SpectralRadar SDK

4.4

Generated by Doxygen 1.8.4

Mon Jun 26 2017 14:01:41

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 -> Spectral-Radar-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 out 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, closeProcessing 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.
- Coloring32BitHandle: Handle to processing routines that can map floating point data to 32 bit color data.

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 approprate 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
closeProcessing(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 angle 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 = createRealData();
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 NO_ERROR 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 Error Handling

Error handling.

Enumerations

- enum ErrorCode {
  NoError = 0x00000000,
  Error = 0xE0000000 }
  This enum is used to describe errors that occur when operating an OCT device.

Functions

- 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.

5.1.1 Detailed Description

Error handling.

5.1.2 Enumeration Type Documentation

5.1.2.1 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.

5.1.3 Function Documentation

5.1.3.1 ErrorCode getError (char * *Message*, int *StringSize*)

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

Parameters

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

5.2 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 DataPropertyInt {
  Data_Dimensions,
  Data_Size1,
  Data_Size2,
  Data_Size3,
  Data_NumberOfElements,
  Data_SizeInBytes,
  Data_BytesPerElement }

  Selects integer point data property.
- enum RawDataPropertyInt {
  RawData_Size1,
  RawData_Size2,
  RawData_Size3,
  RawData_NumberOfElements,
  RawData_SizeInBytes,
  RawData_BytesPerElement,
  RawData_LostFrames }

  Specifies properties of RawData.
- enum DataPropertyFloat {
  Data_Spacing1,
  Data_Spacing2,
  Data_Spacing3,
  Data_Range1,
  Data_Range2,
  Data_Range3 }

  Selects floating point data property.

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

  Selects data property to analyze.
- 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 }

  Selects an appropriate A-Scan analyzation.
- enum DataOrientation {
  DataOrientation_ZXY,
  DataOrientation_ZYX,
  DataOrientation_XZY,
  DataOrientation_XYZ,
  DataOrientation_YXZ,
  DataOrientation_YZX,
  DataOrientation_ZTX,
  DataOrientation_ZXT }

  Selects the orientation of the data.

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)

  Returns a pointer to the content of the specified data.
- 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 chung 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 void copyComplexDataContent (ComplexDataHandle DataSource, ComplexFloat *Destination)

  Copies the content of the complex data to the pointer specified as destination.
- SPECTRALRADAR_API ComplexFloat * getComplexDataPtr (ComplexDataHandle Data)

  Returns a pointer to the data represented by the ComplexDataHandle. The data is still managed by the ComplexDataHandle object.
- 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 Alignment1)

  Copies the data in the specified colored data object (ColoredDataHandle) into the specified pointer. This function assues the data to be aligned accordingly.
- SPECTRALRADAR_API unsigned long * getColoredDataPtr (ColoredDataHandle ColData)

  Returns a pointer to the content of the specified ColoredDataHandle.
- 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)

  Sets the data oritentation of the colored data object to the given orientation.
- SPECTRALRADAR_API void getRawDataSize (RawDataHandle Raw, int ∗SizeX, int ∗SizeY, int ∗SizeZ)

  Returns the size of the specified raw data (RawDataHandle).
- SPECTRALRADAR_API void copyRawDataContent (RawDataHandle RawDataSource, void ∗DataContent)

  Copies the content of the raw data into the specified buffer. The suer needs to assure that enough memory is allocated.
- SPECTRALRADAR_API void ∗ getRawDataPtr (RawDataHandle RawDataSource)

  Returns the pointer to the raw data content. The pointer might no longer after additional actions using the RawDataHandle.
- SPECTRALRADAR_API int getRawDataPropertyInt (RawDataHandle RawData, RawDataPropertyInt Property)

  Returns a raw data property.
- SPECTRALRADAR_API void setRawDataBytesPerPixel (RawDataHandle Raw, int BytesPerPixel)

  Sets the bytes per pixel for raw data.
- SPECTRALRADAR_API void resizeRawData (RawDataHandle Raw, int Size1, int Size2, int Size3)

  Resizes the specified raw data buffer accordingly.
- SPECTRALRADAR_API void setRawDataContent (RawDataHandle RawDataSource, 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)

  Sets the number of the spectra in the raw data that are used for creating A-scan/B-scan data.
- SPECTRALRADAR_API void setApodizationSpectra (RawDataHandle RawData, int NumberOfScanRegions, int ∗ApodizationRegions)

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

5.2.1 Detailed Description

Functions for accessing the information stored in data objects.

5.2.2 Typedef Documentation

5.2.2.1 ColoredDataHandle

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

5.2.2.2 ComplexDataHandle

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

5.2.2.3 DataHandle

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

5.2.2.4 ImageFieldHandle

Handle to the image field description.

5.2.2.5 OCTFileHandle

Handle to the OCT file class.

5.2.2.6 RawDataHandle

Handle to an object holding the unprocessed raw data.

5.2.3 Enumeration Type Documentation

5.2.3.1 enum AScanAnalyzation

Selects an appropriate A-Scan analyzation.

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 hieght 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.

5.2.3.2 enum DataAnalyzation

Selects data property to analyze.

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.

5.2.3.3 enum DataOrientation

Selects the orientation of the data.

5.2.3.4 enum DataPropertyFloat

Selects floating point data property.

Enumerator

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

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

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

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

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

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

5.2.3.5 enum DataPropertyInt

Selects integer point data property.

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.

5.2.3.6 enum RawDataPropertyInt

Specifies properties of RawData.

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.

5.2.4 Function Documentation

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

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

5.2.4.2 void copyColoredDataContent (ColoredDataHandle *Source*, unsigned long * *Destination*)

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

5.2.4.3 void copyColoredDataContentAligned (ColoredDataHandle *ImageSource,* unsigned long * *Destination,* int *Alignment1*)

Copies the data in the specified colored data object (ColoredDataHandle) into the specified pointer. This function assues the data to be aligned accordingly.

5.2.4.4 void copyComplexDataContent (ComplexDataHandle *DataSource,* ComplexFloat * *Destination*)

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

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

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

5.2.4.6 void copyDataContent (DataHandle *DataSource,* float * *Destination*)

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

5.2.4.7 void copyRawDataContent (RawDataHandle *RawDataSource,* void * *DataContent*)

Copies the content of the raw data into the specified buffer. The suer needs to assure that enough memory is allocated.

5.2.4.8 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.

An array needs to be provided that has twice the sice of the number of apodization regions which can be obtained by getNumberOfApodizationRegions()

5.2.4.9 DataOrientation getColoredDataOrientation (ColoredDataHandle *Data*)

Returns the data orientation of the colored data object.

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

Returns the selected integer property of the specified colored data.

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

Returns the selected integer property of the specified colored data.

5.2.4.12 unsigned long * getColoredDataPtr (ColoredDataHandle *ColData*)

Returns a pointer to the content of the specified ColoredDataHandle.

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

Returns the selected integer property of the specified data.

5.2.4.14 ComplexFloat * getComplexDataPtr (ComplexDataHandle *Data*)

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

5.2.4.15 void DataOrientation getDataOrientation (DataHandle *Data*)

Returns the data orientation of the data object.

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

Returns the selected floating point property of the specified data.

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

Returns the selected integer property of the specified data.

5.2.4.18 float * getDataPtr (DataHandle *Data*)

Returns a pointer to the content of the specified data.

5.2.4.19 int getNumberOfApodizationRegions (RawDataHandle *Raw*)

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

5.2.4.20 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.

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

Returns a raw data property.

5.2.4.22 void * getRawDataPtr (RawDataHandle *RawDataSource*)

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

5.2.4.23 void getRawDataSize (RawDataHandle *Raw,* int * *SizeX,* int * *SizeY,* int * *SizeZ*)

Returns the size of the specified raw data (RawDataHandle).

5.2.4.24 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.

An array needs to be provided that has twice the sice of the number of scan regions which can be obtained by getNumberOfScanRegions()

5.2.4.25 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.

5.2.4.26 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.

5.2.4.27 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.

5.2.4.28 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.

5.2.4.29 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.

5.2.4.30 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.

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

Resizes the specified raw data buffer accordingly.

5.2.4.32 void setApodizationSpectra (RawDataHandle *RawData,* int *NumberOfScanRegions,* int * *ApodizationRegions*)

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

5.2.4.33 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).

5.2.4.34 void setColoredDataOrientation (ColoredDataHandle *Data,* DataOrientation)

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

5.2.4.35 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.

5.2.4.36 void setComplexDataContent (ComplexDataHandle *Data,* ComplexFloat * *NewContent*)

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

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

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

5.2.4.38 void setDataContent (DataHandle *Data,* float * *NewContent*)

Sets the data content of the 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(float).

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

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

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

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

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

Sets the bytes per pixel for raw data.

5.2.4.42 void setRawDataContent (RawDataHandle *RawDataSource,* 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.

5.2.4.43 void setScanSpectra (RawDataHandle *RawData*, int *NumberOfScanRegions*, int * *ScanRegions*)

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

5.3 Data Creation and Clearing

Functions to create and clear object containing data.

Functions

- SPECTRALRADAR_API RawDataHandle createRawData (void)

  Creates a raw data object (RawDataHandle).
- SPECTRALRADAR_API void clearRawData (RawDataHandle Raw)

  Clears a raw data object (RawDataHandle)
- 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, DataHandle, DataHandle or DataHandle objects.
- 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.
- SPECTRALRADAR_API void clearComplexData (ComplexDataHandle Data)

  Clears a data object holding complex data.

5.3.1 Detailed Description

Functions to create and clear object containing data.

5.3.2 Function Documentation

5.3.2.1 void clearColoredData (ColoredDataHandle *Volume*)

Clears a colored volume object.

5.3.2.2 void clearComplexData (ComplexDataHandle *Data*)

Clears a data object holding complex data.

5.3.2.3 void clearData (DataHandle *Data*)

Clears the specified DataHandle, DataHandle, DataHandle or DataHandle objects.

5.3.2.4 void clearRawData (RawDataHandle *Raw*)

Clears a raw data object (RawDataHandle)

5.3.2.5 ColoredDataHandle createColoredData (void)

Creates a colored data object (ColoredDataHandle).

5.3.2.6 ComplexDataHandle createComplexData (void)

Creates a data object holding complex data.

5.3.2.7 DataHandle createData (void)

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

5.3.2.8 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).

5.3.2.9 RawDataHandle createRawData (void)

Creates a raw data object (RawDataHandle).

5.4 Hardware

Functions providing direct access to OCT Hardware functionality.

Typedefs

- 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 }

  Properties of the device that can be read or measured.
- enum DevicePropertyInt {
  Device_SpectrumElements,
  Device_BytesPerElement,
  Device_MaxLiveVolumeRenderingScans,
  Device_BitDepth,
  Device_DataIsSigned }

  Properties of the device that can be read or measured.
- enum ScanAxis {
  ScanAxis_X = 0,
  ScanAxis_Y = 1 }

  used to select the axis for manual galvo operations.
- enum Device_CameraPreset {
  Device_CameraPreset_Default,
  Device_CameraPreset_1,
  Device_CameraPreset_2,
  Device_CameraPreset_3,
  Device_CameraPreset_4 }

  Enum identifying sensitivity and acquisition speed of the device.

Functions

- SPECTRALRADAR_API OCTDeviceHandle initDevice (void)

  Initializes the installed device.
- SPECTRALRADAR_API void getDeviceType (OCTDeviceHandle Dev, char DevName[], int BufferSize)

  Gives the name of the device type that is given by the OCTDeviceHandle.
- SPECTRALRADAR_API int getDeviceRevision (OCTDeviceHandle Dev)

  Returns the revision of the device given by the OCTDeviceHandle.

- SPECTRALRADAR_API void getDeviceSerialNumber (OCTDeviceHandle Dev, char DevName[], int BufferSize)

  Returns the serial number of the device given by the OCTDeviceHandle.
- SPECTRALRADAR_API int getDevicePropertyInt (OCTDeviceHandle Dev, DevicePropertyInt 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 void closeDevice (OCTDeviceHandle Dev)

  Closes the device opened previously with initDevice.
- SPECTRALRADAR_API BOOL isDeviceOn (OCTDeviceHandle Handle)

  Returns if the device is switched on.
- SPECTRALRADAR_API BOOL isVideoCameraAvailable (OCTDeviceHandle Dev)

  Returns if the video camera is available.
- SPECTRALRADAR_API BOOL isSLDAvailable (OCTDeviceHandle Dev)

  Returns whethter the SLD is available.
- SPECTRALRADAR_API void setSLD (OCTDeviceHandle Dev, BOOL OnOff)

  switches the SLD of the SpectralRadar device on and off.
- SPECTRALRADAR_API void moveScanner (OCTDeviceHandle Dev, ProbeHandle Probe, ScanAxis Axis, double Position)

  manually moves the scanner to a given position
- SPECTRALRADAR_API void setLaserDiode (OCTDeviceHandle Dev, BOOL OnOff)

  switches the LaserDiode of the SpectralRadar device on and off.
- SPECTRALRADAR_API double getWavelengthAtPixel (OCTDeviceHandle Dev, int Pixel)

  Returns the wavelength at a speicified pixel of the spectrometer.
- SPECTRALRADAR_API int getCameraPreset (OCTDeviceHandle Dev)

  Gets the currently used device preset.

5.4.1 Detailed Description

Functions providing direct access to OCT Hardware functionality.

5.4.2 Typedef Documentation

5.4.2.1 OCTDeviceHandle

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

5.4.3 Enumeration Type Documentation

5.4.3.1 enum Device_CameraPreset

Enum identifying sensitivity and acquisition speed of the device.

Enumerator

Device_CameraPreset_Default Default device preset. Most common compromise of acquisition speed and sensitivity.

Device_CameraPreset_1 Device preset 1.

Device_CameraPreset_2 Device preset 2.

Device_CameraPreset_3 Device preset 3.

Device_CameraPreset_4 Device preset 4.

5.4.3.2 enum DevicePropertyFloat

Properties of the device that can be read or measured.

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 spectral width of the spectrometer.

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

5.4.3.3 enum DevicePropertyInt

Properties of the device that can be read or measured.

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_DataIsSigned Flag indicating if the data is signed.

5.4.3.4 enum ScanAxis

used to select the axis for manual galvo operations.

Enumerator

ScanAxis_X X-Axis of the scanner.

ScanAxis_Y Y-Axis of the scanner.

5.4.4 Function Documentation

5.4.4.1 void closeDevice (OCTDeviceHandle *Dev*)

Closes the device opened previously with initDevice.

Parameters

Dev	The OCTDeviceHandle that was initially provided by initDevice.

5.4.4.2 void Device_CameraPreset getCameraPreset (OCTDeviceHandle *Dev*)

Gets the currently used device preset.

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

Returns properties of the device belonging to the specfied OCTDeviceHandle.

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

Returns properties of the device belonging to the specfied OCTDeviceHandle.

5.4.4.5 int getDeviceRevision (OCTDeviceHandle *Dev*)

Returns the revision of the device given by the OCTDeviceHandle.

5.4.4.6 int void getDeviceSerialNumber (OCTDeviceHandle *Dev*, char *DevName[]*, int *BufferSize*)

Returns the serial number of the device given by the OCTDeviceHandle.

5.4.4.7 void getDeviceType (OCTDeviceHandle *Dev*, char *DevName[]*, int *BufferSize*)

Gives the name of the device type that is given by the OCTDeviceHandle.

5.4.4.8 double getWavelengthAtPixel (OCTDeviceHandle *Dev*, int *Pixel*)

Returns the wavelength at a speicified pixel of the spectrometer.

Warning

This function is still experimental and results might be incorrect.

5.4.4.9 OCTDeviceHandle initDevice (void)

Initializes the installed device.

Returns

Handle to the initialized OCT device.

5.4.4.10 BOOL isDeviceOn (OCTDeviceHandle *Handle*)

Returns if the device is switched on.

Parameters

Dev	The OCTDeviceHandle that was initially provided by initDevice.

5.4.4.11 BOOL isSLDAvailable (OCTDeviceHandle *Dev*)

Returns whethter the SLD is available.

Parameters

Dev	The OCTDeviceHandle that was initially provided by initDevice.

5.4.4.12 BOOL isVideoCameraAvailable (OCTDeviceHandle *Dev*)

Returns if the video camera is available.

Parameters

Dev	The OCTDeviceHandle that was initially provided by initDevice.

5.4.4.13 void moveScanner (OCTDeviceHandle *Dev,* ProbeHandle *Probe,* ScanAxis *Axis,* double *Position*)

manually moves the scanner to a given position

Parameters

Dev	the OCTDeviceHandle that was initially provided by initDevice.
Probe	A handle to the probe (ProbeHandle); whose galvo position is to be set.
Axis	the axis in which you want to set the position manually
Position	the actual position you want to move the galvo to.

5.4.4.14 void setLaserDiode (OCTDeviceHandle *Dev,* BOOL *OnOff*)

switches the LaserDiode of the SpectralRadar device on and off.

Warning

Not all devices are equiped

Parameters

Dev	handle to the OCTDeviceHandle that was initially provided by initDevice.
OnOff	TRUE switches the VisLD on, FALSE swichted the VisLD off.

5.4.4.15 void setSLD (OCTDeviceHandle *Dev,* BOOL *OnOff*)

switches the SLD of the SpectralRadar device on and off.

Parameters

Dev	handle to the OCTDeviceHandle that was initially provided by initDevice.
OnOff	TRUE switches the SLD on, FALSE swichted the SLD off.

5.5 Internal Values

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

Functions

- SPECTRALRADAR_API int getNumberOfInternalValues (OCTDeviceHandle Dev)

  Returns the number of Analog-to-Digital Converter present in the device.
- SPECTRALRADAR_API void getInternalValueName (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 getInternalValueByName (OCTDeviceHandle Dev, const char ∗Name)

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

  Returns the value of the selected ADC.

5.5.1 Detailed Description

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

5.5.2 Function Documentation

5.5.2.1 double getInternalValueByIndex (OCTDeviceHandle *Dev,* int *Index*)

Returns the value of the selected ADC.

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

5.5.2.2 double getInternalValueByName (OCTDeviceHandle *Dev,* const char ∗ *Name*)

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

The ADC is specified by the name returned by getInternalValueName.

5.5.2.3 void getInternalValueName (OCTDeviceHandle *Dev,* int *Index,* char ∗ *Name,* int *NameStringSize,* char ∗ *Unit,* int *UnitStringSize*)

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

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

5.5.2.4 int getNumberOfInternalValues (OCTDeviceHandle *Dev*)

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

5.6 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.

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_WhiteBalanceRed,
  Probe_WhiteBalanceGreen,
  Probe_WhiteBalanceBlue,
  Probe_RangeMaxX,
  Probe_RangeMaxY,
  Probe_MaximumSlope_XY,
  Probe_SpeckleSize,
  Probe_ApoPosX,
  Probe_ApoPosY,
  Probe_ReferenceStageOffset }

  Parameters describing the behaviour of the Probe, such as calibration factors and scan parameters.
- enum ProbeParameterInt {
  Probe_ApodizationCycles,
  Probe_Oversampling,
  Probe_WhiteBalanceAutomatic,
  Probe_Oversampling_SlowAxis,
  Probe_SpeckleReduction,
  Probe_MaxScanRangeShape }

  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 ProbeType {
  ProbeType_Standard,
  ProbeType_Handheld,
  ProbeType_Scientific }

  Determines the kind of probe types.

Functions

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

  Initializes a probe specified by ProbeFile.
- SPECTRALRADAR_API ProbeHandle initStandardProbe (OCTDeviceHandle Dev)

  Creates a standard probe using the Probe.ini file. If this configuration file is not found, standard parameters without valid calibration will be used.
- SPECTRALRADAR_API ProbeHandle initProbeWithType (OCTDeviceHandle Dev, ProbeType Type)

  Creates a standard probe for the given probe type but without valid calibration data .
- 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.
- 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 getProbeName (ProbeHandle Probe, char ProbeName[], int BufferSize)

  Returns the name of the specified probe.
- SPECTRALRADAR_API void setProbeName (ProbeHandle Probe, const char *ProbeName)

  Sets the given name of the specified probe.
- SPECTRALRADAR_API void getProbeSerialNo (ProbeHandle Probe, char SerialNo[], int BufferSize)

  Gets the serial number of the specified probe.
- SPECTRALRADAR_API void setProbeSerialNo (ProbeHandle Probe, const char *SerialNo)

  Gets the serial number of the specified probe.
- SPECTRALRADAR_API void getProbeType (ProbeHandle Probe, char Type[], int BufferSize)

  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 getProbeObjective (ProbeHandle Probe, char Objective[], int BufferSize)

  Gets the objective of the specified probe.
- SPECTRALRADAR_API void setProbeObjective (ProbeHandle Probe, const char *Objective)

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

  Closes the probe and frees all memory associated with it.
- SPECTRALRADAR_API void blendEnFaceInCamera (ProbeHandle Probe, ScanPatternHandle Pattern, ColoredDataHandle EnFace2D, ColoredDataHandle Image, float Ratio, BOOL DenseView)

  Blends the en-face image of a given volume acquisition on top of the video image. Can be used to calibrate the probe manually.
- 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 needs to be assured that the device is properly calibrated.
- 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.

