataObject.

5.23.4.18 void getFileComplexData (OCTFileHandle *Handle*, ComplexDataHandle *Data*, size_t *Index*)

Retrieves a ComplexData object from the OCT file at the given index with 0 <= index < getFileDataObjectCount(OCTFileHandle handle). Users must ensure that the data handle is properly prepared and destroyed.

Parameters

in	*Handle*	A valid (non null) handle of OCTFile (OCTFileHandle), obtained with the function createOCTFile.
out	*Data*	A valid (non null) complex data handle of the existing data (ComplexDataHandle), previously obtained with the function createComplexData. It will be filled in with the data read from the OCT file at the given *Index*.

Parameters

in	*Index*	Index of the data inside the OCT file, e.g. returned by findFileDataObject.

5.23.4.19 int getFileDataObjectCount (OCTFileHandle *Handle*)

Returns the number of data objects in the OCT file. This number will vary depending on the file's format and contents (Files with the .oct extension may contain multiple OCT data objects depending on their internal structure).

Parameters

in	*Handle*	A valid (non null) handle of OCTFile (OCTFileHandle), obtained with the function createOCTFile.

5.23.4.20 void getFileDataObjectName (OCTFileHandle *Handle,* int *Index,* char ∗ *Filename,* int *Length*)

Returns the name of the data object at the given *Index* in the OCT file.

Parameters

in	*Handle*	A valid (non null) handle of OCTFile (OCTFileHandle), obtained with the function createOCTFile.
in	*Index*	Index of the data inside the OCT file, 0 <= Index < getFileDataObjectCount()
out	*Filename*	Name of the requested file
in	*Length*	Length of the user-provided buffer at *Filename*

5.23.4.21 DataObjectType getFileDataObjectType (OCTFileHandle *Handle,* int *Index*)

Returns the type of the data object at the given *Index* in the OCT file.

Parameters

in	*Handle*	A valid (non null) handle of OCTFile (OCTFileHandle), obtained with the function createOCTFile.
in	*Index*	Index of the data inside the OCT file, e.g. returned by findFileDataObject.

Returns

The type of the selected data object or DataObjectType_Unknown in case of an error.

5.23.4.22 float getFileDataRangeX (OCTFileHandle *Handle,* size_t *Index*)

Returns the range (usually in mm) in X of the data object at the given *Index* in the OCT file.

Parameters

in	*Handle*	A valid (non null) handle of OCTFile (OCTFileHandle), obtained with the function createOCTFile.
in	*Index*	Index of the data inside the OCT file, e.g. returned by findFileDataObject.

Returns

Range in X of the data object or 0.0f in case of an error

5.23.4.23 float getFileDataRangeY (OCTFileHandle *Handle,* size_t *Index*)

Returns the range (usually in mm) in Y of the data object at the given *Index* in the OCT file.

Parameters

in	*Handle*	A valid (non null) handle of OCTFile (OCTFileHandle), obtained with the function createOCTFile.
in	*Index*	Index of the data inside the OCT file, e.g. returned by findFileDataObject.

Returns

Range in Y of the data object or 0.0f in case of an error

5.23.4.24 float getFileDataRangeZ (OCTFileHandle *Handle,* size_t *Index*)

Returns the range (usually in mm) in Z of the data object at the given *Index* in the OCT file.

Parameters

in	*Handle*	A valid (non null) handle of OCTFile (OCTFileHandle), obtained with the function createOCTFile.
in	*Index*	Index of the data inside the OCT file, e.g. returned by findFileDataObject.

Returns

Range in Z of the data object or 0.0f in case of an error

5.23.4.25 int getFileDataSizeX (OCTFileHandle *Handle,* size_t *Index*)

Returns the pixel count in X of the data object at the given *Index* in the OCT file.

Parameters

in	*Handle*	A valid (non null) handle of OCTFile (OCTFileHandle), obtained with the function createOCTFile.
in	*Index*	Index of the data inside the OCT file, e.g. returned by findFileDataObject.

Returns

Pixel count in X of the data object or 0 in case of an error

5.23.4.26 int getFileDataSizeY (OCTFileHandle *Handle,* size_t *Index*)

Returns the pixel count in Y of the data object at the given *Index* in the OCT file.

Parameters

in	*Handle*	A valid (non null) handle of OCTFile (OCTFileHandle), obtained with the function createOCTFile.
in	*Index*	Index of the data inside the OCT file, e.g. returned by findFileDataObject.

Returns

Pixel count in Y of the data object or 0 in case of an error

5.23.4.27 int getFileDataSizeZ (OCTFileHandle *Handle*, size_t *Index*)

Returns the pixel count in Z of the data object at the given *Index* in the OCT file.

Parameters

in	*Handle*	A valid (non null) handle of OCTFile (OCTFileHandle), obtained with the function createOCTFile.
in	*Index*	Index of the data inside the OCT file, e.g. returned by findFileDataObject.

Returns

Pixel count in Z of the data object or 0 in case of an error

5.23.4.28 BOOL getFileMetadataFlag (OCTFileHandle *Handle*, FileMetadataFlag *Boolfield*)

Gets the boolean value of the given file metadata field.

Parameters

in	*Handle*	A valid (non null) handle of OCTFile (OCTFileHandle), obtained with the function createOCTFile.
in	*Boolfield*	Metadata field to read.

5.23.4.29 double getFileMetadataFloat (OCTFileHandle *Handle*, FileMetadataFloat *Floatfield*)

Returns the value of the given file metadata field as a floating point number if found.

Parameters

in	*Handle*	A valid (non null) handle of OCTFile (OCTFileHandle), obtained with the function createOCTFile.
in	*Floatfield*	Metadata field to read.

5.23.4.30 int getFileMetadataInt (OCTFileHandle *Handle*, FileMetadataInt *Intfield*)

Returns the value of the given file metadata field as an integer if found.

Parameters

in	*Handle*	A valid (non null) handle of OCTFile (OCTFileHandle), obtained with the function createOCTFile.
in	*Intfield*	Metadata field to read.

5.23.4.31 int getFileMetadataNumberOfPresets (OCTFileHandle *Handle*)

Gets the number of presets that were set during the acquisition.

Parameters

in	*Handle*	A valid (non null) handle of OCTFile (OCTFileHandle), obtained with the function createOCTFile.

5.23.4.32 const char * getFileMetadataPresetCategory (OCTFileHandle *Handle,* int *Index*)

Gets the preset category belonging to the preset with given *Index*.

Parameters

in	*Handle*	A valid (non null) handle of OCTFile (OCTFileHandle), obtained with the function createOCTFile.
in	*Index*	Index of the preset inside the OCT file, 0 <= Index < getFileMetadataNumberOfPresets

5.23.4.33 const char * getFileMetadataPresetDescription (OCTFileHandle *Handle,* int *Index*)

Gets the preset description belonging to the preset with given *Index*.

Parameters

in	*Handle*	A valid (non null) handle of OCTFile (OCTFileHandle), obtained with the function createOCTFile.
in	*Index*	Index of the preset inside the OCT file, 0 <= Index < getFileMetadataNumberOfPresets

5.23.4.34 const char * getFileMetadataString (OCTFileHandle *Handle,* FileMetadataString *Stringfield*)

Returns the value of the given file metadata field as a string if found.

Parameters

in	*Handle*	A valid (non null) handle of OCTFile (OCTFileHandle), obtained with the function createOCTFile.
in	*Stringfield*	Metadata field to read.

5.23.4.35 void getFileRawData (OCTFileHandle *Handle,* RawDataHandle *Data,* size_t *Index*)

Retrieves a RawData object from the OCT file at the given index with 0 <= index < getFileDataObjectCount(OCTFileHandle handle). Users must ensure that the data handle is properly prepared and destroyed.

Parameters

in	*Handle*	A valid (non null) handle of OCTFile (OCTFileHandle), obtained with the function createOCTFile.
out	*Data*	A valid (non null) raw data handle of the existing data (RawDataHandle), previously obtained with the function createRawData. It will be filled in with the data read from the OCT file at the given *Index*.
in	*Index*	Index of the data inside the OCT file, e.g. returned by findFileDataObject. Notice that raw data refers to the spectra as acquired, without processing of any kind.

5.23.4.36 void getFileRealData (OCTFileHandle *Handle*, DataHandle *Data*, int *Index*)

Retrieves a RealData object from the OCT file at the given index with 0 <= index < getFileDataObjectCount(OC↵ TFileHandle handle). Users must ensure that the data handle is properly prepared and destroyed.

Parameters

in	*Handle*	A valid (non null) handle of OCTFile (OCTFileHandle), obtained with the function createOCTFile.
out	*Data*	A valid (non null) data handle of the existing data (DataHandle), previously obtained with the function createData. It will be filled in with the data read from the OCT file at the given *Index*.
in	*Index*	Index of the data inside the OCT file, e.g. returned by findFileDataObject.

5.23.4.37 void loadCalibrationFromFile (OCTFileHandle *Handle*, ProcessingHandle *Proc*)

Loads Chirp, Offset, and Apodization vectors from the given OCT file into the given processing object.

Parameters

in	*Handle*	A valid (non null) handle of OCTFile (OCTFileHandle), obtained with the function createOCTFile.
out	*Proc*	A valid (non null) handle of the processing routines (ProcessingHandle), previously obtained through one of the functions createProcessing, createProcessingForDevice, createProcessingForDeviceEx or createProcessingForOCTFile.

5.23.4.38 void loadCalibrationFromFileEx (OCTFileHandle *Handle*, ProcessingHandle *Proc*, const int *CameraIndex*)

Loads Chirp, Offset, and Apodization vectors from the given OCT file into the given processing object.

Parameters

in	*Handle*	A valid (non null) handle of OCTFile (OCTFileHandle), obtained with the function createOCTFile.
in	*Proc*	A valid (non null) handle of the processing routines (ProcessingHandle), previously obtained through one of the functions createProcessing, createProcessingForDevice, createProcessingForDeviceEx or createProcessingForOCTFile.
in	*CameraIndex*	The camera index (0-based, i.e. zero for the first, one for the second, and so on).

5.23.4.39 void loadFile (OCTFileHandle *Handle*, const char ∗ *Filename*)

Loads the actual OCT data file from a file system. The file must have the format given in createOCTFile().

Parameters

in	*Handle*	A valid (non null) handle of OCTFile (OCTFileHandle), obtained with the function createOCTFile.
in	*Filename*	Name of the data file to load.

5.23.4.40 void saveCalibrationToFile (OCTFileHandle *Handle,* ProcessingHandle *Proc*)

Saves Chirp, Offset, and Apodization vectors from the given processing object into the given OCT file.

Parameters

in	*Handle*	A valid (non null) handle of OCTFile (OCTFileHandle), obtained with the function createOCTFile.
in	*Proc*	A valid (non null) handle of the processing routines (ProcessingHandle), previously obtained through one of the functions createProcessing, createProcessingForDevice, createProcessingForDeviceEx or createProcessingForOCTFile.

5.23.4.41 void saveCalibrationToFileEx (OCTFileHandle *Handle,* ProcessingHandle *Proc,* int *CameraIndex*)

Saves Chirp, Offset, and Apodization vectors from the given processing object into the given OCT file.

Parameters

in	*Handle*	A valid (non null) handle of OCTFile (OCTFileHandle), obtained with the function createOCTFile.
in	*Proc*	A valid (non null) handle of the processing routines (ProcessingHandle), previously obtained through one of the functions createProcessing, createProcessingForDevice, createProcessingForDeviceEx or createProcessingForOCTFile.
in	*CameraIndex*	The camera index (0-based, i.e. zero for the first, one for the second, and so on).

5.23.4.42 void saveChangesToFile (OCTFileHandle *Handle*)

Saves the OCT data file in the file previously opened with loadFile(). Only changes will be saved.

Parameters

in	*Handle*	A valid (non null) handle of OCTFile (OCTFileHandle), obtained with the function createOCTFile.

5.23.4.43 void saveFile (OCTFileHandle *Handle,* const char ∗ *Filename*)

Saves the OCT data file in the given fully qualified path name.

Parameters

in	*Handle*	A valid (non null) handle of OCTFile (OCTFileHandle), obtained with the function createOCTFile.
in	*Filename*	Name to which the OCT data file will be written.

5.23.4.44 void saveFileMetadata (OCTFileHandle *Handle,* OCTDeviceHandle *Dev,* ProcessingHandle *Proc,* ProbeHandle *Probe,* ScanPatternHandle *Pattern*)

Saves meta information from the given device, processing, probe and scan pattern instances in the metadata block of the given file handle. This information will be available in files of type FileFormat_OCITY; mileage on other formats may vary according to their description.

Parameters

in	*Handle*	A valid (non null) handle of OCTFile (OCTFileHandle), obtained with the function createOCTFile.
in	*Dev*	A valid (non null) OCT device handle (OCTDeviceHandle), previously generated with the function initDevice.
in	*Proc*	A valid (non null) handle of the processing routines (ProcessingHandle), previously obtained through one of the functions createProcessing, createProcessingForDevice, createProcessingForDeviceEx or createProcessingForOCTFile.
in	*Probe*	A valid (non null) handle of an initialized probem, obtained through initProbe.
in	*Pattern*	A valid (non null) handle of a scan pattern.

5.23.4.45 void saveFileMetadataDoppler (OCTFileHandle *Handle*, DopplerProcessingHandle *DopplerProc*)

Saves meta information from the given DopplerProcessingHandle. A corresponding DopplerProcessingHandle can then be recreated using createDopplerProcessingForFile.

Parameters

in	*Handle*	A valid (non null) handle of OCTFile (OCTFileHandle), obtained with the function createOCTFile. This describes the files then handle data is stored to.
in	*DopplerProc*	A valid (non null) handle of Doppler processing obtained by createDopplerProcessing. This is the handle whose data is stored.

5.23.4.46 void saveFileMetadataSpeckle (OCTFileHandle *Handle*, SpeckleVarianceHandle *SpeckleVarianceProc*)

Saves meta information from the given SpeckleVarianceHandle. A corresponding SpeckleVarianceHandle can then be recreated using initSpeckleVarianceForFile.

Parameters

in	*Handle*	A valid (non null) handle of OCTFile (OCTFileHandle), obtained with the function createOCTFile. This describes the files then handle data is stored to.
in	*SpeckleVarianceProc*	A valid (non null) handle of speckle variance processing obtained by initSpeckleVariance. This is the handle whose data is stored.

5.23.4.47 void setFileMetadataFlag (OCTFileHandle *Handle*, FileMetadataFlag *Boolfield*, BOOL *Value*)

Sets the boolean value of the given file metadata field.

Parameters

in	*Handle*	A valid (non null) handle of OCTFile (OCTFileHandle), obtained with the function createOCTFile.
in	*Boolfield*	Metadata field to set.
in	*Value*	Boolean value to set on the field.

5.23.4.48 void setFileMetadataFloat (OCTFileHandle *Handle*, FileMetadataFloat *Floatfield*, double *Value*)

Sets the value of the given file metadata field as a floating point number.

Parameters

in	*Handle*	A valid (non null) handle of OCTFile (OCTFileHandle), obtained with the function createOCTFile.
in	*Floatfield*	Metadata field to set.
in	*Value*	Double value to set on the field.

5.23.4.49 void setFileMetadataInt (**OCTFileHandle** *Handle,* **FileMetadataInt** *Intfield,* int *Value*)

Sets the value of the given file metadata field as an integer.

Parameters

in	*Handle*	A valid (non null) handle of OCTFile (OCTFileHandle), obtained with the function createOCTFile.
in	*Intfield*	Metadata field to set.
in	*Value*	int value to set on the field.

5.23.4.50 void setFileMetadataString (**OCTFileHandle** *Handle,* **FileMetadataString** *Stringfield,* const char ∗ *Content*)

Sets the value of the given file metadata field as a string.

Parameters

in	*Handle*	A valid (non null) handle of OCTFile (OCTFileHandle), obtained with the function createOCTFile.
in	*Stringfield*	Metadata field to set.
in	*Content*	String value to set on the field.

5.24 External trigger

Functions to inquire, setup, and deal with an external trigger. Whether this functionality is supported, and to what extent, depends on the hardware.

Functions

- SPECTRALRADAR_API void setTriggerMode (OCTDeviceHandle Dev, DeviceTriggerType TriggerMode)

  Sets the trigger mode for the OCT device used for acquisition. Additional hardware may be needed.
- SPECTRALRADAR_API DeviceTriggerType getTriggerMode (OCTDeviceHandle Dev)

  Returns the trigger mode used for acquisition.
- SPECTRALRADAR_API BOOL isTriggerModeAvailable (OCTDeviceHandle Dev, DeviceTriggerType TriggerMode)

  Returns whether the specified trigger mode is possible or not for the used device.
- SPECTRALRADAR_API void setTriggerTimeout_s (OCTDeviceHandle Dev, int Timeout_s)

  Sets the timeout of the camera in seconds (useful in external trigger mode).
- SPECTRALRADAR_API int getTriggerTimeout_s (OCTDeviceHandle Dev)

  Returns the timeout of the camera in seconds (not used in trigger mode Trigger_FreeRunning).

5.24.1 Detailed Description

Functions to inquire, setup, and deal with an external trigger. Whether this functionality is supported, and to what extent, depends on the hardware.

5.24.2 Function Documentation

5.24.2.1 DeviceTriggerType getTriggerMode (OCTDeviceHandle *Dev*)

Returns the trigger mode used for acquisition.

5.24.2.2 int getTriggerTimeout_s (OCTDeviceHandle *Dev*)

Returns the timeout of the camera in seconds (not used in trigger mode Trigger_FreeRunning).

5.24.2.3 BOOL isTriggerModeAvailable (OCTDeviceHandle *Dev,* DeviceTriggerType *TriggerMode*)

Returns whether the specified trigger mode is possible or not for the used device.

5.24.2.4 void setTriggerMode (OCTDeviceHandle *Dev,* DeviceTriggerType *TriggerMode*)

Sets the trigger mode for the OCT device used for acquisition. Additional hardware may be needed.

5.24.2.5 void setTriggerTimeout_s (OCTDeviceHandle *Dev,* int *Timeout_s*)

Sets the timeout of the camera in seconds (useful in external trigger mode).

5.25 Post Processing

Algorithms and functions used for post processing of floating point data.

Enumerations

- enum PepperFilterType {
  PepperFilter_Horizontal,
  PepperFilter_Vertical,
  PepperFilter_Star,
  PepperFilter_Block }

  Specifies the type of pepper filter to be applied.
- enum ComplexFilterType2D { FilterComplex2D_PhaseContrast }

  Specifies the type of filter to be applied to complex data.
- enum FilterType1D { Filter1D_Gaussian_5 }

  Specifies the type of 1D-filter to be applied. All filters are normalized.
- enum FilterType2D {
  Filter2D_Gaussian_3x3,
  Filter2D_Gaussian_5x5,
  Filter2D_Prewitt_Horizontal_3x3,
  Filter2D_Prewitt_Vertical_3x3,
  Filter2D_NonlinearPrewitt_3x3,
  Filter2D_Sobel_Horizontal_3x3,
  Filter2D_Sobel_Vertical_3x3,
  Filter2D_NonlinearSobel_3x3,
  Filter2D_Laplacian_NoDiagonal_3x3,
  Filter2D_Laplacian_3x3 }

  Specifies the type of 2D-filter to be applied. All filters are normalized.
- enum FilterType3D { Filter3D_Gaussian_3x3x3 }

  Specifies the type of 3D-filter to be applied. All filters are normalized.

Functions

- SPECTRALRADAR_API void determineDynamicRange_dB (DataHandle Data, double ∗MinRange_dB, double ∗MaxRange_dB)

  Gives a rough estimation of the dynamic range of the specified data object.
- SPECTRALRADAR_API void determineDynamicRangeWithMinRange_dB (DataHandle Data, double ∗MinRange_dB, double ∗MaxRange_dB, double MinDynamicRange_dB)

  Gives a rough estimation of the dynamic range of the specified data object.
- SPECTRALRADAR_API void medianFilter1D (DataHandle Data, int Rank, Direction FilterDirection)

  Computes a 1D-median filter on the specified data.
- SPECTRALRADAR_API void medianFilter2D (DataHandle Data, int Rank, Direction FilterNormalDirection)

  Computes a 2D-median filter on the specified 2D data.
- SPECTRALRADAR_API void pepperFilter2D (DataHandle Data, PepperFilterType Type, float Threshold, Direction FilterNormalDirection)

  Removes pepper-noise (very low values, i. e. dark spots in the data). This enhances the visual (colored) representation of the data.
- SPECTRALRADAR_API void convolutionFilter1D (DataHandle Data, int FilterSize, float ∗FilterKernel, Direction FilterDirection)

  Calculates a mathematical convolution of the Data and the 1D-FilterKernel.
- SPECTRALRADAR_API void convolutionFilter2D (DataHandle Data, int FilterSize1, int FilterSize2, float ∗FilterKernel, Direction FilterNormalDirection)

Calculates a mathematical convolution of the Data and the 2D-FilterKernel.

- SPECTRALRADAR_API void convolutionFilter3D (DataHandle Data, int FilterSize1, int FilterSize2, int FilterSize3, float ∗FilterKernel)

  Calculates a mathematical convolution of the Data and the 3D-FilterKernel.
- SPECTRALRADAR_API void predefinedFilter1D (DataHandle Data, FilterType1D Filter, Direction FilterDirection)

  Applies the predefined 1D-Filter to the Data.
- SPECTRALRADAR_API void predefinedFilter2D (DataHandle Data, FilterType2D Filter, Direction FilterNormalDirection)

  Applies the predefined 2D-Filter to the Data.
- SPECTRALRADAR_API void predefinedFilter3D (DataHandle Data, FilterType3D FilterType)

  Applies the predefined 3D-Filter to the Data.
- SPECTRALRADAR_API void predefinedComplexFilter2D (ComplexDataHandle ComplexData, ComplexFilterType2D Type, Direction FilterNormalDirection)

  Applies the predefined 2D-Filter to the ComplexData.
- SPECTRALRADAR_API void darkFieldComplexFilter2D (ComplexDataHandle ComplexData, double Radius, Direction FilterNormalDirection)

  Filters the image such that the image contrast comes from light scattered by the sample.
- SPECTRALRADAR_API void brightFieldComplexFilter2D (ComplexDataHandle ComplexData, double Radius, Direction FilterNormalDirection)

  Filters the image such that the image contrast comes from absorbance of light in the sample.

5.25.1 Detailed Description

Algorithms and functions used for post processing of floating point data.

5.25.2 Enumeration Type Documentation

5.25.2.1 enum ComplexFilterType2D

Specifies the type of filter to be applied to complex data.

Enumerator

FilterComplex2D_PhaseContrast A filter applied to complex data to get a phase contrast image.

5.25.2.2 enum FilterType1D

Specifies the type of 1D-filter to be applied. All filters are normalized.

Enumerator

Filter1D_Gaussian_5 A gaussian 1D-filter of size 5 to smooth the data.

5.25.2.3 enum FilterType2D

Specifies the type of 2D-filter to be applied. All filters are normalized.

Enumerator

Filter2D_Gaussian_3x3 A gaussian filter of size 3x3 to smooth the data.

Filter2D_Gaussian_5x5 A gaussian filter of size 5x5 to smooth the data.

Filter2D_Prewitt_Horizontal_3x3 Horizontal prewitt filter of size 3x3 to detect edges in horizontal direction.

Filter2D_Prewitt_Vertical_3x3 Vertical prewitt filter of size 3x3 to detect edges in vertical direction.

Filter2D_NonlinearPrewitt_3x3 Maximum of horizontal and vertical prewitt filter each of size 3x3 to detect edges.

Filter2D_Sobel_Horizontal_3x3 Horizontal sobel filter of size 3x3 to detect edges in horizontal direction while smoothing in vertical direction.

Filter2D_Sobel_Vertical_3x3 Vertical prewitt filter of size 3x3 to detect edges in vertical direction while smoothing in horizontal direction.

Filter2D_NonlinearSobel_3x3 Maximum of horizontal and vertical sobel filter each of size 3x3 to detect edges while smoothing the data simultaneously.

Filter2D_Laplacian_NoDiagonal_3x3 Laplacian filter of size 3x3 to detect horizontal and vertical edges, no diagonal egdes.