5.6.1 Detailed Description

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

5.6.2 Typedef Documentation

5.6.2.1 ProbeHandle

Handle for controlling the galvo scanner.

5.6.3 Enumeration Type Documentation

5.6.3.1 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.

5.6.3.2 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_WhiteBalanceRed White balance settings will only take effect on initialization of the probe. White balance value for red channel of video camera (if -1, adjustment will be automatic).

Probe_WhiteBalanceGreen White balance value for green channel of video camera (if -1, adjustment will be automatic).

Probe_WhiteBalanceBlue White balance value for blue channel of video camera (if -1, adjustment will be automatic).

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_ApoPosX X-position used to acquire the apodization spectrum.

Probe_ApoPosY Y-position used to acquire the apodization spectrum.

Probe_ReferenceStageOffset Offset for reference stage marking the zero delay line.

5.6.3.3 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_WhiteBalanceAutomatic Automatic white balance for video camera, 0 == off, not 0 == on.

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.

Probe_MaxScanRangeShape Shape of the maximum scan range: 0 is a rectangle, 1 is an ellipse.

5.6.3.4 enum ProbeType

Determines the kind of probe types.

Enumerator

ProbeType_Standard Specifies the standard or general probe.

ProbeType_Handheld Specfies the handheld probe.

ProbeType_Scientific Specifies the scientific probe.

5.6.4 Function Documentation

5.6.4.1 void blendEnFaceInCamera (ProbeHandle *Probe,* ScanPatternHandle *Pattern,* ColoredDataHandle *EnFace2D,* ColoredDataHandle *Image,* float *Ratio,* BOOL *DenseView*)

Blends the en-face image of a given volume acquisition on top of the video image. Can be used to calibrate the probe manually.

5.6.4.2 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 needs to be assured that the device is properly calibrated.

5.6.4.3 void closeProbe (ProbeHandle *Probe*)

Closes the probe and frees all memory associated with it.

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

Returns the selected boolean value of the specified probe.

5.6.4.5 void getProbeName (ProbeHandle *Probe,* char *ProbeName[],* int *BufferSize*)

Returns the name of the specified probe.

5.6.4.6 void getProbeObjective (ProbeHandle *Probe,* char *Objective[],* int *BufferSize*)

Gets the objective of the specified probe.

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

Gets floating point parameters of the specified probe.

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

Gets integer parameters of the specified probe.

5.6.4.9 void getProbeSerialNo (ProbeHandle *Probe,* char *SerialNo[],* int *BufferSize*)

Gets the serial number of the specified probe.

5.6.4.10 void getProbeType (ProbeHandle *Probe,* char *Type[],* int *BufferSize*)

Gets the type of the specified probe.

5.6.4.11 ProbeHandle initProbe (OCTDeviceHandle *Dev,* const char * *ProbeFile*)

Initializes a probe specified by ProbeFile.

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.

```
It is recommended to use #initStandardProbe for systems manufactured in or after May 2011.
```

5.6.4.12 ProbeHandle initProbeWithType (OCTDeviceHandle *Dev,* ProbeType *Type*)

Creates a standard probe for the given probe type but without valid calibration data .

5.6.4.13 ProbeHandle initStandardProbe (OCTDeviceHandle *Dev*)

Creates a standard probe using the Probe.ini file. If this configuration file is not found, standard parameters without valid calibration will be used.

5.6.4.14 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.

5.6.4.15 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.

5.6.4.16 void setProbeName (ProbeHandle *Probe,* const char * *ProbeName*)

Sets the given name of the specified probe.

5.6.4.17 void setProbeObjective (ProbeHandle *Probe,* const char * *Objective*)

Sets the given objective of the specified probe.

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

Sets floating point parameters of the specified probe.

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

Sets.

5.6.4.20 void setProbeSerialNo (ProbeHandle *Probe,* const char * *SerialNo*)

Gets the serial number of the specified probe.

5.6.4.21 void setProbeType (ProbeHandle *Probe,* const char * *Type*)

Sets the type of the specified probe.

5.7 Scan Pattern

Functions that describe the movement of the Scanner during measurement.

Typedefs

- typedef C_ScanPattern * ScanPatternHandle

  Handle for controlling the scan pattern.

Functions

- SPECTRALRADAR_API ScanPatternHandle createPointScanPattern (ProbeHandle Probe, int Size, double PosX, double PosY)

  Creates a scan pattern that consists of a single point (PosX, PosY). The galvo doesn't move from there. Use this pattern for point scans and/or non-scanning probes.
- SPECTRALRADAR_API ScanPatternHandle createNoScanPattern (ProbeHandle Probe, int Scans, int NumberOfScans)

  Creates a simple scan pattern that does not move the galvo. Use this pattern for point scans and/or non-scanning probes.
- SPECTRALRADAR_API ScanPatternHandle createTriggerPattern (ProbeHandle Probe, int Scans)

  Creates a pattern only consisting of a specified amount of trigger signals.
- SPECTRALRADAR_API ScanPatternHandle createBScanPattern (ProbeHandle Probe, double Range, int AScans, BOOL apodization)

  Creates a simple B-scan pattern that moves the galvo over a specified range.
- SPECTRALRADAR_API ScanPatternHandle createBilateralBScanPattern (ProbeHandle Probe, double Range, int AScans, double Shift)

  Creates a bilateral scan pattern. The contouring error can be influenced using the Shift parameter.
- SPECTRALRADAR_API ScanPatternHandle createBScanPatternManual (ProbeHandle Probe, double StartX, double StartY, double StopX, double StopY, int AScans, BOOL apodization)

  Creates a B-scan pattern specified by start and end points.
- SPECTRALRADAR_API ScanPatternHandle createIdealBScanPattern (ProbeHandle Probe, double Range, 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, int AScans)

  Creates a circle scan pattern.
- SPECTRALRADAR_API ScanPatternHandle createVolumePattern (ProbeHandle Probe, double RangeX, int SizeX, double RangeY, int SizeY)

  Creates a simple volume pattern.
- SPECTRALRADAR_API ScanPatternHandle createBScanStackPattern (ProbeHandle Probe, double RangeX, int SizeX, double RangeY, int SizeY)

  Creates a simple stack pattern.
- SPECTRALRADAR_API ScanPatternHandle createFreeformScanPattern (ProbeHandle Probe, float *positions, int size_x, int size_y, BOOL apodization)

  Creates a freeform scan pattern based on an array of positions.
- SPECTRALRADAR_API ScanPatternHandle createFragmentedScanPattern (ProbeHandle Probe, int ChunkSize, int NumberOfChunks)

  Creates a scan pattern which can be used to acquire a dataset of <NumberOfChunks> times <ChunkSize> A-scans at position 0/0. The Fragmented scan pattern can be compared in structure to a B-scan stack pattern with x and y ranges of 0; however the fragmented scan pattern behaves like a volume pattern in that it shows no delay between the respective chunks.
- SPECTRALRADAR_API void updateScanPattern (ScanPatternHandle Pattern)

Updates the specfied pattern (ScanPatternHandle);.

- SPECTRALRADAR_API void rotateScanPattern (ScanPatternHandle Pattern, double Angle)

  Rotates the specfied pattern (ScanPatternHandle);.
- SPECTRALRADAR_API void rotateScanPatternExt (ScanPatternHandle Pattern, double Angle, int index)

  Rotates the scan #index (0-based) of the specfied pattern (ScanPatternHandle).
- SPECTRALRADAR_API void shiftScanPattern (ScanPatternHandle Pattern, double ShiftX, double ShiftY)

  Shifts the specified pattern (ScanPatternHandle).
- SPECTRALRADAR_API void shiftScanPatternExt (ScanPatternHandle Pattern, double ShiftX, double ShiftY, BOOL ShiftApo, int Index)

  Shifts the scan #index (0-based) of the specified pattern (ScanPatternHandle).
- SPECTRALRADAR_API void zoomScanPattern (ScanPatternHandle Pattern, double Factor)

  Zooms the specified pattern (ScanPatternHandle).
- SPECTRALRADAR_API int getScanPatternLUTSize (ScanPatternHandle Pattern)

  Returns the number of data points the specified pattern (ScanPatternHandle) used.
- SPECTRALRADAR_API void getScanPatternLUT (ScanPatternHandle Pattern, double *PosX, double *PosY)

  Returns the actual positions to be scanned with the specified pattern (ScanPatternHandle).
- SPECTRALRADAR_API void clearScanPattern (ScanPatternHandle Pattern)

  Clears the specified scan pattern (ScanPatternHandle).

5.7.1 Detailed Description

Functions that describe the movement of the Scanner during measurement.

5.7.2 Typedef Documentation

5.7.2.1 ScanPatternHandle

Handle for controlling the scan pattern.

5.7.3 Function Documentation

5.7.3.1 void clearScanPattern (ScanPatternHandle *Pattern*)

Clears the specified scan pattern (ScanPatternHandle).

5.7.3.2 ScanPatternHandle createBilateralBScanPattern (ProbeHandle *Probe,* double *Range,* int *AScans,* double *Shift*)

Creates a bilateral scan pattern. The contouring error can be influenced using the Shift parameter.

5.7.3.3 ScanPatternHandle createBScanPattern (ProbeHandle *Probe,* double *Range,* int *AScans,* BOOL *apodization*)

Creates a simple B-scan pattern that moves the galvo over a specified range.

5.7.3.4 ScanPatternHandle createBScanPatternManual (ProbeHandle *Probe,* double *StartX,* double *StartY,* double *StopX,* double *StopY,* int *AScans,* BOOL *apodization*)

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

5.7.3.5 ScanPatternHandle createBScanStackPattern (ProbeHandle *Probe,* double *RangeX,* int *SizeX,* double *RangeY,* int *SizeY*)

Creates a simple stack pattern.

The BScan stack pattern is a volume measurement which consists of several shifted B-Scan measurements. The resulting data will be identical to a volume (see createVolumePattern()) but an apodization is performed for each slice (B-scan). The volume will be returned slice-by-slice by calling getRawData().

5.7.3.6 ScanPatternHandle createCirclePattern (ProbeHandle *Probe,* double *Radius,* int *AScans*)

Creates a circle scan pattern.

Warning

Circle patterns cannot be rotated properly.

5.7.3.7 ScanPatternHandle createFragmentedScanPattern (ProbeHandle *Probe,* int *ChunkSize,* int *NumberOfChunks*)

Creates a scan pattern which can be used to acquire a dataset of <NumberOfChunks> times <ChunkSize> A-scans at position 0/0. The Fragmented scan pattern can be compared in structure to a B-scan stack pattern with x and y ranges of 0; however the fragmented scan pattern behaves like a volume pattern in that it shows no delay between the respective chunks.

5.7.3.8 ScanPatternHandle createFreeformScanPattern (ProbeHandle *Probe,* float * *positions,* int *size_x,* int *size_y,* BOOL *apodization*)

Creates a freeform scan pattern based on an array of positions.

The positions array must consist of pairs of x/y coordinates inside the valid scanning limits of the probe and contain all points of the scan pattern. size_x/size_y can be used analogue to the B-scan stack pattern to discern the length (size_x) and number (size_y) of single lines or sections inside the freeform scan pattern. The position 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.

5.7.3.9 ScanPatternHandle createIdealBScanPattern (ProbeHandle *Probe,* double *Range,* 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.

5.7.3.10 ScanPatternHandle createNoScanPattern (ProbeHandle *Probe,* int *Scans,* int *NumberOfScans*)

Creates a simple scan pattern that does not move the galvo. Use this pattern for point scans and/or non-scanning probes.

5.7.3.11 ScanPatternHandle createPointScanPattern (ProbeHandle *Probe,* int *Size,* double *PosX,* double *PosY*)

Creates a scan pattern that consists of a single point (PosX, PosY). The galvo doesn't move from there. Use this pattern for point scans and/or non-scanning probes.

5.7.3.12 ScanPatternHandle createTriggerPattern (ProbeHandle *Probe,* int *Scans*)

Creates a pattern only consisting of a specified amount of trigger signals.

5.7.3.13 ScanPatternHandle createVolumePattern (ProbeHandle *Probe,* double *RangeX,* int *SizeX,* double *RangeY,* int *SizeY*)

Creates a simple volume pattern.

The volume pattern consists of a single uninterrupted scan and all data is acquired in a single measurement. In contrast to a B-scan stack pattern (see createBScanStackPattern()) only one apodization is performed for the complete volume. The complete volume will be returned in one raw data (RawDataHandle).

5.7.3.14 void getScanPatternLUT (ScanPatternHandle *Pattern,* double * *PosX,* double * *PosY*)

Returns the actual positions to be scanned with the specified pattern (ScanPatternHandle).

5.7.3.15 int getScanPatternLUTSize (ScanPatternHandle *Pattern*)

Returns the number of data points the specified pattern (ScanPatternHandle) used.

5.7.3.16 void rotateScanPattern (ScanPatternHandle *Pattern,* double *Angle*)

Rotates the specfied pattern (ScanPatternHandle);.

5.7.3.17 void rotateScanPatternExt (ScanPatternHandle *Pattern,* double *Angle,* int *index*)

Rotates the scan #index (0-based) of the specfied pattern (ScanPatternHandle).

5.7.3.18 void shiftScanPattern (ScanPatternHandle *Pattern,* double *ShiftX,* double *ShiftY*)

Shifts the specified pattern (ScanPatternHandle).

5.7.3.19 void shiftScanPatternExt (ScanPatternHandle *Pattern,* double *ShiftX,* double *ShiftY,* BOOL *ShiftApo,* int *Index*)

Shifts the scan #index (0-based) of the specified pattern (ScanPatternHandle).

5.7.3.20 void updateScanPattern (ScanPatternHandle *Pattern*)

Updates the specfied pattern (ScanPatternHandle);.

5.7.3.21 void zoomScanPattern (ScanPatternHandle *Pattern,* double *Factor*)

Zooms the specified pattern (ScanPatternHandle).

5.8 Acquisition

Functions for acquisition.

Enumerations

- enum AcquisitionType {
  Acquisition_AsyncContinuous,
  Acquisition_AsyncFinite,
  Acquisition_Sync }

  Determines the kind of acquisition process.

Functions

- 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 N spectra of raw data without moving galvo scanners.
- SPECTRALRADAR_API void measureSpectraEx (OCTDeviceHandle Dev, int N, RawDataHandle Raw, int CameraIndex)

  Acquires N spectra of raw data without moving galvo scanners. Supports multiple cameras (e.g. PS-OCT).

5.8.1 Detailed Description

Functions for acquisition.

5.8.2 Enumeration Type Documentation

5.8.2.1 enum AcquisitionType

Determines the kind of acquisition process.

Enumerator

Acquisition_AsyncContinuous Specifies an asynchronous infinite/continuous measurement. The internal memory management is constructed with a loop of buffers. Note that you may lose data if the acquisition is faster than the collection with getRawData().

Acquisition_AsyncFinite Specifies an asynchronous finite measurement. The reserved memory for the data is as large as required to make sure that no data will be lost.

Acquisition_Sync Specfies a synchronous measurement. Each measurement of e.g. a single B-scan will be started with getRawData(). Since the acquisition is synchronized with the software it is not a continious measurement over the time of e.g. several B-scans.

5.8.3 Function Documentation

5.8.3.1 void getRawData (OCTDeviceHandle *Dev*, RawDataHandle *RawData*)

Acquires data and stores the data unprocessed.

5.8.3.2 void void getRawDataEx (OCTDeviceHandle *Dev*, RawDataHandle *RawData*, int *CameraIdx*)

Acquires data with the specific camera given with camera index and stores the data unprocessed.

5.8.3.3 void measureSpectra (OCTDeviceHandle *Dev*, int *NumberOfSpectra*, RawDataHandle *Raw*)

Acquires N spectra of raw data without moving galvo scanners.

5.8.3.4 void measureSpectraEx (OCTDeviceHandle *Dev*, int *N*, RawDataHandle *Raw*, int *CameraIndex*)

Acquires N spectra of raw data without moving galvo scanners. Supports multiple cameras (e.g. PS-OCT).

5.8.3.5 void startMeasurement (OCTDeviceHandle *Dev*, ScanPatternHandle *Pattern*, AcquisitionType *type*)

starts a continuous measurement BScans.

Scanning takes place according to the specified scan pattern handle. Data can be recorded using the getRawData() function. If you are done, call stopMeasurement().

Parameters

Dev	The OCTDeviceHandle that was initially provided by initDevice.
Pattern	the ScanPatternHandle

5.8.3.6 void stopMeasurement (OCTDeviceHandle *Dev*)

stops the current measurement.

Parameters

Dev	The OCTDeviceHandle that was initially provided by initDevice.

5.9 Processing

Standard Processing Routines.

Typedefs

- typedef struct C_Processing * ProcessingHandle

  Handle for a processing routine.

Enumerations

- enum ProcessingType {
  Processing_StandardFFT,
  Processing_StandardNDFT,
  Processing_iFFT1,
  Processing_iFFT2,
  Processing_iFFT3,
  Processing_iFFT4,
  Processing_NFFT1,
  Processing_NFFT2,
  Processing_NFFT3,
  Processing_NFFT4 }

  defindes the algorithm used for dechirping the input signal and Fourier transformation
- 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 }

  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 }

  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 }

  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 createProcessingForDevice (OCTDeviceHandle Dev)

  Creates suitable standard processing routines for the specified device (OCTDeviceHandle).
- SPECTRALRADAR_API ProcessingHandle createProcessingForDeviceEx (OCTDeviceHandle Dev, int CameraIndex)

  Creates suitable standard processing routines for the specified device (OCTDeviceHandle) with camera index.
- SPECTRALRADAR_API int getInputSize (ProcessingHandle Proc)

  Returns the expected input size (pixels per spectrum); of the processing algorithms.
- SPECTRALRADAR_API int getAScanSize (ProcessingHandle Handle)

  gives the number of pixels in an A-Scan of the SpectralRadar device. This number is identical to the number of rows in a finished B-Scan.
- SPECTRALRADAR_API void setApodizationWindow (ProcessingHandle Proc, ApodizationWindow Window)

  Sets the window function that is to be used for apodization. The selected function will be used in all subsequent processings.
- SPECTRALRADAR_API int getApodizationWindow (ProcessingHandle Proc)

  Gets the window function that is being used for apodization.
- 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.