Filter2D_Laplacian_3x3 Laplacian filter of size 3x3 to detect horizontal, vertical and diagonal edges.

5.25.2.4 enum FilterType3D

Specifies the type of 3D-filter to be applied. All filters are normalized.

Enumerator

Filter3D_Gaussian_3x3x3 A gaussian filter of size 3x3 to smooth the data.

5.25.2.5 enum PepperFilterType

Specifies the type of pepper filter to be applied.

Enumerator

PepperFilter_Horizontal Values along the horizontal axis are taken into account for the pepper filter.

PepperFilter_Vertical Values along the vertical axis are taken into account for the pepper filter.

PepperFilter_Star Values along the vertical and horizontal axis (star shape) are taken into account for the pepper filter.

PepperFilter_Block Values in a block surrounding the destination pixel are taken into account.

5.25.3 Function Documentation

5.25.3.1 void brightFieldComplexFilter2D (ComplexDataHandle *ComplexData,* double *Radius,* Direction *FilterNormalDirection*)

Filters the image such that the image contrast comes from absorbance of light in the sample.

Parameters

ComplexData	The ComplexDataHandle the filter will be applied to.
Radius	Parameter to adjust the image contrast.
FilterNormalDirection	The normal of the direction the 2D-filter will be applied to the complex data, e.g. Direction_3 for filtering each single B-scan

5.25.3.2 void convolutionFilter1D (**DataHandle** *Data*, int *Size*, float ∗ *FilterKernel*, **Direction** *FilterDirection*)

Calculates a mathematical convolution of the Data and the 1D-FilterKernel.

Parameters

Data	The DataHandle the filter will be applied to
Size	Size of the filter
FilterKernel	Pointer to the array containing the filter kernel
FilterDirection	The filter direction the 1D-filter will be applied to the data, e.g. Direction_1 for filtering each single A-scan

5.25.3.3 void convolutionFilter2D (**DataHandle** *Data*, int *FilterSize1*, int *FilterSize2*, float ∗ *FilterKernel*, **Direction** *FilterNormalDirection*)

Calculates a mathematical convolution of the Data and the 2D-FilterKernel.

Parameters

Data	The DataHandle the filter will be applied to
FilterSize1	Size of the first dimension of the filter
FilterSize2	Size of the second dimension of the filter
FilterKernel	Pointer to the array containing the filter kernel
FilterNormalDirection	The normal of the direction the 2D-filter will be applied to the data, e.g. Direction_3 for filtering each single B-scan

5.25.3.4 void convolutionFilter3D (**DataHandle** *Data*, int *FilterSize1*, int *FilterSize2*, int *FilterSize3*, float ∗ *FilterKernel*)

Calculates a mathematical convolution of the Data and the 3D-FilterKernel.

Parameters

Data	The DataHandle the filter will be applied to
FilterSize1	Size of the first dimension of the filter
FilterSize2	Size of the second dimension of the filter
FilterSize3	Size of the third dimension of the filter
FilterKernel	Pointer to the array containing the filter kernel

5.25.3.5 void darkFieldComplexFilter2D (**ComplexDataHandle** *ComplexData*, double *Radius*, **Direction** *FilterNormalDirection*)

Filters the image such that the image contrast comes from light scattered by the sample.

Parameters

ComplexData	The ComplexDataHandle the filter will be applied to.
Radius	Parameter to adjust the image contrast.
FilterNormalDirection	The normal of the direction the 2D-filter will be applied to the complex data, e.g. Direction_3 for filtering each single B-scan

5.25.3.6 void determineDynamicRange_dB (DataHandle *Data,* double ∗ *MinRange_dB,* double ∗ *MaxRange_dB*)

Gives a rough estimation of the dynamic range of the specified data object.

This functions assumes that the data contains an A-scan and performs A-scan specific analysis on it.

Parameters

Data	The DataHandle the filter will be applied to
MinRange_dB	Used to return the lower bound of the dynamic range
MaxRange_dB	Used to return the upper bound of the dynamic range

5.25.3.7 void determineDynamicRangeWithMinRange_dB (DataHandle *Data,* double ∗ *MinRange_dB,* double ∗ *MaxRange_dB,* double *MinDynamicRange_dB*)

Gives a rough estimation of the dynamic range of the specified data object.

Parameters

Data	The DataHandle the filter will be applied to
MinRange_dB	Used to return the lower bound of the dynamic range
MaxRange_dB	Used to return the upper bound of the dynamic range
MinDynamicRange_dB	Minimal size of the returned dynamic range interval in dB

5.25.3.8 void medianFilter1D (DataHandle *Data,* int *Rank,* Direction *FilterDirection*)

Computes a 1D-median filter on the specified data.

Parameters

Data	The DataHandle the filter will be applied to
Rank	The size of the filter
FilterDirection	The direction the 1D-filter will be applied to the data.

5.25.3.9 void medianFilter2D (DataHandle *Data,* int *Rank,* Direction *FilterNormalDirection*)

Computes a 2D-median filter on the specified 2D data.

Parameters

Data	The DataHandle the filter will be applied to

Parameters

Rank	The size of the filter
FilterNormalDirection	The normal of the direction the 2D-filter will be applied to the data.

5.25.3.10 void pepperFilter2D (DataHandle *Data*, PepperFilterType *Type*, float *Threshold*, Direction *FilterNormalDirection*)

Removes pepper-noise (very low values, i. e. dark spots in the data). This enhances the visual (colored) representation of the data.

Parameters

Data	The DataHandle the filter will be applied to
Type	The type of the pepper filter chosen from PepperFilterType
Threshold	If the value is lower than the given value it will be replaced by the mean
FilterNormalDirection	The normal of the direction the 2D-filter will be applied to the data

The pepper filter compares all pixels to a mean of surrounding pixels. The surrounding pixels taking into account are specified by PepperFilterType. If the pixels is lower than specified by the Threshold the pixel will be replaced by the mean.

5.25.3.11 void predefinedComplexFilter2D (ComplexDataHandle *ComplexData*, ComplexFilterType2D *Type*, Direction *FilterNormalDirection*)

Applies the predefined 2D-Filter to the ComplexData.

Parameters

ComplexData	The ComplexDataHandle the filter will be applied to
Type	Chosen predefined filter for complex data. See ComplexFilterType2D for selection.
FilterNormalDirection	The normal of the direction the 2D-filter will be applied to the complex data, e.g. Direction_3 for filtering each single B-scan

5.25.3.12 void predefinedFilter1D (DataHandle *Data*, FilterType1D *Filter*, Direction *FilterDirection*)

Applies the predefined 1D-Filter to the Data.

Parameters

Data	The DataHandle the filter will be applied to
Filter	Selection of a predefined filter FilterType1D
FilterDirection	The filter direction the 1D-filter will be applied to the data, e.g. Direction_1 for filtering each single A-scan

5.25.3.13 void predefinedFilter2D (DataHandle *Data*, FilterType2D *Filter*, Direction *FilterNormalDirection*)

Applies the predefined 2D-Filter to the Data.

Parameters

Data	The DataHandle the filter will be applied to
Filter	Selection of a predefined filter FilterType2D
FilterNormalDirection	The normal of the direction the 2D-filter will be applied to the data, e.g. Direction_3 for filtering each single B-scan

5.25.3.14 void predefinedFilter3D (DataHandle *Data,* FilterType3D *FilterType*)

Applies the predefined 3D-Filter to the Data.

Parameters

Data	The DataHandle the filter will be applied to
FilterType	Selection of a predefined filter FilterType3D

5.26 Polarization

Polarization Sensitive OCT Processing Routines.

Typedefs

- typedef struct C_PolarizationProcessing ∗ PolarizationProcessingHandle

  Handle used for Polarization processing.

Enumerations

- enum PolarizationDOPUFilterType {
  PolarizationProcessing_DOPU_Median,
  PolarizationProcessing_DOPU_Average,
  PolarizationProcessing_DOPU_Gaussian,
  PolarizationProcessing_DOPU_GaussianWithFFT }

  Values that determine the behaviour of temporal filter, if enabled.
- enum PolarizationPropertyInt {
  PolarizationProcessing_DOPU_Z = 0,
  PolarizationProcessing_DOPU_X = 1,
  PolarizationProcessing_DOPU_Y = 2,
  PolarizationProcessing_DOPU_FilterType = 3,
  PolarizationProcessing_BScanAveraging = 4,
  PolarizationProcessing_AveragingZ = 5,
  PolarizationProcessing_AveragingX = 6,
  PolarizationProcessing_AveragingY = 7,
  PolarizationProcessing_AScanAveraging = 8 }

  Values that determine the behaviour of the Polarization processing routines.
- enum PolarizationPropertyFloat {
  PolarizationProcessing_IntensityThreshold_dB = 0,
  PolarizationProcessing_PMDCorrectionAngle_rad = 1,
  PolarizationProcessing_CentralWavelength_nm = 2,
  PolarizationProcessing_OpticalAxisOffset_rad = 3 }

  Values that determine the behaviour of the Polarization processing routines.
- enum PolarizationRetarder {
  Retarder_Quarter_Wave = 0,
  Retarder_Half_Wave = 1 }

  List of available polarization retarders in a polarization control unit.

Functions

- SPECTRALRADAR_API PolarizationProcessingHandle createPolarizationProcessing (void)

  Returns a Polarization processing handle to the Processing routines for polarization analysis.
- SPECTRALRADAR_API void clearPolarizationProcessing (PolarizationProcessingHandle Polarization)

  Clears the polarization processing routines and frees the memory that has been allocated for these to work properly.
- SPECTRALRADAR_API int getPolarizationPropertyInt (PolarizationProcessingHandle Polarization, PolarizationPropertyInt Property)

  Gets the desired polarization processing property.
- SPECTRALRADAR_API void setPolarizationPropertyInt (PolarizationProcessingHandle Polarization, PolarizationPropertyInt Property, int Value)

  Sets polarization processing properties.

- SPECTRALRADAR_API double getPolarizationPropertyFloat (PolarizationProcessingHandle Polarization, PolarizationPropertyFloat Property)

  Gets the desired polarization processing floating-point property.
- SPECTRALRADAR_API void setPolarizationPropertyFloat (PolarizationProcessingHandle Polarization, PolarizationPropertyFloat Property, double Value)

  Sets the desired polarization processing floating-point property.
- SPECTRALRADAR_API void setPolarizationOutputI (PolarizationProcessingHandle Polarization, Data↩ Handle Intensity)

  Sets the location of the resulting polarization intensity output (Stokes parameter I).
- SPECTRALRADAR_API void setPolarizationOutputQ (PolarizationProcessingHandle Polarization, Data↩ Handle StokesQ)

  Sets the location of the resulting Stokes parameter Q.
- SPECTRALRADAR_API void setPolarizationOutputU (PolarizationProcessingHandle Polarization, Data↩ Handle StokesU)

  Sets the location of the resulting Stokes parameter U.
- SPECTRALRADAR_API void setPolarizationOutputV (PolarizationProcessingHandle Polarization, Data↩ Handle StokesV)

  Sets the location of the resulting Stokes parameter U.
- SPECTRALRADAR_API void setPolarizationOutputDOPU (PolarizationProcessingHandle Polarization, DataHandle DOPU)

  Sets the location of the resulting DOPU.
- SPECTRALRADAR_API void setPolarizationOutputRetardation (PolarizationProcessingHandle Polarization, DataHandle Retardation)

  Sets the location of the resulting retardation.
- SPECTRALRADAR_API void setPolarizationOutputOpticAxis (PolarizationProcessingHandle Polarization, DataHandle OpticAxis)

  Sets the location of the resulting optic axis.
- SPECTRALRADAR_API void executePolarizationProcessing (PolarizationProcessingHandle Polarization, ComplexDataHandle Data_P_Camera1, ComplexDataHandle PData_S_Camera0)

  Executes the polarization processing of the input data and returns, if previously setup, intensity, retardation, and phase differences.
- SPECTRALRADAR_API void saveFileMetadataPolarization (OCTFileHandle FileHandle, Polarization↩ ProcessingHandle PolProc)

  Saves metadata to the specified file. These metadata specify the operational arguments needed by the polarization processing routines to redo the polarization-analysis starting from two ComplexDataHandle delivered by `Proc_0` and `Proc_1`.
- SPECTRALRADAR_API PolarizationProcessingHandle createPolarizationProcessingForFile (OCTFile↩ Handle FileHandle)

  Loads metadata to the specified file. These metadata specify the operational arguments needed by the polarization processing routines to redo the polarization-analysis starting from two ComplexDataHandle delivered by `Proc_0` and `Proc_1`, exactly as they were done before the file was written.

5.26.1 Detailed Description

Polarization Sensitive OCT Processing Routines.

This section deals with polarization sensitive OCT (PS-OCT).

5.26.2 Typedef Documentation

5.26.2.1 PolarizationProcessingHandle

Handle used for Polarization processing.

5.26.3 Enumeration Type Documentation

5.26.3.1 enum PolarizationDOPUFilterType

Values that determine the behaviour of temporal filter, if enabled.

Enumerator

- ***PolarizationProcessing_DOPU_Median*** Median.
- ***PolarizationProcessing_DOPU_Average*** Average.
- ***PolarizationProcessing_DOPU_Gaussian*** Convolution with a Gaussian kernel.
- ***PolarizationProcessing_DOPU_GaussianWithFFT*** FFT convolution with a Gaussian kernel (preumably more efficient for very large kernels).

5.26.3.2 enum PolarizationPropertyFloat

Values that determine the behaviour of the Polarization processing routines.

Enumerator

- ***PolarizationProcessing_IntensityThreshold_dB*** Threshold value to enable/disable features of PS computation based on the total intensity value.
- ***PolarizationProcessing_PMDCorrectionAngle_rad*** Correction angle (in radians) to get circularly polarized light at the upper surface of the sample. This angle is a compensation fo the polarization-mode-dispersion (PMD). More in detail, this angle is used to compute a phasor ($\exp(i\alpha)$), that will be applied to the complex reflectivities vector associated with camera 0.
- ***PolarizationProcessing_CentralWavelength_nm*** Assuming a gaussian light source, the value of the wavenumber with maximal intensity (in nm).
- ***PolarizationProcessing_OpticalAxisOffset_rad*** Refer to a particular orientation on the sample holder. The angle should be expressed in radians.

5.26.3.3 enum PolarizationPropertyInt

Values that determine the behaviour of the Polarization processing routines.

Enumerator

- ***PolarizationProcessing_DOPU_Z*** Number of pixels for DOPU averaging in the z-direction.
- ***PolarizationProcessing_DOPU_X*** Number of pixels for DOPU averaging in the z-direction.
- ***PolarizationProcessing_DOPU_Y*** Number of pixels for DOPU averaging in the y-direction.
- ***PolarizationProcessing_DOPU_FilterType*** DOPU filter specification. See PolarizationDOPUFilterType.
- ***PolarizationProcessing_BScanAveraging*** Number of frames for averaging.
- ***PolarizationProcessing_AveragingZ*** Number of pixels for averaging along the x axis.
- ***PolarizationProcessing_AveragingX*** Number of pixels for averaging along the y axis.
- ***PolarizationProcessing_AveragingY*** Number of pixels for averaging along the z axis.
- ***PolarizationProcessing_AScanAveraging*** A-Scan averaging. This parameter influences the way data get acquired, it cannot be changed for offline processing.

5.26.3.4 enum **PolarizationRetarder**

List of available polarization retarders in a polarization control unit.

5.26.4 Function Documentation

5.26.4.1 void clearPolarizationProcessing (**PolarizationProcessingHandle** *Polarization*)

Clears the polarization processing routines and frees the memory that has been allocated for these to work properly.

5.26.4.2 **PolarizationProcessingHandle** createPolarizationProcessing (void)

Returns a Polarization processing handle to the Processing routines for polarization analysis.

5.26.4.3 **PolarizationProcessingHandle** createPolarizationProcessingForFile (**OCTFileHandle** *FileHandle*)

Loads metadata to the specified file. These metadata specify the operational arguments needed by the polarization processing routines to redo the polarization-analysis starting from two ComplexDataHandle delivered by `Proc_0` and `Proc_1`, exactly as they were done before the file was written.

Parameters

in	*FileHandle*	A valid (non null) handle of OCTFile (OCTFileHandle), previously obtained with the function createOCTFile.

Returns

A valid (non null) polarization-processing handle to the processing routines for polarization analysis.

5.26.4.4 void executePolarizationProcessing (**PolarizationProcessingHandle** *Polarization,* **ComplexDataHandle** *Data_P_Camera1,* **ComplexDataHandle** *PData_S_Camera0*)

Executes the polarization processing of the input data and returns, if previously setup, intensity, retardation, and phase differences.

5.26.4.5 double getPolarizationPropertyFloat (**PolarizationProcessingHandle** *Polarization,* **PolarizationPropertyFloat** *Property*)

Gets the desired polarization processing floating-point property.

5.26.4.6 int getPolarizationPropertyInt (**PolarizationProcessingHandle** *Polarization,* **PolarizationPropertyInt** *Property*)

Gets the desired polarization processing property.

5.26.4.7 void saveFileMetadataPolarization (**OCTFileHandle** *FileHandle,* **PolarizationProcessingHandle** *PolProc*)

Saves metadata to the specified file. These metadata specify the operational arguments needed by the polarization processing routines to redo the polarization-analysis starting from two ComplexDataHandle delivered by `Proc_0` and `Proc_1`.

Parameters

in	*FileHandle*	A valid (non null) handle of OCTFile (OCTFileHandle), previously obtained with the function createOCTFile.
in	*PolProc*	A valid (non null) polarization-processing handle to the processing routines for polarization analysis.

5.26.4.8 void setPolarizationOutputDOPU (**PolarizationProcessingHandle** *Polarization,* **DataHandle** *DOPU*)

Sets the location of the resulting DOPU.

The range of the DOPU is [0,1]. It takes the value when light appearsto be completely unpolarized, and 1 when the opposite is the case.

5.26.4.9 void setPolarizationOutputI (**PolarizationProcessingHandle** *Polarization,* **DataHandle** *Intensity*)

Sets the location of the resulting polarization intensity output (Stokes parameter I).

5.26.4.10 void setPolarizationOutputOpticAxis (**PolarizationProcessingHandle** *Polarization,* **DataHandle** *OpticAxis*)

Sets the location of the resulting optic axis.

The range of the optic axil is [-pi/2,pi/2].

5.26.4.11 void setPolarizationOutputQ (**PolarizationProcessingHandle** *Polarization,* **DataHandle** *StokesQ*)

Sets the location of the resulting Stokes parameter Q.

The range of Q is [-1,1]. It takes the value -1 when the polarization is 100% parallel (zero degrees), and the value 1 when the polarization is 100% perpendicular (ninety degrees).

5.26.4.12 void setPolarizationOutputRetardation (**PolarizationProcessingHandle** *Polarization,* **DataHandle** *Retardation*)

Sets the location of the resulting retardation.

The range of the retardation is [0,pi/2].

5.26.4.13 void setPolarizationOutputU (**PolarizationProcessingHandle** *Polarization,* **DataHandle** *StokesU*)

Sets the location of the resulting Stokes parameter U.

The range of U is [-1,1]. It takes the value -1 when the polarization is 100% at -45 degrees, and the value 1 when the polarization is 100% at 45 degrees.

5.26.4.14 void setPolarizationOutputV (**PolarizationProcessingHandle** *Polarization,* **DataHandle** *StokesV*)

Sets the location of the resulting Stokes parameter U.

The range of V is [-1,1]. It takes the value -1 when the reflected light is 100% left-hand circularly polarized, and the value 1 when the reflected light is 100% right-hand circularly polarized.

5.26.4.15 void setPolarizationPropertyFloat (**PolarizationProcessingHandle** *Polarization,* **PolarizationPropertyFloat** *Property,* double *Value*)

Sets the desired polarization processing floating-point property.

5.26.4.16 void setPolarizationPropertyInt (**PolarizationProcessingHandle** *Polarization,* **PolarizationPropertyInt** *Property,* int *Value*)

Sets polarization processing properties.

5.27 Polarization Adjustment

Typedefs

- typedef void(__stdcall ∗ cbRetardationChanged) (PolarizationRetarder, double)

  Defines the function prototype for the polarization adjustment retardation callback (see also setPolarizationAdjustmentRetardationChangedCallback()). The argument contains the current (unitless) position (see also setPolarizationAdjustmentRetardation()) of the specified PolarizationRetarder.

Functions

- SPECTRALRADAR_API BOOL isPolarizationAdjustmentAvailable (OCTDeviceHandle Dev)

  Returns whether or not a motorized polarization adjustment stage is available for the specified device.
- SPECTRALRADAR_API void setPolarizationAdjustmentRetardationChangedCallback (OCTDeviceHandle Dev, cbRetardationChanged Callback)

  Registers the callback to get notified when the polarization adjustment retardation has changed.
- SPECTRALRADAR_API void setPolarizationAdjustmentRetardation (OCTDeviceHandle Dev, PolarizationRetarder Retarder, double Retardation, WaitForCompletion Wait)

  Sets the retardation of the specified retarder in the polarization adjustment. The retardation is a unitless value between 0 and 1, which represents the full adjustment range of the retarder. The retarder may take some time to physically reach the new Retardation. Use the Wait parameter to choose if the function should block until the new position is reached.
- SPECTRALRADAR_API double getPolarizationAdjustmentRetardation (OCTDeviceHandle Dev, PolarizationRetarder Retarder)

  Gets the current retardation of the specified retarder in the polarization adjustment. If setPolarizationAdjustmentRetardation was used in a non-blocking fashion, the function returns the current position of the retarder, not the final target position.

5.27.1 Detailed Description

5.27.2 Typedef Documentation

5.27.2.1 typedef void(__stdcall∗ cbRetardationChanged) (**PolarizationRetarder**, double)

Defines the function prototype for the polarization adjustment retardation callback (see also setPolarizationAdjustmentRetardationChangedCallback()). The argument contains the current (unitless) position (see also setPolarizationAdjustmentRetardation()) of the specified PolarizationRetarder.

5.27.3 Function Documentation

5.27.3.1 double getPolarizationAdjustmentRetardation (**OCTDeviceHandle** *Dev,* **PolarizationRetarder** *Retarder*)

Gets the current retardation of the specified retarder in the polarization adjustment. If setPolarizationAdjustmentRetardation was used in a non-blocking fashion, the function returns the current position of the retarder, not the final target position.

Parameters

Dev	the OCTDeviceHandle that was initially provided by initDevice.
Retarder	the PolarizationRetarder which shall be queried

Returns

The current unitless Retardation of the selected Retarder (0 <= Retardation <= 1)

5.27.3.2 SPECTRALRADAR_API BOOL isPolarizationAdjustmentAvailable (OCTDeviceHandle *Dev*)

Returns whether or not a motorized polarization adjustment stage is available for the specified device.

Parameters

Dev	the OCTDeviceHandle that was initially provided by initDevice.

Returns

true if a polarization adjustment is available

5.27.3.3 double setPolarizationAdjustmentRetardation (OCTDeviceHandle *Dev*, PolarizationRetarder *Retarder*, double *Retardation*, WaitForCompletion *Wait*)

Sets the retardation of the specified retarder in the polarization adjustment. The retardation is a unitless value between 0 and 1, which represents the full adjustment range of the retarder. The retarder may take some time to physically reach the new Retardation. Use the Wait parameter to choose if the function should block until the new position is reached.

Parameters

Dev	the OCTDeviceHandle that was initially provided by initDevice.
Retarder	the PolarizationRetarder which shall be adjusted
Retardation	the new retardation value (0 <= retardation <= 1)
Wait	specify WaitForCompletion Wait to block until the new Retardation value has been reached

5.27.3.4 void setPolarizationAdjustmentRetardationChangedCallback (OCTDeviceHandle *Dev*, cbRetardationChanged *Callback*)

Registers the callback to get notified when the polarization adjustment retardation has changed.

Parameters

Dev	the OCTDeviceHandle that was initially provided by initDevice.
Callback	the Callback to register.

5.28 Reference Intensity Control

Typedefs

- typedef void(__stdcall ∗ cbReferenceIntensityControlValueChanged) (double)

  Defines the function prototype for the reference intensity control status callback (see also setReferenceIntensityControlCallback()). The argument contains the current (unitless) intensity between 0 and 1 (see also setReferenceIntensityControlValue()).

Functions

- SPECTRALRADAR_API BOOL isReferenceIntensityControlAvailable (OCTDeviceHandle Dev)

  Returns whether or not an automated reference intensity control is available for the specified device.
- SPECTRALRADAR_API void setReferenceIntensityControlCallback (OCTDeviceHandle Dev, cbReferenceIntensityControlValueChanged Callback)

  Registers the callback to get notified when the reference intensity has changed.
- SPECTRALRADAR_API void setReferenceIntensityControlValue (OCTDeviceHandle Dev, double ReferenceIntensity, WaitForCompletion Wait)

  Sets the reference intensity of the specified device. The intensity is a unitless value between 0 and 1, which represents the full adjustment range of the reference intensity control, but may or may not be linear. The control may take some time to physically reach the new intensity. Use the Wait parameter to choose if the function should block until the new intensity is reached.
- SPECTRALRADAR_API double getReferenceIntensityControlValue (OCTDeviceHandle Dev)

  Gets the current reference intensity of the specified device. If setReferenceIntensityControlValue was used in a non-blocking fashion, the function returns the current value of the control, not the final target value.

5.28.1 Detailed Description

5.28.2 Typedef Documentation

5.28.2.1 typedef void(__stdcall∗ cbReferenceIntensityControlValueChanged) (double)

Defines the function prototype for the reference intensity control status callback (see also setReferenceIntensityControlCallback()). The argument contains the current (unitless) intensity between 0 and 1 (see also setReferenceIntensityControlValue()).

5.28.3 Function Documentation

5.28.3.1 double getReferenceIntensityControlValue (**OCTDeviceHandle** *Dev*)

Gets the current reference intensity of the specified device. If setReferenceIntensityControlValue was used in a non-blocking fashion, the function returns the current value of the control, not the final target value.

Parameters

Dev	the OCTDeviceHandle that was initially provided by initDevice.

Returns

The current unitless reference intensity of the selected device (0 <= ReferenceIntensity <= 1)

5.28.3.2 SPECTRALRADAR_API BOOL isReferenceIntensityControlAvailable (OCTDeviceHandle *Dev*)

Returns whether or not an automated reference intensity control is available for the specified device.

Parameters

Dev	the OCTDeviceHandle that was initially provided by initDevice.

Returns

true if a reference intensity control is available

5.28.3.3 void setReferenceIntensityControlCallback (OCTDeviceHandle *Dev*, cbReferenceIntensityControlValue↩ Changed *Callback*)

Registers the callback to get notified when the reference intensity has changed.

Parameters

Dev	the OCTDeviceHandle that was initially provided by initDevice.
Callback	the Callback to register.

5.28.3.4 void setReferenceIntensityControlValue (OCTDeviceHandle *Dev*, double *ReferenceIntensity*, WaitForCompletion *Wait*)

Sets the reference intensity of the specified device. The intensity is a unitless value between 0 and 1, which represents the full adjustment range of the reference intensity control, but may or may not be linear. The control may take some time to physically reach the new intensity. Use the Wait parameter to choose if the function should block until the new intensity is reached.

Parameters

Dev	the OCTDeviceHandle that was initially provided by initDevice.
ReferenceIntensity	the new reference intensity value (0 <= ReferenceIntensity <= 1)
Wait	specify WaitForCompletion Wait to block until the new intensity value has been reached

5.29 Amplification Control

Functions

- SPECTRALRADAR_API BOOL isAmplificationControlAvailable (OCTDeviceHandle Dev)

  Returns whether or not the sampling amplification of specified device can be adjusted.
- SPECTRALRADAR_API int getAmplificationControlNumberOfSteps (OCTDeviceHandle Dev)

  Gets the number of discrete amplification control steps available on the specified device. Please note that the largest amplification step is getAmplificationControlNumberOfSteps() - 1.
- SPECTRALRADAR_API void setAmplificationControlStep (OCTDeviceHandle Dev, int Step)

  Sets the sampling amplification on the the specified device. The lowest amplification is always 0. In general, the amplification should be set as high as possible without going into saturation.
- SPECTRALRADAR_API int getAmplificationControlStep (OCTDeviceHandle Dev)

  Gets the current sampling amplification of the specified device.

5.29.1 Detailed Description

5.29.2 Function Documentation

5.29.2.1 int getAmplificationControlNumberOfSteps (OCTDeviceHandle *Dev*)

Gets the number of discrete amplification control steps available on the specified device. Please note that the largest amplification step is getAmplificationControlNumberOfSteps() - 1.

Parameters

Dev	the OCTDeviceHandle that was initially provided by initDevice.

Returns

The number of amplification steps.

5.29.2.2 int getAmplificationControlStep (OCTDeviceHandle *Dev*)

Gets the current sampling amplification of the specified device.

Parameters

Dev	the OCTDeviceHandle that was initially provided by initDevice.

Returns

The current amplification step of the selected device (0 <= Step <= getAmplificationControlNumberOfSteps())

5.29.2.3 SPECTRALRADAR_API BOOL isAmplificationControlAvailable (OCTDeviceHandle *Dev*)

Returns whether or not the sampling amplification of specified device can be adjusted.

Parameters

Dev	the OCTDeviceHandle that was initially provided by initDevice.

Returns

true if an amplification control is available

5.29.2.4 void setAmplificationControlStep (OCTDeviceHandle *Dev,* int *Step*)

Sets the sampling amplification on the the specified device. The lowest amplification is always 0. In general, the amplification should be set as high as possible without going into saturation.

Parameters

Dev	the OCTDeviceHandle that was initially provided by initDevice.
Step	Which amplification step to use. 0 <= AmplificationStep < getAmplificationControlNumberOfSteps()

6 Data Structure Documentation

6.1 ComplexFloat Struct Reference

A standard complex data type that is used to access complex data.

Data Fields

- float data [2]

  data[0] is the real part and data[1] is the imaginary part.

6.1.1 Detailed Description

A standard complex data type that is used to access complex data.

This data structure is an ANSI C equivalent to the C++ data type

```
std::complex<float>
```

. Notice that arrays of complex data are always in interleaved format (real and imaginary parts of each element in contiguous memory addresses) and not in split format, where real and imaginary parts are stored in separate arrays.

6.1.2 Field Documentation

6.1.2.1 float data[2]

data[0] is the real part and data[1] is the imaginary part.

7 File Documentation

7.1 SpectralRadar.h File Reference

Header containing all functions of the Spectral Radar SDK. This SDK can be used for Callisto, Ganymede, Hyperion, Telesto and Vega devices.

Data Structures

- struct ComplexFloat

  A standard complex data type that is used to access complex data.

Macros

- #define SPECTRALRADAR_API __declspec(dllimport)

  Export/Import of define of DLL members.
- #define TRUE 1

  TRUE for use with data type BOOL.
- #define FALSE 0

  FALSE for use with data type BOOL.

Typedefs

- typedef int BOOL

  A standard boolean data type used in the API.
- typedef struct C_RawData ∗ RawDataHandle

  Handle to an object holding the unprocessed raw data.
- typedef struct C_Data ∗ DataHandle

  Handle to an object holding 1-, 2- or 3-dimensional floating point data.
- typedef struct C_ColoredData ∗ ColoredDataHandle

  Handle to an object holding 1-, 2- or 3-dimensional colored data.
- typedef struct C_ComplexData ∗ ComplexDataHandle

  Handle to an object holding complex 1-, 2- or 3-dimensional complex floating point data.
- typedef struct C_Buffer ∗ BufferHandle

  The BufferHandle identifies a data buffer.
- typedef struct C_OCTDevice ∗ OCTDeviceHandle

  The OCTDeviceHandle type is used as Handle for using the SpectralRadar.
- typedef struct C_Probe ∗ ProbeHandle

  Handle for controlling the galvo scanner.
- typedef struct C_ScanPattern ∗ ScanPatternHandle

  Handle for creating, manipulating, and discarding a scan pattern.
- typedef struct C_Processing ∗ ProcessingHandle

  Handle for a processing routine.
- typedef struct C_DopplerProcessing ∗ DopplerProcessingHandle

  Handle used for Doppler processing.
- typedef struct C_SpeckleVariance ∗ SpeckleVarianceHandle

  Handle used for SpeckleVariance processing.
- typedef struct C_PolarizationProcessing ∗ PolarizationProcessingHandle

  Handle used for Polarization processing.
- typedef struct C_Coloring32Bit ∗ ColoringHandle

  Handle for routines that color avaible scans for displaying.
- typedef struct C_ImageFieldCorrection ∗ ImageFieldHandle

  Handle to the image field description.
- typedef struct C_VisualCalibration ∗ VisualCalibrationHandle

  Handle to the visual galvo calibration class.
- typedef struct C_MarkerList ∗ MarkerListHandle

  Handle to the marker list class.
- typedef struct C_FileHandling ∗ OCTFileHandle

  Handle to the OCT file class.
- typedef struct C_Settings ∗ SettingsHandle

  Handle for saving settings on disk.
- typedef void(__stdcall ∗ cbProbeMessageReceived) (int)

The prototype for callback functions registered for probe button events. As of the creation time of this document, only the OCTH probe is equipped with buttons.

- typedef void(__stdcall ∗ cbRefstageStatusChanged) (RefstageStatus)

  Defines the function prototype for the reference stage status callback (see also setRefstageStatusCallback()). The argument contains the current status of the reference stage when called.
- typedef void(__stdcall ∗ cbRefstagePositionChanged) (double)

  Defines the function prototype for the reference stage position change callback (see also setRefstagePosChangedCallback()). The argument contains the reference stage position in mm when called.
- typedef void(__stdcall ∗ lightSourceStateCallback) (LightSourceState)

  Defines the function prototype for the light source callback(see also setLightSourceTimeoutCallback()). The argument contains the current state of the light source.
- typedef void(__stdcall ∗ cbRetardationChanged) (PolarizationRetarder, double)

  Defines the function prototype for the polarization adjustment retardation callback (see also setPolarizationAdjustmentRetardationChangedCallback()). The argument contains the current (unitless) position (see also setPolarizationAdjustmentRetardation()) of the specified PolarizationRetarder.
- typedef void(__stdcall ∗ cbReferenceIntensityControlValueChanged) (double)

  Defines the function prototype for the reference intensity control status callback (see also setReferenceIntensityControlCallback()). The argument contains the current (unitless) intensity between 0 and 1 (see also setReferenceIntensityControlValue()).

Enumerations

- enum ErrorCode {
  NoError = 0x0000,
  Error = 0xE000 }

  This enum is used to describe errors that occur when operating an OCT device.
- enum LogOutputType {
  Standard,
  File,
  None }

  Specifies where to write text output by the SDK.
- enum RawDataPropertyInt {
  RawData_Size1,
  RawData_Size2,
  RawData_Size3,
  RawData_NumberOfElements,
  RawData_SizeInBytes,
  RawData_BytesPerElement,
  RawData_LostFrames }

  Integer properties of raw data (RawDataHandle) that can be retrieved with the function getRawDataPropertyInt.
- enum DataPropertyInt {
  Data_Dimensions,
  Data_Size1,
  Data_Size2,
  Data_Size3,
  Data_NumberOfElements,
  Data_SizeInBytes,
  Data_BytesPerElement }

  Integer properties of data (DataHandle) that can be retrieved with the function getDataPropertyInt.
- enum DataPropertyFloat {
  Data_Spacing1,
  Data_Spacing2,
  Data_Spacing3,
  Data_Range1,
  Data_Range2,
  Data_Range3 }

Floating point properties of data (DataHandle), that can be retrieved with the function getDataPropertyFloat.

- enum DataAnalyzation {
Data_Min,
Data_Mean,
Data_Max,
Data_MaxDepth }

  Analysis types accepted by the functions analyzeData and computeDataProjection.

- enum AScanAnalyzation {
Data_Noise_dB,
Data_Noise_electrons,
Data_PeakPos_Pixel,
Data_PeakPos_PhysUnits,
Data_PeakHeight_dB,
Data_PeakWidth_6dB,
Data_PeakWidth_20dB,
Data_PeakWidth_40dB,
Data_PeakPhase,
Data_PeakRealPart,
Data_PeakImagPart }

  Analysis types accepted by the functions analyzeAScan and analyzeComplexAScan.

- enum DataOrientation {
DataOrientation_ZXY,
DataOrientation_ZYX,
DataOrientation_XZY,
DataOrientation_XYZ,
DataOrientation_YXZ,
DataOrientation_YZX,
DataOrientation_ZTX,
DataOrientation_ZXT }

  Supported data orientations. The default orientation is the first one.

- enum DevicePropertyFloat {
Device_FullWellCapacity,
Device_zSpacing,
Device_zRange,
Device_SignalAmplitudeMin_dB,
Device_SignalAmplitudeLow_dB,
Device_SignalAmplitudeHigh_dB,
Device_SignalAmplitudeMax_dB,
Device_BinToElectronScaling,
Device_Temperature,
Device_SLD_OnTime_sec,
Device_CenterWavelength_nm,
Device_SpectralWidth_nm,
Device_MaxTriggerFrequency_Hz,
Device_LineRate_Hz }

  Floating point properties of the device that can be retrieved with the function getDevicePropertyFloat.

- enum DevicePropertyInt {
Device_SpectrumElements,
Device_BytesPerElement,
Device_MaxLiveVolumeRenderingScans,
Device_BitDepth,
Device_NumOfCameras,
Device_RevisionNumber }

  Integer properties of the device that can be retrieved with the function getDevicePropertyInt.

- enum DevicePropertyString {

Device_Type,
Device_Series,
Device_SerialNumber,
Device_HardwareConfig }

String-properties of the device that can be retrieved with the function getDevicePropertyString.

- enum DeviceFlag {
Device_On,
Device_CameraAvailable,
Device_SLDAvailable,
Device_SLDStatus,
Device_LaserDiodeStatus,
Device_CameraShowScanPattern,
Device_ProbeControllerAvailable,
Device_DataIsSigned,
Device_IsSweptSource }

  Boolean properties of the device that can be retrieved with the function getDeviceFlag.

- enum ScanAxis {
ScanAxis_X = 0,
ScanAxis_Y = 1 }

  Axis selection for the function moveScanner.

- enum ProbeParameterFloat {
Probe_FactorX,
Probe_OffsetX,
Probe_FactorY,
Probe_OffsetY,
Probe_FlybackTime_Sec,
Probe_ExpansionTime_Sec,
Probe_RotationTime_Sec,
Probe_ExpectedScanRate_Hz,
Probe_CameraScalingX,
Probe_CameraOffsetX,
Probe_CameraScalingY,
Probe_CameraOffsetY,
Probe_CameraAngle,
Probe_RangeMaxX,
Probe_RangeMaxY,
Probe_MaximumSlope_XY,
Probe_SpeckleSize,
Probe_ApoVoltageX,
Probe_ApoVoltageY,
Probe_ReferenceStageOffset,
Probe_FiberOpticalPathLength_mm,
Probe_ProbeOpticalPathLength_mm,
Probe_ObjectiveOpticalPathLength_mm,
Probe_ObjectiveFocalLength_mm }

  Parameters describing the behaviour of the Probe, such as calibration factors and scan parameters.

- enum ProbeParameterString {
Probe_Name,
Probe_SerialNumber,
Probe_Description,
Probe_Objective }

  Parameters describing the composition of the probe. These properties refer to a probe that has already been created and for which a valid ProbeHandle has been obtained.

- enum ProbeParameterInt {
Probe_ApodizationCycles,
Probe_Oversampling,
Probe_Oversampling_SlowAxis,

Probe_SpeckleReduction }

Parameters describing the behaviour of the Probe, such as calibration factors and scan parameters.

- enum ProbeFlag {
Probe_CameraInverted_X,
Probe_CameraInverted_Y,
Probe_HasMEMSScanner }

  Boolean parameters describing the behaviour of the Probe.
- enum ScanPatternAcquisitionOrder {
ScanPattern_AcqOrderFrameByFrame,
ScanPattern_AcqOrderAll }

  Parameters describing the behaviour of the scan pattern.
- enum ScanPatternApodizationType {
ScanPattern_ApoOneForAll,
ScanPattern_ApoEachBScan }

  Parameters describing how often the apodization spectra will be acquired. If you want to create a scan pattern without an apodization please use (setProbeParameterInt) and (Probe_ApodizationCycles) to set the size of apodization to zero.
- enum InflationMethod { Inflation_NormalDirection }

  Describes how to use a 2D freeform scan pattern to create a 3D scan pattern.
- enum InterpolationMethod {
Interpolation_Linear,
Interpolation_Spline }

  Selects the interpolation method.
- enum BoundaryCondition {
BoundaryCondition_Standard,
BoundaryCondition_Natural,
BoundaryCondition_Periodic }

  Selects the boundary conditions for the interpolation.
- enum ScanPointsDataFormat {
ScanPoints_DataFormat_TXT,
ScanPoints_DataFormat_RAWandSRM }

  Selects format with the functions loadScanPointsFromFile or saveScanPointsToFile to import or export data points.
- enum AcquisitionType {
Acquisition_AsyncContinuous,
Acquisition_AsyncFinite,
Acquisition_Sync }

  Determines the kind of acquisition process. The type of acquisition process affects e.g. whether consecutive B-scans are acquired or if it is possible to lose some data.
- enum Processing_FFTType {
Processing_StandardFFT,
Processing_StandardNDFT,
Processing_iFFT,
Processing_NFFT1,
Processing_NFFT2,
Processing_NFFT3,
Processing_NFFT4 }

  defindes the algorithm used for dechirping the input signal and Fourier transformation
- enum DispersionCorrectionType {
Dispersion_None,
Dispersion_QuadraticCoeff,
Dispersion_Preset,
Dispersion_Manual }

  To select the dispersion correction algorithm.

- enum ApodizationWindow {
  Apodization_Hann = 0,
  Apodization_Hamming = 1,
  Apodization_Gauss = 2,
  Apodization_TaperedCosine = 3,
  Apodization_Blackman = 4,
  Apodization_BlackmanHarris = 5,
  Apodization_LightSourceBased = 6,
  Apodization_Unknown = 999 }

  To select the apodization window function.

- enum ProcessingParameterInt {
  Processing_SpectrumAveraging,
  Processing_AScanAveraging,
  Processing_BScanAveraging,
  Processing_ZeroPadding,
  Processing_NumberOfThreads,
  Processing_FourierAveraging }

  Parameters that set the behavious of the processing algorithms.

- enum ProcessingParameterFloat {
  Processing_ApodizationDamping,
  Processing_MinElectrons,
  Processing_FFTOversampling,
  Processing_MaxSensorValue }

  Parameters that set the behaviour of the processing algorithms.

- enum CalibrationData {
  Calibration_OffsetErrors,
  Calibration_ApodizationSpectrum,
  Calibration_ApodizationVector,
  Calibration_Dispersion,
  Calibration_Chirp,
  Calibration_ExtendedAdjust,
  Calibration_FixedPattern }

  Data describing the calibration of the processing routines.

- enum ProcessingFlag {
  Processing_UseOffsetErrors,
  Processing_RemoveDCSpectrum,
  Processing_RemoveAdvancedDCSpectrum,
  Processing_UseApodization,
  Processing_UseScanForApodization,
  Processing_UseUndersamplingFilter,
  Processing_UseDispersionCompensation,
  Processing_UseDechirp,
  Processing_UseExtendedAdjust,
  Processing_FullRangeOutput,
  Processing_FilterDC,
  Processing_UseAutocorrCompensation,
  Processing_UseDEFR,
  Processing_OnlyWindowing,
  Processing_RemoveFixedPattern,
  Processing_CalculateSaturation }

  Flags that set the behaviour of the processing algorithms.

- enum ProcessingAveragingAlgorithm {

Processing_Averaging_Min,
Processing_Averaging_Mean,
Processing_Averaging_Median,
Processing_Averaging_Norm2,
Processing_Averaging_Max,
Processing_Averaging_Fourier_Min,
Processing_Averaging_Fourier_Norm4,
Processing_Averaging_Fourier_Max,
Processing_Averaging_StandardDeviationAbs,
Processing_Averaging_PhaseMatched }

This sets the averaging algorithm to be used for processing.

- enum ApodizationWindowParameter {
ApodizationWindowParameter_Sigma,
ApodizationWindowParameter_Ratio,
ApodizationWindowParameter_Frequency }

  Sets certain parameters that are used by the window functions to be applied during apodization.

- enum DataExportFormat {
DataExport_SRM,
DataExport_RAW,
DataExport_CSV,
DataExport_TXT,
DataExport_TableTXT,
DataExport_Fits,
DataExport_VFF,
DataExport_VTK,
DataExport_TIFF }

  Export format for any data represented by a DataHandle.

- enum ComplexDataExportFormat { ComplexDataExport_RAW }

  Export format for complex data.

- enum ColoredDataExportFormat {
ColoredDataExport_SRM,
ColoredDataExport_RAW,
ColoredDataExport_BMP,
ColoredDataExport_PNG,
ColoredDataExport_JPG,
ColoredDataExport_PDF,
ColoredDataExport_TIFF }

  Export format for images (ColoredDataHandle).

- enum Direction {
Direction_1,
Direction_2,
Direction_3 }

  Specifies a direction. In the default orientation, the first orientation is the Z-axis (parallel to the illumination-ray during the measurement), the second is the X-axis, and the third is the Y-axis.

- enum DataImportFormat { DataImport_SRM }

  Supported import format to load data from disk.

- enum RawDataExportFormat {
RawDataExport_RAW,
RawDataExport_SRR }

  Supported raw data export formats to store data to disk.

- enum RawDataImportFormat { RawDataImport_SRR }

  Supported raw data import formats to load data from disk.

- enum Plane2D {
Plane2D_12,
Plane2D_23,
Plane2D_13 }

Planes for slices of the volume data.

- enum DopplerPropertyInt {
Doppler_Averaging_1,
Doppler_Averaging_2,
Doppler_Stride_1,
Doppler_Stride_2 }

  Values that determine the behaviour of the Doppler processing routines.
- enum DopplerPropertyFloat {
Doppler_RefractiveIndex,
Doppler_ScanRate_Hz,
Doppler_CenterWavelength_nm,
Doppler_DopplerAngle_Deg }

  Values that determine the behaviour of the Doppler processing routines.
- enum DopplerFlag { Doppler_VelocityScaling }

  Flats that determine the behaviour of the Doppler processing routines.
- enum ColorScheme {
ColorScheme_BlackAndWhite = 0,
ColorScheme_Inverted = 1,
ColorScheme_Color = 2,
ColorScheme_BlackAndOrange = 3,
ColorScheme_BlackAndRed = 4,
ColorScheme_BlackRedAndYellow = 5,
ColorScheme_DopplerPhase = 6,
ColorScheme_BlueAndBlack = 7,
ColorScheme_PolarizationRetardation = 8,
ColorScheme_GreenBlueAndBlack = 9,
ColorScheme_BlackAndRedYellow = 10,
ColorScheme_TransparentAndWhite = 11,
ColorScheme_GreenBlueWhiteRedYellow = 12,
ColorScheme_BlueGreenBlackYellowRed = 13,
ColorScheme_RedGreenBlue = 14,
ColorScheme_GreenBlueRed = 15,
ColorScheme_BlueRedGreen = 16,
ColorScheme_GreenBlueRedGreen = 17,
ColorScheme_BlueRedGreenBlue = 18,
ColorScheme_Inverse_RedGreenBlue = 19,
ColorScheme_Inverse_GreenBlueRed = 20,
ColorScheme_Inverse_BlueRedGreen = 21,
ColorScheme_Inverse_GreenBlueRedGreen = 22,
ColorScheme_Inverse_BlueRedGreenBlue = 23,
ColorScheme_RedYellowGreenBlueRed = 24,
ColorScheme_RedGreenBlueRed = 25,
ColorScheme_Inverse_RedGreenBlueRed = 26,
ColorScheme_RedYellowBlue = 27,
ColorScheme_Inverse_RedYellowBlue = 28,
ColorScheme_DEM_Normal = 29,
ColorScheme_Inverse_DEM_Normal = 30,
ColorScheme_DEM_Blind = 31,
ColorScheme_Inverse_DEM_Blind = 32,
ColorScheme_WhiteBlackWhite = 33,
ColorScheme_BlackWhiteBlack = 34 }

  selects the ColorScheme of the data to transform real data to colored data.
- enum ColoringByteOrder {
Coloring_RGBA = 0,
Coloring_BGRA = 1,
Coloring_ARGB = 2 }

  Selects the byte order of the coloring to be applied.