- SPECTRALRADAR_API double getApodizationWindowParameter (ProcessingHandle Proc, ApodizationWindowParameter Selection)

  Gets the apodization window parameter, such as window width or ratio between constant and cosine part.
- SPECTRALRADAR_API void setDechirpAlgorithm (ProcessingHandle Proc, ProcessingType Type)

  Sets the algorithm that is to be sued 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 double getProcessingParameterFloat (ProcessingHandle Proc, ProcessingParameterFloat Selection)

  Gets the specified processing paramter.
- 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, ProcessingAveragingAlgorithm Algorithm)

  Sets the algorithm that is used for averaing by the processing.
- SPECTRALRADAR_API void setCalibration (ProcessingHandle Proc, CalibrationData Selection, DataHandle Data)

  Sets the current active calibration data.
- SPECTRALRADAR_API void getCalibration (ProcessingHandle Proc, CalibrationData Selection, DataHandle Data)

  Returns the currently active calibration parameter.
- 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 measureSpectrum (OCTDeviceHandle Dev, ProbeHandle Probe, ProcessingHandle Proc, BOOL moveToApoPos)

  Measures apodization spectrum and uses them in subsequent processing.
- SPECTRALRADAR_API void saveCalibrationAuto (ProcessingHandle Proc, CalibrationData Selection)

  Saves the selected calibration in its default path.
- 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 use 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 ProcessingCorrectedSpectrum)

  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 ComplexBScan)

  Sets the pointer the resulting complex B-Scan of the next processing is written to.
- SPECTRALRADAR_API void setProcessedDataOutput (ProcessingHandle Proc, DataHandle Scan)

  Sets the pointer the resulting B-Scan of the next processing is written to.
- SPECTRALRADAR_API void setHorMirroredDataOutput (ProcessingHandle Proc, DataHandle Scan)

  Sets the pointer the resulting B-Scan of the next processing is written to. The result will be written mirrored at the horizontal axis.
- SPECTRALRADAR_API void setColoredDataOutput (ProcessingHandle Proc, ColoredDataHandle BScan, Coloring32BitHandle Color)

  Sets the pointer the resulting colored B-Scan of the next processing is written to.
- SPECTRALRADAR_API void setTransposedColoredDataOutput (ProcessingHandle Proc, ColoredDataHandle BScan, Coloring32BitHandle Color)

  Sets the pointer the resulting colored B-Scan of the next processing is written to. The data will be transposed so that the first axis is the x-axis.
- SPECTRALRADAR_API void executeProcessing (ProcessingHandle Proc, RawDataHandle RawData)

  Execute the processing.
- SPECTRALRADAR_API void closeProcessing (ProcessingHandle Proc)

  Closes the processing and frees all temporary memory that was associated with it. Processing threads will be stopped.
- SPECTRALRADAR_API void computeDispersion (DataHandle Spectrum1In, DataHandle Spectrum2In, DataHandle ChirpOut, DataHandle DispOut)

  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 QuadraticIn, DataHandle ChirpIn, DataHandle DispOut)

  Computes dispersion by a quadratic approximation specified by the quadratic factor.
- SPECTRALRADAR_API void computeDispersionByImage (DataHandle LinearKSpectraIn, DataHandle ChirpIn, DataHandle DispOut)

  Guesses the dispersion based on the raw data specified. The raw data needs to be linearized in k before applying to 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 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.

5.9.1 Detailed Description

Standard Processing Routines.

5.9.2 Typedef Documentation

5.9.2.1 ProcessingHandle

Handle for a processing routine.

5.9.3 Enumeration Type Documentation

5.9.3.1 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.

5.9.3.2 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.}

5.9.3.3 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.

5.9.3.4 enum ProcessingAveragingAlgorithm

This sets the averaging algorithm to be used for processing.

Warning

{This features is still experimental and might contain bugs.}

5.9.3.5 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.

5.9.3.6 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.}

5.9.3.7 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.

5.9.3.8 enum ProcessingType

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_iFFT1 Linear interpolation prior to FFT.

Processing_iFFT2 Linear interpolation with 2x oversampling prior to FFT.

Processing_iFFT3 Linear interpolation with 3x oversampling prior to FFT.

Processing_iFFT4 Linear interpolation with 4x oversampling 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.

5.9.4 Function Documentation

5.9.4.1 void closeProcessing (ProcessingHandle *Proc*)

Closes the processing and frees all temporary memory that was associated with it. Processing threads will be stopped.

5.9.4.2 void computeDispersion (DataHandle *Spectrum1In,* DataHandle *Spectrum2In,* DataHandle *ChirpOut,* DataHandle *DispOut*)

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.

5.9.4.3 void computeDispersionByCoeff (double *QuadraticIn,* DataHandle *ChirpIn,* DataHandle *DispOut*)

Computes dispersion by a quadratic approximation specified by the quadratic factor.

5.9.4.4 void computeDispersionByImage (DataHandle *LinearKSpectraIn,* DataHandle *ChirpIn,* DataHandle *DispOut*)

Guesses the dispersion based on the raw data specified. The raw data needs to be linearized in k before applying to this function.

5.9.4.5 void computeLinearKRawData (ComplexDataHandle *ComplexDataAfterFFT,* DataHandle *LinearKData*)

Computes the linear k raw data of the complex data after FFT by an inverse Fourier transform.

5.9.4.6 ProcessingHandle createProcessingForDevice (OCTDeviceHandle *Dev*)

Creates suitable standard processing routines for the specified device (OCTDeviceHandle).

5.9.4.7 ProcessingHandle createProcessingForDeviceEx (OCTDeviceHandle *Dev,* int *CameraIndex*)

Creates suitable standard processing routines for the specified device (OCTDeviceHandle) with camera index.

5.9.4.8 void executeProcessing (ProcessingHandle *Proc,* RawDataHandle *RawData*)

Execute the processing.

The specified raw data will be transformed. Results will be written to data objects specified by setProcessedDataOutput(), setComplexDataOutput(), setColoredDataOutput(), etc.

5.9.4.9 int getApodizationWindow (ProcessingHandle *Proc*)

Gets the window function that is being used for apodization.

Parameters

Proc	handle to the OCTDeviceHandle that was initially provided by initDevice.

5.9.4.10 double getApodizationWindowParameter (ProcessingHandle *Proc,* ApodizationWindowParameter *Selection*)

Gets the apodization window parameter, such as window width or ratio between constant and cosine part.

5.9.4.11 int getAScanSize (ProcessingHandle *Proc*)

gives the number of pixels in an A-Scan of the SpectralRadar device. This number is identical to the number of rows in a finished B-Scan.

Parameters

Proc	Processing that is used to get the A-Scan.

Returns

The number of pixels in an A-Scan of the SpectralRadar device.

5.9.4.12 void getCalibration (ProcessingHandle *Proc,* CalibrationData *Selection,* DataHandle *Data*)

Returns the currently active calibration parameter.

5.9.4.13 const char * getDispersionPresetName (ProcessingHandle *Proc,* int *Index*)

Gets the name of the dispersion preset specified with index.

5.9.4.14 int getInputSize (ProcessingHandle *Proc*)

Returns the expected input size (pixels per spectrum); of the processing algorithms.

This function is provided for convenience as processing routines can be used independently of the device.

5.9.4.15 int getNumberOfDispersionPresets (ProcessingHandle *Proc*)

Gets the number of dispersion presets.

5.9.4.16 BOOL getProcessingFlag (ProcessingHandle *Proc,* ProcessingFlag *Flag*)

Returns TRUE if the specified processing flag is set, FALSE otherwise.

5.9.4.17 double getProcessingParameterFloat (ProcessingHandle *Proc,* ProcessingParameterFloat *Selection*)

Gets the specified processing paramter.

5.9.4.18 int getProcessingParameterInt (ProcessingHandle *Proc,* ProcessingParameterInt *Selection*)

Returns the specified integer value processing parameter.

5.9.4.19 void linearizeSpectralData (DataHandle *SpectraIn,* DataHandle *SpectraOut,* DataHandle *Chirp*)

Linearizes the spectral data using the given chirp vector.

5.9.4.20 void loadCalibration (ProcessingHandle *Proc,* CalibrationData *Selection,* const char *Path[]*)

Will load a specified calibration file and use for subsequent processing.

5.9.4.21 void measureCalibration (OCTDeviceHandle *Dev,* ProcessingHandle *Proc,* CalibrationData *Selection*)

Measures the specified calibration parameters and uses them in subsequent processing.

5.9.4.22 void 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.

5.9.4.23 void measureSpectrum (OCTDeviceHandle *Dev,* ProbeHandle *Probe,* ProcessingHandle *Proc,* BOOL *moveToApoPos*)

Measures apodization spectrum and uses them in subsequent processing.

5.9.4.24 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.

5.9.4.25 void saveCalibrationAuto (ProcessingHandle *Proc,* CalibrationData *Selection*)

Saves the selected calibration in its default path.

Warning

This will override your default calibration of the device.

5.9.4.26 void setApodizationWindow (ProcessingHandle *Proc,* ApodizationWindow *Window*)

Sets the window function that is to be used for apodization. The selected function will be used in all subsequent processings.

If this function is not explicitly called a Hann window will be used.

Parameters

Handle	Processing handle.
Window	The apodization window that is used for data processing.

5.9.4.27 void setApodizationWindowParameter (ProcessingHandle *Proc,* ApodizationWindowParameter *Selection,* double *Value*)

Sets the apodization window parameter, such as window width or ratio between constant and cosine part.

5.9.4.28 void setApodizedSpectrumOutput (ProcessingHandle *Proc,* DataHandle *ApodizedSpectrum*)

Sets the location for the resulting apodized spectral data.

5.9.4.29 void setCalibration (ProcessingHandle *Proc,* CalibrationData *Selection,* DataHandle *Data*)

Sets the current active calibration data.

5.9.4.30 void setColoredDataOutput (ProcessingHandle *Proc,* ColoredDataHandle *BScan,* Coloring32BitHandle *Color*)

Sets the pointer the resulting colored B-Scan of the next processing is written to.

5.9.4.31 void setComplexDataOutput (ProcessingHandle *Proc,* ComplexDataHandle *ComplexBScan*)

Sets the pointer the resulting complex B-Scan of the next processing is written to.

If set to 0 no complex data result will be created in the next processing.

5.9.4.32 void setDCCorrectedSpectrumOutput (ProcessingHandle *Proc,* DataHandle *ProcessingCorrectedSpectrum*)

Sets the location for the resulting DC removed spectral data.

5.9.4.33 void setDechirpAlgorithm (ProcessingHandle *Proc,* ProcessingType *Type*)

Sets the algorithm that is to be sued for dechirping the input spectra.

5.9.4.34 void setDispersionPresetByIndex (ProcessingHandle *Proc,* int *Index*)

Sets the dispersion preset specified with index.

5.9.4.35 void setDispersionPresetByName (ProcessingHandle *Proc,* const char * *Name*)

Sets the dispersion preset specified with name.

5.9.4.36 void setDispersionPresets (ProcessingHandle *Proc,* ProbeHandle *Probe*)

Sets the dispersion presets for the probe.

5.9.4.37 void setHorMirroredDataOutput (ProcessingHandle *Proc,* DataHandle *Scan*)

Sets the pointer the resulting B-Scan of the next processing is written to. The result will be written mirrored at the horizontal axis.

If set to 0 no floating point processed data in dB will be created in the next processing.

5.9.4.38 void setOffsetCorrectedSpectrumOutput (ProcessingHandle *Proc,* DataHandle *OffsetCorrectedSpectrum*)

Sets the location for the resulting offset corrected spectral data.

5.9.4.39 void setProcessedDataOutput (ProcessingHandle *Proc,* DataHandle *Scan*)

Sets the pointer the resulting B-Scan of the next processing is written to.

If set to 0 no processed floating point data in dB will be created in the next processing.

5.9.4.40 void setProcessingAveragingAlgorithm (ProcessingHandle *Proc,* ProcessingAveragingAlgorithm *Algorithm*)

Sets the algorithm that is used for averaing by the processing.

5.9.4.41 void setProcessingFlag (ProcessingHandle *Proc,* ProcessingFlag *Flag,* BOOL *Value*)

Sets the specified processing flag.

5.9.4.42 setProcessingParameterInt (ProcessingHandle *Proc,* ProcessingParameterInt *Selection,* int *Value*)

Sets the specified integer value processing parameter.

5.9.4.43 void setSpectrumOutput (ProcessingHandle *Proc,* DataHandle *Spectrum*)

Sets the location for the resulting spectral data.

5.9.4.44 void setTransposedColoredDataOutput (ProcessingHandle *Proc,* ColoredDataHandle *BScan,* Coloring32BitHandle *Color*)

Sets the pointer the resulting colored B-Scan of the next processing is written to. The data will be transposed so that the first axis is the x-axis.

5.10 Export and Import

Export functionality to store data to disk and load it from there.

Enumerations

- enum Data1DExportFormat {
  Data1DExport_RAW,
  Data1DExport_TXT,
  Data1DExport_CSV,
  Data1DExport_TableTXT,
  Data1DExport_Fits }

  Export format for 1-dimensional data (DataHandle).
- enum Data2DExportFormat {
  Data2DExport_SRM,
  Data2DExport_RAW,
  Data2DExport_TXT,
  Data2DExport_CSV,
  Data2DExport_TableTXT,
  Data2DExport_Fits }

  Export format for 2-dimensional data (DataHandle).
- enum Data3DExportFormat {
  Data3DExport_SRM,
  Data3DExport_RAW,
  Data3DExport_TXT,
  Data3DExport_CSV,
  Data3DExport_VFF,
  Data3DExport_VTK,
  Data3DExport_Fits,
  Data3DExport_TIFF }

  Export format for 3-dimensional data (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 exportData1D (DataHandle Data, Data1DExportFormat Format, const char *Path)

Exports 1-dimensional data (DataHandle).

- SPECTRALRADAR_API void exportData2D (DataHandle Data, Data2DExportFormat Format, const char *Path)

  Exports 2-dimensional data (DataHandle).

- SPECTRALRADAR_API void exportData3D (DataHandle Volume, Data3DExportFormat Format, const char *Path)

  Exports 3-dimensional data (DataHandle).

- SPECTRALRADAR_API void exportComplexData (ComplexDataHandle, ComplexDataExportFormat, const char *)

  Exports 1-, 2- and 3-dimensional complex data (ComplexDataHandle)

- SPECTRALRADAR_API void exportColoredData (ColoredDataHandle Image, ColoredDataExportFormat Format, const char *fileName)

  Exports colored data (ColoredDataHandle)

- SPECTRALRADAR_API void importColoredData (ColoredDataHandle ColoredData, DataImportFormat Format, const char *Path)

  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 *Path)

  Imports data with the specified format and copies it into a dat data object (DataHandle).

- SPECTRALRADAR_API void exportRawData (RawDataHandle Raw, RawDataExportFormat Format, const char *Path)

  Exports the specified data to disk.

- SPECTRALRADAR_API void importRawData (RawDataHandle Raw, RawDataImportFormat Format, const char *Path)

  Imports the specified data from disk.

5.10.1 Detailed Description

Export functionality to store data to disk and load it from there.

5.10.2 Enumeration Type Documentation

5.10.2.1 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 (ColoredDataHandle). 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.

5.10.2.2 enum ComplexDataExportFormat

Export format for complex data.

Enumerator

ComplexDataExport_RAW RAW data format containg binary data.

5.10.2.3 enum Data1DExportFormat

Export format for 1-dimensional data (DataHandle).

Enumerator

Data1DExport_RAW RAW data format containing the data of the object as binary, single precision floating point values, little endian.

Data1DExport_TXT TXT is a text file having all values stored space seperated and human readable.

Data1DExport_CSV CSV (Comma Seperated Values) is a text file having all values stored, comma seperated and human readable.

Data1DExport_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.

Data1DExport_Fits FITS Data format.

5.10.2.4 enum Data2DExportFormat

Export format for 2-dimensional data (DataHandle).

Enumerator

Data2DExport_SRM Spectral Radar Metaformat, containing no data but all additinal parameters, such as spacing, size, etc.

Data2DExport_RAW RAW data format containing the data of the object as binary, single precision floating point values, little endian.

Data2DExport_TXT TXT is a text file having all values stored space seperated and human readable.

Data2DExport_CSV CSV (Comma Seperated Values) is a text file having all values stored, comma seperated and human readable.

Data2DExport_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.

Data2DExport_Fits FITS Data format.

5.10.2.5 enum Data3DExportFormat

Export format for 3-dimensional data (DataHandle).

Enumerator

Data3DExport_SRM Spectral Radar Metaformat, containing no data but all additinal parameters, such as spacing, size, etc.

Data3DExport_RAW RAW data format containing the data of the object as binary, single precision floating point values, little endian.

Data3DExport_TXT TXT is a text file having all values stored space seperated and human readable.

Data3DExport_CSV CSV (Comma Separated Values) is a text file having all values stored, comma seperated and human readable.

Data3DExport_VFF VFF data format.

Data3DExport_VTK VTK data format.

Data3DExport_Fits FITS Data format.

Data3DExport_TIFF TIFF Data format.

5.10.2.6 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.

5.10.2.7 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.

5.10.2.8 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.

5.10.3 Function Documentation

5.10.3.1 void exportColoredData (ColoredDataHandle *Image,* ColoredDataExportFormat *Format,* const char * *fileName*)

Exports colored data (ColoredDataHandle)

5.10.3.2 void exportComplexData (ComplexDataHandle , ComplexDataExportFormat , const char *)

Exports 1-, 2- and 3-dimensional complex data (ComplexDataHandle)

5.10.3.3 void exportData1D (DataHandle *Data,* Data1DExportFormat *Format,* const char * *Path*)

Exports 1-dimensional data (DataHandle).

5.10.3.4 void exportData2D (DataHandle *Data,* Data2DExportFormat *Format,* const char * *Path*)

Exports 2-dimensional data (DataHandle).

5.10.3.5 void exportData3D (DataHandle *Volume,* Data3DExportFormat *Format,* const char * *Path*)

Exports 3-dimensional data (DataHandle).

5.10.3.6 void exportRawData (RawDataHandle *Raw,* RawDataExportFormat *Format,* const char * *Path*)

Exports the specified data to disk.

5.10.3.7 void importColoredData (ColoredDataHandle *ColoredData,* DataImportFormat *Format,* const char * *Path*)

Imports colored data (ColoredDataHandle) with the specified format and copied it into a data object (ColoredDataHandle)

5.10.3.8 void importData (DataHandle *Data,* DataImportFormat *Format,* const char * *Path*)

Imports data with the specified format and copies it into a dat data object (DataHandle).

5.10.3.9 void importRawData (RawDataHandle *Raw,* RawDataImportFormat *Format,* const char * *Path*)

Imports the specified data from disk.

5.11 Volume

Functionality to store and access volume data.

Enumerations

- enum Direction {
  Direction_1,
  Direction_2,
  Direction_3 }

  Specifies a direction.
- enum Plane2D {
  Plane2D_12,
  Plane2D_23,
  Plane2D_13 }

  Planes for slices of the volume data.

Functions

- SPECTRALRADAR_API void appendRawData (RawDataHandle Data, RawDataHandle DataToAppend, Direction direction)

  Appends the new raw data to the old raw data in the specified direction.
- SPECTRALRADAR_API void getRawDataSliceIndex (RawDataHandle Data, RawDataHandle Slice, Direction SliceNormalDirection, int Index)

  Returns a slice of raw data in the specified direction at the specified index.
- SPECTRALRADAR_API double analyzeData (DataHandle Data, DataAnalyzation Selection)

  Performs the selected analyzation of the specified data and returns the resulting value.
- SPECTRALRADAR_API double analyzeAScan (DataHandle Data, AScanAnalyzation Selection)

  Performs the selected analyzation of the specified A-scan and returns the resulting value.
- SPECTRALRADAR_API void determineDynamicRange (DataHandle Data, float *MinRange_dB, float *MaxRange_dB)

  Gives a rough estimation of the dynamic range of the specified data object.
- SPECTRALRADAR_API void transpose (DataHandle DataIn, DataHandle DataOut)

  Transposes the given data and writes the result to DataOut.
- SPECTRALRADAR_API void transposeInplace (DataHandle Data)

  Transposes the given Data.
- SPECTRALRADAR_API void transposeAndScaleData (DataHandle DataIn, DataHandle DataOut, float Min, float Max)

  Transposes the given data and scales it to the range [Min, Max].
- SPECTRALRADAR_API void normalizeData (DataHandle Data, float Min, float Max)

  Scales the given data to the range [Min, Max].
- SPECTRALRADAR_API void lockData (DataHandle Data)

  Locks the given data.
- SPECTRALRADAR_API void unlockData (DataHandle Data)

  Unlocks the given data.
- SPECTRALRADAR_API void getDataSlicePos (DataHandle Data, DataHandle Slice, Direction SliceNormalDirection, double Pos)

  Returns a slice of data in the specified direction at the specified position.
- SPECTRALRADAR_API void getDataSliceIndex (DataHandle Data, DataHandle Slice, Direction SliceNormalDirection, int Index)

  Returns a slice of data in the specified direction at the specified index.
- SPECTRALRADAR_API void getDataSliceAnalyzed (DataHandle Data, DataHandle Slice, Direction SliceNormalDirection, DataAnalyzation Selection)

Returns a slice of data that has been computed of all slice using the specified analyzation method.