- enum ColorEnhancement {
  ColorEnhancement_None = 0,
  ColorEnhancement_Sine = 1,
  ColorEnhancement_Parable = 2,
  ColorEnhancement_Cubic = 3,
  ColorEnhancement_Sqrt = 4 }

  Selects the byte order of the coloring to be applied.

- enum OCTFileFormat {
  FileFormat_OCITY,
  FileFormat_IMG,
  FileFormat_SDR,
  FileFormat_SRM,
  FileFormat_TIFF32 }

  Enum identifying possible file formats.

- enum DataObjectType {
  DataObjectType_Real,
  DataObjectType_Colored,
  DataObjectType_Complex,
  DataObjectType_Raw,
  DataObjectType_Binary,
  DataObjectType_Text,
  DataObjectType_Unknown = 999 }

  Enum identifying.

- enum FileMetadataFloat {
  FileMetadata_RefractiveIndex,
  FileMetadata_RangeX,
  FileMetadata_RangeY,
  FileMetadata_RangeZ,
  FileMetadata_CenterX,
  FileMetadata_CenterY,
  FileMetadata_Angle,
  FileMetadata_BinToElectronScaling,
  FileMetadata_CentralWavelength_nm,
  FileMetadata_SourceBandwidth_nm,
  FileMetadata_MinElectrons,
  FileMetadata_QuadraticDispersionCorrectionFactor,
  FileMetadata_SpeckleVarianceThreshold,
  FileMetadata_ScanTime_Sec,
  FileMetadata_ReferenceIntensity,
  FileMetadata_ScanPause_Sec,
  FileMetadata_Zoom,
  FileMetadata_MinPointDistance,
  FileMetadata_MaxPointDistance,
  FileMetadata_FFTOversampling,
  FileMetadata_FullWellCapacity,
  FileMetadata_Saturation,
  FileMetadata_CameraLineRate_Hz,
  FileMetadata_PMDCorrectionAngle_rad,
  FileMetadata_OpticalAxisOffset_rad }

  Enum identifying file metadata fields of floating point type.

- enum FileMetadataInt {

FileMetadata_ProcessState,
FileMetadata_SizeX,
FileMetadata_SizeY,
FileMetadata_SizeZ,
FileMetadata_Oversampling,
FileMetadata_IntensityAveragedSpectra,
FileMetadata_IntensityAveragedAScans,
FileMetadata_IntensityAveragedBScans,
FileMetadata_DopplerAverageX,
FileMetadata_DopplerAverageZ,
FileMetadata_ApoWindow,
FileMetadata_DeviceBitDepth,
FileMetadata_SpectrometerElements,
FileMetadata_ExperimentNumber,
FileMetadata_DeviceBytesPerPixel,
FileMetadata_SpeckleAveragingFastAxis,
FileMetadata_SpeckleAveragingSlowAxis,
FileMetadata_Processing_FFTType,
FileMetadata_NumOfCameras,
FileMetadata_SelectedCamera,
FileMetadata_ApodizationType,
FileMetadata_AcquisitionOrder,
FileMetadata_DOPUFilter,
FileMetadata_DOPUAverageZ,
FileMetadata_DOPUAverageX,
FileMetadata_DOPUAverageY,
FileMetadata_PolarizationAverageZ,
FileMetadata_PolarizationAverageX,
FileMetadata_PolarizationAverageY }

Enum identifying file metadata fields of integral type.

- enum FileMetadataString {
FileMetadata_DeviceSeries,
FileMetadata_DeviceName,
FileMetadata_Serial,
FileMetadata_Comment,
FileMetadata_CustomInfo,
FileMetadata_AcquisitionMode,
FileMetadata_Study,
FileMetadata_DispersionPreset,
FileMetadata_ProbeName,
FileMetadata_FreeformScanPatternInterpolation,
FileMetadata_HardwareConfig,
FileMetadata_OrigVersion,
FileMetadata_LastModVersion }

Enum identifying file metadata fields of character string type.

- enum FileMetadataFlag {

FileMetadata_OffsetApplied,
FileMetadata_DCSubtracted,
FileMetadata_ApoApplied,
FileMetadata_DechirpApplied,
FileMetadata_UndersamplingFilterApplied,
FileMetadata_DispersionCompensationApplied,
FileMetadata_QuadraticDispersionCorrectionUsed,
FileMetadata_ImageFieldCorrectionApplied,
FileMetadata_ScanLineShown,
FileMetadata_AutoCorrCompensationUsed,
FileMetadata_BScanCrossCorrelation,
FileMetadata_DCSubtractedAdvanced,
FileMetadata_OnlyWindowing,
FileMetadata_RawDataIsSigned,
FileMetadata_FreeformScanPatternIsActive,
FileMetadata_FreeformScanPatternCloseLoop,
FileMetadata_IsSweptSource }

Enum identifying file metadata fields of bool type.

- enum SpeckleVarianceType {
SpeckleVariance_LogscaleVariance_Linear,
SpeckleVariance_LogscaleVariance_Logscale,
SpeckleVariance_LinearVariance_Linear,
SpeckleVariance_LinearVariance_Logscale,
SpeckleVariance_ComplexVariance_Linear,
SpeckleVariance_ComplexVariance_Logscale }

  Enum identifying different speckle variance processing types.
- enum SpeckleVariancePropertyInt {
SpeckleVariance_Averaging_1,
SpeckleVariance_Averaging_2,
SpeckleVariance_Averaging_3 }

  Enum identifying different properties of typ int for speckle variance processing.
- enum SpeckleVariancePropertyFloat { **SpeckleVariance_Threshold** }

  Enum identifying different properties of typ float for speckle variance processing.
- enum DeviceTriggerType {
Trigger_FreeRunning,
Trigger_TrigBoard_ExternalStart,
Trigger_External_AScan }

  Enum identifying trigger types for the OCT system.
- enum ScanPatternPropertyInt {
ScanPattern_SizeTotal,
ScanPattern_Cycles,
ScanPattern_SizeCycle,
ScanPattern_SizePreparationCycle,
ScanPattern_SizeImagingCycle }

  Enum identifying different properties of typ int of the specified scan pattern.
- enum ScanPatternPropertyFloat {
ScanPattern_RangeX,
ScanPattern_RangeY,
ScanPattern_CenterX,
ScanPattern_CenterY,
ScanPattern_Angle,
ScanPattern_MeanLength_mm }

  Enum identifying different floating-type properties of the specified scan pattern.
- enum PepperFilterType {
PepperFilter_Horizontal,
PepperFilter_Vertical,
PepperFilter_Star,

PepperFilter_Block }

Specifies the type of pepper filter to be applied.

- enum ComplexFilterType2D { FilterComplex2D_PhaseContrast }

  Specifies the type of filter to be applied to complex data.
- enum FilterType1D { Filter1D_Gaussian_5 }

  Specifies the type of 1D-filter to be applied. All filters are normalized.
- enum FilterType2D {
  Filter2D_Gaussian_3x3,
  Filter2D_Gaussian_5x5,
  Filter2D_Prewitt_Horizontal_3x3,
  Filter2D_Prewitt_Vertical_3x3,
  Filter2D_NonlinearPrewitt_3x3,
  Filter2D_Sobel_Horizontal_3x3,
  Filter2D_Sobel_Vertical_3x3,
  Filter2D_NonlinearSobel_3x3,
  Filter2D_Laplacian_NoDiagonal_3x3,
  Filter2D_Laplacian_3x3 }

  Specifies the type of 2D-filter to be applied. All filters are normalized.
- enum FilterType3D { Filter3D_Gaussian_3x3x3 }

  Specifies the type of 3D-filter to be applied. All filters are normalized.
- enum ObjectivePropertyString {
  Objective_DisplayName,
  Objective_Mount }

  Properties of the objective mounted to the scanner such as the name.
- enum ObjectivePropertyInt {
  Objective_RangeMaxX_mm,
  Objective_RangeMaxY_mm }

  Properties of the objective mounted to the scanner such as valid scan range in mm.
- enum ObjectivePropertyFloat {
  Objective_FocalLength_mm,
  Objective_OpticalPathLength }

  Properties of the objective mounted to the scanner such as the focal length of the lens.
- enum ProbeScanRangeShape {
  Probe_ScanRange_Rectangular,
  Probe_ScanRange_Round }

  The shape of the maximal valid scan range.
- enum RefstageStatus {
  RefStage_Status_Idle = 0,
  RefStage_Status_Homing = 1,
  RefStage_Status_Moving = 2,
  RefStage_Status_MovingTo = 3,
  RefStage_Status_Stopping = 4,
  RefStage_Status_NotAvailable = 5,
  RefStage_Status_Undefined = -1 }

  Defines the status of the motorized reference stage.
- enum RefstageSpeed {
  RefStage_Speed_Slow = 0,
  RefStage_Speed_Fast = 1,
  RefStage_Speed_VerySlow = 2,
  RefStage_Speed_VeryFast = 3 }

  Defines the velocity of movement for the motorized reference stage.
- enum RefstageWaitForMovement {
  RefStage_Movement_Wait = 0,
  RefStage_Movement_Continue = 1 }

Defines the behaviour whether the the function should wait until the movement of the motorized reference stage has stopped to return.

- enum RefstageMovementDirection {
RefStage_MoveShorter = 0,
RefStage_MoveLonger = 1 }

  Defines the direction of movement for the motorized reference stage. Please note that not in all systems a motorized reference stage is present.
- enum LightSourceState {
Activating,
On,
Off }

  Values that define the state of the light source.
- enum PolarizationDOPUFilterType {
PolarizationProcessing_DOPU_Median,
PolarizationProcessing_DOPU_Average,
PolarizationProcessing_DOPU_Gaussian,
PolarizationProcessing_DOPU_GaussianWithFFT }

  Values that determine the behaviour of temporal filter, if enabled.
- enum PolarizationPropertyInt {
PolarizationProcessing_DOPU_Z = 0,
PolarizationProcessing_DOPU_X = 1,
PolarizationProcessing_DOPU_Y = 2,
PolarizationProcessing_DOPU_FilterType = 3,
PolarizationProcessing_BScanAveraging = 4,
PolarizationProcessing_AveragingZ = 5,
PolarizationProcessing_AveragingX = 6,
PolarizationProcessing_AveragingY = 7,
PolarizationProcessing_AScanAveraging = 8 }

  Values that determine the behaviour of the Polarization processing routines.
- enum PolarizationPropertyFloat {
PolarizationProcessing_IntensityThreshold_dB = 0,
PolarizationProcessing_PMDCorrectionAngle_rad = 1,
PolarizationProcessing_CentralWavelength_nm = 2,
PolarizationProcessing_OpticalAxisOffset_rad = 3 }

  Values that determine the behaviour of the Polarization processing routines.
- enum WaitForCompletion {
Wait = 0,
Continue = 1 }

  Defines the behaviour whether a function should wait for the operation to complete or return immediately.
- enum PolarizationRetarder {
Retarder_Quarter_Wave = 0,
Retarder_Half_Wave = 1 }

  List of available polarization retarders in a polarization control unit.

Functions

- SPECTRALRADAR_API ErrorCode isError (void)

  Returns error code. The error flag will not be cleared; a following call to getError thus provides detailed error information.
- SPECTRALRADAR_API ErrorCode getError (char ∗Message, int StringSize)

  Returns an error code and a message if an error occurred. The error flag will be cleared.
- SPECTRALRADAR_API void setLog (LogOutputType Type, const char ∗Filename)

  Specifies where to write text output by the SDK. The respective text output might help to debug applications or identify errors and faults.

- SPECTRALRADAR_API int getDataPropertyInt (DataHandle Data, DataPropertyInt Selection)

  Returns the selected integer property of the specified data.
- SPECTRALRADAR_API double getDataPropertyFloat (DataHandle Data, DataPropertyFloat Selection)

  Returns the selected floating point property of the specified data.
- SPECTRALRADAR_API void copyData (DataHandle DataSource, DataHandle DataDestination)

  Copies the content of the specified source to the specified destination.
- SPECTRALRADAR_API void copyDataContent (DataHandle DataSource, float ∗Destination)

  Copies the data in the specified data object (DataHandle) into the specified pointer.
- SPECTRALRADAR_API float ∗ getDataPtr (DataHandle Data)

  The returned pointer points to memory owned by SpectralRadar.dll. The user should not attempt to free it.
- SPECTRALRADAR_API void reserveData (DataHandle Data, int Size1, int Size2, int Size3)

  Reserves the amount of data specified. This might improve performance if appending data to the DataHandle as no additional memory needs to be reserved then.
- SPECTRALRADAR_API void resizeData (DataHandle Data, int Size1, int Size2, int Size3)

  Resizes the respective data object. In general the data will be 1-dimensional if Size2 and Size3 are equal to 1, 2-dimensional if Size3 is equal to 1 dn 3-dimensional if all, Size1, Size2, Size3, are unequal to 1.
- SPECTRALRADAR_API void setDataRange (DataHandle Data, double range1, double range2, double range3)

  Sets the range in mm in the 3 axes represented in the RealData buffer.
- SPECTRALRADAR_API void setDataContent (DataHandle Data, float ∗NewContent)

  Sets the data content of the data object. The data chunk pointed to by NewContent needs to be of the size expected by the data object, i. e. Size1∗Size2∗Size∗sizeof(float).
- SPECTRALRADAR_API DataOrientation getDataOrientation (DataHandle Data)

  Returns the data orientation of the data object.
- SPECTRALRADAR_API void setDataOrientation (DataHandle Data, DataOrientation Orientation)

  Sets the data oritentation of the data object to the given orientation.
- SPECTRALRADAR_API int getComplexDataPropertyInt (ComplexDataHandle Data, DataPropertyInt Selection)

  Returns the selected integer property of the specified data.
- SPECTRALRADAR_API double getComplexDataPropertyFloat (ComplexDataHandle Data, DataPropertyFloat Selection)

  Returns the selected floating-point property of the specified data.
- SPECTRALRADAR_API void copyComplexDataContent (ComplexDataHandle DataSource, ComplexFloat ∗Destination)

  Copies the content of the complex data to the pointer specified as destination.
- SPECTRALRADAR_API void copyComplexData (ComplexDataHandle DataSource, ComplexDataHandle DataDestination)

  Copies the contents of the specified ComplexDataHandle to the specified destination ComplexDataHandle.
- SPECTRALRADAR_API ComplexFloat ∗ getComplexDataPtr (ComplexDataHandle Data)

  The returned pointer points to memory owned by SpectralRadar.dll. The user should not attempt to free it.
- SPECTRALRADAR_API void setComplexDataContent (ComplexDataHandle Data, ComplexFloat ∗NewContent)

  Sets the data content of the ComplexDataHandle to the content specified by the pointer.
- SPECTRALRADAR_API void reserveComplexData (ComplexDataHandle Data, int Size1, int Size2, int Size3)

  Reserves the amount of data specified. This might improve performance if appending data to the ComplexDataHandle as no additional memory needs to be reserved then.
- SPECTRALRADAR_API void resizeComplexData (ComplexDataHandle Data, int Size1, int Size2, int Size3)

  Resizes the respective data object. In general the data will be 1-dimensional if Size2 and Size3 are equal to 1, 2-dimensional if Size3 is equal to 1 dn 3-dimensional if all, Size1, Size2, Size3, are unequal to 1.
- SPECTRALRADAR_API void setComplexDataRange (ComplexDataHandle Data, double range1, double range2, double range3)

  Sets the range in mm in the 3 axes represented in the RealData buffer.

- SPECTRALRADAR_API int getColoredDataPropertyInt (ColoredDataHandle ColData, DataPropertyInt Selection)

  Returns the selected integer property of the specified colored data.
- SPECTRALRADAR_API double getColoredDataPropertyFloat (ColoredDataHandle ColData, DataPropertyFloat Selection)

  Returns the selected integer property of the specified colored data.
- SPECTRALRADAR_API void copyColoredData (ColoredDataHandle ImageSource, ColoredDataHandle ImageDestionation)

  Copies the contents of the specified ColoredDataHandle to the specified destination ColoredDataHandle.
- SPECTRALRADAR_API void copyColoredDataContent (ColoredDataHandle Source, unsigned long ∗Destination)

  Copies the data in the specified colored data object (ColoredDataHandle) into the specified pointer.
- SPECTRALRADAR_API void copyColoredDataContentAligned (ColoredDataHandle ImageSource, unsigned long ∗Destination, int Stride)

  Copies the data in the specified colored data object (ColoredDataHandle) into the specified pointer.
- SPECTRALRADAR_API unsigned long ∗ getColoredDataPtr (ColoredDataHandle ColData)

  The returned pointer points to memory owned by SpectralRadar.dll. The user should not attempt to free it.
- SPECTRALRADAR_API void resizeColoredData (ColoredDataHandle ColData, int Size1, int Size2, int Size3)

  Resizes the respective colored data object. In general the data will be 1-dimensional if Size2 and Size3 are equal to 1, 2-dimensional if Size3 is equal to 1 dn 3-dimensional if all, Size1, Size2, Size3, are unequal to 1.
- SPECTRALRADAR_API void reserveColoredData (ColoredDataHandle ColData, int Size1, int Size2, int Size3)

  Reserves the amount of colored data specified. This might improve performance if appending data to the ColoredDataHandle as no additional memory needs to be reserved then.
- SPECTRALRADAR_API void setColoredDataContent (ColoredDataHandle ColData, unsigned long ∗NewContent)

  Sets the data content of the colored data object. The data chung pointed to by NewContent needs to be of the size expected by the data object, i. e. Size1∗Size2∗Size∗sizeof(unsigned long).
- SPECTRALRADAR_API void setColoredDataRange (ColoredDataHandle Data, double range1, double range2, double range3)

  Sets the range in mm in the 3 axes represented in the data object buffer.
- SPECTRALRADAR_API DataOrientation getColoredDataOrientation (ColoredDataHandle Data)

  Returns the data orientation of the colored data object.
- SPECTRALRADAR_API void setColoredDataOrientation (ColoredDataHandle Data, DataOrientation Orientation)

  Sets the data oritentation of the colored data object to the given orientation.
- SPECTRALRADAR_API void copyRawDataContent (RawDataHandle RawDataSource, void ∗DataContent)

  Copies the content of the raw data into the specified buffer.
- SPECTRALRADAR_API void copyRawData (RawDataHandle RawDataSource, RawDataHandle RawDataTarget)

  Copies raw data content and metadata into the specified target handle.
- SPECTRALRADAR_API void ∗ getRawDataPtr (RawDataHandle RawDataSource)

  Notice that raw data refers to the spectra as acquired, without processing of any kind.
- SPECTRALRADAR_API int getRawDataPropertyInt (RawDataHandle RawData, RawDataPropertyInt Property)

  Notice that raw data refers to the spectra as acquired, without processing of any kind.
- SPECTRALRADAR_API void setRawDataBytesPerPixel (RawDataHandle Raw, int BytesPerPixel)

  Sets the bytes per pixel for raw data.
- SPECTRALRADAR_API void reserveRawData (RawDataHandle Raw, int Size1, int Size2, int Size3)

  Reserves the amount of data specified. This might improve performance if appending data to the RawDataHandle as no additional memory needs to be reserved then.
- SPECTRALRADAR_API void resizeRawData (RawDataHandle Raw, int Size1, int Size2, int Size3)

  Resizes the specified raw data buffer accordingly.

- SPECTRALRADAR_API void setRawDataContent (RawDataHandle RawData, void ∗NewContent)

  Sets the content of the raw data buffer. The size of the RawDataHandle needs to be adjusted first, as otherwise not all data might be copied.
- SPECTRALRADAR_API void setScanSpectra (RawDataHandle RawData, int NumberOfScanRegions, int ∗ScanRegions)

  Notice that raw data refers to the spectra as acquired, without processing of any kind.

  .
- SPECTRALRADAR_API void setApodizationSpectra (RawDataHandle RawData, int NumberOfApoRegions, int ∗ApodizationRegions)

  Notice that raw data refers to the spectra as acquired, without processing of any kind.
- SPECTRALRADAR_API int getNumberOfScanRegions (RawDataHandle Raw)

  Returns the number of regions that have been acquired that contain scan data, i. e. spectra that are used to compute A-scans.
- SPECTRALRADAR_API int getNumberOfApodizationRegions (RawDataHandle Raw)

  Returns the number of regions in the raw data containing spectra that are supposed to be used for apodization.
- SPECTRALRADAR_API void getScanSpectra (RawDataHandle Raw, int ∗SpectraIndex)

  Returns the indices of spectra that contain scan data, i. e. spectra that are supposed to be used to compute A-scans.
- SPECTRALRADAR_API void getApodizationSpectra (RawDataHandle Raw, int ∗SpectraIndex)

  Returns the indices of spectra that contain apodization data, i. e. spectra that are supposed to be used as input for apodization.
- SPECTRALRADAR_API RawDataHandle createRawData (void)

  Notice that raw data refers to the spectra as acquired, without processing of any kind.
- SPECTRALRADAR_API void clearRawData (RawDataHandle Raw)

  Notice that raw data refers to the spectra as acquired, without processing of any kind.
- SPECTRALRADAR_API DataHandle createData (void)

  Creates a 1-dimensional data object, containing floating point data.
- SPECTRALRADAR_API DataHandle createGradientData (int Size)

  Creates a 1-dimensional data object, containing floating point data with equidistant arranged values between [0, size-1] with distance 1/(size-1).
- SPECTRALRADAR_API void clearData (DataHandle Data)

  Clears the specified DataHandle object.
- SPECTRALRADAR_API ColoredDataHandle createColoredData (void)

  Creates a colored data object (ColoredDataHandle).
- SPECTRALRADAR_API void clearColoredData (ColoredDataHandle Volume)

  Clears a colored volume object.
- SPECTRALRADAR_API ComplexDataHandle createComplexData (void)

  Creates a data object holding complex data (ComplexDataHandle).
- SPECTRALRADAR_API void clearComplexData (ComplexDataHandle Data)

  Clears a data object holding complex data (ComplexDataHandle).
- SPECTRALRADAR_API OCTDeviceHandle initDevice (void)

  Initializes the installed device.
- SPECTRALRADAR_API int getDevicePropertyInt (OCTDeviceHandle Dev, DevicePropertyInt Selection)

  Returns properties of the device belonging to the specfied OCTDeviceHandle.
- SPECTRALRADAR_API const char ∗ getDevicePropertyString (OCTDeviceHandle Dev, DevicePropertyString Selection)

  Returns properties of the device belonging to the specfied OCTDeviceHandle.
- SPECTRALRADAR_API double getDevicePropertyFloat (OCTDeviceHandle Dev, DevicePropertyFloat Selection)

  Returns properties of the device belonging to the specfied OCTDeviceHandle.
- SPECTRALRADAR_API BOOL getDeviceFlag (OCTDeviceHandle Dev, DeviceFlag Selection)

  Returns properties of the device belonging to the specified OCTDeviceHandle.
- SPECTRALRADAR_API void setDeviceFlag (OCTDeviceHandle Dev, DeviceFlag Selection, BOOL Value)

Sets the selcted flag of the device belonging to the specified OCTDeviceHandle.

- SPECTRALRADAR_API void closeDevice (OCTDeviceHandle Dev)

  Closes the device opened previously with initDevice.
- SPECTRALRADAR_API void moveScanner (OCTDeviceHandle Dev, ProbeHandle Probe, ScanAxis Axis, double Position_mm)

  Manually moves the scanner to a given position.
- SPECTRALRADAR_API void moveScannerToApoPosition (OCTDeviceHandle Dev, ProbeHandle Probe)

  Moves the scanner to the apodization position.
- SPECTRALRADAR_API int getNumberOfDevicePresetCategories (OCTDeviceHandle Dev)

  If the hardware supports multiple presets, the funciton returns the number of categories in which presets can be set.
- SPECTRALRADAR_API const char ∗ getDevicePresetCategoryName (OCTDeviceHandle Dev, int Category)

  Gets a descriptor/name for the respective preset category.
- SPECTRALRADAR_API int getDevicePresetCategoryIndex (OCTDeviceHandle Dev, const char ∗Name)

  Gets the index of a preset category from the name of the category.
- SPECTRALRADAR_API void setDevicePreset (OCTDeviceHandle Dev, int Category, ProbeHandle Probe, ProcessingHandle Proc, int Preset)

  Sets the preset of the device. Using presets the sensitivity and acquisition speed of the device can be influenced.
- SPECTRALRADAR_API int getDevicePreset (OCTDeviceHandle Dev, int Category)

  Gets the currently used device preset.
- SPECTRALRADAR_API const char ∗ getDevicePresetDescription (OCTDeviceHandle Dev, int Category, int Preset)

  Returns a description of the selected device preset. Using the description more information about sensitivity and acquisition speed of the respective set can be found.
- SPECTRALRADAR_API int getNumberOfDevicePresets (OCTDeviceHandle Dev, int Category)

  Returns the number of available device presets.
- SPECTRALRADAR_API void setRequiredSLDOnTime_s (int Time_s)

  Sets the time the SLD needs to be switched on before any measurement can be started. Default is 3 seconds.
- SPECTRALRADAR_API void resetCamera (void)

  Resets the spectrometer camera.
- SPECTRALRADAR_API BOOL isDeviceAvailable (void)

  Returns whethter any supported Base-Unit is available.
- SPECTRALRADAR_API int getNumberOfInternalDeviceValues (OCTDeviceHandle Dev)

  Returns the number of Analog-to-Digital Converter present in the device.
- SPECTRALRADAR_API void getInternalDeviceValueName (OCTDeviceHandle Dev, int Index, char ∗Name, int NameStringSize, char ∗Unit, int UnitStringSize)

  Returns names and unit for the specified Analog-to-Digital Converter.
- SPECTRALRADAR_API double getInternalDeviceValueByName (OCTDeviceHandle Dev, const char ∗Name)

  Returns the value of the specified Analog-to-Digital Converter (ADC);.
- SPECTRALRADAR_API double getInternalDeviceValueByIndex (OCTDeviceHandle Dev, int Index)

  Returns the value of the selected ADC.
- SPECTRALRADAR_API void setInternalDeviceValueByIndex (OCTDeviceHandle Dev, int Index, double Value)

  Sets the value of the selected ADC.
- SPECTRALRADAR_API ProbeHandle initProbe (OCTDeviceHandle Dev, const char ∗ProbeFile)

  Initializes a probe specified by ProbeFile.
- SPECTRALRADAR_API ProbeHandle initDefaultProbe (OCTDeviceHandle Dev, const char ∗Type, const char ∗Objective)

  Creates a standard probe using standard parameters for the specified probe type.
- SPECTRALRADAR_API ProbeHandle initProbeFromOCTFile (OCTDeviceHandle Dev, OCTFileHandle File)

  Creates a probe using the parameters from the specified OCT file.
- SPECTRALRADAR_API void saveProbe (ProbeHandle Probe, const char ∗ProbeFile)

Saves the current properties of the ProbeHandle to a specified INI file to be reloaded using the initProbe() function.

- SPECTRALRADAR_API void setProbeParameterInt (ProbeHandle Probe, ProbeParameterInt Selection, int Value)

  Sets integer parameter of the specified probe.
- SPECTRALRADAR_API void setProbeParameterFloat (ProbeHandle Probe, ProbeParameterFloat Selection, double Value)

  Sets floating point parameters of the specified probe.
- SPECTRALRADAR_API int getProbeParameterInt (ProbeHandle Probe, ProbeParameterInt Selection)

  Gets integer parameters of the specified probe.
- SPECTRALRADAR_API double getProbeParameterFloat (ProbeHandle Probe, ProbeParameterFloat Selection)

  Gets floating point parameters of the specified probe.
- SPECTRALRADAR_API BOOL getProbeFlag (ProbeHandle Probe, ProbeFlag Selection)

  Returns the selected boolean value of the specified probe.
- SPECTRALRADAR_API void setProbeParameterString (ProbeHandle Probe, ProbeParameterString Selection, const char ∗Value)

  Sets a string property of the specified probe.
- SPECTRALRADAR_API const char ∗ getProbeParameterString (ProbeHandle Probe, ProbeParameterString Selection)

  Gets the desired string property of the specified probe.
- SPECTRALRADAR_API const char ∗ getProbeType (ProbeHandle Probe)

  Gets the type of the specified probe.
- SPECTRALRADAR_API void setProbeType (ProbeHandle Probe, const char ∗Type)

  Sets the type of the specified probe.
- SPECTRALRADAR_API void closeProbe (ProbeHandle Probe)

  Closes the probe and frees all memory associated with it.
- SPECTRALRADAR_API void CameraPixelToPosition (ProbeHandle Probe, ColoredDataHandle Image, int PixelX, int PixelY, double ∗PosX, double ∗PosY)

  Computes the physical position of a camera pixel of the video camera in the probe. It assumes a properly calibrated device.
- SPECTRALRADAR_API void PositionToCameraPixel (ProbeHandle Probe, ColoredDataHandle Image, double PosX, double PosY, int ∗PixelX, int ∗PixelY)

  Computes the pixel of the video camera corresponding to a physical position. It needs to be assured that the device is properly calibrated.
- SPECTRALRADAR_API void visualizeScanPatternOnDevice (OCTDeviceHandle Dev, ProbeHandle Probe, ScanPatternHandle Pattern, BOOL ShowRawPattern)

  Visualizes the scan pattern on top of the camera image; if appropriate hardware is used for visualization.
- SPECTRALRADAR_API void visualizeScanPatternOnImage (ProbeHandle Probe, ScanPatternHandle ScanPattern, ColoredDataHandle VideoImage)

  Visualizes the scan pattern on top of the camera image; scan pattern data is written into the image.
- SPECTRALRADAR_API ScanPatternHandle createNoScanPattern (ProbeHandle Probe, int AScans, int NumberOfScans)

  Creates a simple scan pattern that does not move the galvo. Use this pattern for point scans and/or non-scanning probes. The pattern will however use a specified amount of trigger signals. For continuous acquisition use Number↩ OfScans set to 1.
- SPECTRALRADAR_API ScanPatternHandle createAScanPattern (ProbeHandle Probe, int AScans, double PosX_mm, double PosY_mm)

  Creates a scan pattern used to acquire a specific amount of Ascans at a specific position.
- SPECTRALRADAR_API ScanPatternHandle createBScanPattern (ProbeHandle Probe, double Range_mm, int AScans)

  Creates a horizontal rectilinear-segment B-scan pattern that moves the galvo over a specified range.
- SPECTRALRADAR_API ScanPatternHandle createBScanPatternManual (ProbeHandle Probe, double StartX_mm, double StartY_mm, double StopX_mm, double StopY_mm, int AScans)

Creates a B-scan pattern specified by start and end points.

- SPECTRALRADAR_API ScanPatternHandle createIdealBScanPattern (ProbeHandle Probe, double Range_mm, int AScans)

  Creates an ideal B-scan pattern assuming scanners with infinite speed. No correction factors are taken into account. This is only used for internal purposes and not as a scan pattern designed to be output to the galvo drivers.

- SPECTRALRADAR_API ScanPatternHandle createCirclePattern (ProbeHandle Probe, double Radius_mm, int AScans)

  Creates a circle scan pattern.

- SPECTRALRADAR_API ScanPatternHandle createVolumePattern (ProbeHandle Probe, double Range↩X_mm, int SizeX, double RangeY_mm, int SizeY, ScanPatternApodizationType ApoType, ScanPattern↩AcquisitionOrder AcqOrder)

  Creates a simple volume pattern.

- SPECTRALRADAR_API ScanPatternHandle createVolumePatternEx (ProbeHandle Probe, double Range↩X_mm, int SizeX, double RangeY_mm, int SizeY, double CenterX_mm, double CenterY_mm, double Angle↩_rad, ScanPatternApodizationType ApoType, ScanPatternAcquisitionOrder AcqOrder)

  Creates a simple volume pattern.

- SPECTRALRADAR_API void updateScanPattern (ScanPatternHandle Pattern)

  Updates the specfied pattern (ScanPatternHandle) and computes the full look-up-table.

- SPECTRALRADAR_API void rotateScanPattern (ScanPatternHandle Pattern, double Angle_rad)

  Rotates the specfied pattern (ScanPatternHandle), counter-clockwise. The rotation is relative to current angle, not to the horizontal. That is, after multiple invokations of this function the final rotation is the addition of all rotations.

- SPECTRALRADAR_API void rotateScanPatternEx (ScanPatternHandle Pattern, double Angle_rad, int Index)

  Counter-clockwise rotates the scan `Index` (0-based, i.e. zero for the first, one for the second, and so on) of the specfied volume scan pattern (ScanPatternHandle). The rotation is relative to current angle, not to the horizontal. That is, after multiple invokations of this function the final rotation is the addition of all rotations.

- SPECTRALRADAR_API void shiftScanPattern (ScanPatternHandle Pattern, double ShiftX_mm, double ShiftY_mm)

  Shifts the specified pattern (ScanPatternHandle). The shift is relative to current position, not to (0,0). That is, after multiple invokations of this function the final shift is the addition of all shifts.

- SPECTRALRADAR_API void shiftScanPatternEx (ScanPatternHandle Pattern, double ShiftX_mm, double ShiftY_mm, BOOL ShiftApo, int Index)

  Shifts the scan `Index` (0-based, i.e. zero for the first, one for the second, and so on) of the specified volume pattern (ScanPatternHandle). The shift is relative to current position, not to (0,0). That is, after multiple invokations of this function the final shift is the addition of all shifts.

- SPECTRALRADAR_API void zoomScanPattern (ScanPatternHandle Pattern, double Factor)

  Zooms the specified pattern (ScanPatternHandle) around the optical center that coincides with the center of the camera image and the physical coordinates (0 mm,0 mm). The apodization position will not be modified.

- SPECTRALRADAR_API int getScanPatternLUTSize (ScanPatternHandle Pattern)

  Returns the number of points in the specified scan pattern (ScanPatternHandle), including apodization and flyback.

- SPECTRALRADAR_API void getScanPatternLUT (ScanPatternHandle Pattern, double ∗VoltX, double ∗VoltY)

  Returns the voltages that will be applied to reach the positions to be scanned, in the specified scan pattern (Scan↩PatternHandle).

- SPECTRALRADAR_API int getScanPointsSize (ScanPatternHandle Pattern)

  Returns the number of points in the specified scan pattern (ScanPatternHandle), including apodization and flyback.

- SPECTRALRADAR_API void getScanPoints (ScanPatternHandle Pattern, double ∗PosX_mm, double ∗PosY_mm)

  Returns the position coordinates (in mm) of the points that in the specified scan pattern (ScanPatternHandle).

- SPECTRALRADAR_API void clearScanPattern (ScanPatternHandle Pattern)

  Clears the specified scan pattern (ScanPatternHandle).

- SPECTRALRADAR_API ScanPatternHandle createFreeformScanPattern2D (ProbeHandle Probe, double ∗PosX_mm, double ∗PosY_mm, int Size, int AScans, InterpolationMethod InterpolationMethod, BOOL CloseScanPattern)

  Creates a B-scan scan pattern of arbitrary form with equidistant sampled scan points.

- SPECTRALRADAR_API ScanPatternHandle createFreeformScanPattern2DFromLUT (ProbeHandle Probe, double ∗PosX_mm, double ∗PosY_mm, int Size, BOOL ClosedScanPattern)

  Creates a B-scan scan pattern of arbitrary form with the specified scan points. The voltages array is taken as-is, so care must be taken to use sensible values with regard to the capabilities of the utilized scanner system and to the resolution of the system resp. the desired resolution of your scan pattern.

- SPECTRALRADAR_API ScanPatternHandle createFreeformScanPattern3DFromLUT (ProbeHandle Probe, double ∗PosX_mm, double ∗PosY_mm, int AScansPerBScan, int NumberOfBScans, BOOL ClosedScan←Pattern, ScanPatternApodizationType ApoType, ScanPatternAcquisitionOrder AcqOrder)

  Creates a volume scan pattern of arbitrary form with the specified scan voltages. The voltages array is taken as-is, so care must be taken to use sensible values with regard to the capabilities of the utilized scanner system and to the resolution of the system resp. the desired resolution of your scan pattern. With this function the definition of each single scan point is required. In order to create a scan pattern specifying only the end coordinates, please consider createFreeformScanPattern3D.

- SPECTRALRADAR_API ScanPatternHandle createFreeformScanPattern3D (ProbeHandle Probe, double ∗PosX_mm, double ∗PosY_mm, int ∗ScanIndices, int Size, int NumberOfAScansPerBScan, Interpolation←Method InterpolationMethod, BOOL CloseScanPattern, ScanPatternApodizationType ApoType, Scan←PatternAcquisitionOrder AcqOrder)

  Creates a volume scan pattern of arbitrary form with equidistant sampled scan points.

- SPECTRALRADAR_API void interpolatePoints2D (double ∗OrigPosX, double ∗OrigPosY, int Size, double ∗InterpPosX, double ∗InterpPosY, int NewSize, InterpolationMethod InterpolationMet, BoundaryCondition BoundaryCond)

  Interpolates the imaginary curve defined by the given sequence of points with the specified InterpolationMethod. The coordinates are abstract and this funcion has no sideffects that could affect any physical property. The original and the interpolated coordinates have a meaning for the user, but no consequence for SpectralRadar.

- SPECTRALRADAR_API void inflatePoints (double ∗PosX, double ∗PosY, int Size, double ∗InflatedPosX, double ∗InflatedPosY, int NumberOfInflationLines, double RangeOfInflation, InflationMethod Method)

  Inflates the provided curve in space with the specified InflationMethod. It can be used to create scan patterns of arbitrary forms with createFreeformScanPattern3DFromLUT if the used positions correspond to coordinates of the valid scan field in mm.

- SPECTRALRADAR_API void saveScanPointsToFile (double ∗ScanPosX_mm, double ∗ScanPosY_mm, int ∗ScanIndices, int Size, const char ∗Filename, ScanPointsDataFormat DataFormat)

  Saves the scan points and scan indices to a file with the specified ScanPointsDataFormat.

- SPECTRALRADAR_API int getSizeOfScanPointsFromFile (const char ∗Filename, ScanPointsDataFormat DataFormat)

  Returns the number of scan points in the specified file.

- SPECTRALRADAR_API void loadScanPointsFromFile (double ∗ScanPosX_mm, double ∗ScanPosY_mm, int ∗ScanIndices, int Size, const char ∗Filename, ScanPointsDataFormat DataFormat)

  Copies the scan points and scan indices from the file to the provided arrays.

- SPECTRALRADAR_API int getSizeOfScanPointsFromDataHandle (DataHandle ScanPoints)

  Returns the size of the scan points and scan indices in the DataHandle.

- SPECTRALRADAR_API void getScanPointsFromDataHandle (DataHandle ScanPoints, double ∗PosX_mm, double ∗PosY_mm, int ∗ScanIndices, int Length)

  Copies the scan points and scan indices from the DataHandle to the provided arrays.

- SPECTRALRADAR_API DataHandle createDataHandleFromScanPoints (double ∗PosX_mm, double ∗Pos←Y_mm, int ∗ScanIndices, int Length)

  Creates a DataHandle from the specified scan points and corresponding indices.

- SPECTRALRADAR_API size_t projectMemoryRequirement (OCTDeviceHandle Handle, ScanPatternHandle Pattern, AcquisitionType type)

  Returns the size of the required memory, e.g. for a raw data object, in bytes to acquire the scan pattern once.

- SPECTRALRADAR_API void startMeasurement (OCTDeviceHandle Dev, ScanPatternHandle Pattern, AcquisitionType Type)

  starts a continuous measurement BScans.

- SPECTRALRADAR_API void getRawData (OCTDeviceHandle Dev, RawDataHandle RawData)

  Acquires data and stores the data unprocessed.

- SPECTRALRADAR_API void getRawDataEx (OCTDeviceHandle Dev, RawDataHandle RawData, int CameraIdx)

  Acquires data with the specific camera given with camera index and stores the data unprocessed.
- SPECTRALRADAR_API void stopMeasurement (OCTDeviceHandle Dev)

  stops the current measurement.
- SPECTRALRADAR_API void measureSpectra (OCTDeviceHandle Dev, int NumberOfSpectra, RawDataHandle Raw)

  Acquires the desired number of spectra (raw data without processing) without moving galvo scanners.
- SPECTRALRADAR_API void measureSpectraEx (OCTDeviceHandle Dev, int NumberOfSpectra, RawDataHandle Raw, int CameraIndex)

  Acquires the desired number of spectra (raw data without processing) without moving galvo scanners, for the desired camera.
- SPECTRALRADAR_API ProcessingHandle createProcessing (int SpectrumSize, int BytesPerRawPixel, BOOL Signed, float ScalingFactor, float MinElectrons, Processing_FFTType Type, float FFTOversampling)

  Creates processing routines with the desired properties.
- SPECTRALRADAR_API ProcessingHandle createProcessingForDevice (OCTDeviceHandle Dev)

  Creates processing routines for the specified device (OCTDeviceHandle).
- SPECTRALRADAR_API ProcessingHandle createProcessingForDeviceEx (OCTDeviceHandle Dev, int CameraIndex)

  Creates processing routines for the specified device (OCTDeviceHandle) with camera index.
- SPECTRALRADAR_API ProcessingHandle createProcessingForOCTFile (OCTFileHandle File)

  Creates processing routines for the specified OCT file (OCTFileHandle), such that the processing conditions are exactly the same as those when the file had been saved.
- SPECTRALRADAR_API ProcessingHandle createProcessingForOCTFileEx (OCTFileHandle File, const int CameraIndex)

  Creates processing routines for the specified OCT file (OCTFileHandle), such that the processing conditions are exactly the same as those when the file had been saved.
- SPECTRALRADAR_API int getInputSize (ProcessingHandle Proc)

  Returns the expected input size (pixels per spectrum) of the processing algorithms.
- SPECTRALRADAR_API int getAScanSize (ProcessingHandle Proc)

  Returns the number of pixels in an A-Scan that can be obtained (computed) with the given processing routines.
- SPECTRALRADAR_API void setApodizationWindow (ProcessingHandle Proc, ApodizationWindow Window)

  Sets the windowing function that will be used for apodization (this apodization has nothing to do with the reference spectra measured without a sample!). The selected windowing function will be used in all subsequent processings right before the fast Fourier transformation.
- SPECTRALRADAR_API ApodizationWindow getApodizationWindow (ProcessingHandle Proc)

  Returns the current windowing function that is being used for apodization, ApodizationWindow (this apodization is not the reference spectrum measured without a sample!).
- SPECTRALRADAR_API void setApodizationWindowParameter (ProcessingHandle Proc, ApodizationWindowParameter Selection, double Value)

  Sets the apodization window parameter, such as window width or ratio between constant and cosine part. Notice that this apodization is unrelated to the reference spectrum measured without a sample!.
- SPECTRALRADAR_API double getApodizationWindowParameter (ProcessingHandle Proc, ApodizationWindowParameter Selection)

  Gets the apodization window parameter, such as window width or ratio between constant and cosine part. Notice that this apodization is unrelated to the reference spectrum measured without a sample!.
- SPECTRALRADAR_API void getCurrentApodizationEdgeChannels (ProcessingHandle Proc, int ∗LeftPix, int ∗RightPix)

  Returns the pixel positions of the left/right edge channels of the current apodization. Here apodization refers to the reference spectra measured without sample.
- SPECTRALRADAR_API void setProcessingDechirpAlgorithm (ProcessingHandle Proc, Processing_FFTType Type, float Oversampling)

  Sets the algorithm to be used for dechirping the input spectra.

- SPECTRALRADAR_API void setProcessingParameterInt (ProcessingHandle Proc, ProcessingParameterInt Selection, int Value)

  Sets the specified integer value processing parameter.
- SPECTRALRADAR_API int getProcessingParameterInt (ProcessingHandle Proc, ProcessingParameterInt Selection)

  Returns the specified integer value processing parameter.
- SPECTRALRADAR_API void setProcessingParameterFloat (ProcessingHandle Proc, Processing←ParameterFloat Selection, double Value)

  Sets the specified floating point processing parameter.
- SPECTRALRADAR_API double getProcessingParameterFloat (ProcessingHandle Proc, Processing←ParameterFloat Selection)

  Gets the specified floating point processing parameter.
- SPECTRALRADAR_API void setProcessingFlag (ProcessingHandle Proc, ProcessingFlag Flag, BOOL Value)

  Sets the specified processing flag.
- SPECTRALRADAR_API BOOL getProcessingFlag (ProcessingHandle Proc, ProcessingFlag Flag)

  Returns TRUE if the specified processing flag is set, FALSE otherwise.
- SPECTRALRADAR_API void setProcessingAveragingAlgorithm (ProcessingHandle Proc, Processing←AveragingAlgorithm Algorithm)

  Sets the algorithm that will be used for averaging during the processing.
- SPECTRALRADAR_API void setCalibration (ProcessingHandle Proc, CalibrationData Selection, DataHandle Data)

  Sets the calibration data.
- SPECTRALRADAR_API void getCalibration (ProcessingHandle Proc, CalibrationData Selection, DataHandle Data)

  Retrieves the desired calibration vector.
- SPECTRALRADAR_API void measureCalibration (OCTDeviceHandle Dev, ProcessingHandle Proc, CalibrationData Selection)

  Measures the specified calibration parameters and uses them in subsequent processing.
- SPECTRALRADAR_API void measureCalibrationEx (OCTDeviceHandle Dev, ProcessingHandle Proc, CalibrationData Selection, int CameraIndex)

  Measures the specified calibration parameters and uses them in subsequent processing with specified camera index.
- SPECTRALRADAR_API void measureApodizationSpectra (OCTDeviceHandle Dev, ProbeHandle Probe, ProcessingHandle Proc)

  Measures the apodization spectra in the defined apodization position and size and uses them in subsequent processing.
- SPECTRALRADAR_API void saveCalibrationDefault (ProcessingHandle Proc, CalibrationData Selection)

  Saves the selected calibration in its default path. This same default path will be used by SpectralRadar in subsequent executions to retrieve the calibration data.
- SPECTRALRADAR_API void saveCalibrationDefaultEx (ProcessingHandle Proc, CalibrationData Selection, int CameraIndex)

  Saves the selected calibration in its default path, for the selected camera. This same default path will be used by SpectralRadar in.
- SPECTRALRADAR_API void saveCalibration (ProcessingHandle Proc, CalibrationData Selection, const char ∗Path)

  Saves the selected calibration in the specified path.
- SPECTRALRADAR_API void loadCalibration (ProcessingHandle Proc, CalibrationData Selection, const char ∗Path)

  Will load a specified calibration file and its content will be used for subsequent processing.
- SPECTRALRADAR_API void setSpectrumOutput (ProcessingHandle Proc, DataHandle Spectrum)

  Sets the location for the resulting spectral data.
- SPECTRALRADAR_API void setOffsetCorrectedSpectrumOutput (ProcessingHandle Proc, DataHandle OffsetCorrectedSpectrum)

Sets the location for the resulting offset corrected spectral data.

- SPECTRALRADAR_API void setDCCorrectedSpectrumOutput (ProcessingHandle Proc, DataHandle DC↩ CorrectedSpectrum)

  Sets the location for the resulting DC removed spectral data.

- SPECTRALRADAR_API void setApodizedSpectrumOutput (ProcessingHandle Proc, DataHandle ApodizedSpectrum)

  Sets the location for the resulting apodized spectral data.

- SPECTRALRADAR_API void setComplexDataOutput (ProcessingHandle Proc, ComplexDataHandle ComplexScan)

  Sets the pointer to the resulting complex scans that will be written after subsequent processing executions.

- SPECTRALRADAR_API void setProcessedDataOutput (ProcessingHandle Proc, DataHandle Scan)

  Sets the pointer to the resulting scans that will be written after subsequent processing executions.

- SPECTRALRADAR_API void setColoredDataOutput (ProcessingHandle Proc, ColoredDataHandle Scan, ColoringHandle Color)

  Sets the pointer to the resulting colored scans that will be written after subsequent processing executions.