- SPECTRALRADAR_API void appendData (DataHandle Data, DataHandle NewData, Direction direction)

  Appends the new data to the old data in the specified direction.
- SPECTRALRADAR_API void cropData (DataHandle Data, Direction direction, int Index)

  Crops the data at the specific direction at the given index. The result will contain the data with range [0, index] at the cropping dirction.
- SPECTRALRADAR_API void cropDataEx (DataHandle Data, Direction direction, int IndexMax, int IndexMin)

  Crops the data at the specific direction at the given indeces. The result will contain the data with range [IndexMin, IndexMax] at the cropping dirction.
- 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 be in Data1, the second separated in Data2.
- SPECTRALRADAR_API void flipData (DataHandle Data, Direction FlippingDirection)

  Flips the data around the specific direction.
- SPECTRALRADAR_API void getComplexDataSlicePos (ComplexDataHandle Data, ComplexDataHandle Slice, Direction SliceNormalDirection, double Pos)

  Returns a slice of data in the specified direction at the specified position.
- SPECTRALRADAR_API void getComplexDataSliceIndex (ComplexDataHandle Data, ComplexDataHandle Slice, Direction SliceNormalDirection, int Index)

  Returns a slice of data in the specified direction at the specified index.
- SPECTRALRADAR_API void appendComplexData (ComplexDataHandle Data, ComplexDataHandle DataToAppend, Direction direction)

  Appends the new data to the old data in the specified direction.
- SPECTRALRADAR_API void cropComplexData (ComplexDataHandle Data, Direction CroppingDirection, int IndexMax, int IndexMin)

  Crops the complex data at the specific direction at the given indeces. The result will contain the data with range [IndexMin, IndexMax] at the cropping dirction.
- SPECTRALRADAR_API void cropColoredData (ColoredDataHandle Data, Direction CroppingDirection, int IndexMax, int IndexMin)

  Crops the colored data at the specific direction at the given indeces. The result will contain the data with range [IndexMin, IndexMax] at the cropping dirction.
- SPECTRALRADAR_API void appendColoredData (ColoredDataHandle Data, ColoredDataHandle DataToAppend, Direction AppendingDirection)

  Appends the new colored data to the old colored data in the specified direction.
- SPECTRALRADAR_API void getColoredDataSlicePos (ColoredDataHandle Data, ColoredDataHandle Slice, Direction SliceNormalDirection, double Pos)

  Get a slice of the colored data with specific slicing direction at given index position.
- SPECTRALRADAR_API void getColoredDataSliceIndex (ColoredDataHandle Data, ColoredDataHandle Slice, Direction SliceNormalDirection, int Index)

  Get a slice of the colored data with specific slicing direction at given index.
- SPECTRALRADAR_API ImageFieldHandle createImageField (void)

  Creates an object holding image field data.
- 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, DataHandle Surface)

  Determines the image field correction of the surface.
- SPECTRALRADAR_API void determineImageFieldForProbe (ProbeHandle Probe, DataHandle Surface)

  Determines the image field correction of the surface for the specified probe handle.

- SPECTRALRADAR_API void determineImageFieldForProbeWithMap (ProbeHandle Probe, DataHandle Surface, DataHandle Map)

  Determines the image field correction of the surface for the specified probe handle using the given map. Values != 0 in the map specifies to use the data in the surface handle otherwise thex will be interpolated.
- SPECTRALRADAR_API void correctImageField (ImageFieldHandle ImageField, ScanPatternHandle Pattern, DataHandle Data)

  Applies the image field correction to the B-Scan or volume data .
- SPECTRALRADAR_API void correctSurface (ImageFieldHandle ImageField, DataHandle Surface)

  Applies the image field correction to the given Surface.
- SPECTRALRADAR_API void correctImageFieldFromProbe (ProbeHandle Probe, ScanPatternHandle Pattern, DataHandle Data)

  Applies the image field correction saved in the probe handle to the B-Scan or volume data .

5.11.1 Detailed Description

Functionality to store and access volume data.

5.11.2 Enumeration Type Documentation

5.11.2.1 enum Direction

Specifies a direction.

Enumerator

Direction_1 The 1-axis direction.

Direction_2 The 2-axis direction.

Direction_3 The 3-axis direction.

5.11.2.2 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.

5.11.3 Function Documentation

5.11.3.1 double analyzeAScan (DataHandle *Data,* AScanAnalyzation *Selection*)

Performs the selected analyzation of the specified A-scan and returns the resulting value.

5.11.3.2 double analyzeData (DataHandle *Data,* DataAnalyzation *Selection*)

Performs the selected analyzation of the specified data and returns the resulting value.

5.11.3.3 void appendColoredData (ColoredDataHandle *Data,* ColoredDataHandle *DataToAppend,* Direction *AppendingDirection*)

Appends the new colored data to the old colored data in the specified direction.

5.11.3.4 void appendComplexData (ComplexDataHandle *Data,* ComplexDataHandle *DataToAppend,* Direction *direction*)

Appends the new data to the old data in the specified direction.

5.11.3.5 void appendData (DataHandle *Data,* DataHandle *NewData,* Direction *direction*)

Appends the new data to the old data in the specified direction.

5.11.3.6 void appendRawData (RawDataHandle *Data,* RawDataHandle *DataToAppend,* Direction *direction*)

Appends the new raw data to the old raw data in the specified direction.

5.11.3.7 void clearImageField (ImageFieldHandle *ImageField*)

Frees an object holding image field data.

5.11.3.8 void correctImageField (ImageFieldHandle *ImageField,* ScanPatternHandle *Pattern,* DataHandle *Data*)

Applies the image field correction to the B-Scan or volume data .

5.11.3.9 void correctImageFieldFromProbe (ProbeHandle *Probe,* ScanPatternHandle *Pattern,* DataHandle *Data*)

Applies the image field correction saved in the probe handle to the B-Scan or volume data .

5.11.3.10 void correctSurface (ImageFieldHandle *ImageField,* DataHandle *Surface*)

Applies the image field correction to the given Surface.

5.11.3.11 ImageFieldHandle createImageField (void)

Creates an object holding image field data.

5.11.3.12 void cropColoredData (ColoredDataHandle *Data,* Direction *CroppingDirection,* int *IndexMax,* int *IndexMin*)

Crops the colored data at the specific direction at the given indeces. The result will contain the data with range [IndexMin, IndexMax] at the cropping dirction.

5.11.3.13 void cropComplexData (ComplexDataHandle *Data,* Direction *CroppingDirection,* int *IndexMax,* int *IndexMin*)

Crops the complex data at the specific direction at the given indeces. The result will contain the data with range [IndexMin, IndexMax] at the cropping dirction.

5.11.3.14 void cropData (DataHandle *Data,* Direction *direction,* int *Index*)

Crops the data at the specific direction at the given index. The result will contain the data with range [0, index] at the cropping dirction.

5.11.3.15 void cropDataEx (DataHandle *Data,* Direction *direction,* int *IndexMax,* int *IndexMin*)

Crops the data at the specific direction at the given indeces. The result will contain the data with range [IndexMin, IndexMax] at the cropping dirction.

5.11.3.16 void determineDynamicRange (DataHandle *Data,* float * *MinRange_dB,* float * *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.

5.11.3.17 void determineImageField (ImageFieldHandle *ImageField,* DataHandle *Surface*)

Determines the image field correction of the surface.

5.11.3.18 void determineImageFieldForProbe (ProbeHandle *Probe,* DataHandle *Surface*)

Determines the image field correction of the surface for the specified probe handle.

5.11.3.19 void determineImageFieldForProbeWithMap (ProbeHandle *Probe,* DataHandle *Surface,* DataHandle *Map*)

Determines the image field correction of the surface for the specified probe handle using the given map. Values != 0 in the map specifies to use the data in the surface handle otherwise thex will be interpolated.

5.11.3.20 void void flipData (DataHandle *Data,* Direction *FlippingDirection*)

Flips the data around the specific direction.

5.11.3.21 void getColoredDataSliceIndex (ColoredDataHandle *Data,* ColoredDataHandle *Slice,* Direction *SliceNormalDirection,* int *Index*)

Get a slice of the colored data with specific slicing direction at given index.

5.11.3.22 void getColoredDataSlicePos (ColoredDataHandle *Data,* ColoredDataHandle *Slice,* Direction *SliceNormalDirection,* double *Pos*)

Get a slice of the colored data with specific slicing direction at given index position.

5.11.3.23 void getComplexDataSliceIndex (ComplexDataHandle *Data,* ComplexDataHandle *Slice,* Direction *SliceNormalDirection,* int *Index*)

Returns a slice of data in the specified direction at the specified index.

5.11.3.24 void getComplexDataSlicePos (ComplexDataHandle *Data,* ComplexDataHandle *Slice,* Direction *SliceNormalDirection,* double *Pos*)

Returns a slice of data in the specified direction at the specified position.

5.11.3.25 void getDataSliceAnalyzed (DataHandle *Data,* DataHandle *Slice,* Direction *SliceNormalDirection,* DataAnalyzation *Selection*)

Returns a slice of data that has been computed of all slice using the specified analyzation method.

5.11.3.26 void getDataSliceIndex (DataHandle *Data,* DataHandle *Slice,* Direction *SliceNormalDirection,* int *Index*)

Returns a slice of data in the specified direction at the specified index.

5.11.3.27 void getDataSlicePos (DataHandle *Data,* DataHandle *Slice,* Direction *SliceNormalDirection,* double *Pos*)

Returns a slice of data in the specified direction at the specified position.

5.11.3.28 void getRawDataSliceIndex (RawDataHandle *Data,* RawDataHandle *Slice,* Direction *SliceNormalDirection,* int *Index*)

Returns a slice of raw data in the specified direction at the specified index.

5.11.3.29 void loadImageField (ImageFieldHandle *ImageField,* const char * *Path*)

Loads data containing image field data.

5.11.3.30 void lockData (DataHandle *Data*)

Locks the given data.

5.11.3.31 void normalizeData (DataHandle *Data,* float *Min,* float *Max*)

Scales the given data to the range [Min, Max].

5.11.3.32 void saveImageField (ImageFieldHandle *ImageField,* const char * *Path*)

Saves data containing image field data.

5.11.3.33 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 be in Data1, the second separated in Data2.

5.11.3.34 void void transpose (DataHandle *DataIn,* DataHandle *DataOut*)

Transposes the given data and writes the result to DataOut.

5.11.3.35 void transposeAndScaleData (DataHandle *DataIn,* DataHandle *DataOut,* float *Min,* float *Max*)

Transposes the given data and scales it to the range [Min, Max].

5.11.3.36 void transposeInplace (DataHandle *Data*)

Transposes the given Data.

5.11.3.37 void unlockData (DataHandle *Data*)

Unlocks the given data.

5.12 ProbeCalibration

Functionality to perform the probe calibration. Please use the ThorImageOCT software to perform the probe calibration if necessary.

Functionality to perform the probe calibration. Please use the ThorImageOCT software to perform the probe calibration if necessary.

5.13 Doppler

Doppler Processing Routines.

Typedefs

- typedef struct C_DopplerProcessing * DopplerProcessingHandle

  Handle used for Doppler processing.

Enumerations

- enum DopplerPropertyInt { DopplerAveraging_1, DopplerAveraging_2, DopplerStride_1, DopplerStride_2 }

  Values that determine the behaviour of the Doppler processing routines.
- enum DopplerPropertyFloat { DopplerRefractiveIndex }

  Values that determine the behaviour of the Doppler processing routines.
- enum DopplerFlag { DopplerVelocityScaling }

  Flats that determine the behaviour of the Doppler processing routines.

Functions

- SPECTRALRADAR_API DopplerProcessingHandle createDopplerProcessing (void)
- SPECTRALRADAR_API void createDopplerProcessingForProcessing (DopplerProcessingHandle *Doppler, ProcessingHandle Proc)
- SPECTRALRADAR_API void setDopplerPropertyInt (DopplerProcessingHandle Doppler, DopplerPropertyInt Property, int Value)

  Sets Doppler processing properties.
- SPECTRALRADAR_API void setDopplerPropertyFloat (DopplerProcessingHandle Doppler, DopplerPropertyFloat Property, float Value)

  Sets Doppler processing properties.
- SPECTRALRADAR_API void setDopplerFlag (DopplerProcessingHandle Doppler, DopplerFlag Flag, BOOL OnOff)

  Sets the Doppler processing flags.
- SPECTRALRADAR_API void setDopplerAmplitudeOutput (DopplerProcessingHandle Doppler, DataHandle AmpOut)

  Sets the location of the resulting doppler amplitude output.
- SPECTRALRADAR_API void setDopplerPhaseOutput (DopplerProcessingHandle Doppler, DataHandle PhasesOut)

  Sets the location of the resulting doppler phase output.
- SPECTRALRADAR_API void executeDopplerProcessing (DopplerProcessingHandle Doppler, ComplexDataHandle Input)

  Executes the Doppler processing of the input data and returns phases and amplitudes.
- SPECTRALRADAR_API void closeDopplerProcessing (DopplerProcessingHandle Doppler)

  Closes the Doppler processing routines and frees the memory that has been allocated for these to work properly.

5.13.1 Detailed Description

Doppler Processing Routines.

5.13.2 Typedef Documentation

5.13.2.1 DopplerProcessingHandle

Handle used for Doppler processing.

5.13.3 Enumeration Type Documentation

5.13.3.1 enum DopplerFlag

Flats that determine the behaviour of the Doppler processing routines.

Enumerator

DopplerVelocityScaling Averaging along the first axis, usually the longitudinal axis (z)

5.13.3.2 enum DopplerPropertyFloat

Values that determine the behaviour of the Doppler processing routines.

Enumerator

DopplerRefractiveIndex Averaging along the first axis, usually the longitudinal axis (z)

5.13.3.3 enum DopplerPropertyInt

Values that determine the behaviour of the Doppler processing routines.

Enumerator

DopplerAveraging_1 Averaging along the first axis, usually the longitudinal axis (z)

DopplerAveraging_2 Averaging along the first axis, usually the first transversal axis (x)

DopplerStride_1 Step size for calculating the doppler processing in the longitudinal axis (z). Stride needs to be smaller or equal to DopplerAveraging_1 and larger or equal to 1.

DopplerStride_2 Step size for calculating the doppler processing in the transversal axis (x). Stride needs to be smaller or equal to DopplerAveraging_2 and larger or equal to 1.

5.13.4 Function Documentation

5.13.4.1 void closeDopplerProcessing (DopplerProcessingHandle *Doppler*)

Closes the Doppler processing routines and frees the memory that has been allocated for these to work properly.

5.13.4.2 DopplerProcessingHandle createDopplerProcessing (void)

Returns

DopplerProcessingHandle to the created Doppler routines.

5.13.4.3 void createDopplerProcessingForProcessing (DopplerProcessingHandle * *Doppler,* ProcessingHandle *Proc*)

Parameters

Doppler	Handle to the Doppler processing.
Proc	Handle to the Processing routines whose results are used as input for Doppler routines.

5.13.4.4 void executeDopplerProcessing (DopplerProcessingHandle *Doppler,* ComplexDataHandle *Input*)

Executes the Doppler processing of the input data and returns phases and amplitudes.

5.13.4.5 void setDopplerAmplitudeOutput (DopplerProcessingHandle *Doppler,* DataHandle *AmpOut*)

Sets the location of the resulting doppler amplitude output.

5.13.4.6 void setDopplerFlag (DopplerProcessingHandle *Doppler,* DopplerFlag *Flag,* BOOL *OnOff*)

Sets the Doppler processing flags.

5.13.4.7 void setDopplerPhaseOutput (DopplerProcessingHandle *Doppler,* DataHandle *PhasesOut*)

Sets the location of the resulting doppler phase output.

5.13.4.8 void setDopplerPropertyFloat (DopplerProcessingHandle *Doppler,* DopplerPropertyFloat *Property,* float *Value*)

Sets Doppler processing properties.

5.13.4.9 void setDopplerPropertyInt (DopplerProcessingHandle *Doppler,* DopplerPropertyInt *Property,* int *Value*)

Sets Doppler processing properties.

5.14 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.

5.14.1 Detailed Description

Service functions for additional analyzing of OCT functionality.

5.14.2 Function Documentation

5.14.2.1 void calcContrast (DataHandle *ApodizedSpectrum,* DataHandle *Contrast*)

Computes the contrast for the specified (apodized); spectrum.

5.15 Settings

Direct access to INI files and settings.

Typedefs

- typedef struct C_Settings * SettingsHandle

  Handle for saving settings on disk.

Functions

- SPECTRALRADAR_API SettingsHandle loadSettingsFile (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 loadSettingsFile);.
- SPECTRALRADAR_API double getSettingsEntryDouble (SettingsHandle SettingsFile, const char *Node, double DefaultValue)

  Gets an floating point number from the specified ini file (see SettingsHandle and loadSettingsFile);.
- SPECTRALRADAR_API void getSettingsEntryString (SettingsHandle SettingsFile, const char *Node, const char *Default, char *Data, int MaxDataSize)

  Gets a string from the specified ini file (see SettingsHandle and loadSettingsFile);.
- SPECTRALRADAR_API void setSettingsEntryInt (SettingsHandle SettingsFile, const char *Node, int Value)

  Sets an integer entry in the specified ini file (see SettingsHandle and loadSettingsFile);.
- SPECTRALRADAR_API void setSettingsEntryFloat (SettingsHandle SettingsFile, const char *Node, double Value)

  Sets a floating point entry in the specified ini file (see SettingsHandle and loadSettingsFile);.
- SPECTRALRADAR_API void setSettingsEntryString (SettingsHandle SettingsFile, const char *Node, const char *Value)

  Sets a string in the specified ini file (see SettingsHandle and loadSettingsFile);.
- SPECTRALRADAR_API void saveSettings (SettingsHandle SettingsFile)

  Saves the changes to the specified Settings file.
- SPECTRALRADAR_API void closeSettingsFile (SettingsHandle SettingsFile)

  Closes the specified ini file and stores the set entries (see SettingsHandle and loadSettingsFile);.

5.15.1 Detailed Description

Direct access to INI files and settings.

5.15.2 Typedef Documentation

5.15.2.1 SettingsHandle

Handle for saving settings on disk.

5.15.3 Function Documentation

5.15.3.1 void closeSettingsFile (SettingsHandle *SettingsFile*)

Closes the specified ini file and stores the set entries (see SettingsHandle and loadSettingsFile);.

5.15.3.2 double getSettingsEntryDouble (SettingsHandle *SettingsFile,* const char * *Node,* double *DefaultValue*)

Gets an floating point number from the specified ini file (see SettingsHandle and loadSettingsFile);.

5.15.3.3 int getSettingsEntryInt (SettingsHandle *SettingsFile,* const char * *Node,* int *DefaultValue*)

Gets an integer number from the specified ini file (see SettingsHandle and loadSettingsFile);.

5.15.3.4 void getSettingsEntryString (SettingsHandle *SettingsFile,* const char * *Node,* const char * *Default,* char * *Data,* int *MaxDataSize*)

Gets a string from the specified ini file (see SettingsHandle and loadSettingsFile);.

5.15.3.5 SettingsHandle loadSettingsFile (const char * *Path*)

Loads a settings file (usually *.ini); and prepares its properties to be read.

5.15.3.6 void saveSettings (SettingsHandle *SettingsFile*)

Saves the changes to the specified Settings file.

5.15.3.7 void setSettingsEntryFloat (SettingsHandle *SettingsFile,* const char * *Node,* double *Value*)

Sets a floating point entry in the specified ini file (see SettingsHandle and loadSettingsFile);.

5.15.3.8 void setSettingsEntryInt (SettingsHandle *SettingsFile,* const char * *Node,* int *Value*)

Sets an integer entry in the specified ini file (see SettingsHandle and loadSettingsFile);.

5.15.3.9 void setSettingsEntryString (SettingsHandle *SettingsFile,* const char * *Node,* const char * *Value*)

Sets a string in the specified ini file (see SettingsHandle and loadSettingsFile);.

5.16 Coloring

Functions used for coloring of floating point data.

Typedefs

- typedef struct C_Coloring32Bit * Coloring32BitHandle

  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 }

  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
  Coloring32BitHandle 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
  Coloring32BitHandle createCustomColoring32Bit (int LUTSize, unsigned long LUT[])

  Create custom coloring using the specified color look-up-table.
- SPECTRALRADAR_API void setColoringBoundaries (Coloring32BitHandle 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 (Coloring32BitHandle Coloring, ColorEnhancement Enhancement)

  Selects a function for non-linear coloring to enhance (subjective) image impression.
- SPECTRALRADAR_API void colorizeData (Coloring32BitHandle Coloring, DataHandle Data, ColoredDataHandle ColoredData, BOOL Transpose)

  Colors a given data object (DataHandle) into a given colored object (ColoredDataHandle).

- SPECTRALRADAR_API void colorizeDopplerData (Coloring32BitHandle AmpColoring, Coloring32BitHandle 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.
- SPECTRALRADAR_API void clearColoring32Bit (Coloring32BitHandle Coloring)

  Clears the coloring previously created by createColoring32Bit().

5.16.1 Detailed Description

Functions used for coloring of floating point data.

5.16.2 Typedef Documentation

5.16.2.1 Coloring32BitHandle

Handle for routines that color avaible scans for displaying.

5.16.3 Enumeration Type Documentation

5.16.3.1 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.

5.16.3.2 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.

5.16.3.3 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

5.16.4 Function Documentation

5.16.4.1 void clearColoring32Bit (Coloring32BitHandle *Coloring*)

Clears the coloring previously created by createColoring32Bit().

5.16.4.2 void colorizeData (Coloring32BitHandle *Coloring,* DataHandle *Data,* ColoredDataHandle *ColoredData,* BOOL *Transpose*)

Colors a given data object (DataHandle) into a given colored object (ColoredDataHandle).

5.16.4.3 oid colorizeDopplerData (Coloring32BitHandle *AmpColoring,* Coloring32BitHandle *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.

5.16.4.4 Coloring32BitHandle 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 (Coloring32BitHandle) to the coloring algorithm.

5.16.4.5 Coloring32BitHandle createCustomColoring32Bit (int *LUTSize,* unsigned long *LUT[]*)

Create custom coloring using the specified color look-up-table.

5.16.4.6 void setColoringBoundaries (Coloring32BitHandle *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.

5.16.4.7 void setColoringEnhancement (Coloring32BitHandle *Coloring,* ColorEnhancement *Enhancement*)

Selects a function for non-linear coloring to enhance (subjective) image impression.

5.17 Camera

Functions for acquiring camera video images.

Enumerations

- enum CameraPropertyFloat {
  Camera_Saturation,
  Camera_Brightness,
  Camera_Contrast,
  Camera_WB_Red,
  Camera_WB_Green,
  Camera_WB_Blue,
  Camera_WB_Auto }

  Enum identifying properties of the camera.

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, int SizeX, int SizeY, ColoredDataHandle Image)

  Gets a camera image.
- SPECTRALRADAR_API void getMirroredCameraImage (OCTDeviceHandle Dev, int SizeX, int SizeY, ColoredDataHandle Image)

  Gets a camera image.
- SPECTRALRADAR_API void setCameraPropertyFloat (OCTDeviceHandle Dev, CameraPropertyFloat Selection, double Value)

  Sets saturation, brightness and contrast for the camera images if this option is available for the current device.
- SPECTRALRADAR_API void setCameraShowScanPattern (OCTDeviceHandle Dev, BOOL Value)

  Enables to turn on/off the scan pattern overlay in the video camera image.
- SPECTRALRADAR_API void visualizeScanPattern (OCTDeviceHandle Dev, ProbeHandle Probe, ScanPatternHandle Pattern, BOOL showRawPattern)

  Visualizes the scan pattern in top of the camera image.

5.17.1 Detailed Description

Functions for acquiring camera video images.

5.17.2 Enumeration Type Documentation

5.17.2.1 enum CameraPropertyFloat

Enum identifying properties of the camera.

Enumerator

Camera_Saturation Saturation of the video camera.

Camera_Brightness Brightness of the video camera.

Camera_Contrast Contrast of the video camera.

Camera_WB_Red Red white-balance value.

Camera_WB_Green Red white-balance value.

Camera_WB_Blue Red white-balance value.

Camera_WB_Auto Automatic setting of white balance values.

5.17.3 Function Documentation

5.17.3.1 void getCameraImage (OCTDeviceHandle *Dev,* int *SizeX,* int *SizeY,* ColoredDataHandle *Image*)

Gets a camera image.

5.17.3.2 void getMaxCameraImageSize (OCTDeviceHandle *Dev,* int * *SizeX,* int * *SizeY*)

Returns the maximum possible camera image size for the current device.

5.17.3.3 void getMirroredCameraImage (OCTDeviceHandle *Dev,* int *SizeX,* int *SizeY,* ColoredDataHandle *Image*)

Gets a camera image.

The returned camera image is mirrored in the X axis.

5.17.3.4 void setCameraPropertyFloat (OCTDeviceHandle *Dev,* CameraPropertyFloat *Selection,* double *Value*)

Sets saturation, brightness and contrast for the camera images if this option is available for the current device.

5.17.3.5 void setCameraShowScanPattern (OCTDeviceHandle *Dev,* BOOL *Value*)

Enables to turn on/off the scan pattern overlay in the video camera image.

5.17.3.6 void void visualizeScanPattern (OCTDeviceHandle *Dev,* ProbeHandle *Probe,* ScanPatternHandle *Pattern,* BOOL *showRawPattern*)

Visualizes the scan pattern in top of the camera image.

5.18 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 int getBufferSize (BufferHandle)

  Returns the currently avaiable data sets 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)

  Clears the buffer and frees all data and colored data objects in it.

5.18.1 Detailed Description

Functions for acquiring camera video images.

5.18.2 Typedef Documentation

5.18.2.1 BufferHandle

The BufferHandle identifies a data buffer.

5.18.3 Function Documentation

5.18.3.1 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.

5.18.3.2 void clearBuffer (BufferHandle)

Clears the buffer and frees all data and colored data objects in it.

5.18.3.3 BufferHandle createMemoryBuffer (void)

Creates a buffer holding data and colored data.

5.18.3.4 DataHandle getBufferData (BufferHandle , int *Index*)

Returns the data in the buffer.

5.18.3.5 int getBufferSize (BufferHandle)

Returns the currently avaiable data sets in the buffer.

5.18.3.6 ColoredDataHandle getColoredBufferData (BufferHandle , int *Index*)

Returns the colored data in the buffer.

5.19 File Handling

Enumerations

- enum OCTFileFormat {
  FF_Unknown,
  FF_SRM,
  FF_RAW,
  FF_SDR,
  FF_PHS,
  FF_IMG,
  FF_CSV,
  FF_TXT,
  FF_TABLETXT,
  FF_OCITY,
  FF_FITS,
  FF_VTK,
  FF_VFF,
  FF_TIFF,
  FF_JPG,
  FF_BMP,
  FF_PNG }

  Enum identifying possible file formats.
- enum DataKind {
  dkReal,
  dkColored,
  dkComplex,
  dkRaw,
  dkBinary,
  dkText,
  dkUnknown = 999 }

  Enum identifying.
- enum FileMetadataFloatField {
  FMD_RefractiveIndex,
  FMD_RangeX,
  FMD_RangeY,
  FMD_RangeZ,
  FMD_CenterX,
  FMD_CenterY,
  FMD_Angle,
  FMD_BinToElectronScaling,
  FMD_CentralWavelength,
  FMD_SourceBandwidth,
  FMD_ElectronFloor,
  FMD_DynamicRange_Lower,
  FMD_DynamicRange_Upper,
  FMD_Rotation3D_X,
  FMD_Rotation3D_Y,
  FMD_ClipPlaneDepth3D,
  FMD_QuadraticDispersionCorrectionFactor,
  FMD_SpeckleVarianceThreshold,
  FMD_ScanTime,
  FMD_ReferenceIntensity,
  FMD_ScanPause,
  FMD_Zoom,
  FMD_MinPointDistance,
  FMD_MaxPointDistance,
  FMD_MaxExternalTriggerFrequency }

Enum identifying file metadata fields of floating point type.

- enum FileMetadataIntField {
  FMD_ProcessState,
  FMD_SizeX,
  FMD_SizeY,
  FMD_SizeZ,
  FMD_Oversampling,
  FMD_IntensityAveragedSpectra,
  FMD_IntensityAveragedAScans,
  FMD_IntensityAveragedBScans,
  FMD_DopplerAverageX,
  FMD_DopplerAverageZ,
  FMD_ApoWindow,
  FMD_DeviceBitDepth,
  FMD_SpectrometerElements,
  FMD_Colormap,
  FMD_Aspect,
  FMD_ExperimentNumber,
  FMD_DevicePreset,
  FMD_Timestamp,
  FMD_CompressionLevel,
  FMD_DeviceBytesPerPixel,
  FMD_SpeckleAveragingFastAxis,
  FMD_SpeckleAveragingSlowAxis }

  Enum identifying file metadata fields of integral type.

- enum FileMetadataStringField {
  FMD_AxisOrder,
  FMD_DeviceName,
  FMD_Serial,
  FMD_Comment,
  FMD_CustomInfo,
  FMD_AcquisitionMode,
  FMD_Study,
  FMD_DispersionPreset,
  FMD_ProbeName,
  FMD_DevicePresetDescription }

  Enum identifying file metadata fields of character string type.

- enum FileMetadataBoolField {
  FMD_OffsetApplied,
  FMD_DCSubracted,
  FMD_ApoApplied,
  FMD_DechirpApplied,
  FMD_UndersamplingFilterApplied,
  FMD_DispersionCompensationApplied,
  FMD_QuadraticDispersionCorrectionUsed,
  FMD_ImageFieldCorrectionApplied,
  FMD_ScanLineShown,
  FMD_AutoCorrCompensationUsed,
  FMD_BScanCrossCorrelation,
  FMD_DCSubractedAdvanced,
  FMD_OnlyWindowing,
  FMD_RawDataIsSigned,
  FMD_FreeformScanPatternIsActive,
  FMD_FreeformScanPatternCloseLoop,
  FMD_FreeformScanPatternSplineInterpolation,
  FMD_ExternalTriggerActive }

  Enum identifying file metadata fields of bool type.

5.19.1 Detailed Description

5.19.2 Enumeration Type Documentation

5.19.2.1 enum DataKind

Enum identifying.

5.19.2.2 enum FileMetadataBoolField

Enum identifying file metadata fields of bool type.

5.19.2.3 enum FileMetadataFloatField

Enum identifying file metadata fields of floating point type.

Enumerator

FMD_RefractiveIndex The refractive index applied to the whole image.

FMD_RangeX The FOV in longitudinal axis (z) in mm.

FMD_RangeY The FOV in axial direction (x) in mm.

FMD_RangeZ The FOV in axial direction (y) in mm.

FMD_CenterX The center of the scan pattern in axial direction (x) in mm.

FMD_CenterY The center of the scan pattern in axial direction (y) in mm.

FMD_Angle The angle betwenn the scanner and the video camera image.

FMD_BinToElectronScaling Ratio between the binary value from the camera to the count of electrons.

FMD_CentralWavelength Central wavelength of the device.

FMD_SourceBandwidth Bandwidth of the light source.

FMD_DynamicRange_Lower Lower border of the dynamic range.

FMD_DynamicRange_Upper Upper border of the dynamic range.

FMD_SpeckleVarianceThreshold Threshold for speckle variance mode.

FMD_ScanTime Time needed for data acqusition. The orocessing and saving time is not included.

FMD_ReferenceIntensity Value for the reference intensity.

FMD_Zoom Zooms the scan pattern.

FMD_MinPointDistance Minimum distance between two points of the scan pattern used for freeform scan patterns.

FMD_MaxPointDistance Maximum distance between two points of the scan pattern used for freeform scan patterns.

FMD_MaxExternalTriggerFrequency maximal external trigger frequency for the chosen preset

5.19.2.4 enum FileMetadataIntField

Enum identifying file metadata fields of integral type.

Enumerator

FMD_ProcessState Contains the specifif data format.

FMD_SizeX Number of pixels in x.

FMD_SizeY Number of pixels in y.

FMD_SizeZ Number of pixels in z.

FMD_Oversampling Oversampling parameter.

FMD_IntensityAveragedAScans A-scan averaging.

FMD_IntensityAveragedBScans B-scan averaging.

FMD_DopplerAverageX Averaging for doppler processing in x-direction.

FMD_DopplerAverageZ Averaging for doppler processing in z-direction.

FMD_ApoWindow Type of window used for apodization.

FMD_DeviceBitDepth Bits per pixel of the camera.

FMD_SpectrometerElements Number of elements of the spectrometer.

FMD_Colormap Actual colormpa.

FMD_Aspect Ratio between pixel and real size.

FMD_ExperimentNumber Serial number of the dataset.

FMD_DevicePreset Preset for scan speed.

FMD_Timestamp Timestamp of acquired dataset.

FMD_CompressionLevel Compression used to create the zipped container (.oct-file). No compression used right now.

FMD_DeviceBytesPerPixel Bytes per pixel of the camera.

FMD_SpeckleAveragingFastAxis Averaging parameter of the fast scan axis in speckle variance mode.

FMD_SpeckleAveragingSlowAxis Averaging parameter of the slow scan axis in speckle variance mode.

5.19.2.5 enum FileMetadataStringField

Enum identifying file metadata fields of character string type.

Enumerator

FMD_AxisOrder Order of the axis, e.g. ZXY.

FMD_DeviceName Name of the OCT device.

FMD_Serial Serial number of the OCT device.

FMD_Comment Comment of the OCT data file.

FMD_AcquisitionMode Acquisition mode of the OCT data file.

FMD_Study Study of the OCT data file.

FMD_DispersionPreset Dispersion Preset of the OCT data file.

FMD_ProbeName Name of the probe.

FMD_DevicePresetDescription Description of the scan speed, e.g. "Default (48kHz").

5.19.2.6 enum OCTFileFormat

Enum identifying possible file formats.

5.20 BETA_API

BETA_API.

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.

5.20.1 Detailed Description

BETA_API.

5.20.2 Enumeration Type Documentation

5.20.2.1 enum BoundaryCondition

Selects the boundary conditions for the interpolation.

5.20.2.2 enum InterpolationMethod

Selects the interpolation method.

Enumerator

Interpolation_Linear Linear interpolation.

Interpolation_Spline Cubic B-Spline interpolation.

5.21 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.

Functions

- SPECTRALRADAR_API void medianFilter (DataHandle Data, int Rank)

  Computes a median filter on the specified 2D data.
- SPECTRALRADAR_API void levelData (DataHandle Data)

  Levels the specified data and removes tilt.
- SPECTRALRADAR_API void pepperFilter (DataHandle Data, PepperFilterType Type, float Threshold)

  Removes pepper-noise (very low values, i. e. dark spots in the data). This enhances the visual (colored) representation of the data.

5.21.1 Detailed Description

Algorithms and functions used for post processing of floating point data.

5.21.2 Enumeration Type Documentation

5.21.2.1 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.

5.21.3 Function Documentation

5.21.3.1 void levelData (DataHandle *Data*)

Levels the specified data and removes tilt.

5.21.3.2 void medianFilter (DataHandle *Data,* int *Rank*)

Computes a median filter on the specified 2D data.

5.21.3.3 void pepperFilter (DataHandle *Data,* PepperFilterType *Type,* float *Threshold*)

Removes pepper-noise (very low values, i. e. dark spots in the data). This enhances the visual (colored) representation of the data.

The pepper filter compares all pixels to a mean of surrounding pixels. The surrouding 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.

5.22 Polarization

Polarization code.

Typedefs

- typedef struct C_PolarizationProcessing * PolarizationProcessingHandle
  Handle used for Polarization processing.

Enumerations

- enum PolarizationPropertyFloat { **PolarizationProcessing_SurfaceThreshold** }
  Values that determine the behaviour of the Polarization processing routines.
- enum PolarizationFlag { PolarizationVelocityScaling }
  Flags that determine the behaviour of the Polarization processing routines.

5.22.1 Detailed Description

Polarization code. Polarization Sensitive OCT Processing Routines.

5.22.2 Typedef Documentation

5.22.2.1 PolarizationProcessingHandle

Handle used for Polarization processing.

5.22.3 Enumeration Type Documentation

5.22.3.1 enum PolarizationFlag

Flags that determine the behaviour of the Polarization processing routines.

Enumerator

PolarizationVelocityScaling Averaging along the first axis, usually the longitudinal axis (z)

5.22.3.2 enum PolarizationPropertyFloat

Values that determine the behaviour of the Polarization processing routines.

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.

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 and Telesto 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 C_ScanPattern * ScanPatternHandle

  Handle for controlling the 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_PolarizationProcessing * PolarizationProcessingHandle

  Handle used for Polarization processing.
- typedef struct C_Coloring32Bit * Coloring32BitHandle

  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 struct C_SpeckleVariance * **SpeckleVarianceHandle**
- typedef struct C_FullRange * **FullRangeHandle**
- typedef void(__stdcall * **cbRefstageStatusChanged**)(RefstageStatus)
- typedef void(__stdcall * **cbRefstagePositionChanged**)(double)

Enumerations

- enum ErrorCode {
  NoError = 0x00000000,
  Error = 0xE0000000 }

  This enum is used to describe errors that occur when operating an OCT device.
- enum RawDataPropertyInt {
  RawData_Size1,
  RawData_Size2,
  RawData_Size3,
  RawData_NumberOfElements,
  RawData_SizeInBytes,
  RawData_BytesPerElement,
  RawData_LostFrames }

Specifies properties of RawData.

- enum DataPropertyInt {
  Data_Dimensions,
  Data_Size1,
  Data_Size2,
  Data_Size3,
  Data_NumberOfElements,
  Data_SizeInBytes,
  Data_BytesPerElement }

  Selects integer point data property.
- enum DataPropertyFloat {
  Data_Spacing1,
  Data_Spacing2,
  Data_Spacing3,
  Data_Range1,
  Data_Range2,
  Data_Range3 }

  Selects floating point data property.
- enum DataAnalyzation {
  Data_Min,
  Data_Mean,
  Data_Max,
  Data_MaxDepth }

  Selects data property to analyze.
- 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 }

  Selects an appropriate A-Scan analyzation.
- enum DataOrientation {
  DataOrientation_ZXY,
  DataOrientation_ZYX,
  DataOrientation_XZY,
  DataOrientation_XYZ,
  DataOrientation_YXZ,
  DataOrientation_YZX,
  DataOrientation_ZTX,
  DataOrientation_ZXT }

  Selects the orientation of the data.
- 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,

Boolean parameters describing the behaviour of the Probe.

- enum ProbeType {
  ProbeType_Standard,
  ProbeType_Handheld,
  ProbeType_Scientific }

  Determines the kind of probe types.

- enum AcquisitionType {
  Acquisition_AsyncContinuous,
  Acquisition_AsyncFinite,
  Acquisition_Sync }

  Determines the kind of acquisition process.

- enum ProcessingType {
  Processing_StandardFFT,
  Processing_StandardNDFT,
  Processing_iFFT1,
  Processing_iFFT2,
  Processing_iFFT3,
  Processing_iFFT4,
  Processing_NFFT1,
  Processing_NFFT2,
  Processing_NFFT3,
  Processing_NFFT4 }

  defindes the algorithm used for dechirping the input signal and Fourier transformation

- 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 }

  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 }

  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 }

  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 Data1DExportFormat {
  Data1DExport_RAW,
  Data1DExport_TXT,
  Data1DExport_CSV,
  Data1DExport_TableTXT,
  Data1DExport_Fits }

  Export format for 1-dimensional data (DataHandle).
- enum Data2DExportFormat {
  Data2DExport_SRM,
  Data2DExport_RAW,
  Data2DExport_TXT,
  Data2DExport_CSV,
  Data2DExport_TableTXT,
  Data2DExport_Fits }

  Export format for 2-dimensional data (DataHandle).
- enum Data3DExportFormat {
  Data3DExport_SRM,
  Data3DExport_RAW,
  Data3DExport_TXT,
  Data3DExport_CSV,
  Data3DExport_VFF,
  Data3DExport_VTK,
  Data3DExport_Fits,
  Data3DExport_TIFF }

Export format for 3-dimensional data (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.

- 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 **CornerPos** {
  NotRecognized,
  BottomLeft,
  BottomRight,
  TopRight,
  TopLeft }

- enum DopplerPropertyInt {
  DopplerAveraging_1,
  DopplerAveraging_2,
  DopplerStride_1,
  DopplerStride_2 }

  Values that determine the behaviour of the Doppler processing routines.

- enum DopplerPropertyFloat { DopplerRefractiveIndex }

  Values that determine the behaviour of the Doppler processing routines.

- enum DopplerFlag { DopplerVelocityScaling }

  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 }

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 CameraPropertyFloat {
  Camera_Saturation,
  Camera_Brightness,
  Camera_Contrast,
  Camera_WB_Red,
  Camera_WB_Green,
  Camera_WB_Blue,
  Camera_WB_Auto }

  Enum identifying properties of the camera.