- SPECTRALRADAR_API void setTransposedColoredDataOutput (ProcessingHandle Proc, ColoredData↩ Handle Scan, ColoringHandle Color)

  Sets the pointer to the resulting colored scans that will be written after subsequent processing executions. The orientation of the colored data will be transposed in such a way that the first axis (normally z-axis) will be the x-axis (the depth of each individual A-scan) and the second axis (normally x-axis) will be the z-axis.

- SPECTRALRADAR_API void executeProcessing (ProcessingHandle Proc, RawDataHandle RawData)

  Executes the processing. The results will be stored as requested through the functions setProcessedDataOutput(), setComplexDataOutput(), setColoredDataOutput() (including coloring properties) and similar ones. In all cases, sizes and ranges will be adjusted automatically to the right values.

- SPECTRALRADAR_API void clearProcessing (ProcessingHandle Proc)

  Clears the processing instance and frees all temporary memory that was associated with it. Processing threads will be stopped.

- SPECTRALRADAR_API void computeDispersion (DataHandle Spectrum1, DataHandle Spectrum2, Data↩ Handle Chirp, DataHandle Disp)

  Computes the dispersion and chirp of the two provided spectra, where both spectra need to have been subjected to same dispersion mismatch. Both spectra need to have been acquired for different path length differences.

- SPECTRALRADAR_API void computeDispersionByCoeff (double Quadratic, DataHandle Chirp, DataHandle Disp)

  Computes dispersion by a quadratic approximation specified by the quadratic factor.

- SPECTRALRADAR_API void computeDispersionByImage (DataHandle LinearKSpectra, DataHandle Chirp, DataHandle Disp)

  Guesses the dispersion based on the spectral data specified. The spectral data needs to be linearized in wavenumber before using this function.

- SPECTRALRADAR_API int getNumberOfDispersionPresets (ProcessingHandle Proc)

  Gets the number of dispersion presets.

- SPECTRALRADAR_API const char ∗ getDispersionPresetName (ProcessingHandle Proc, int Index)

  Gets the name of the dispersion preset specified with index.

- SPECTRALRADAR_API void setDispersionPresetByName (ProcessingHandle Proc, const char ∗Name)

  Sets the dispersion preset specified with name.

- SPECTRALRADAR_API void setDispersionPresetByIndex (ProcessingHandle Proc, int Index)

  Sets the dispersion preset specified with index.

- SPECTRALRADAR_API void setDispersionPresets (ProcessingHandle Proc, ProbeHandle Probe)

  Sets the dispersion presets for the probe.

- SPECTRALRADAR_API Processing_FFTType getProcessing_FFTType (ProcessingHandle Proc)

  Retrieve the active FFT Type.

- SPECTRALRADAR_API void setDispersionCorrectionType (ProcessingHandle Proc, DispersionCorrection↩ Type Type)

  Sets the active dispersion correction type.

- SPECTRALRADAR_API DispersionCorrectionType getDispersionCorrectionType (ProcessingHandle Proc)

  Sets the active dispersion correction type.
- SPECTRALRADAR_API void setDispersionQuadraticCoeff (ProcessingHandle Proc, double Coeff)

  Sets the coefficient for the quadratic correction of the dispersion.
- SPECTRALRADAR_API double getDispersionQuadraticCoeff (ProcessingHandle Proc)

  Sets the coefficient for the quadratic correction of the dispersion.
- SPECTRALRADAR_API const char ∗ getCurrentDispersionPresetName (ProcessingHandle Proc)

  Gets the name of the active dispersion preset.
- SPECTRALRADAR_API void exportData (DataHandle Data, DataExportFormat Format, const char ∗FileName)

  Exports data (DataHandle) to a file. The number of dimensions is handled automatically upon analysis of the first argument.
- SPECTRALRADAR_API void exportDataAsImage (DataHandle Data, ColoringHandle Color, ColoredDataExportFormat Format, Direction SliceNormalDirection, const char ∗FileName, int ExportOptionMask)

  Exports 2-dimensional and 3-dimensional data (DataHandle) as image data (such as BMP, PNG, JPEG, ...).
- SPECTRALRADAR_API void exportComplexData (ComplexDataHandle Data, ComplexDataExportFormat Format, const char ∗FileName)

  Exports 1-, 2- and 3-dimensional complex data (ComplexDataHandle)
- SPECTRALRADAR_API void exportColoredData (ColoredDataHandle Data, ColoredDataExportFormat Format, Direction SliceNormalDirection, const char ∗FileName, int ExportOptionMask)

  Exports colored data (ColoredDataHandle).
- SPECTRALRADAR_API void importColoredData (ColoredDataHandle ColoredData, DataImportFormat Format, const char ∗FileName)

  Imports colored data (ColoredDataHandle) with the specified format and copied it into a data object (ColoredDataHandle)
- SPECTRALRADAR_API void importData (DataHandle Data, DataImportFormat Format, const char ∗FileName)

  Imports data with the specified format and copies it into a data object (DataHandle).
- SPECTRALRADAR_API void exportRawData (RawDataHandle Raw, RawDataExportFormat Format, const char ∗FileName)

  Exports the specified data to disk.
- SPECTRALRADAR_API void importRawData (RawDataHandle Raw, RawDataImportFormat Format, const char ∗FileName)

  Imports the specified data from disk.
- SPECTRALRADAR_API void appendRawData (RawDataHandle Raw, RawDataHandle DataToAppend, Direction Dir)

  Appends the new raw data to the old raw data perpendicular to the specified direction.
- SPECTRALRADAR_API void getRawDataSliceAtIndex (RawDataHandle Raw, RawDataHandle Slice, Direction SliceNormalDirection, int Index)

  Returns a slice of raw data perpendicular to the specified direction at the specified index.
- SPECTRALRADAR_API double analyzeData (DataHandle Data, DataAnalyzation Selection)

  Analyzes the given data, extracts the selected feature, and returns the computed value.
- SPECTRALRADAR_API double analyzeAScan (DataHandle Data, AScanAnalyzation Selection)

  Analyzes the given A-scan data, extracts the selected feature, and returns the computed value.
- SPECTRALRADAR_API void transposeData (DataHandle DataIn, DataHandle DataOut)

  Transposes the given data and writes the result to DataOut. First and second axes will be swaped.
- SPECTRALRADAR_API void transposeDataInplace (DataHandle Data)

  Transposes the given Data. First and second axes will be swaped.
- SPECTRALRADAR_API void transposeAndScaleData (DataHandle DataIn, DataHandle DataOut, float Min, float Max)

  Transposes the given data and writes the result to DataOut. First and second axes will be swaped, and the range of the entries will be scaled in such a way, that the range [Min,Max] will be mapped onto the range [0,1].
- SPECTRALRADAR_API void normalizeData (DataHandle Data, float Min, float Max)

Scales the given data in such a way, that the range [Min, Max] is mapped onto the range [0,1].

- SPECTRALRADAR_API void getDataSliceAtPos (DataHandle Data, DataHandle Slice, Direction SliceNormalDirection, double Pos_mm)

  Returns a slice of data perpendicular to the specified direction at the specified position.
- SPECTRALRADAR_API void getComplexDataSlicePos (ComplexDataHandle Data, ComplexDataHandle Slice, Direction SliceNormalDirection, double Pos_mm)

  Returns a slice of complex data perpendicular to the specified direction at the specified position.
- SPECTRALRADAR_API void getColoredDataSlicePos (ColoredDataHandle Data, ColoredDataHandle Slice, Direction SliceNormalDirection, double Pos_mm)

  Returns a slice of colored data perpendicular to the specified direction at the specified position.
- SPECTRALRADAR_API void getDataSliceAtIndex (DataHandle Data, DataHandle Slice, Direction SliceNormalDirection, int Index)

  Returns a slice of data perpendicular to the specified direction at the specified index.
- SPECTRALRADAR_API void getComplexDataSliceIndex (ComplexDataHandle Data, ComplexDataHandle Slice, Direction SliceNormalDirection, int Index)

  Returns a slice of complex data perpendicular to the specified direction at the specified index.
- SPECTRALRADAR_API void getColoredDataSliceIndex (ColoredDataHandle Data, ColoredDataHandle Slice, Direction SliceNormalDirection, int Index)

  Returns a slice of colored data perpendicular to the specified direction at the specified index.
- SPECTRALRADAR_API void computeDataProjection (DataHandle Data, DataHandle Slice, Direction ProjectionDirection, DataAnalyzation Selection)

  Returns a single slice of data, in which each pixel value is the feature extracted through an analysis along the specified direction.
- SPECTRALRADAR_API void appendData (DataHandle Data, DataHandle DataToAppend, Direction Dir)

  Appends the new data to the provided data, perpendicular to the specified direction.
- SPECTRALRADAR_API void appendComplexData (ComplexDataHandle Data, ComplexDataHandle DataToAppend, Direction Dir)

  Appends the new data to the provided data, perpendicular to the specified direction.
- SPECTRALRADAR_API void appendColoredData (ColoredDataHandle Data, ColoredDataHandle DataToAppend, Direction Dir)

  Appends the new data to the provided data, perpendicular to the specified direction.
- SPECTRALRADAR_API void cropData (DataHandle Data, Direction Dir, int IndexMax, int IndexMin)

  Crops the data along the desired direction at the given indices. Upon return the data will only contain those slices whose indices where in the interval [IndexMin, IndexMax), counted along the cropping direction.
- SPECTRALRADAR_API void cropComplexData (ComplexDataHandle Data, Direction Dir, int IndexMax, int IndexMin)

  Crops the complex data along the desired direction at the given indices. Upon return the data will only contain those slices whose indices where in the interval [IndexMin, IndexMax), counted along the cropping direction.
- SPECTRALRADAR_API void cropColoredData (ColoredDataHandle Data, Direction Dir, int IndexMax, int IndexMin)

  Crops the colored data along the desired direction at the given indices. Upon return the data will only contain those slices whose indices where in the interval [IndexMin, IndexMax), counted along the cropping direction.
- SPECTRALRADAR_API void separateData (DataHandle Data1, DataHandle Data2, int SeparationIndex, Direction Dir)

  Separates the data at the given index at specific separation direction. The first part of the separated data will remain in Data1, the second separated in Data2.
- SPECTRALRADAR_API void separateComplexData (ComplexDataHandle Data1, ComplexDataHandle Data2, int SeparationIndex, Direction Dir)

  Separates the data at the given index at specific separation direction. The first part of the separated data will remain in Data1, the second separated in Data2.
- SPECTRALRADAR_API void separateColoredData (ColoredDataHandle Data1, ColoredDataHandle Data2, int SeparationIndex, Direction Dir)

  Separates the data at the given index at specific separation direction. The first part of the separated data will remain in Data1, the second separated in Data2.

- SPECTRALRADAR_API void flipData (DataHandle Data, Direction FlippingDir)

  Mirrors the data across a plane perpendicular to the given direction.
- SPECTRALRADAR_API void flipComplexData (ComplexDataHandle Data, Direction FlippingDir)

  Mirrors the data across a plane perpendicular to the given direction.
- SPECTRALRADAR_API void flipColoredData (ColoredDataHandle Data, Direction FlippingDir)

  Mirrors the data across a plane perpendicular to the given direction.
- SPECTRALRADAR_API ImageFieldHandle createImageField (void)

  Creates an object holding image field data.
- SPECTRALRADAR_API ImageFieldHandle createImageFieldFromProbe (ProbeHandle Probe)

  Creates an object holding image field data from the specified Probe Handle.
- SPECTRALRADAR_API void clearImageField (ImageFieldHandle ImageField)

  Frees an object holding image field data.
- SPECTRALRADAR_API void saveImageField (ImageFieldHandle ImageField, const char ∗Path)

  Saves data containing image field data.
- SPECTRALRADAR_API void loadImageField (ImageFieldHandle ImageField, const char ∗Path)

  Loads data containing image field data.
- SPECTRALRADAR_API void determineImageField (ImageFieldHandle ImageField, ScanPatternHandle Pattern, DataHandle Surface)

  Determines the image field correction for the given surface data, previously measured with the given scan pattern.
- SPECTRALRADAR_API void determineImageFieldWithMask (ImageFieldHandle ImageField, ScanPatternHandle Pattern, DataHandle Surface, DataHandle Mask)

  Determines the image field correction for the given surface data, previously measured with the given scan pattern. The positive entries of the mask determine the points that actually enter in the computation.
- SPECTRALRADAR_API void correctImageField (ImageFieldHandle ImageField, ScanPatternHandle Pattern, DataHandle Data)

  Applies the image field correction to the given B-Scan or volume data.
- SPECTRALRADAR_API void correctImageFieldComplex (ImageFieldHandle ImageField, ScanPatternHandle Pattern, ComplexDataHandle Data)

  Applies the image field correction to the complex B-Scan or volume complex data.
- SPECTRALRADAR_API void correctSurface (ImageFieldHandle ImageField, ScanPatternHandle Pattern, DataHandle Surface)

  Applies the image field correction to the given Surface. Surface must contain depth values as a function of x/y coordinates.
- SPECTRALRADAR_API void setImageFieldInProbe (ImageFieldHandle ImageField, ProbeHandle Probe)

  Sets the specified image field to the specified Probe handle. Notice that no probe file will be automatically saved.
- SPECTRALRADAR_API VisualCalibrationHandle createVisualCalibration (OCTDeviceHandle Device, double TargetCornerLength_mm, BOOL CheckAngle, BOOL SaveData)

  Creates handle used for visual calibration.
- SPECTRALRADAR_API void clearVisualCalibration (VisualCalibrationHandle Handle)

  Clear handle and frees all related memory.
- SPECTRALRADAR_API BOOL visualCalibrate_1st_CameraScaling (VisualCalibrationHandle Handle, ProbeHandle Probe, ColoredDataHandle Image)

  This is the first step in visual calibration. For this, the calibration the target needs to be placed under the objective. Returns TRUE if the first step succeeds.
- SPECTRALRADAR_API BOOL visualCalibrate_2nd_Galvo (VisualCalibrationHandle Handle, ProbeHandle Probe, ColoredDataHandle Image)

  This is the second step in visual calibration. For this, the calibration target or and infrared viewing card needs to be placed under the objective. Returns TRUE if the second step succeeds.
- SPECTRALRADAR_API BOOL visualCalibrate_previewImage (VisualCalibrationHandle Handle, ColoredDataHandle Image)

  Provides a preview image for the current calibration.
- SPECTRALRADAR_API void visualCalibration_getHoles (VisualCalibrationHandle Handle, int ∗x0, int ∗y0, int ∗x1, int ∗y1, int ∗x2, int ∗y2)

provides currently located hole positions of the three-hole target.

- SPECTRALRADAR_API const char ∗ visualCalibrate_Status (VisualCalibrationHandle Handle)

  Gives a status message of the currently executed visual calibration.
- SPECTRALRADAR_API DopplerProcessingHandle createDopplerProcessing (void)

  Returns a handle for the use of Doppler-computation routines.
- SPECTRALRADAR_API DopplerProcessingHandle createDopplerProcessingForFile (OCTFileHandle File)

  Returns a handle for the use of Doppler-computation routines. The handle is created based on a saved OCT file.
- SPECTRALRADAR_API int getDopplerPropertyInt (DopplerProcessingHandle Handle, DopplerPropertyInt Property)

  Gets the value of the given Doppler processing property.
- SPECTRALRADAR_API void setDopplerPropertyInt (DopplerProcessingHandle Handle, DopplerPropertyInt Property, int Value)

  Sets the value of the given Doppler processing property.
- SPECTRALRADAR_API double getDopplerPropertyFloat (DopplerProcessingHandle Doppler, DopplerPropertyFloat Property)

  Gets the value of the given Doppler processing property.
- SPECTRALRADAR_API void setDopplerPropertyFloat (DopplerProcessingHandle Handle, DopplerPropertyFloat Property, float Value)

  Sets the value of the given Doppler processing property.
- SPECTRALRADAR_API BOOL getDopplerFlag (DopplerProcessingHandle Handle, DopplerFlag Flag)

  Gets the given Doppler processing flag.
- SPECTRALRADAR_API void setDopplerFlag (DopplerProcessingHandle Handle, DopplerFlag Flag, BOOL OnOff)

  Sets the given Doppler processing flag.
- SPECTRALRADAR_API void setDopplerAmplitudeOutput (DopplerProcessingHandle Handle, DataHandle AmpOut)

  Sets the location of the resulting Doppler amplitude output.
- SPECTRALRADAR_API void setDopplerPhaseOutput (DopplerProcessingHandle Handle, DataHandle PhasesOut)

  Sets the location of the resulting Doppler phase output.
- SPECTRALRADAR_API void executeDopplerProcessing (DopplerProcessingHandle Handle, ComplexDataHandle Input)

  Executes the Doppler processing of the input data and returns phases and amplitudes.
- SPECTRALRADAR_API void dopplerPhaseToVelocity (DopplerProcessingHandle Doppler, DataHandle InOut)

  Scales phases computed by Doppler OCT to actual flow velocities in scan direction.
- SPECTRALRADAR_API void dopplerVelocityToPhase (DopplerProcessingHandle Doppler, DataHandle InOut)

  Scales flow velocities computed by Doppler OCT back to original phase differencees.
- SPECTRALRADAR_API void clearDopplerProcessing (DopplerProcessingHandle Handle)

  Closes the Doppler processing routines and frees the memory that has been allocated for these to work properly.
- SPECTRALRADAR_API void getDopplerOutputSize (DopplerProcessingHandle Handle, int Size1In, int Size2In, int ∗Size1Out, int ∗Size2Out)

  Returns the final size of the Doppler output if executeDopplerProcessing is executed using data of the specified input size.
- SPECTRALRADAR_API void calcContrast (DataHandle ApodizedSpectrum, DataHandle Contrast)

  Computes the contrast for the specified (apodized) spectrum.
- SPECTRALRADAR_API SettingsHandle initSettingsFile (const char ∗Path)

  Loads a settings file (usually ∗.ini); and prepares its properties to be read.
- SPECTRALRADAR_API int getSettingsEntryInt (SettingsHandle SettingsFile, const char ∗Node, int DefaultValue)

  Gets an integer number from the specified ini file (see SettingsHandle and initSettingsFile);.

- SPECTRALRADAR_API double getSettingsEntryFloat (SettingsHandle SettingsFile, const char ∗Node, double DefaultValue)

  Gets an floating point number from the specified ini file (see SettingsHandle and initSettingsFile);.
- SPECTRALRADAR_API void getSettingsEntryFloatArray (SettingsHandle SettingsFile, const char ∗Node, const double ∗DefaultValues, double ∗Values, int ∗Size)

  Gets an array of floating point numbers from the specified ini file (see SettingsHandle and initSettingsFile);.
- SPECTRALRADAR_API const char ∗ getSettingsEntryString (SettingsHandle SettingsFile, const char ∗Node, const char ∗Default)

  Gets a string from the specified ini file (see SettingsHandle and initSettingsFile);. The resulting const char∗ ptr will be valid until the settings file is closed by closeSettingsFile).
- SPECTRALRADAR_API void setSettingsEntryInt (SettingsHandle SettingsFile, const char ∗Node, int Value)

  Sets an integer entry in the specified ini file (see SettingsHandle and initSettingsFile);.
- SPECTRALRADAR_API void setSettingsEntryFloat (SettingsHandle SettingsFile, const char ∗Node, double Value)

  Sets a floating point entry in the specified ini file (see SettingsHandle and initSettingsFile);.
- SPECTRALRADAR_API void setSettingsEntryString (SettingsHandle SettingsFile, const char ∗Node, const char ∗Value)

  Sets a string in the specified ini file (see SettingsHandle and initSettingsFile);.
- SPECTRALRADAR_API void saveSettings (SettingsHandle SettingsFile)

  Saves the changes to the specified Settings file.
- SPECTRALRADAR_API void closeSettingsFile (SettingsHandle Handle)

  Closes the specified ini file and stores the set entries (.
- SPECTRALRADAR_API ColoringHandle createColoring32Bit (ColorScheme Color, ColoringByteOrder ByteOrder)

  Creates processing that can be used to color given floating point B-scans to 32 bit colored images.
- SPECTRALRADAR_API ColoringHandle createCustomColoring32Bit (int LUTSize, unsigned long ∗LUT)

  Create custom coloring using the specified color look-up-table.
- SPECTRALRADAR_API void setColoringBoundaries (ColoringHandle Colorng, float Min_dB, float Max_dB)

  Sets the boundaries in dB which are used by the coloring algorithm to map colors to floating point values in dB.
- SPECTRALRADAR_API void setColoringEnhancement (ColoringHandle Coloring, ColorEnhancement Enhancement)

  Selects a function for non-linear coloring to enhance (subjective) image impression.
- SPECTRALRADAR_API void colorizeData (ColoringHandle Coloring, DataHandle Data, ColoredDataHandle ColoredData, BOOL Transpose)

  Colors a given data object (DataHandle) into a given colored object (ColoredDataHandle).
- SPECTRALRADAR_API void colorizeDopplerData (ColoringHandle AmpColoring, ColoringHandle Phase←Coloring, DataHandle AmpData, DataHandle PhaseData, ColoredDataHandle Output, double MinSignal_dB, BOOL Transpose)

  Colors a two given data object (DataHandle) using overlay and intensity to represent phase and amplitude data. Used for Doppler imaging.
- SPECTRALRADAR_API void colorizeDopplerDataEx (ColoringHandle AmpColoring, ColoringHandle PhaseColoring[2], DataHandle AmpData, DataHandle PhaseData, ColoredDataHandle Output, double MinSignal_dB, BOOL Transpose)

  Colors a two given data object (DataHandle) using overlay and intensity to represent phase and amplitude data. Used for Doppler imaging. In the extended version, two ColoringHandles can be specified, two provide different coloring for increasing and decreasing phase, for example.
- SPECTRALRADAR_API void clearColoring (ColoringHandle Handle)

  Clears the coloring previously created by createColoring32Bit.
- SPECTRALRADAR_API void getMaxCameraImageSize (OCTDeviceHandle Dev, int ∗SizeX, int ∗SizeY)

  Returns the maximum possible camera image size for the current device.
- SPECTRALRADAR_API void getCameraImage (OCTDeviceHandle Dev, ColoredDataHandle Image)

  Gets a camera image.
- SPECTRALRADAR_API unsigned long InterpretReferenceIntensity (float intensity)

interprets the reference intensity and gives a color code that reflects its state.

- SPECTRALRADAR_API void getConfigPath (char ∗Path, int StrSize)

  Returns the path that hold the config files.
- SPECTRALRADAR_API void getPluginPath (char ∗Path, int StrSize)

  Returns the path that hold the plugins.
- SPECTRALRADAR_API void getInstallationPath (char ∗Path, int StrSize)

  Returns the installation path.
- SPECTRALRADAR_API double getReferenceIntensity (ProcessingHandle Proc)

  Returns an absolute value that indicates the refernce intensity that was present when the currently used apodization was determined.
- SPECTRALRADAR_API double getRelativeReferenceIntensity (OCTDeviceHandle Dev, ProcessingHandle Proc)

  Returns a value larger than 0.0 and smaller than 1.0 that indicates the reference intensity (relative to saturation) that was present when the currently used apodization was determined.
- SPECTRALRADAR_API double getRelativeSaturation (ProcessingHandle Proc)

  Returns a value larger than 0.0 and smaller than 1.0 that indicates the saturation of the sensor that was present during the last processing cycle.
- SPECTRALRADAR_API BufferHandle createMemoryBuffer (void)

  Creates a buffer holding data and colored data.
- SPECTRALRADAR_API void appendToBuffer (BufferHandle, DataHandle, ColoredDataHandle)

  Appends specified data and colored data to the requested buffer.
- SPECTRALRADAR_API void purgeBuffer (BufferHandle)

  Discards all data.
- SPECTRALRADAR_API int getBufferSize (BufferHandle)

  Returns the currently avaiable data sets in the buffer.
- SPECTRALRADAR_API int getBufferFirstIndex (BufferHandle)

  Returns the index of the first data sets available in the buffer.
- SPECTRALRADAR_API int getBufferLastIndex (BufferHandle)

  Returns the index of one past the last data sets available in the buffer.
- SPECTRALRADAR_API DataHandle getBufferData (BufferHandle, int Index)

  Returns the data in the buffer.
- SPECTRALRADAR_API ColoredDataHandle getColoredBufferData (BufferHandle, int Index)

  Returns the colored data in the buffer.
- SPECTRALRADAR_API void clearBuffer (BufferHandle BufferHandle)

  Clears the buffer and frees all data and colored data objects in it.
- SPECTRALRADAR_API int getNumberOfOutputDeviceValues (OCTDeviceHandle Dev)

  Returns the number of output values.
- SPECTRALRADAR_API void getOutputDeviceValueName (OCTDeviceHandle Dev, int Index, char ∗Name, int NameStringSize, char ∗Unit, int UnitStringSize)

  Returns names and units of the requested output values.
- SPECTRALRADAR_API BOOL doesOutputDeviceValueExist (OCTDeviceHandle Dev, const char ∗Name)

  Returns whether the requested output device values exists or not.
- SPECTRALRADAR_API void setOutputDeviceValueByName (OCTDeviceHandle Dev, const char ∗Name, double value)

  Sets the specified output value.
- SPECTRALRADAR_API void setOutputValueByIndex (OCTDeviceHandle Dev, int Index, double Value)

  Sets the specified output value.
- SPECTRALRADAR_API void getOutputDeviceValueRangeByName (OCTDeviceHandle Dev, const char ∗Name, double ∗Min, double ∗Max)

  Gives the range of the specified output value.
- SPECTRALRADAR_API void getOutputValueRangeByIndex (OCTDeviceHandle Dev, int Index, double ∗Min, double ∗Max)

Gives the range of the specified output value.