- enum OCTFileFormat {
  FF_Unknown,
  FF_SRM,
  FF_RAW,
  FF_SDR,
  FF_PHS,
  FF_IMG,
  FF_CSV,
  FF_TXT,
  FF_TABLETXT,
  FF_OCITY,
  FF_FITS,
  FF_VTK,
  FF_VFF,
  FF_TIFF,
  FF_JPG,
  FF_BMP,
  FF_PNG }

  Enum identifying possible file formats.

- enum DataKind {
  dkReal,
  dkColored,
  dkComplex,
  dkRaw,
  dkBinary,
  dkText,
  dkUnknown = 999 }

  Enum identifying.

- enum FileMetadataFloatField {

FMD_RefractiveIndex,
FMD_RangeX,
FMD_RangeY,
FMD_RangeZ,
FMD_CenterX,
FMD_CenterY,
FMD_Angle,
FMD_BinToElectronScaling,
FMD_CentralWavelength,
FMD_SourceBandwidth,
FMD_ElectronFloor,
FMD_DynamicRange_Lower,
FMD_DynamicRange_Upper,
FMD_Rotation3D_X,
FMD_Rotation3D_Y,
FMD_ClipPlaneDepth3D,
FMD_QuadraticDispersionCorrectionFactor,
FMD_SpeckleVarianceThreshold,
FMD_ScanTime,
FMD_ReferenceIntensity,
FMD_ScanPause,
FMD_Zoom,
FMD_MinPointDistance,
FMD_MaxPointDistance,
FMD_MaxExternalTriggerFrequency }

Enum identifying file metadata fields of floating point type.

- enum FileMetadataIntField {
FMD_ProcessState,
FMD_SizeX,
FMD_SizeY,
FMD_SizeZ,
FMD_Oversampling,
FMD_IntensityAveragedSpectra,
FMD_IntensityAveragedAScans,
FMD_IntensityAveragedBScans,
FMD_DopplerAverageX,
FMD_DopplerAverageZ,
FMD_ApoWindow,
FMD_DeviceBitDepth,
FMD_SpectrometerElements,
FMD_Colormap,
FMD_Aspect,
FMD_ExperimentNumber,
FMD_DevicePreset,
FMD_Timestamp,
FMD_CompressionLevel,
FMD_DeviceBytesPerPixel,
FMD_SpeckleAveragingFastAxis,
FMD_SpeckleAveragingSlowAxis }

Enum identifying file metadata fields of integral type.

- enum FileMetadataStringField {

FMD_AxisOrder,
FMD_DeviceName,
FMD_Serial,
FMD_Comment,
FMD_CustomInfo,
FMD_AcquisitionMode,
FMD_Study,
FMD_DispersionPreset,
FMD_ProbeName,
FMD_DevicePresetDescription }

Enum identifying file metadata fields of character string type.

- enum FileMetadataBoolField {
FMD_OffsetApplied,
FMD_DCSubracted,
FMD_ApoApplied,
FMD_DechirpApplied,
FMD_UndersamplingFilterApplied,
FMD_DispersionCompensationApplied,
FMD_QuadraticDispersionCorrectionUsed,
FMD_ImageFieldCorrectionApplied,
FMD_ScanLineShown,
FMD_AutoCorrCompensationUsed,
FMD_BScanCrossCorrelation,
FMD_DCSubractedAdvanced,
FMD_OnlyWindowing,
FMD_RawDataIsSigned,
FMD_FreeformScanPatternIsActive,
FMD_FreeformScanPatternCloseLoop,
FMD_FreeformScanPatternSplineInterpolation,
FMD_ExternalTriggerActive }

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 Device_TriggerType {
Trigger_FreeRunning,
Trigger_TrigBoard_ExternalStart,
Trigger_External_AScan }
- enum ScanPatternPropertyInt {
ScanPattern_SizeTotal,
ScanPattern_Cycles,
ScanPattern_SizeCycle,
ScanPattern_SizePreparationCycle,
ScanPattern_SizeImagingCycle }
- enum ScanPattern_AcquisitionOrder {
ScanPattern_AcqOrderFrameByFrame,
ScanPattern_AcqOrderAll }
- enum **ScanPatternPropertyFloat** {
ScanPattern_RangeX,
ScanPattern_RangeY,
ScanPattern_CenterX,
ScanPattern_CenterY,
ScanPattern_Angle }

- 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 PepperFilterType {
  PepperFilter_Horizontal,
  PepperFilter_Vertical,
  PepperFilter_Star,
  PepperFilter_Block }

  Specifies the type of pepper filter to be applied.
- enum **GaussianFilterType** {
  GaussianFilter_3x3,
  GaussianFilter_5x5 }
- enum **LaplacianFilterType** { **LaplacianFilter_3x3** }
- enum **PrewittFilterType** {
  PrewittFilter_Horizontal_3x3,
  PrewittFilter_Vertical_3x3 }
- enum **SobelFilterType** {
  SobelFilter_Horizontal_3x3,
  SobelFilter_Vertical_3x3 }
- enum **FilterType** {
  Filter_DarkField,
  Filter_BrightField,
  Filter_PhaseContrast }
- enum **PolarizationPropertyInt**
- enum PolarizationPropertyFloat { **PolarizationProcessing_SurfaceThreshold** }

  Values that determine the behaviour of the Polarization processing routines.
- enum PolarizationFlag { PolarizationVelocityScaling }

  Flags that determine the behaviour of the Polarization processing routines.
- enum **USBProbeButtonID** {
  USB_BTN1 = 0xA0u,
  USB_BTN2 = 0xA1u,
  USB_BTN3 = 0xA2u,
  USB_BTN4 = 0xA3u,
  USB_BTN5 = 0xA4u }
- enum **USBProbeMessage** {
  USB_INT_ERR_INVALID_CFG = 0x10,
  USB_INT_BTN = 0xA0u,
  USB_INT_BTN1 = 0xA0u,
  USB_INT_BTN2 = 0xA1u,
  USB_INT_BTN3 = 0xA2u,
  USB_INT_BTN4 = 0xA3u,
  USB_INT_BTN5 = 0xA4u,
  USB_INT_MSG_START = 0xB0u,
  USB_INT_MSG_STOP = 0xB1u,
  USB_INT_MSG_SNAPSHOT = 0xB2u,
  USB_INT_MSG_LED1_ON = 0xB3u,
  USB_INT_MSG_LED1_OFF = 0xB4u,
  USB_INT_MSG_LED2_ON = 0xB5u,
  USB_INT_MSG_LED2_OFF = 0xB6u,
  USB_INT_MSG_FILESIZE = 0xB7u,
  USB_INT_MSG_FIRMWARE_VER = 0xB8u,

USB_INT_MSG_BITMAPSIZE = 0xB9u }
- enum **USBProbeCommand** {
USB_CMD_CONFIGURE = 0x80u,
USB_CMD_RINGLIGHT_SET = 0x81u,
USB_CMD_LED1_TOGGLE = 0x82u,
USB_CMD_LED2_TOGGLE = 0x83u,
USB_CMD_GET_FIRMWARE_VER = 0x84u,
USB_CMD_READ_USER_SIG = 0x87u,
USB_CMD_READ_FILE = 0x88u,
USB_CMD_READ_KEY_BITMAP = 0x89u }
- enum **USBProbeConfiguration** {
USB_CFG_RINGLIGHT_UP = 0x90u,
USB_CFG_RINGLIGHT_DOWN = 0x91u,
USB_CFG_ACQ_STARTSTOP = 0x92u,
USB_CFG_ACQ_SNAPSHOT = 0x93u,
USB_CFG_NOFUNCTION = 0x9fu }
- enum **USBProbeMisc** {
USB_INT_NOFUNCTION = 0x9Fu,
USB_INT_ACK = 0x99u }
- enum **RefstageStatus** {
REFSTAGE_STATUS_IDLE = 0,
REFSTAGE_STATUS_HOMING = 1,
REFSTAGE_STATUS_MOVING = 2,
REFSTAGE_STATUS_MOVING_TO = 3,
REFSTAGE_STATUS_UNDEFINED = -1 }
- enum **RefstageSpeed** {
REFSTAGE_SPEED_SLOW = 0,
REFSTAGE_SPEED_FAST = 1,
REFSTAGE_SPEED_SLOWER = 2,
REFSTAGE_SPEED_FASTER = 3 }

Functions

- SPECTRALRADAR_API unsigned long InterpretReferenceIntensity (float intensity)

  interprets the reference intensity and gives a color code that reflects its state.
- 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 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)

  Returns a pointer to the content of the specified data.
- 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 chung 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 void copyComplexDataContent (ComplexDataHandle DataSource, ComplexFloat *Destination)

  Copies the content of the complex data to the pointer specified as destination.
- SPECTRALRADAR_API ComplexFloat * getComplexDataPtr (ComplexDataHandle Data)

  Returns a pointer to the data represented by the ComplexDataHandle. The data is still managed by the ComplexDataHandle object.
- 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 Alignment1)

  Copies the data in the specified colored data object (ColoredDataHandle) into the specified pointer. This function assues the data to be aligned accordingly.
- SPECTRALRADAR_API unsigned long * getColoredDataPtr (ColoredDataHandle ColData)

  Returns a pointer to the content of the specified ColoredDataHandle.
- 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)

  Sets the data oritentation of the colored data object to the given orientation.
- SPECTRALRADAR_API void getRawDataSize (RawDataHandle Raw, int ∗SizeX, int ∗SizeY, int ∗SizeZ)

  Returns the size of the specified raw data (RawDataHandle).
- SPECTRALRADAR_API void copyRawDataContent (RawDataHandle RawDataSource, void ∗DataContent)

  Copies the content of the raw data into the specified buffer. The suer needs to assure that enough memory is allocated.
- SPECTRALRADAR_API void ∗ getRawDataPtr (RawDataHandle RawDataSource)

  Returns the pointer to the raw data content. The pointer might no longer after additional actions using the RawDataHandle.
- SPECTRALRADAR_API int getRawDataPropertyInt (RawDataHandle RawData, RawDataPropertyInt Property)

  Returns a raw data property.
- SPECTRALRADAR_API void setRawDataBytesPerPixel (RawDataHandle Raw, int BytesPerPixel)

  Sets the bytes per pixel for raw data.
- SPECTRALRADAR_API void resizeRawData (RawDataHandle Raw, int Size1, int Size2, int Size3)

  Resizes the specified raw data buffer accordingly.
- SPECTRALRADAR_API void **reserveRawData** (RawDataHandle Raw, int Size1, int Size2, int Size3)
- SPECTRALRADAR_API void setRawDataContent (RawDataHandle RawDataSource, 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)

  Sets the number of the spectra in the raw data that are used for creating A-scan/B-scan data.
- SPECTRALRADAR_API void setApodizationSpectra (RawDataHandle RawData, int NumberOfScanRegions, int ∗ApodizationRegions)

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

  Creates a raw data object (RawDataHandle).
- SPECTRALRADAR_API void clearRawData (RawDataHandle Raw)

  Clears a raw data object (RawDataHandle)
- 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, DataHandle, DataHandle or DataHandle objects.
- 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.
- SPECTRALRADAR_API void clearComplexData (ComplexDataHandle Data)

  Clears a data object holding complex data.
- SPECTRALRADAR_API BOOL **isDeviceAvailable** ()
- SPECTRALRADAR_API void **expectedDevice** (char *Buffer, int Size, int *Rev)
- SPECTRALRADAR_API OCTDeviceHandle initDevice (void)

  Initializes the installed device.
- SPECTRALRADAR_API void getDeviceType (OCTDeviceHandle Dev, char DevName[], int BufferSize)

  Gives the name of the device type that is given by the OCTDeviceHandle.
- SPECTRALRADAR_API int getDeviceRevision (OCTDeviceHandle Dev)

  Returns the revision of the device given by the OCTDeviceHandle.
- SPECTRALRADAR_API void getDeviceSerialNumber (OCTDeviceHandle Dev, char DevName[], int BufferSize)

  Returns the serial number of the device given by the OCTDeviceHandle.
- SPECTRALRADAR_API int getDevicePropertyInt (OCTDeviceHandle Dev, DevicePropertyInt 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 void closeDevice (OCTDeviceHandle Dev)

  Closes the device opened previously with initDevice.
- SPECTRALRADAR_API BOOL isDeviceOn (OCTDeviceHandle Handle)

  Returns if the device is switched on.
- SPECTRALRADAR_API BOOL isVideoCameraAvailable (OCTDeviceHandle Dev)

  Returns if the video camera is available.
- SPECTRALRADAR_API BOOL isSLDAvailable (OCTDeviceHandle Dev)

  Returns whethter the SLD is available.
- SPECTRALRADAR_API void setSLD (OCTDeviceHandle Dev, BOOL OnOff)

  switches the SLD of the SpectralRadar device on and off.
- SPECTRALRADAR_API void moveScanner (OCTDeviceHandle Dev, ProbeHandle Probe, ScanAxis Axis, double Position)

  manually moves the scanner to a given position
- SPECTRALRADAR_API void setLaserDiode (OCTDeviceHandle Dev, BOOL OnOff)

  switches the LaserDiode of the SpectralRadar device on and off.
- SPECTRALRADAR_API double getWavelengthAtPixel (OCTDeviceHandle Dev, int Pixel)

  Returns the wavelength at a speicified pixel of the spectrometer.
- SPECTRALRADAR_API void **setCameraPreset** (OCTDeviceHandle Dev, ProbeHandle Probe, ProcessingHandle Proc, int Preset)
- SPECTRALRADAR_API int getCameraPreset (OCTDeviceHandle Dev)

  Gets the currently used device preset.
- SPECTRALRADAR_API void **getCameraPresetDescription** (OCTDeviceHandle Dev, int Preset, char *Description, int BufferSize)
- SPECTRALRADAR_API int **getNumberOfCameraPresets** (OCTDeviceHandle Dev)

- SPECTRALRADAR_API int getNumberOfInternalValues (OCTDeviceHandle Dev)

  Returns the number of Analog-to-Digital Converter present in the device.
- SPECTRALRADAR_API void getInternalValueName (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 getInternalValueByName (OCTDeviceHandle Dev, const char ∗Name)

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

  Returns the value of the selected ADC.
- SPECTRALRADAR_API ProbeHandle initProbe (OCTDeviceHandle Dev, const char ∗ProbeFile)

  Initializes a probe specified by ProbeFile.
- SPECTRALRADAR_API ProbeHandle initStandardProbe (OCTDeviceHandle Dev)

  Creates a standard probe using the Probe.ini file. If this configuration file is not found, standard parameters without valid calibration will be used.
- SPECTRALRADAR_API ProbeHandle initProbeWithType (OCTDeviceHandle Dev, ProbeType Type)

  Creates a standard probe for the given probe type but without valid calibration data .
- 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.
- 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 getProbeName (ProbeHandle Probe, char ProbeName[], int BufferSize)

  Returns the name of the specified probe.
- SPECTRALRADAR_API void setProbeName (ProbeHandle Probe, const char ∗ProbeName)

  Sets the given name of the specified probe.
- SPECTRALRADAR_API void getProbeSerialNo (ProbeHandle Probe, char SerialNo[], int BufferSize)

  Gets the serial number of the specified probe.
- SPECTRALRADAR_API void setProbeSerialNo (ProbeHandle Probe, const char ∗SerialNo)

  Gets the serial number of the specified probe.
- SPECTRALRADAR_API void getProbeType (ProbeHandle Probe, char Type[], int BufferSize)

  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 getProbeObjective (ProbeHandle Probe, char Objective[], int BufferSize)

  Gets the objective of the specified probe.
- SPECTRALRADAR_API void setProbeObjective (ProbeHandle Probe, const char ∗Objective)

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

  Closes the probe and frees all memory associated with it.
- SPECTRALRADAR_API void blendEnFaceInCamera (ProbeHandle Probe, ScanPatternHandle Pattern, ColoredDataHandle EnFace2D, ColoredDataHandle Image, float Ratio, BOOL DenseView)

Blends the en-face image of a given volume acquisition on top of the video image. Can be used to calibrate the probe manually.

- 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 needs to be assured that the device is properly calibrated.
- 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 ScanPatternHandle createPointScanPattern (ProbeHandle Probe, int Size, double PosX, double PosY)

  Creates a scan pattern that consists of a single point (PosX, PosY). The galvo doesn't move from there. Use this pattern for point scans and/or non-scanning probes.
- SPECTRALRADAR_API ScanPatternHandle createNoScanPattern (ProbeHandle Probe, int Scans, int NumberOfScans)

  Creates a simple scan pattern that does not move the galvo. Use this pattern for point scans and/or non-scanning probes.
- SPECTRALRADAR_API ScanPatternHandle createTriggerPattern (ProbeHandle Probe, int Scans)

  Creates a pattern only consisting of a specified amount of trigger signals.
- SPECTRALRADAR_API ScanPatternHandle createBScanPattern (ProbeHandle Probe, double Range, int AScans, BOOL apodization)

  Creates a simple B-scan pattern that moves the galvo over a specified range.
- SPECTRALRADAR_API ScanPatternHandle createBilateralBScanPattern (ProbeHandle Probe, double Range, int AScans, double Shift)

  Creates a bilateral scan pattern. The contouring error can be influenced using the Shift parameter.
- SPECTRALRADAR_API ScanPatternHandle createBScanPatternManual (ProbeHandle Probe, double StartX, double StartY, double StopX, double StopY, int AScans, BOOL apodization)

  Creates a B-scan pattern specified by start and end points.
- SPECTRALRADAR_API ScanPatternHandle createIdealBScanPattern (ProbeHandle Probe, double Range, 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, int AScans)

  Creates a circle scan pattern.
- SPECTRALRADAR_API ScanPatternHandle createVolumePattern (ProbeHandle Probe, double RangeX, int SizeX, double RangeY, int SizeY)

  Creates a simple volume pattern.
- SPECTRALRADAR_API ScanPatternHandle createBScanStackPattern (ProbeHandle Probe, double RangeX, int SizeX, double RangeY, int SizeY)

  Creates a simple stack pattern.
- SPECTRALRADAR_API ScanPatternHandle createFreeformScanPattern (ProbeHandle Probe, float *positions, int size_x, int size_y, BOOL apodization)

  Creates a freeform scan pattern based on an array of positions.
- SPECTRALRADAR_API ScanPatternHandle createFragmentedScanPattern (ProbeHandle Probe, int ChunkSize, int NumberOfChunks)

  Creates a scan pattern which can be used to acquire a dataset of <NumberOfChunks> times <ChunkSize> A-scans at position 0/0. The Fragmented scan pattern can be compared in structure to a B-scan stack pattern with x and y ranges of 0; however the fragmented scan pattern behaves like a volume pattern in that it shows no delay between the respective chunks.
- SPECTRALRADAR_API void updateScanPattern (ScanPatternHandle Pattern)

  Updates the specfied pattern (ScanPatternHandle);.
- SPECTRALRADAR_API void rotateScanPattern (ScanPatternHandle Pattern, double Angle)

Rotates the specfied pattern (ScanPatternHandle);.

- SPECTRALRADAR_API void rotateScanPatternExt (ScanPatternHandle Pattern, double Angle, int index)

  Rotates the scan #index (0-based) of the specfied pattern (ScanPatternHandle).
- SPECTRALRADAR_API void shiftScanPattern (ScanPatternHandle Pattern, double ShiftX, double ShiftY)

  Shifts the specified pattern (ScanPatternHandle).
- SPECTRALRADAR_API void shiftScanPatternExt (ScanPatternHandle Pattern, double ShiftX, double ShiftY, BOOL ShiftApo, int Index)

  Shifts the scan #index (0-based) of the specified pattern (ScanPatternHandle).
- SPECTRALRADAR_API void zoomScanPattern (ScanPatternHandle Pattern, double Factor)

  Zooms the specified pattern (ScanPatternHandle).
- SPECTRALRADAR_API int getScanPatternLUTSize (ScanPatternHandle Pattern)

  Returns the number of data points the specified pattern (ScanPatternHandle) used.
- SPECTRALRADAR_API void getScanPatternLUT (ScanPatternHandle Pattern, double ∗PosX, double ∗PosY)

  Returns the actual positions to be scanned with the specified pattern (ScanPatternHandle).
- SPECTRALRADAR_API void clearScanPattern (ScanPatternHandle Pattern)

  Clears the specified scan pattern (ScanPatternHandle).
- SPECTRALRADAR_API size_t **projectMemoryRequirement** (OCTDeviceHandle Handle, ScanPatternHandle Pattern, AcquisitionType type)
- 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 N spectra of raw data without moving galvo scanners.
- SPECTRALRADAR_API void measureSpectraEx (OCTDeviceHandle Dev, int N, RawDataHandle Raw, int CameraIndex)

  Acquires N spectra of raw data without moving galvo scanners. Supports multiple cameras (e.g. PS-OCT).
- SPECTRALRADAR_API ProcessingHandle **createProcessing** (int SpectrumSize, int BytesPerRawPixel, BOOL Signed, float ScalingFactor, float MinElectrons, ProcessingType Type, float FFTOversampling)
- SPECTRALRADAR_API ProcessingHandle createProcessingForDevice (OCTDeviceHandle Dev)

  Creates suitable standard processing routines for the specified device (OCTDeviceHandle).
- SPECTRALRADAR_API ProcessingHandle createProcessingForDeviceEx (OCTDeviceHandle Dev, int CameraIndex)

  Creates suitable standard processing routines for the specified device (OCTDeviceHandle) with camera index.
- SPECTRALRADAR_API int getInputSize (ProcessingHandle Proc)

  Returns the expected input size (pixels per spectrum); of the processing algorithms.
- SPECTRALRADAR_API int getAScanSize (ProcessingHandle Handle)

  gives the number of pixels in an A-Scan of the SpectralRadar device. This number is identical to the number of rows in a finished B-Scan.
- SPECTRALRADAR_API int getApodizationWindow (ProcessingHandle Proc)

  Gets the window function that is being used for apodization.
- SPECTRALRADAR_API void setApodizationWindow (ProcessingHandle Proc, ApodizationWindow Window)

  Sets the window function that is to be used for apodization. The selected function will be used in all subsequent processings.

- 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.
- SPECTRALRADAR_API double getApodizationWindowParameter (ProcessingHandle Proc, ApodizationWindowParameter Selection)

  Gets the apodization window parameter, such as window width or ratio between constant and cosine part.
- SPECTRALRADAR_API void setDechirpAlgorithm (ProcessingHandle Proc, ProcessingType Type)

  Sets the algorithm that is to be sued 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, ProcessingParameterFloat Selection, double Value)
- SPECTRALRADAR_API double getProcessingParameterFloat (ProcessingHandle Proc, ProcessingParameterFloat Selection)

  Gets the specified processing paramter.
- 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, ProcessingAveragingAlgorithm Algorithm)

  Sets the algorithm that is used for averaing by the processing.
- SPECTRALRADAR_API void setCalibration (ProcessingHandle Proc, CalibrationData Selection, DataHandle Data)

  Sets the current active calibration data.
- SPECTRALRADAR_API void getCalibration (ProcessingHandle Proc, CalibrationData Selection, DataHandle Data)

  Returns the currently active calibration parameter.
- 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 measureSpectrum (OCTDeviceHandle Dev, ProbeHandle Probe, ProcessingHandle Proc, BOOL moveToApoPos)

  Measures apodization spectrum and uses them in subsequent processing.
- SPECTRALRADAR_API void saveCalibrationAuto (ProcessingHandle Proc, CalibrationData Selection)

  Saves the selected calibration in its default path.
- 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 use 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 ProcessingCorrectedSpectrum)

  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 setProcessedDataOutput (ProcessingHandle Proc, DataHandle Scan)

  Sets the pointer the resulting B-Scan of the next processing is written to.
- SPECTRALRADAR_API void setHorMirroredDataOutput (ProcessingHandle Proc, DataHandle Scan)

  Sets the pointer the resulting B-Scan of the next processing is written to. The result will be written mirrored at the horizontal axis.
- SPECTRALRADAR_API void setColoredDataOutput (ProcessingHandle Proc, ColoredDataHandle BScan, Coloring32BitHandle Color)

  Sets the pointer the resulting colored B-Scan of the next processing is written to.
- SPECTRALRADAR_API void setTransposedColoredDataOutput (ProcessingHandle Proc, ColoredDataHandle BScan, Coloring32BitHandle Color)

  Sets the pointer the resulting colored B-Scan of the next processing is written to. The data will be transposed so that the first axis is the x-axis.
- SPECTRALRADAR_API void setComplexDataOutput (ProcessingHandle Proc, ComplexDataHandle ComplexBScan)

  Sets the pointer the resulting complex B-Scan of the next processing is written to.
- SPECTRALRADAR_API void executeProcessing (ProcessingHandle Proc, RawDataHandle RawData)

  Execute the processing.
- SPECTRALRADAR_API void closeProcessing (ProcessingHandle Proc)

  Closes the processing and frees all temporary memory that was associated with it. Processing threads will be stopped.
- SPECTRALRADAR_API void computeDispersion (DataHandle Spectrum1In, DataHandle Spectrum2In, DataHandle ChirpOut, DataHandle DispOut)

  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 QuadraticIn, DataHandle ChirpIn, DataHandle DispOut)

  Computes dispersion by a quadratic approximation specified by the quadratic factor.
- SPECTRALRADAR_API void computeDispersionByImage (DataHandle LinearKSpectraIn, DataHandle ChirpIn, DataHandle DispOut)

  Guesses the dispersion based on the raw data specified. The raw data needs to be linearized in k before applying to 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 void exportData1D (DataHandle Data, Data1DExportFormat Format, const char *Path)

  Exports 1-dimensional data (DataHandle).

- SPECTRALRADAR_API void exportData2D (DataHandle Data, Data2DExportFormat Format, const char *Path)

  Exports 2-dimensional data (DataHandle).
- SPECTRALRADAR_API void exportData3D (DataHandle Volume, Data3DExportFormat Format, const char *Path)

  Exports 3-dimensional data (DataHandle).
- SPECTRALRADAR_API void **exportData2DAsImage** (DataHandle Data, Coloring32BitHandle Color, ColoredDataExportFormat format, const char *fileName, BOOL drawScale, BOOL drawMarkers, BOOL physicalAspectRatio)
- SPECTRALRADAR_API void **exportData3DAsImage** (DataHandle Data, Coloring32BitHandle Color, ColoredDataExportFormat format, Direction SliceNormalDirection, const char *fileName, BOOL drawScale, BOOL drawMarkers, BOOL physicalAspectRatio)
- SPECTRALRADAR_API void importData (DataHandle Data, DataImportFormat Format, const char *Path)

  Imports data with the specified format and copies it into a dat data object (DataHandle).
- SPECTRALRADAR_API void exportComplexData (ComplexDataHandle, ComplexDataExportFormat, const char *)

  Exports 1-, 2- and 3-dimensional complex data (ComplexDataHandle)
- SPECTRALRADAR_API void exportColoredData (ColoredDataHandle Image, ColoredDataExportFormat Format, const char *fileName)

  Exports colored data (ColoredDataHandle)
- SPECTRALRADAR_API void importColoredData (ColoredDataHandle ColoredData, DataImportFormat Format, const char *Path)

  Imports colored data (ColoredDataHandle) with the specified format and copied it into a data object (ColoredDataHandle)
- SPECTRALRADAR_API void exportRawData (RawDataHandle Raw, RawDataExportFormat Format, const char *Path)

  Exports the specified data to disk.
- SPECTRALRADAR_API void importRawData (RawDataHandle Raw, RawDataImportFormat Format, const char *Path)

  Imports the specified data from disk.
- SPECTRALRADAR_API void appendRawData (RawDataHandle Data, RawDataHandle DataToAppend, Direction direction)

  Appends the new raw data to the old raw data in the specified direction.
- SPECTRALRADAR_API void getRawDataSliceIndex (RawDataHandle Data, RawDataHandle Slice, Direction SliceNormalDirection, int Index)

  Returns a slice of raw data in the specified direction at the specified index.
- SPECTRALRADAR_API double analyzeData (DataHandle Data, DataAnalyzation Selection)

  Performs the selected analyzation of the specified data and returns the resulting value.
- SPECTRALRADAR_API double analyzeAScan (DataHandle Data, AScanAnalyzation Selection)

  Performs the selected analyzation of the specified A-scan and returns the resulting value.
- SPECTRALRADAR_API void determineDynamicRange (DataHandle Data, float *MinRange_dB, float *MaxRange_dB)

  Gives a rough estimation of the dynamic range of the specified data object.
- SPECTRALRADAR_API void transpose (DataHandle DataIn, DataHandle DataOut)

  Transposes the given data and writes the result to DataOut.
- SPECTRALRADAR_API void transposeInplace (DataHandle Data)

  Transposes the given Data.
- SPECTRALRADAR_API void transposeAndScaleData (DataHandle DataIn, DataHandle DataOut, float Min, float Max)

  Transposes the given data and scales it to the range [Min, Max].
- SPECTRALRADAR_API void normalizeData (DataHandle Data, float Min, float Max)

  Scales the given data to the range [Min, Max].
- SPECTRALRADAR_API void lockData (DataHandle Data)

Locks the given data.

- SPECTRALRADAR_API void unlockData (DataHandle Data)

  Unlocks the given data.

- SPECTRALRADAR_API void getDataSlicePos (DataHandle Data, DataHandle Slice, Direction SliceNormalDirection, double Pos)

  Returns a slice of data in the specified direction at the specified position.

- SPECTRALRADAR_API void getDataSliceIndex (DataHandle Data, DataHandle Slice, Direction SliceNormalDirection, int Index)

  Returns a slice of data in the specified direction at the specified index.

- SPECTRALRADAR_API void getDataSliceAnalyzed (DataHandle Data, DataHandle Slice, Direction SliceNormalDirection, DataAnalyzation Selection)

  Returns a slice of data that has been computed of all slice using the specified analyzation method.

- SPECTRALRADAR_API void appendData (DataHandle Data, DataHandle NewData, Direction direction)

  Appends the new data to the old data in the specified direction.

- SPECTRALRADAR_API void cropData (DataHandle Data, Direction direction, int Index)

  Crops the data at the specific direction at the given index. The result will contain the data with range [0, index] at the cropping dirction.

- SPECTRALRADAR_API void cropDataEx (DataHandle Data, Direction direction, int IndexMax, int IndexMin)

  Crops the data at the specific direction at the given indeces. The result will contain the data with range [IndexMin, IndexMax] at the cropping dirction.

- 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 be in Data1, the second separated in Data2.

- SPECTRALRADAR_API void flipData (DataHandle Data, Direction FlippingDirection)

  Flips the data around the specific direction.

- SPECTRALRADAR_API void appendComplexData (ComplexDataHandle Data, ComplexDataHandle DataToAppend, Direction direction)

  Appends the new data to the old data in the specified direction.

- SPECTRALRADAR_API void getComplexDataSlicePos (ComplexDataHandle Data, ComplexDataHandle Slice, Direction SliceNormalDirection, double Pos)

  Returns a slice of data in the specified direction at the specified position.

- SPECTRALRADAR_API void getComplexDataSliceIndex (ComplexDataHandle Data, ComplexDataHandle Slice, Direction SliceNormalDirection, int Index)

  Returns a slice of data in the specified direction at the specified index.

- SPECTRALRADAR_API void cropComplexData (ComplexDataHandle Data, Direction CroppingDirection, int IndexMax, int IndexMin)

  Crops the complex data at the specific direction at the given indeces. The result will contain the data with range [IndexMin, IndexMax] at the cropping dirction.

- SPECTRALRADAR_API void appendColoredData (ColoredDataHandle Data, ColoredDataHandle DataToAppend, Direction AppendingDirection)

  Appends the new colored data to the old colored data in the specified direction.

- SPECTRALRADAR_API void cropColoredData (ColoredDataHandle Data, Direction CroppingDirection, int IndexMax, int IndexMin)

  Crops the colored data at the specific direction at the given indeces. The result will contain the data with range [IndexMin, IndexMax] at the cropping dirction.

- SPECTRALRADAR_API void getColoredDataSlicePos (ColoredDataHandle Data, ColoredDataHandle Slice, Direction SliceNormalDirection, double Pos)

  Get a slice of the colored data with specific slicing direction at given index position.

- SPECTRALRADAR_API void getColoredDataSliceIndex (ColoredDataHandle Data, ColoredDataHandle Slice, Direction SliceNormalDirection, int Index)

  Get a slice of the colored data with specific slicing direction at given index.

- SPECTRALRADAR_API ImageFieldHandle createImageField (void)

Creates an object holding image field data.

- 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, DataHandle Surface)

  Determines the image field correction of the surface.
- SPECTRALRADAR_API void determineImageFieldForProbe (ProbeHandle Probe, DataHandle Surface)

  Determines the image field correction of the surface for the specified probe handle.
- SPECTRALRADAR_API void determineImageFieldForProbeWithMap (ProbeHandle Probe, DataHandle Surface, DataHandle Map)

  Determines the image field correction of the surface for the specified probe handle using the given map. Values != 0 in the map specifies to use the data in the surface handle otherwise thex will be interpolated.
- SPECTRALRADAR_API void correctImageField (ImageFieldHandle ImageField, ScanPatternHandle Pattern, DataHandle Data)

  Applies the image field correction to the B-Scan or volume data .
- SPECTRALRADAR_API void correctImageFieldFromProbe (ProbeHandle Probe, ScanPatternHandle Pattern, DataHandle Data)

  Applies the image field correction saved in the probe handle to the B-Scan or volume data .
- SPECTRALRADAR_API void correctSurface (ImageFieldHandle ImageField, DataHandle Surface)

  Applies the image field correction to the given Surface.
- SPECTRALRADAR_API VisualCalibrationHandle **createVisualCalibration** (OCTDeviceHandle Device, double TargetCornerLength_mm, BOOL CheckAngle, BOOL SaveData)
- SPECTRALRADAR_API void **clearVisualCalibration** (VisualCalibrationHandle Handle)
- SPECTRALRADAR_API BOOL **visualCalibrate_1st_CameraScaling** (VisualCalibrationHandle Handle, ProbeHandle Probe, ColoredDataHandle Image)
- SPECTRALRADAR_API BOOL **visualCalibrate_2nd_Galvo** (VisualCalibrationHandle Handle, ProbeHandle Probe, ColoredDataHandle Image)
- SPECTRALRADAR_API BOOL **visualCalibrate_previewImage** (VisualCalibrationHandle Handle, ColoredDataHandle Image)
- SPECTRALRADAR_API void **visualCalibration_getHoles** (VisualCalibrationHandle Handle, int *x0, int *y0, int *x1, int *y1, int *x2, int *y2)
- SPECTRALRADAR_API const char * **visualCalibrate_Status** (VisualCalibrationHandle Handle)
- SPECTRALRADAR_API BOOL **visualCalibrate_CameraCenter** (VisualCalibrationHandle Handle, OCTDeviceHandle Device, ColoredDataHandle Image)
- SPECTRALRADAR_API BOOL **visualCalibrate_getVideoCameraCenterImage** (VisualCalibrationHandle Handle, OCTDeviceHandle Device, ColoredDataHandle Image)
- SPECTRALRADAR_API DopplerProcessingHandle createDopplerProcessing (void)
- SPECTRALRADAR_API void createDopplerProcessingForProcessing (DopplerProcessingHandle *Doppler, ProcessingHandle Proc)
- SPECTRALRADAR_API void closeDopplerProcessing (DopplerProcessingHandle Doppler)

  Closes the Doppler processing routines and frees the memory that has been allocated for these to work properly.
- SPECTRALRADAR_API void setDopplerPropertyInt (DopplerProcessingHandle Doppler, DopplerPropertyInt Property, int Value)

  Sets Doppler processing properties.
- SPECTRALRADAR_API void setDopplerPropertyFloat (DopplerProcessingHandle Doppler, DopplerPropertyFloat Property, float Value)

  Sets Doppler processing properties.
- SPECTRALRADAR_API void setDopplerFlag (DopplerProcessingHandle Doppler, DopplerFlag Flag, BOOL OnOff)

Sets the Doppler processing flags.

- SPECTRALRADAR_API void setDopplerAmplitudeOutput (DopplerProcessingHandle Doppler, DataHandle AmpOut)

  Sets the location of the resulting doppler amplitude output.
- SPECTRALRADAR_API void setDopplerPhaseOutput (DopplerProcessingHandle Doppler, DataHandle PhasesOut)

  Sets the location of the resulting doppler phase output.
- SPECTRALRADAR_API void getDopplerOutputSize (DopplerProcessingHandle Doppler, 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 executeDopplerProcessing (DopplerProcessingHandle Doppler, ComplexDataHandle Input)

  Executes the Doppler processing of the input data and returns phases and amplitudes.
- SPECTRALRADAR_API void calcContrast (DataHandle ApodizedSpectrum, DataHandle Contrast)

  Computes the contrast for the specified (apodized); spectrum.
- SPECTRALRADAR_API SettingsHandle loadSettingsFile (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 loadSettingsFile);.
- SPECTRALRADAR_API double getSettingsEntryDouble (SettingsHandle SettingsFile, const char ∗Node, double DefaultValue)

  Gets an floating point number from the specified ini file (see SettingsHandle and loadSettingsFile);.
- SPECTRALRADAR_API void **getSettingsEntryFloatArray** (SettingsHandle SettingsFile, const char ∗Node, const double ∗DefaultValues, double ∗Values, int ∗Size)
- SPECTRALRADAR_API void getSettingsEntryString (SettingsHandle SettingsFile, const char ∗Node, const char ∗Default, char ∗Data, int MaxDataSize)

  Gets a string from the specified ini file (see SettingsHandle and loadSettingsFile);.
- SPECTRALRADAR_API void setSettingsEntryInt (SettingsHandle SettingsFile, const char ∗Node, int Value)

  Sets an integer entry in the specified ini file (see SettingsHandle and loadSettingsFile);.
- SPECTRALRADAR_API void setSettingsEntryFloat (SettingsHandle SettingsFile, const char ∗Node, double Value)

  Sets a floating point entry in the specified ini file (see SettingsHandle and loadSettingsFile);.
- SPECTRALRADAR_API void saveSettings (SettingsHandle SettingsFile)

  Saves the changes to the specified Settings file.
- SPECTRALRADAR_API void closeSettingsFile (SettingsHandle SettingsFile)

  Closes the specified ini file and stores the set entries (see SettingsHandle and loadSettingsFile);.
- SPECTRALRADAR_API void setSettingsEntryString (SettingsHandle SettingsFile, const char ∗Node, const char ∗Value)

  Sets a string in the specified ini file (see SettingsHandle and loadSettingsFile);.
- SPECTRALRADAR_API Coloring32BitHandle 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 Coloring32BitHandle createCustomColoring32Bit (int LUTSize, unsigned long LUT[])

  Create custom coloring using the specified color look-up-table.
- SPECTRALRADAR_API void setColoringBoundaries (Coloring32BitHandle 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 (Coloring32BitHandle Coloring, ColorEnhancement Enhancement)

  Selects a function for non-linear coloring to enhance (subjective) image impression.

- SPECTRALRADAR_API void colorizeData (Coloring32BitHandle Coloring, DataHandle Data, ColoredDataHandle ColoredData, BOOL Transpose)

  Colors a given data object (DataHandle) into a given colored object (ColoredDataHandle).
- SPECTRALRADAR_API void colorizeDopplerData (Coloring32BitHandle AmpColoring, Coloring32BitHandle 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.
- SPECTRALRADAR_API void clearColoring32Bit (Coloring32BitHandle Coloring)

  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, int SizeX, int SizeY, ColoredDataHandle Image)

  Gets a camera image.
- SPECTRALRADAR_API void **getFlippedCameraImage** (OCTDeviceHandle Dev, int SizeX, int SizeY, ColoredDataHandle Image, BOOL InvertX, BOOL InvertY)
- SPECTRALRADAR_API void getMirroredCameraImage (OCTDeviceHandle Dev, int SizeX, int SizeY, ColoredDataHandle Image)

  Gets a camera image.
- SPECTRALRADAR_API void setCameraPropertyFloat (OCTDeviceHandle Dev, CameraPropertyFloat Selection, double Value)

  Sets saturation, brightness and contrast for the camera images if this option is available for the current device.
- SPECTRALRADAR_API void setCameraShowScanPattern (OCTDeviceHandle Dev, BOOL Value)

  Enables to turn on/off the scan pattern overlay in the video camera image.
- SPECTRALRADAR_API void visualizeScanPattern (OCTDeviceHandle Dev, ProbeHandle Probe, ScanPatternHandle Pattern, BOOL showRawPattern)

  Visualizes the scan pattern in top of the camera image.
- 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 refernce intensity that was present when the currently used apodization was determined.
- 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 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 int getBufferSize (BufferHandle)

  Returns the currently avaiable data sets 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)

Clears the buffer and frees all data and colored data objects in it.