- SPECTRALRADAR_API void computeLinearKRawData (ComplexDataHandle ComplexDataAfterFFT, DataHandle LinearKData)

  Computes the linear k raw data of the complex data after FFT by an inverse Fourier transform.
- SPECTRALRADAR_API void linearizeSpectralData (DataHandle SpectraIn, DataHandle SpectraOut, DataHandle Chirp)

  Linearizes the spectral data using the given chirp vector.
- SPECTRALRADAR_API const char ∗ DataObjectName_SpectralData (int index)

  Returns the filename of the spectral-data object with the specified index.
- SPECTRALRADAR_API OCTFileHandle createOCTFile (OCTFileFormat format)

  Creates a handle to an OCT file of the given format.
- SPECTRALRADAR_API void clearOCTFile (OCTFileHandle Handle)

  Clears the given OCT file handle and frees its resources.
- SPECTRALRADAR_API int getFileDataObjectCount (OCTFileHandle Handle)

  Returns the number of data objects in the OCT file. This number will vary depending on the file's format and contents (Files with the .oct extension may contain multiple OCT data objects depending on their internal structure).
- SPECTRALRADAR_API void loadFile (OCTFileHandle Handle, const char ∗Filename)

  Loads the actual OCT data file from a file system. The file must have the format given in createOCTFile().
- SPECTRALRADAR_API void saveFile (OCTFileHandle Handle, const char ∗Filename)

  Saves the OCT data file in the given fully qualified path name.
- SPECTRALRADAR_API void saveChangesToFile (OCTFileHandle Handle)

  Saves the OCT data file in the file previously opened with loadFile(). Only changes will be saved.
- SPECTRALRADAR_API void copyFileMetadata (OCTFileHandle SrcHandle, OCTFileHandle DstHandle)

  Copies metadata from one OCT file to another.
- SPECTRALRADAR_API double getFileMetadataFloat (OCTFileHandle Handle, FileMetadataFloat Floatfield)

  Returns the value of the given file metadata field as a floating point number if found.
- SPECTRALRADAR_API void setFileMetadataFloat (OCTFileHandle Handle, FileMetadataFloat Floatfield, double Value)

  Sets the value of the given file metadata field as a floating point number.
- SPECTRALRADAR_API int getFileMetadataInt (OCTFileHandle Handle, FileMetadataInt Intfield)

  Returns the value of the given file metadata field as an integer if found.
- SPECTRALRADAR_API void setFileMetadataInt (OCTFileHandle Handle, FileMetadataInt Intfield, int Value)

  Sets the value of the given file metadata field as an integer.
- SPECTRALRADAR_API const char ∗ getFileMetadataString (OCTFileHandle Handle, FileMetadataString Stringfield)

  Returns the value of the given file metadata field as a string if found.
- SPECTRALRADAR_API void setFileMetadataString (OCTFileHandle Handle, FileMetadataString Stringfield, const char ∗Content)

  Sets the value of the given file metadata field as a string.
- SPECTRALRADAR_API BOOL getFileMetadataFlag (OCTFileHandle Handle, FileMetadataFlag Boolfield)

  Gets the boolean value of the given file metadata field.
- SPECTRALRADAR_API void setFileMetadataFlag (OCTFileHandle Handle, FileMetadataFlag Boolfield, BOOL Value)

  Sets the boolean value of the given file metadata field.
- SPECTRALRADAR_API void saveFileMetadata (OCTFileHandle Handle, OCTDeviceHandle Dev, ProcessingHandle Proc, ProbeHandle Probe, ScanPatternHandle Pattern)

  Saves meta information from the given device, processing, probe and scan pattern instances in the metadata block of the given file handle. This information will be available in files of type FileFormat_OCITY; mileage on other formats may vary according to their description.
- SPECTRALRADAR_API void saveFileMetadataDoppler (OCTFileHandle Handle, DopplerProcessingHandle DopplerProc)

  Saves meta information from the given DopplerProcessingHandle. A corresponding DopplerProcessingHandle can then be recreated using createDopplerProcessingForFile.

- SPECTRALRADAR_API void saveFileMetadataSpeckle (OCTFileHandle Handle, SpeckleVarianceHandle SpeckleVarianceProc)

  Saves meta information from the given SpeckleVarianceHandle. A corresponding SpeckleVarianceHandle can then be recreated using initSpeckleVarianceForFile.
- SPECTRALRADAR_API void loadCalibrationFromFile (OCTFileHandle Handle, ProcessingHandle Proc)

  Loads Chirp, Offset, and Apodization vectors from the given OCT file into the given processing object.
- SPECTRALRADAR_API void loadCalibrationFromFileEx (OCTFileHandle Handle, ProcessingHandle Proc, const int CameraIndex)

  Loads Chirp, Offset, and Apodization vectors from the given OCT file into the given processing object.
- SPECTRALRADAR_API void saveCalibrationToFile (OCTFileHandle Handle, ProcessingHandle Proc)

  Saves Chirp, Offset, and Apodization vectors from the given processing object into the given OCT file.
- SPECTRALRADAR_API void saveCalibrationToFileEx (OCTFileHandle Handle, ProcessingHandle Proc, int CameraIndex)

  Saves Chirp, Offset, and Apodization vectors from the given processing object into the given OCT file.
- SPECTRALRADAR_API void getFileRealData (OCTFileHandle Handle, DataHandle Data, int Index)

  Retrieves a RealData object from the OCT file at the given index with 0 <= index < getFileDataObjectCount(OCT←FileHandle handle). Users must ensure that the data handle is properly prepared and destroyed.
- SPECTRALRADAR_API void getFileColoredData (OCTFileHandle Handle, ColoredDataHandle Data, size←_t Index)

  Retrieves a ColoredData object from the OCT file at the given index with 0 <= index < getFileDataObjectCount(O←CTFileHandle handle). Users must ensure that the data handle is properly prepared and destroyed.
- SPECTRALRADAR_API void getFileComplexData (OCTFileHandle Handle, ComplexDataHandle Data, size_t Index)

  Retrieves a ComplexData object from the OCT file at the given index with 0 <= index < getFileDataObjectCount(←OCTFileHandle handle). Users must ensure that the data handle is properly prepared and destroyed.
- SPECTRALRADAR_API void getFileRawData (OCTFileHandle Handle, RawDataHandle Data, size_t Index)

  Retrieves a RawData object from the OCT file at the given index with 0 <= index < getFileDataObjectCount(OCT←FileHandle handle). Users must ensure that the data handle is properly prepared and destroyed.
- SPECTRALRADAR_API void getFile (OCTFileHandle Handle, size_t Index, const char ∗FilenameOnDisk)

  Retrieves a data object of arbitrary type from the OCT file at the given index with 0 <= index < getFileDataObject←Count(OCTFileHandle handle) and stores it at the given fully qualified path.
- SPECTRALRADAR_API int findFileDataObject (OCTFileHandle Handle, const char ∗Search)

  Searches for a data object the name of which contains the given string and returns its index, -1 if not found.
- SPECTRALRADAR_API BOOL containsFileDataObject (OCTFileHandle Handle, const char ∗Search)

  Searches for a data object the name of which contains the given string and returns TRUE if at least one data object name matches.
- SPECTRALRADAR_API BOOL containsFileRawData (OCTFileHandle Handle)

  Returns TRUE if the file contains raw data objects.
- SPECTRALRADAR_API void addFileRealData (OCTFileHandle Handle, DataHandle Data, const char ∗DataObjectName)

  Adds a RealData object to the OCT file; dataObjectName will be its name inside the OCT file if applicable. The object that the DataHandle refers to must live until after saveFile() has been called.
- SPECTRALRADAR_API void addFileColoredData (OCTFileHandle Handle, ColoredDataHandle Data, const char ∗DataObjectName)

  Adds a ColoredData object to the OCT file; dataObjectName will be its name inside the OCT file if applicable. The object that the ColoredDataHandle refers to must live until after saveFile() has been called.
- SPECTRALRADAR_API void addFileComplexData (OCTFileHandle Handle, ComplexDataHandle Data, const char ∗DataObjectName)

  Adds a ComplexData object to the OCT file; dataObjectName will be its name inside the OCT file if applicable. The object that the ComplexDataHandle refers to must live until after saveFile() has been called.
- SPECTRALRADAR_API void addFileRawData (OCTFileHandle Handle, RawDataHandle Data, const char ∗DataObjectName)

  Adds raw Data object to the OCT file; DataObjectName will be its name inside the OCT file if applicable. The object that the RawDataHandle refers to must live until after saveFile has been called.

- SPECTRALRADAR_API void addFileText (OCTFileHandle Handle, const char ∗FilenameOnDisk, const char ∗DataObjectName)

  Adds a text object read from FilenameOnDisk to the OCT file; DataObjectName will be its name inside the OCT file if applicable. The file identified by filenameOnDisk must exist until after saveFile() has been called.
- SPECTRALRADAR_API DataObjectType getFileDataObjectType (OCTFileHandle Handle, int Index)

  Returns the type of the data object at the given Index in the OCT file.
- SPECTRALRADAR_API void getFileDataObjectName (OCTFileHandle Handle, int Index, char ∗Filename, int Length)

  Returns the name of the data object at the given Index in the OCT file.
- SPECTRALRADAR_API int getFileDataSizeX (OCTFileHandle Handle, size_t Index)

  Returns the pixel count in X of the data object at the given Index in the OCT file.
- SPECTRALRADAR_API int getFileDataSizeY (OCTFileHandle Handle, size_t Index)

  Returns the pixel count in Y of the data object at the given Index in the OCT file.
- SPECTRALRADAR_API int getFileDataSizeZ (OCTFileHandle Handle, size_t Index)

  Returns the pixel count in Z of the data object at the given Index in the OCT file.
- SPECTRALRADAR_API float getFileDataRangeX (OCTFileHandle Handle, size_t Index)

  Returns the range (usually in mm) in X of the data object at the given Index in the OCT file.
- SPECTRALRADAR_API float getFileDataRangeY (OCTFileHandle Handle, size_t Index)

  Returns the range (usually in mm) in Y of the data object at the given Index in the OCT file.
- SPECTRALRADAR_API float getFileDataRangeZ (OCTFileHandle Handle, size_t Index)

  Returns the range (usually in mm) in Z of the data object at the given Index in the OCT file.
- SPECTRALRADAR_API void copyMarkerListFromRealData (OCTFileHandle Handle, DataHandle Data)

  coordinates, so re-use is possible.
- SPECTRALRADAR_API void copyMarkerListToRealData (OCTFileHandle Handle, DataHandle Data)

  coordinates, so re-use is possible.
- SPECTRALRADAR_API void addFileMetadataPreset (OCTFileHandle Handle, const char ∗Category, const char ∗PresetDescription)

  Adds one of the presets set during acquisition for the OCTFileHandle.
- SPECTRALRADAR_API int getFileMetadataNumberOfPresets (OCTFileHandle Handle)

  Gets the number of presets that were set during the acquisition.
- SPECTRALRADAR_API const char ∗ getFileMetadataPresetCategory (OCTFileHandle Handle, int Index)

  Gets the preset category belonging to the preset with given Index.
- SPECTRALRADAR_API const char ∗ getFileMetadataPresetDescription (OCTFileHandle Handle, int Index)

  Gets the preset description belonging to the preset with given Index.
- SPECTRALRADAR_API SpeckleVarianceHandle initSpeckleVariance (void)

  Initializes the speckle variance contrast processing instance.
- SPECTRALRADAR_API SpeckleVarianceHandle initSpeckleVarianceForFile (OCTFileHandle File)

  Initializes the speckle variance contrast processing instance, based on the parameters stored in an OCT file.
- SPECTRALRADAR_API void closeSpeckleVariance (SpeckleVarianceHandle Handle)

  Closes the speckle variance contrast processing instance and frees all used resources.
- SPECTRALRADAR_API void setSpeckleVariancePropertyInt (SpeckleVarianceHandle Handle, Speckle↩ VariancePropertyInt Property, int value)

  Sets the given integer property to the given value.
- SPECTRALRADAR_API int getSpeckleVariancePropertyInt (SpeckleVarianceHandle Handle, Speckle↩ VariancePropertyInt Property)

  Sets the given floating point property to the given value.
- SPECTRALRADAR_API void setSpeckleVariancePropertyFloat (SpeckleVarianceHandle Handle, Speckle↩ VariancePropertyFloat Property, double value)

  Returns the value of the given integer property.
- SPECTRALRADAR_API double getSpeckleVariancePropertyFloat (SpeckleVarianceHandle Handle, SpeckleVariancePropertyFloat Property)

  Returns the value of the given floating point property.

- SPECTRALRADAR_API void setSpeckleVarianceType (SpeckleVarianceHandle SpeckleVar, SpeckleVarianceType Type)

  Sets the speckle variance type to the given value.
- SPECTRALRADAR_API SpeckleVarianceType getSpeckleVarianceType (SpeckleVarianceHandle SpeckleVar)

  Returns the speckle variance type the instance is using.
- SPECTRALRADAR_API void computeSpeckleVariance (SpeckleVarianceHandle SpeckleVar, ComplexDataHandle CompDataIn, DataHandle DataOutMean, DataHandle DataOutVar)

  Computes the speckle variance contrast and returns the mean and variance values in DataOutMean and DataOutVar.
- SPECTRALRADAR_API void setTriggerMode (OCTDeviceHandle Dev, DeviceTriggerType TriggerMode)

  Sets the trigger mode for the OCT device used for acquisition. Additional hardware may be needed.
- SPECTRALRADAR_API DeviceTriggerType getTriggerMode (OCTDeviceHandle Dev)

  Returns the trigger mode used for acquisition.
- SPECTRALRADAR_API BOOL isTriggerModeAvailable (OCTDeviceHandle Dev, DeviceTriggerType TriggerMode)

  Returns whether the specified trigger mode is possible or not for the used device.
- SPECTRALRADAR_API void setTriggerTimeout_s (OCTDeviceHandle Dev, int Timeout_s)

  Sets the timeout of the camera in seconds (useful in external trigger mode).
- SPECTRALRADAR_API int getTriggerTimeout_s (OCTDeviceHandle Dev)

  Returns the timeout of the camera in seconds (not used in trigger mode Trigger_FreeRunning).
- SPECTRALRADAR_API int getScanPatternPropertyInt (ScanPatternHandle ScanPattern, ScanPatternPropertyInt Property)

  Returns the specified property of the scan pattern.
- SPECTRALRADAR_API double getScanPatternPropertyFloat (ScanPatternHandle Pattern, ScanPatternPropertyFloat Selection)

  Returns the specified property of the scan pattern.
- SPECTRALRADAR_API double expectedAcquisitionTime_s (ScanPatternHandle ScanPattern, OCTDeviceHandle Dev)

  Returns the expected acquisition time of the scan pattern. Please.
- SPECTRALRADAR_API ScanPatternAcquisitionOrder getScanPatternAcqOrder (ScanPatternHandle ScanPattern)

  Returns the acquisition order of the scan pattern. See definition of ScanPatternAcquisitionOrder for detailed information.
- SPECTRALRADAR_API BOOL isAcqTypeForScanPatternAvailable (ScanPatternHandle ScanPattern, AcquisitionType AcqType)

  Returns whether the acquisition type is available for the scan pattern.
- SPECTRALRADAR_API BOOL checkAvailableMemoryForRawData (OCTDeviceHandle Dev, ScanPatternHandle Pattern, ptrdiff_t AdditionalMemory)

  Checks whether sufficient memory is available for raw data acquired with the specified scan pattern.
- SPECTRALRADAR_API double QuantumEfficiency (OCTDeviceHandle Dev, double CenterWavelength_nm, double PowerIntoSpectrometer_W, DataHandle Spectrum_e)

  Calculates the quantum efficiency from the processed input spectrum in the Data instance.
- SPECTRALRADAR_API void determineSurface (DataHandle Volume, DataHandle Surface)

  Performs a minimal segmentation of the data, by finding a surface that is compromised of the highes signals fromo each A-scan. From the 3D input data, the output data will 2D data, where each data pixel contains the depth of the respective surface as a funciton of the x- and y-pixel position.
- SPECTRALRADAR_API unsigned long long getFreeMemory ()

  Returns the amount of free system memory. Function is available for convenience.
- SPECTRALRADAR_API void absComplexData (ComplexDataHandle ComplexData, DataHandle Abs)

  Converts the complex values from the ComplexDataHandle to its absolute values and writes them to DataHandle.
- SPECTRALRADAR_API void logAbsComplexData (ComplexDataHandle ComplexData, DataHandle dB)

  Converts the complex values from the ComplexDataHandle to its dB values and writes them to DataHandle.
- SPECTRALRADAR_API void argComplexData (ComplexDataHandle ComplexData, DataHandle Arg)

Converts the complex values from the ComplexDataHandle to its phase angle values and writes them to DataHandle.
- SPECTRALRADAR_API void realComplexData (ComplexDataHandle ComplexData, DataHandle Real)

  Writes the real part of the complex values from the ComplexDataHandle to DataHandle.
- SPECTRALRADAR_API void imagComplexData (ComplexDataHandle ComplexData, DataHandle Imag)

  Writes the imaginary part of the complex values from the ComplexDataHandle to DataHandle.
- SPECTRALRADAR_API void determineDynamicRange_dB (DataHandle Data, double *MinRange_dB, double *MaxRange_dB)

  Gives a rough estimation of the dynamic range of the specified data object.
- SPECTRALRADAR_API void determineDynamicRangeWithMinRange_dB (DataHandle Data, double *MinRange_dB, double *MaxRange_dB, double MinDynamicRange_dB)

  Gives a rough estimation of the dynamic range of the specified data object.
- SPECTRALRADAR_API void medianFilter1D (DataHandle Data, int Rank, Direction FilterDirection)

  Computes a 1D-median filter on the specified data.
- SPECTRALRADAR_API void medianFilter2D (DataHandle Data, int Rank, Direction FilterNormalDirection)

  Computes a 2D-median filter on the specified 2D data.
- SPECTRALRADAR_API void pepperFilter2D (DataHandle Data, PepperFilterType Type, float Threshold, Direction FilterNormalDirection)

  Removes pepper-noise (very low values, i. e. dark spots in the data). This enhances the visual (colored) representation of the data.
- SPECTRALRADAR_API void convolutionFilter1D (DataHandle Data, int FilterSize, float *FilterKernel, Direction FilterDirection)

  Calculates a mathematical convolution of the Data and the 1D-FilterKernel.
- SPECTRALRADAR_API void convolutionFilter2D (DataHandle Data, int FilterSize1, int FilterSize2, float *FilterKernel, Direction FilterNormalDirection)

  Calculates a mathematical convolution of the Data and the 2D-FilterKernel.
- SPECTRALRADAR_API void convolutionFilter3D (DataHandle Data, int FilterSize1, int FilterSize2, int Filter↩ Size3, float *FilterKernel)

  Calculates a mathematical convolution of the Data and the 3D-FilterKernel.
- SPECTRALRADAR_API void predefinedFilter1D (DataHandle Data, FilterType1D Filter, Direction Filter↩ Direction)

  Applies the predefined 1D-Filter to the Data.
- SPECTRALRADAR_API void predefinedFilter2D (DataHandle Data, FilterType2D Filter, Direction Filter↩ NormalDirection)

  Applies the predefined 2D-Filter to the Data.
- SPECTRALRADAR_API void predefinedFilter3D (DataHandle Data, FilterType3D FilterType)

  Applies the predefined 3D-Filter to the Data.
- SPECTRALRADAR_API void predefinedComplexFilter2D (ComplexDataHandle ComplexData, Complex↩ FilterType2D Type, Direction FilterNormalDirection)

  Applies the predefined 2D-Filter to the ComplexData.
- SPECTRALRADAR_API void darkFieldComplexFilter2D (ComplexDataHandle ComplexData, double Radius, Direction FilterNormalDirection)

  Filters the image such that the image contrast comes from light scattered by the sample.
- SPECTRALRADAR_API void brightFieldComplexFilter2D (ComplexDataHandle ComplexData, double Radius, Direction FilterNormalDirection)

  Filters the image such that the image contrast comes from absorbance of light in the sample.
- SPECTRALRADAR_API void polynomialFitAndEval1D (int Size, const float *OrigPosX, const float *OrigY, int DegreePolynom, int EvalSize, const float *EvalPosX, float *EvalY)

  Computes the polynomial fit of the given 1D data.
- SPECTRALRADAR_API float calcParabolaMaximum (float x0, float y0, float yLeft, float yRight, float *peak↩ Height)

  Computes the x-position of the highest peak of the parabola given by the point x0, y0, yLeft, yRight. y0 needs to be the point with the highest value.
- SPECTRALRADAR_API void crossCorrelatedProjection (DataHandle DataIn, DataHandle DataOut)

Upon return DataOut contains an average of all B-Scans in DataIn. Right before averaging, the datasets are cross-correlated to eliminate registration errors.

- SPECTRALRADAR_API void thresholdDopplerData (DataHandle Phase, DataHandle Intensity, float intensityThreshold, float phaseTargetValue)

  At points whose Intensity does not exceed the intensityThreshold, the phase is set to the phaseTargetValue.
- SPECTRALRADAR_API void getCurrentIntensityStatistics (OCTDeviceHandle Dev, ProcessingHandle Proc, float ∗relToRefIntensity, float ∗relToProjAbsIntensity)

  Returns two statistical interpretations of the current light intensity on the sensor.
- SPECTRALRADAR_API int getNumberOfProbeConfigs ()

  Returns the number of available probe configuration files.
- SPECTRALRADAR_API void getProbeConfigName (int Index, char ∗ProbeName, int StringSize)

  Returns the name of the specified probe configuration file.
- SPECTRALRADAR_API int getNumberOfAvailableProbes (void)

  Returns the number of the available probe types.
- SPECTRALRADAR_API void getAvailableProbe (int Index, char ∗ProbeName, int StringSize)

  Returns the name of the desired probe type.
- SPECTRALRADAR_API void getProbeDisplayName (const char ∗ProbeName, char ∗DisplayName, int StringSize)

  Returns the display name for the probe name specified.
- SPECTRALRADAR_API void getObjectiveDisplayName (const char ∗ObjectiveName, char ∗DisplayName, int StringSize)

  Returns the display name for the objective name specified.
- SPECTRALRADAR_API int getNumberOfCompatibleObjectives (const char ∗ProbeName)

  Returns the number of objectives compatible with the specified objective mount.
- SPECTRALRADAR_API void getCompatibleObjective (int Index, const char ∗ProbeName, char ∗Objective, int StringSize)

  Returns the name of the specified objective for the selected probe type.
- SPECTRALRADAR_API ProbeScanRangeShape getProbeMaxScanRangeShape (ProbeHandle Probe)

  Returns the shape of the valid scan range for the ProbeHandle. All possible scan range are defined in ProbeScanRangeShape.
- SPECTRALRADAR_API void setProbeMaxScanRangeShape (ProbeHandle Probe, ProbeScanRangeShape Shape)

  Sets the `Shape` *of the valid scan range for the ProbeHandle. All possible scan-range shapes are defined in ProbeScanRangeShape.*
- SPECTRALRADAR_API int getObjectivePropertyInt (const char ∗Objective, ObjectivePropertyInt Selection)

  Returns the selected ObjectivePropertyInt for the chosen objective.
- SPECTRALRADAR_API double getObjectivePropertyFloat (const char ∗Objective, ObjectivePropertyFloat Selection)

  Returns the selected ObjectivePropertyFloat for the chosen objective.
- SPECTRALRADAR_API const char ∗ getObjectivePropertyString (const char ∗Objective, ObjectivePropertyString Selection)

  Returns the selected ObjectivePropertyString for the chosen objective. Warning: The returned const char∗ will only be valid until the next call to getObjectivePropertyString.
- SPECTRALRADAR_API void addProbeButtonCallback (OCTDeviceHandle Dev, cbProbeMessageReceived Callback)

  Registers a callback function to notify when a button on the probe has been pressed. The int parameter passed to the callback function will contain the pressed button's ID. Caution: Since the callbacks will not be called in separate threads but in the order of addition, make sure that the callback function returns as soon as possible.
- SPECTRALRADAR_API void removeProbeButtonCallback (OCTDeviceHandle Dev, cbProbeMessageReceived Callback)

  Removes a previously registered probe button callback function.
- SPECTRALRADAR_API BOOL isRefstageAvailable (OCTDeviceHandle Dev)

  Returns whether a motorized reference stage is available or not for the specified device. Please note that a motorized reference stage is not included in all systems.