- SPECTRALRADAR_API int getNumberOfOutputValues (OCTDeviceHandle Dev)

  Returns the number of output values.

- SPECTRALRADAR_API void getOutputValueName (OCTDeviceHandle Dev, int Index, char *Name, int NameStringSize, char *Unit, int UnitStringSize)

  Returns names and units of the requested output values.

- SPECTRALRADAR_API BOOL **doesOutputExist** (OCTDeviceHandle Dev, const char *Name)
- SPECTRALRADAR_API void setOutputValueByName (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 getOutputValueRangeByName (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 void **TestFileEngine** ()
- SPECTRALRADAR_API OCTFileHandle **createOCTFile** (OCTFileFormat format)
- SPECTRALRADAR_API void **clearOCTFile** (OCTFileHandle handle)
- SPECTRALRADAR_API int **getFileDataFileCount** (OCTFileHandle handle)
- SPECTRALRADAR_API void **loadFile** (OCTFileHandle handle, const char *fileName)
- SPECTRALRADAR_API void **saveFile** (OCTFileHandle handle, const char *fileName)
- SPECTRALRADAR_API void **copyFileMetadata** (OCTFileHandle src, OCTFileHandle dest)
- SPECTRALRADAR_API double **getFileMetadataFloat** (OCTFileHandle handle, FileMetadataFloatField floatfield)
- SPECTRALRADAR_API void **setFileMetadataFloat** (OCTFileHandle handle, FileMetadataFloatField floatfield, double value)
- SPECTRALRADAR_API int **getFileMetadataInt** (OCTFileHandle handle, FileMetadataIntField intfield)
- SPECTRALRADAR_API void **setFileMetadataInt** (OCTFileHandle handle, FileMetadataIntField intfield, int value)
- SPECTRALRADAR_API size_t **getFileMetadataString** (OCTFileHandle handle, FileMetadataStringField stringfield, char *content, int length)
- SPECTRALRADAR_API void **setFileMetadataString** (OCTFileHandle handle, FileMetadataStringField stringfield, const char *content)
- SPECTRALRADAR_API BOOL **getFileMetadataBool** (OCTFileHandle handle, FileMetadataBoolField boolfield)
- SPECTRALRADAR_API void **setFileMetadataBool** (OCTFileHandle handle, FileMetadataBoolField boolfield, BOOL value)
- SPECTRALRADAR_API DataHandle **getFileRealData** (OCTFileHandle handle, size_t index)
- SPECTRALRADAR_API ColoredDataHandle **getFileColoredData** (OCTFileHandle handle, size_t index)
- SPECTRALRADAR_API ComplexDataHandle **getFileComplexData** (OCTFileHandle handle, size_t index)
- SPECTRALRADAR_API RawDataHandle **getFileRawData** (OCTFileHandle handle, size_t index)
- SPECTRALRADAR_API ColoredDataHandle **getFileBinaryData** (OCTFileHandle handle, size_t index)
- SPECTRALRADAR_API void **getFile** (OCTFileHandle handle, size_t index, const char *filenameOnDisk)
- SPECTRALRADAR_API void **addFileRealData** (OCTFileHandle handle, DataHandle data, const char *filename)

- SPECTRALRADAR_API void **addFileColoredData** (OCTFileHandle handle, ColoredDataHandle data, const char *filename)
- SPECTRALRADAR_API void **addFileComplexData** (OCTFileHandle handle, ComplexDataHandle data, const char *filename)
- SPECTRALRADAR_API void **addFileRawData** (OCTFileHandle handle, RawDataHandle data, const char *filename)
- SPECTRALRADAR_API void **addFileBinaryData** (OCTFileHandle handle, ColoredDataHandle data, const char *filename)
- SPECTRALRADAR_API void **addFileBinary** (OCTFileHandle handle, const char *filenameOnDisk, const char *filename)
- SPECTRALRADAR_API void **addFileText** (OCTFileHandle handle, const char *filenameOnDisk, const char *filename)
- SPECTRALRADAR_API DataKind **getFileDataFileType** (OCTFileHandle handle, int index)
- SPECTRALRADAR_API void **getFileDataFileName** (OCTFileHandle handle, int index, char *filename, int length)
- SPECTRALRADAR_API int **getFileDataSizeX** (OCTFileHandle handle, size_t index)
- SPECTRALRADAR_API int **getFileDataSizeY** (OCTFileHandle handle, size_t index)
- SPECTRALRADAR_API int **getFileDataSizeZ** (OCTFileHandle handle, size_t index)
- SPECTRALRADAR_API float **getFileDataRangeX** (OCTFileHandle handle, size_t index)
- SPECTRALRADAR_API float **getFileDataRangeY** (OCTFileHandle handle, size_t index)
- SPECTRALRADAR_API float **getFileDataRangeZ** (OCTFileHandle handle, size_t index)
- SPECTRALRADAR_API void **setMarkerListFromRealData** (OCTFileHandle handle, DataHandle data)
- SPECTRALRADAR_API void **setMarkerListInRealData** (OCTFileHandle handle, DataHandle data)
- SPECTRALRADAR_API SpeckleVarianceHandle **initSpeckleVariance** (void)
- SPECTRALRADAR_API void **closeSpeckleVariance** (SpeckleVarianceHandle SpeckleVar)
- SPECTRALRADAR_API void **setAveraging** (SpeckleVarianceHandle SpeckleVar, int Av1, int Av2, int Av3)
- SPECTRALRADAR_API void **setSpeckleVarianceType** (SpeckleVarianceHandle SpeckleVar, SpeckleVarianceType Type)
- SPECTRALRADAR_API void **computeSpeckleVariance** (SpeckleVarianceHandle SpeckleVar, ComplexDataHandle CompDataIn, DataHandle DataOutMean, DataHandle DataOutVar)
- SPECTRALRADAR_API void **setSpeckleVarianceThreshold** (SpeckleVarianceHandle SpecleVar, double Threshold)
- SPECTRALRADAR_API void **setTriggerMode** (OCTDeviceHandle Dev, Device_TriggerType TriggerMode)
- SPECTRALRADAR_API Device_TriggerType getTriggerMode (OCTDeviceHandle Dev)

  Returns the trigger mode used for acquisition.
- SPECTRALRADAR_API bool isTriggerModeAvailable (OCTDeviceHandle Dev, Device_TriggerType TriggerMode)

  Returns whether the specified trigger mode is possible or not for the used device.
- SPECTRALRADAR_API void **setTriggerTimeoutSec** (OCTDeviceHandle Dev, int Timeout)
- SPECTRALRADAR_API int getTriggerTimeoutSec (OCTDeviceHandle Dev)

  Returns the timeout of the camera (not used in trigger mode Trigger_FreeRunning).
- SPECTRALRADAR_API int **getScanPatternPropertyInt** (ScanPatternHandle ScanPattern, ScanPatternPropertyInt Property)
- SPECTRALRADAR_API double **expectedAcquisitionTimeSec** (ScanPatternHandle ScanPattern, OCTDeviceHandle Dev)
- SPECTRALRADAR_API void **interpolateScanPattern** (DataHandle DataIn, DataHandle DataOut, InterpolationMethod method, BoundaryCondition condition)
- SPECTRALRADAR_API double **meanFreeformScanPatternPathLength** (DataHandle ScanPattern)
- SPECTRALRADAR_API BOOL checkAvailableMemoryForScanPattern (OCTDeviceHandle Dev, ScanPatternHandle Pattern, ptrdiff_t AdditionalMemory)

  Checks whether sufficient memory is available for acquiring the specified scan pattern.
- SPECTRALRADAR_API void **startMeasurePulseResponse** (OCTDeviceHandle Dev, double time_ms, double voltage, int *samplesPerChannel)

- SPECTRALRADAR_API void **getPulseResponseInput** (OCTDeviceHandle Dev, double *Raw)
- SPECTRALRADAR_API void **getPulseResponse** (OCTDeviceHandle Dev, double *RawX, double *RawY)
- SPECTRALRADAR_API void **stopMeasurePulseRespone** (OCTDeviceHandle Dev)
- SPECTRALRADAR_API void **startScanBenchmark** (OCTDeviceHandle Dev, double *RawX, double *RawY, int Size, double RateHz)
- SPECTRALRADAR_API void **stopScanBenchmark** (OCTDeviceHandle Dev)
- SPECTRALRADAR_API void **getScanFeedback** (OCTDeviceHandle Dev, double *RawX, double *RawY)
- SPECTRALRADAR_API int **getScanFeedbackSize** (OCTDeviceHandle Dev)
- SPECTRALRADAR_API double **QuantumEfficiency** (OCTDeviceHandle Dev, double CenterWavelength_nm, double PowerIntoSpectrometer_W, DataHandle Spectrum_e)
- SPECTRALRADAR_API FullRangeHandle **initFullRange** ()
- SPECTRALRADAR_API void **executeFullRange** (FullRangeHandle FullRange, DataHandle ApodizedDataIn, ComplexDataHandle ApodizedDataOut)
- SPECTRALRADAR_API void **closeFullRange** (FullRangeHandle FullRange)
- SPECTRALRADAR_API void **setFullRangeSensitivity** (FullRangeHandle FullRangeObject, float CutOff, float Smoothness)
- SPECTRALRADAR_API void **executeComplexProcessing** (ProcessingHandle Proc, ComplexDataHandle ApodizedSpectralData)
- SPECTRALRADAR_API void **determineThickness** (DataHandle Data, float *front, float *back)
- SPECTRALRADAR_API void **flattenImage** (DataHandle ImageData)
- SPECTRALRADAR_API void **determineSurface** (DataHandle Volume, DataHandle Surface)
- SPECTRALRADAR_API unsigned long long **getFreeMemory** ()
- SPECTRALRADAR_API void **absComplexData** (ComplexDataHandle ComplexData, DataHandle Abs)
- SPECTRALRADAR_API void **logAbsComplexData** (ComplexDataHandle ComplexData, DataHandle dB)
- SPECTRALRADAR_API void **argComplexData** (ComplexDataHandle ComplexData, DataHandle Arg)
- SPECTRALRADAR_API void **realComplexData** (ComplexDataHandle ComplexData, DataHandle Real)
- SPECTRALRADAR_API void **imagComplexData** (ComplexDataHandle ComplexData, DataHandle Imag)
- SPECTRALRADAR_API void **equalizeColoredDataHistogram** (ColoredDataHandle ColoredData)
- SPECTRALRADAR_API void **equalizeDataHistogram** (DataHandle Data, double Min, double Max)
- SPECTRALRADAR_API void medianFilter (DataHandle Data, int Rank)

  Computes a median filter on the specified 2D data.
- SPECTRALRADAR_API void pepperFilter (DataHandle Data, PepperFilterType Type, float Threshold)

  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 **polynomialFilter** (DataHandle Data, int SizeX, int SizeY)
- SPECTRALRADAR_API void **gaussianFilter** (DataHandle Data, GaussianFilterType Type)
- SPECTRALRADAR_API void **prewittFilter** (DataHandle Data, PrewittFilterType Type)
- SPECTRALRADAR_API void **sobelFilter** (DataHandle Data, SobelFilterType Type)
- SPECTRALRADAR_API void **laplacianFilter** (DataHandle Data, LaplacianFilterType Type)
- SPECTRALRADAR_API void **applyNonlinearSobelFilter2D** (DataHandle Data)
- SPECTRALRADAR_API void **applyNonlinearPrewittFilter2D** (DataHandle Data)
- SPECTRALRADAR_API void **applyConvolutionFilter2D** (DataHandle Data, int *Filter)
- SPECTRALRADAR_API void **applyMedianFilter2D_1x** (DataHandle Data, int Rank)
- SPECTRALRADAR_API void **applyFilter1D** (DataHandle Data, int *Size, float *FilterKernel, Direction FilterDirection=Direction_1, bool Normalization=false)
- SPECTRALRADAR_API void **applyFilter2D** (DataHandle Data, int *Size, float *FilterKernel, Direction FilterNormalDirection=Direction_3, bool Normalization=false)
- SPECTRALRADAR_API void **applyFilter3D** (DataHandle Data, int *Size, float *FilterKernel, bool Normalization=false)
- SPECTRALRADAR_API void **applyFilter** (ComplexDataHandle ComplexData, FilterType Type, double FilterParameter)
- SPECTRALRADAR_API void **normalizeFilter** (float *FilterKernel, int FilterSize)
- SPECTRALRADAR_API void **smoothCurve1D** (int Size, float *Curve, int DegreePolynom, int NumberPolynoms=1)

- SPECTRALRADAR_API float * **polynomialFitAndEval1D** (int Size, float *OrigPosX, float *OrigY, int DegreePolynom, int EvalSize, float *EvalPosX)
- SPECTRALRADAR_API float **calcParableMaximum** (float x0, float y0, float yp1, float ym1, float *peak_height=nullptr)
- SPECTRALRADAR_API void **thresholdData** (DataHandle Data, double Threshold)
- SPECTRALRADAR_API void levelData (DataHandle Data)

  Levels the specified data and removes tilt.
- SPECTRALRADAR_API void **importRealBinaryData** (DataHandle RealData, int Size1, int Size2, int Size3, const char *Path)
- SPECTRALRADAR_API void **filterDC** (DataHandle Data)
- SPECTRALRADAR_API void **crossCorrelatedProjection** (DataHandle DataIn, DataHandle Res)
- SPECTRALRADAR_API void **thresholdDataPtByPt** (DataHandle Phase, DataHandle Intensity, float threshold, float targetValue)
- SPECTRALRADAR_API void **analyzeScatteringSample** (DataHandle Data)
- SPECTRALRADAR_API void **getCurrentIntensityStatistics** (OCTDeviceHandle Dev, ProcessingHandle Proc, float *relToRefIntensity, float *relToProjAbsIntensity)
- SPECTRALRADAR_API void **getCurrentApodizationEdgeChannels** (ProcessingHandle Proc, int *LeftPix, int *RightPix)
- SPECTRALRADAR_API int **getNumberOfProbeConfigs** ()
- SPECTRALRADAR_API void **getProbeConfigName** (int index, char *ProbeName, int StringSize)
- SPECTRALRADAR_API PolarizationProcessingHandle **createPolarizationProcessing** (void)
- SPECTRALRADAR_API void **closePolarizationProcessing** (PolarizationProcessingHandle Polarization)
- SPECTRALRADAR_API void **setPolarizationPropertyInt** (PolarizationProcessingHandle Polarization, PolarizationPropertyInt Property, int Value)
- SPECTRALRADAR_API void **setPolarizationPropertyFloat** (PolarizationProcessingHandle Polarization, PolarizationPropertyFloat Property, float Value)
- SPECTRALRADAR_API void **setPolarizationFlag** (PolarizationProcessingHandle Polarization, PolarizationFlag Flag, BOOL OnOff)
- SPECTRALRADAR_API void **setPolarizationIntensityOutput** (PolarizationProcessingHandle Polarization, DataHandle AmpOut)
- SPECTRALRADAR_API void **setPolarizationRetardationOutput** (PolarizationProcessingHandle Polarization, DataHandle PhasesOut)
- SPECTRALRADAR_API void **executePolarizationProcessing** (PolarizationProcessingHandle Polarization, ComplexDataHandle SData, ComplexDataHandle PData)
- SPECTRALRADAR_API BOOL **initUSBProbeCtrl** (OCTDeviceHandle dev)
- SPECTRALRADAR_API BOOL **configureUSBProbeCtrlButton** (OCTDeviceHandle dev, USBProbeButtonID btn, USBProbeCommand cmd)
- SPECTRALRADAR_API BOOL **getLastUSBProbeMessage** (OCTDeviceHandle dev, char *msg, size_t size)
- SPECTRALRADAR_API BOOL **toggleUSBProbeLED** (OCTDeviceHandle Dev, int LED, BOOL OnOff)
- SPECTRALRADAR_API BOOL **refstageAvailable** (OCTDeviceHandle Dev)
- SPECTRALRADAR_API RefstageStatus **refstageGetStatus** (OCTDeviceHandle Dev)
- SPECTRALRADAR_API double **refstageGetLength_mm** (OCTDeviceHandle Dev, ProbeHandle Probe)
- SPECTRALRADAR_API double **refstageGetPosition_mm** (OCTDeviceHandle Dev, ProbeHandle Probe)
- SPECTRALRADAR_API void **refstageHome** (OCTDeviceHandle Dev, BOOL wait)
- SPECTRALRADAR_API void **refstageMoveLonger** (OCTDeviceHandle Dev)
- SPECTRALRADAR_API void **refstageMoveShorter** (OCTDeviceHandle Dev)
- SPECTRALRADAR_API void **refstageMoveAbsolute** (OCTDeviceHandle Dev, ProbeHandle Probe, double pos_mm)
- SPECTRALRADAR_API void **refstageStop** (OCTDeviceHandle Dev)
- SPECTRALRADAR_API void **refstageSetSpeed** (OCTDeviceHandle Dev, RefstageSpeed speed)
- SPECTRALRADAR_API void **refstageSetStatusCallback** (OCTDeviceHandle Dev, cbRefstageStatusChanged Callback)
- SPECTRALRADAR_API void **refstageSetPosChangeCallback** (OCTDeviceHandle Dev, cbRefstagePositionChanged Callback)

7.1.1 Detailed Description

Header containing all functions of the Spectral Radar SDK. This SDK can be used for Callisto, Ganymede, Hyperion and Telesto 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 MarkerListHandle

Handle to the marker list class.

7.1.3.3 VisualCalibrationHandle

Handle to the visual galvo calibration class.

7.1.4 Enumeration Type Documentation

7.1.4.1 enum Device_TriggerType

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.

7.1.4.2 enum ScanPattern_AcquisitionOrder

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.

7.1.4.3 enum ScanPatternPropertyInt

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.

7.1.5 Function Documentation

7.1.5.1 BOOL checkAvailableMemoryForScanPattern (OCTDeviceHandle *Dev,* ScanPatternHandle *Pattern,* ptrdiff_t *AdditionalMemory*)

Checks whether sufficient memory is available for acquiring 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 getConfigPath (char * *Path,* int *StrSize*)

Returns the path that hold the config files.

7.1.5.3 void getDopplerOutputSize (DopplerProcessingHandle *Doppler,* 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.

Doppler

7.1.5.4 void getInstallationPath (char * *Path,* int *StrSize*)

Returns the installation path.

7.1.5.5 int getNumberOfOutputValues (OCTDeviceHandle *Dev*)

Returns the number of output values.

7.1.5.6 void getOutputValueName (OCTDeviceHandle *Dev,* int *Index,* char * *Name,* int *NameStringSize,* char * *Unit,* int *UnitStringSize*)

Returns names and units of the requested output values.

7.1.5.7 void getOutputValueRangeByIndex (OCTDeviceHandle *Dev,* int *Index,* double * *Min,* double * *Max*)

Gives the range of the specified output value.

7.1.5.8 void getOutputValueRangeByName (OCTDeviceHandle *Dev,* const char * *Name,* double * *Min,* double * *Max*)

Gives the range of the specified output value.

7.1.5.9 void getPluginPath (char * *Path,* int *StrSize*)

Returns the path that hold the plugins.

7.1.5.10 double getReferenceIntensity (ProcessingHandle *Proc*)

Returns an absolute value that indicates the refernce intensity that was present when the currently used apodization was determined.

7.1.5.11 double double getRelativeReferenceIntensity (OCTDeviceHandle *Dev,* ProcessingHandle *Proc*)

Returns a value larger than 0.0 and smaller than 1.0 that indicates the refernce intensity that was present when the currently used apodization was determined.

7.1.5.12 Device_TriggerType getTriggerMode (OCTDeviceHandle *Dev*)

Returns the trigger mode used for acquisition.

7.1.5.13 double getTriggerTimeoutSec (OCTDeviceHandle *Dev*)

Returns the timeout of the camera (not used in trigger mode Trigger_FreeRunning).

7.1.5.14 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

7.1.5.15 bool isTriggerModeAvailable (OCTDeviceHandle *Dev,* Device_TriggerType *TriggerMode*)

Returns whether the specified trigger mode is possible or not for the used device.

7.1.5.16 void setOutputValueByIndex (OCTDeviceHandle *Dev,* int *Index,* double *Value*)

Sets the specified output value.

7.1.5.17 void setOutputValueByName (OCTDeviceHandle *Dev,* const char * *Name,* double *value*)

Sets the specified output value.