- SPECTRALRADAR_API RefstageStatus getRefstageStatus (OCTDeviceHandle Dev)

  Returns the current status of the reference stage, e.g. if it is moving.
- SPECTRALRADAR_API double getRefstageLength_mm (OCTDeviceHandle Dev, ProbeHandle Probe)

  Returns the total length in mm of the reference stage.
- SPECTRALRADAR_API double getRefstagePosition_mm (OCTDeviceHandle Dev, ProbeHandle Probe)

  Returns the current position in mm of the reference stage.
- SPECTRALRADAR_API void homeRefstage (OCTDeviceHandle Dev, RefstageWaitForMovement WaitFor↩ Moving)

  Homes the reference stage to calibrate the zero position.
- SPECTRALRADAR_API void moveRefstageToPosition_mm (OCTDeviceHandle Dev, ProbeHandle Probe, double Pos_mm, RefstageSpeed Speed, RefstageWaitForMovement WaitForMoving)

  Moves the reference stage to the specified position in mm.
- SPECTRALRADAR_API void moveRefstage_mm (OCTDeviceHandle Dev, ProbeHandle Probe, double Length_mm, RefstageMovementDirection Direction, RefstageSpeed Speed, RefstageWaitForMovement WaitForMoving)

  Moves the reference stage with the specified length in mm.
- SPECTRALRADAR_API void startRefstageMovement (OCTDeviceHandle Dev, RefstageMovementDirection Direction, RefstageSpeed Speed)

  Starts the movement of the reference stage with the chosen speed. Please note that the movement does not stop until stopRefstageMovement is called.
- SPECTRALRADAR_API void stopRefstageMovement (OCTDeviceHandle Dev)

  Stops the movement of the reference stage.
- SPECTRALRADAR_API void setRefstageSpeed (OCTDeviceHandle Dev, RefstageSpeed Speed)

  Sets the velocity of the movement of the reference stage.
- SPECTRALRADAR_API void setRefstageStatusCallback (OCTDeviceHandle Dev, cbRefstageStatus↩ Changed Callback)

  Registers the callback to get notified if the reference stage status changed.
- SPECTRALRADAR_API void setRefstagePosChangedCallback (OCTDeviceHandle Dev, cbRefstage↩ PositionChanged Callback)

  Registers the callback to get notified if the reference stage position changed.
- SPECTRALRADAR_API double getRefstageMinPosition_mm (OCTDeviceHandle Dev, ProbeHandle Probe)

  Returns the minimal position in mm the reference stage can move to.
- SPECTRALRADAR_API double getRefstageMaxPosition_mm (OCTDeviceHandle Dev, ProbeHandle Probe)

  Returns the maximal position in mm the reference stage can move to.
- SPECTRALRADAR_API void setLightSourceTimeoutCallback (OCTDeviceHandle Dev, lightSourceState↩ Callback Callback)

  Sets a callback function that will be invoked by the SDK whenever the state of the lightsource of the device changes.
- SPECTRALRADAR_API void setLightSourceTimeout_s (OCTDeviceHandle Dev, double Timeout)

  Sets a the timeout in seconds, after which the OCT lightsource will be turned off if no scanning is performed.
- SPECTRALRADAR_API double getLightSourceTimeout_s (OCTDeviceHandle Dev)

  Gets a the timeout in seconds, after which the OCT lightsource will be turned off if no scanning is performed.
- SPECTRALRADAR_API PolarizationProcessingHandle createPolarizationProcessing (void)

  Returns a Polarization processing handle to the Processing routines for polarization analysis.
- SPECTRALRADAR_API void clearPolarizationProcessing (PolarizationProcessingHandle Polarization)

  Clears the polarization processing routines and frees the memory that has been allocated for these to work properly.
- SPECTRALRADAR_API int getPolarizationPropertyInt (PolarizationProcessingHandle Polarization, PolarizationPropertyInt Property)

  Gets the desired polarization processing property.
- SPECTRALRADAR_API void setPolarizationPropertyInt (PolarizationProcessingHandle Polarization, PolarizationPropertyInt Property, int Value)

  Sets polarization processing properties.

- SPECTRALRADAR_API double getPolarizationPropertyFloat (PolarizationProcessingHandle Polarization, PolarizationPropertyFloat Property)

  Gets the desired polarization processing floating-point property.
- SPECTRALRADAR_API void setPolarizationPropertyFloat (PolarizationProcessingHandle Polarization, PolarizationPropertyFloat Property, double Value)

  Sets the desired polarization processing floating-point property.
- SPECTRALRADAR_API void setPolarizationOutputI (PolarizationProcessingHandle Polarization, Data↩ Handle Intensity)

  Sets the location of the resulting polarization intensity output (Stokes parameter I).
- SPECTRALRADAR_API void setPolarizationOutputQ (PolarizationProcessingHandle Polarization, Data↩ Handle StokesQ)

  Sets the location of the resulting Stokes parameter Q.
- SPECTRALRADAR_API void setPolarizationOutputU (PolarizationProcessingHandle Polarization, Data↩ Handle StokesU)

  Sets the location of the resulting Stokes parameter U.
- SPECTRALRADAR_API void setPolarizationOutputV (PolarizationProcessingHandle Polarization, Data↩ Handle StokesV)

  Sets the location of the resulting Stokes parameter U.
- SPECTRALRADAR_API void setPolarizationOutputDOPU (PolarizationProcessingHandle Polarization, DataHandle DOPU)

  Sets the location of the resulting DOPU.
- SPECTRALRADAR_API void setPolarizationOutputRetardation (PolarizationProcessingHandle Polarization, DataHandle Retardation)

  Sets the location of the resulting retardation.
- SPECTRALRADAR_API void setPolarizationOutputOpticAxis (PolarizationProcessingHandle Polarization, DataHandle OpticAxis)

  Sets the location of the resulting optic axis.
- SPECTRALRADAR_API void executePolarizationProcessing (PolarizationProcessingHandle Polarization, ComplexDataHandle Data_P_Camera1, ComplexDataHandle PData_S_Camera0)

  Executes the polarization processing of the input data and returns, if previously setup, intensity, retardation, and phase differences.
- SPECTRALRADAR_API void saveFileMetadataPolarization (OCTFileHandle FileHandle, Polarization↩ ProcessingHandle PolProc)

  Saves metadata to the specified file. These metadata specify the operational arguments needed by the polarization processing routines to redo the polarization-analysis starting from two ComplexDataHandle delivered by `Proc_0` and `Proc_1`.
- SPECTRALRADAR_API PolarizationProcessingHandle createPolarizationProcessingForFile (OCTFile↩ Handle FileHandle)

  Loads metadata to the specified file. These metadata specify the operational arguments needed by the polarization processing routines to redo the polarization-analysis starting from two ComplexDataHandle delivered by `Proc_0` and `Proc_1`, exactly as they were done before the file was written.
- SPECTRALRADAR_API void updateAfterPresetChange (OCTDeviceHandle Dev, ProbeHandle Probe, ProcessingHandle Proc, int CameraIndex)

  Updates the processing handle after preset change. Please use setDevicePreset first for the first camera (with index 0) and this function to update the corresponding ProcessingHandle for the second camera (with index 1).
- SPECTRALRADAR_API double analyzeComplexAScan (ComplexDataHandle AScanIn, AScanAnalyzation Selection)

  Analyzes the given complex A-scan data, extracts the selected feature, and returns the computed value.
- SPECTRALRADAR_API BOOL isPolarizationAdjustmentAvailable (OCTDeviceHandle Dev)

  Returns whether or not a motorized polarization adjustment stage is available for the specified device.
- SPECTRALRADAR_API void setPolarizationAdjustmentRetardationChangedCallback (OCTDeviceHandle Dev, cbRetardationChanged Callback)

  Registers the callback to get notified when the polarization adjustment retardation has changed.

- SPECTRALRADAR_API void setPolarizationAdjustmentRetardation (OCTDeviceHandle Dev, PolarizationRetarder Retarder, double Retardation, WaitForCompletion Wait)

  Sets the retardation of the specified retarder in the polarization adjustment. The retardation is a unitless value between 0 and 1, which represents the full adjustment range of the retarder. The retarder may take some time to physically reach the new Retardation. Use the Wait parameter to choose if the function should block until the new position is reached.
- SPECTRALRADAR_API double getPolarizationAdjustmentRetardation (OCTDeviceHandle Dev, PolarizationRetarder Retarder)

  Gets the current retardation of the specified retarder in the polarization adjustment. If setPolarizationAdjustmentRetardation was used in a non-blocking fashion, the function returns the current position of the retarder, not the final target position.
- SPECTRALRADAR_API BOOL isReferenceIntensityControlAvailable (OCTDeviceHandle Dev)

  Returns whether or not an automated reference intensity control is available for the specified device.
- SPECTRALRADAR_API void setReferenceIntensityControlCallback (OCTDeviceHandle Dev, cbReferenceIntensityControlValueChanged Callback)

  Registers the callback to get notified when the reference intensity has changed.
- SPECTRALRADAR_API void setReferenceIntensityControlValue (OCTDeviceHandle Dev, double ReferenceIntensity, WaitForCompletion Wait)

  Sets the reference intensity of the specified device. The intensity is a unitless value between 0 and 1, which represents the full adjustment range of the reference intensity control, but may or may not be linear. The control may take some time to physically reach the new intensity. Use the Wait parameter to choose if the function should block until the new intensity is reached.
- SPECTRALRADAR_API double getReferenceIntensityControlValue (OCTDeviceHandle Dev)

  Gets the current reference intensity of the specified device. If setReferenceIntensityControlValue was used in a non-blocking fashion, the function returns the current value of the control, not the final target value.
- SPECTRALRADAR_API BOOL isAmplificationControlAvailable (OCTDeviceHandle Dev)

  Returns whether or not the sampling amplification of specified device can be adjusted.
- SPECTRALRADAR_API int getAmplificationControlNumberOfSteps (OCTDeviceHandle Dev)

  Gets the number of discrete amplification control steps available on the specified device. Please note that the largest amplification step is getAmplificationControlNumberOfSteps() - 1.
- SPECTRALRADAR_API void setAmplificationControlStep (OCTDeviceHandle Dev, int Step)

  Sets the sampling amplification on the the specified device. The lowest amplification is always 0. In general, the amplification should be set as high as possible without going into saturation.
- SPECTRALRADAR_API int getAmplificationControlStep (OCTDeviceHandle Dev)

  Gets the current sampling amplification of the specified device.

Variables

ExportOptions

Specifies additional export options to be used with functions such as exportDataAsImage(). Multiple options can be combined by bit-wise or ("|"). Different options can be used for different export format. If an option is not supported by an export format, it is ignored.

- const int ExportOption_None = 0x00000000
- const int ExportOption_DrawScaleBar = 0x00000001

  Draw scale bar on exported image.
- const int ExportOption_DrawMarkers = 0x00000002

  Draw markers on exported image.
- const int ExportOption_UsePhysicalAspectRatio = 0x00000004

  Honor physical aspect ratio when exporting data (width and height of each pixel will have the same physical dimensions).
- const int ExportOption_Flip_X_Axis = 0x00000008

  Flip X-axis.
- const int ExportOption_Flip_Y_Axis = 0x00000010

  Flip Y-axis.
- const int ExportOption_Flip_Z_Axis = 0x00000020

  Flip Z-axis.

7.1.1 Detailed Description

Header containing all functions of the Spectral Radar SDK. This SDK can be used for Callisto, Ganymede, Hyperion, Telesto and Vega devices.

7.1.2 Macro Definition Documentation

7.1.2.1 #define FALSE 0

FALSE for use with data type BOOL.

7.1.2.2 #define SPECTRALRADAR_API __declspec(dllimport)

Export/Import of define of DLL members.

7.1.2.3 #define TRUE 1

TRUE for use with data type BOOL.

7.1.3 Typedef Documentation

7.1.3.1 BOOL

A standard boolean data type used in the API.

7.1.3.2 cbRefstagePositionChanged

Defines the function prototype for the reference stage position change callback (see also setRefstagePosChanged↩ Callback()). The argument contains the reference stage position in mm when called.

Parameters

double	Current position of the reference stage in mm

7.1.3.3 cbRefstageStatusChanged

Defines the function prototype for the reference stage status callback (see also setRefstageStatusCallback()). The argument contains the current status of the reference stage when called.

Parameters

RefstageStatus	Current status of the reference stage

7.1.4 Enumeration Type Documentation

7.1.4.1 enum SpeckleVariancePropertyFloat

Enum identifying different properties of typ float for speckle variance processing.

7.1.4.2 enum SpeckleVariancePropertyInt

Enum identifying different properties of typ int for speckle variance processing.

7.1.4.3 enum SpeckleVarianceType

Enum identifying different speckle variance processing types.

7.1.5 Function Documentation

7.1.5.1 BOOL checkAvailableMemoryForRawData (OCTDeviceHandle *Dev,* ScanPatternHandle *Pattern,* ptrdiff_t *AdditionalMemory*)

Checks whether sufficient memory is available for raw data acquired with the specified scan pattern.

AdditionalMemory The parameter specifies additional memory that will be required during the measurement (from startMeasurement()

to stopMeasruement()) unknown to the SDK and/or memory that will be freed/available prior to the call of startMeasurement().

7.1.5.2 void clearVisualCalibration (VisualCalibrationHandle *Handle*)

Clear handle and frees all related memory.

Parameters

in	*Handle*	A handle of a visual calibration (VisualCalibrationHandle). If the handle is a nullptr, this function does nothing. In most cases this handle will have been previously created with the function createVisualCalibration.

Warning

ThorImageOCT uses this and other functions to calibrate the galvo, assuming a very specific sequence of actions and conditions, as explained in the ThorImageOCT. For this function to properly work, the user need to re-create the same sequence of actions and conditions. Please use the ThorImageOCT software to perform probe calibrations, if at all necessary.

7.1.5.3 void closeSpeckleVariance (SpeckleVarianceHandle *Handle*)

Closes the speckle variance contrast processing instance and frees all used resources.

Parameters

in	*Handle*	A handle of speckle variance routines (SpeckleVarianceHandle). If the handle is a nullptr, this function does nothing.

7.1.5.4 void computeSpeckleVariance (**SpeckleVarianceHandle** *SpeckleVar,* **ComplexDataHandle** *CompDataIn,* **DataHandle** *DataOutMean,* **DataHandle** *DataOutVar*)

Computes the speckle variance contrast and returns the mean and variance values in DataOutMean and Data↩ OutVar.

7.1.5.5 **ScanPatternHandle** createAScanPattern (**ProbeHandle** *Probe,* int *AScans,* double *PosX_mm,* double *PosY_mm*)

Creates a scan pattern used to acquire a specific amount of Ascans at a specific position.

Parameters

in	*Probe*	A valid (non null) handle of a probe.
in	*AScans*	The number of A-Scans that will be measured.
in	*PosX_mm*	The position of the light spot, in millimeter.
in	*PosY_mm*	The position of the light spot, in millimeter.

Returns

A valid (non null) handle to a scan pattern.

7.1.5.6 **VisualCalibrationHandle** createVisualCalibration (**OCTDeviceHandle** *Device,* double *TargetCornerLength_mm,* **BOOL** *CheckAngle,* **BOOL** *SaveData*)

Creates handle used for visual calibration.

Parameters

in	*Device*	A valid (non null) OCT device handle (OCTDeviceHandle), previously generated with the function initDevice.
in	*TargetCornerLength_mm*	The length of the edge
in	*CheckAngle*	A flag stating if the the sample's position is in a right angle with respect to the camera image (TRUE) or not (FALSE).
in	*SaveData*	If TRUE, debug information will be dumped. Kindly say FALSE.

Returns

A valid handle of a visual calibration (VisualCalibrationHandle).

Warning

ThorImageOCT uses this and other functions to calibrate the galvo, assuming a very specific sequence of actions and conditions, as explained in the ThorImageOCT. For this function to properly work, the user need to re-create the same sequence of actions and conditions. Please use the ThorImageOCT software to perform probe calibrations, if at all necessary.

7.1.5.7 void dopplerPhaseToVelocity (**DopplerProcessingHandle** *Handle,* **DataHandle** *InOut*)

Scales phases computed by Doppler OCT to actual flow velocities in scan direction.

Parameters

in	*Handle*	A valid (non null) handle of Doppler processing routines (DopplerProcessingHandle), obtained with the function createDopplerProcessing.
in,out	*InOut*	A handle of data representing first phase data that will then be modified to conttain velocity data.

This requires the Doppler scan rate, Doppler angle and center velocity of the Doppler object to be set correctly.

7.1.5.8 unsigned long long getFreeMemory ( )

Returns the amount of free system memory. Function is available for convenience.

7.1.5.9 double getSpeckleVariancePropertyFloat (SpeckleVarianceHandle *Handle*, SpeckleVariancePropertyFloat *Property*)

Returns the value of the given floating point property.

7.1.5.10 int getSpeckleVariancePropertyInt (SpeckleVarianceHandle *Handle*, SpeckleVariancePropertyInt *Property*)

Sets the given floating point property to the given value.

7.1.5.11 void SpeckleVarianceType getSpeckleVarianceType (SpeckleVarianceHandle *SpeckleVar*)

Returns the speckle variance type the instance is using.

7.1.5.12 SpeckleVarianceHandle initSpeckleVariance (void)

Initializes the speckle variance contrast processing instance.

7.1.5.13 SPECTRALRADAR_API SpeckleVarianceHandle initSpeckleVarianceForFile (OCTFileHandle *File*)

Initializes the speckle variance contrast processing instance, based on the parameters stored in an OCT file.

Parameters

in	*File*	A handle to the OCT-File used to create the speckle variance processing routines from.

7.1.5.14 void setSpeckleVariancePropertyFloat (SpeckleVarianceHandle *Handle*, SpeckleVariancePropertyFloat *Property*, double *value*)

Returns the value of the given integer property.

7.1.5.15 void setSpeckleVariancePropertyInt (SpeckleVarianceHandle *Handle*, SpeckleVariancePropertyInt *Property*, int *value*)

Sets the given integer property to the given value.

7.1.5.16 void setSpeckleVarianceType (SpeckleVarianceHandle *SpeckleVar,* SpeckleVarianceType *Type*)

Sets the speckle variance type to the given value.

7.1.5.17 BOOL visualCalibrate_1st_CameraScaling (VisualCalibrationHandle *Handle,* ProbeHandle *Probe,* ColoredDataHandle *Image*)

This is the first step in visual calibration. For this, the calibration the target needs to be placed under the objective. Returns TRUE if the first step succeeds.

Parameters

`in`	*Handle*	A handle of a valid (non null) visual calibration (VisualCalibrationHandle), previously created with the function createVisualCalibration.
`in`	*Probe*	A valid (non null) probe handle (ProbeHandle), previously generated with the function initProbe.
`in`	*Image*	Video snapshot to use for calibration

Warning

> ThorImageOCT uses this and other functions to calibrate the galvo, assuming a very specific sequence of actions and conditions, as explained in the ThorImageOCT. For this function to properly work, the user need to re-create the same sequence of actions and conditions. Please use the ThorImageOCT software to perform probe calibrations, if at all necessary.

7.1.5.18 BOOL visualCalibrate_2nd_Galvo (VisualCalibrationHandle *Handle,* ProbeHandle *Probe,* ColoredDataHandle *Image*)

This is the second step in visual calibration. For this, the calibration target or and infrared vieweing card needs to be placed under the objective. Returns TRUE if the second step succeeds.

Parameters

`in`	*Handle*	A handle of a valid (non null) visual calibration (VisualCalibrationHandle), previously created with the function createVisualCalibration.
`in`	*Probe*	A valid (non null) probe handle (ProbeHandle), previously generated with the function initProbe.
`in`	*Image*	Video snapshot to use for calibration

It is assumed that the function visualCalibrate_1st_CameraScaling has been previously successfully invoked.

Warning

> ThorImageOCT uses this and other functions to calibrate the galvo, assuming a very specific sequence of actions and conditions, as explained in the ThorImageOCT. For this function to properly work, the user need to re-create the same sequence of actions and conditions. Please use the ThorImageOCT software to perform probe calibrations, if at all necessary.

7.1.5.19 BOOL visualCalibrate_previewImage (VisualCalibrationHandle *Handle,* ColoredDataHandle *Image*)

Provides a preview image for the current calibration.

Parameters

in	*Handle*	A handle of a valid (non null) visual calibration (VisualCalibrationHandle), previously created with the function createVisualCalibration.
out	*Image*	A valid (non null) handle of colored data (ColoredDataHandle). The preview image will be written here.

Warning

ThorImageOCT uses this and other functions to calibrate the galvo, assuming a very specific sequence of actions and conditions, as explained in the ThorImageOCT. For this function to properly work, the user need to re-create the same sequence of actions and conditions. Please use the ThorImageOCT software to perform probe calibrations, if at all necessary.

7.1.5.20 const char ∗ visualCalibrate_Status (VisualCalibrationHandle *Handle*)

Gives a status message of the currently executed visual calibration.

Parameters

in	*Handle*	A handle of a valid (non null) visual calibration (VisualCalibrationHandle), previously created with the function createVisualCalibration.

Returns

The status message of the currently executed visual calibration.

Warning

ThorImageOCT uses this and other functions to calibrate the galvo, assuming a very specific sequence of actions and conditions, as explained in the ThorImageOCT. For this function to properly work, the user need to re-create the same sequence of actions and conditions. Please use the ThorImageOCT software to perform probe calibrations, if at all necessary.

7.1.5.21 void visualCalibration_getHoles (VisualCalibrationHandle *Handle,* int ∗ *x0,* int ∗ *y0,* int ∗ *x1,* int ∗ *y1,* int ∗ *x2,* int ∗ *y2*)

provides currently located hole positions of the three-hole target.

Parameters

in	*Handle*	A handle of a valid (non null) visual calibration (VisualCalibrationHandle), previously created with the function createVisualCalibration.
out	*x0*	The x-coordinate of the first hole.
out	*y0*	The y-coordinate of the first hole.
out	*x1*	The x-coordinate of the second hole.
out	*y1*	The y-coordinate of the second hole.
out	*x2*	The x-coordinate of the third hole.
out	*y2*	The y-coordinate of the third hole.

Warning

ThorImageOCT uses this and other functions to calibrate the galvo, assuming a very specific sequence of actions and conditions, as explained in the ThorImageOCT. For this function to properly work, the user need to re-create the same sequence of actions and conditions. Please use the ThorImageOCT software to perform probe calibrations, if at all necessary.