

SpectralRadar SDK

5.4

Generated by Doxygen 1.8.20

1 Spectral Radar SDK	1
1.1 SpectralRadar SDK License	1
1.2 Introduction	1
1.2.1 Overview	1
1.2.2 Data Handle (DataHandle, ColoredDataHandle, ComplexDataHandle, RawDataHandle)	1
1.2.3 OCTDeviceHandle	1
1.2.4 ProcessingHandle	2
1.2.5 ProbeHandle	2
1.2.6 ScanPatternHandle	2
1.2.7 Other Handles	3
1.3 First Steps	3
1.3.1 Initializing The Device	3
1.3.2 Creating Processing Routines	3
1.3.3 Creating A Scan Pattern	3
1.3.4 Acquisition	4
1.4 Error Handling	4
1.5 Examples	4
1.5.1 Simple	4
1.5.2 Advanced	6
2 Deprecated List	7
3 Module Index	8
3.1 Modules	8
4 Data Structure Index	9
4.1 Data Structures	9
5 File Index	9
5.1 File List	9
6 Module Documentation	9
6.1 Hardware	9
6.1.1 Detailed Description	14
6.1.2 Typedef Documentation	14
6.1.3 Enumeration Type Documentation	15
6.1.4 Function Documentation	21
6.2 Speckle Variance Contrast Processing	36
6.2.1 Detailed Description	36
6.2.2 Typedef Documentation	36
6.3 Error Handling	37
6.3.1 Detailed Description	37
6.3.2 Enumeration Type Documentation	37
6.3.3 Function Documentation	38

6.4 Data Access	40
6.4.1 Detailed Description	45
6.4.2 Typedef Documentation	45
6.4.3 Enumeration Type Documentation	46
6.4.4 Function Documentation	49
6.5 Data Creation and Clearing	72
6.5.1 Detailed Description	72
6.5.2 Function Documentation	72
6.6 Internal Values	75
6.6.1 Detailed Description	75
6.6.2 Function Documentation	75
6.7 Output Values (digital or analog)	78
6.7.1 Detailed Description	78
6.7.2 Function Documentation	78
6.8 Pattern Factory/Probe	81
6.8.1 Detailed Description	84
6.8.2 Typedef Documentation	84
6.8.3 Enumeration Type Documentation	85
6.8.4 Function Documentation	88
6.9 Scan Pattern	102
6.9.1 Detailed Description	105
6.9.2 Typedef Documentation	105
6.9.3 Enumeration Type Documentation	105
6.9.4 Function Documentation	107
6.10 Mathematical manipulations	123
6.10.1 Detailed Description	123
6.10.2 Enumeration Type Documentation	123
6.10.3 Function Documentation	124
6.11 Acquisition	127
6.11.1 Detailed Description	128
6.11.2 Enumeration Type Documentation	128
6.11.3 Function Documentation	128
6.12 Processing	135
6.12.1 Detailed Description	140
6.12.2 Typedef Documentation	140
6.12.3 Enumeration Type Documentation	140
6.12.4 Function Documentation	144
6.13 Export and Import	167
6.13.1 Detailed Description	168
6.13.2 Enumeration Type Documentation	168
6.13.3 Function Documentation	170
6.13.4 Variable Documentation	174

6.14 Volume	175
6.14.1 Detailed Description	177
6.14.2 Enumeration Type Documentation	177
6.14.3 Function Documentation	178
6.15 ProbeCalibration	198
6.16 Doppler	199
6.16.1 Detailed Description	200
6.16.2 Typedef Documentation	200
6.16.3 Enumeration Type Documentation	200
6.16.4 Function Documentation	201
6.17 Service	207
6.17.1 Detailed Description	207
6.17.2 Function Documentation	207
6.18 Settings	208
6.18.1 Detailed Description	208
6.18.2 Typedef Documentation	208
6.18.3 Function Documentation	209
6.19 Coloring	211
6.19.1 Detailed Description	212
6.19.2 Typedef Documentation	212
6.19.3 Enumeration Type Documentation	213
6.19.4 Function Documentation	214
6.20 Camera	217
6.20.1 Detailed Description	217
6.20.2 Function Documentation	217
6.21 Helper function	218
6.21.1 Detailed Description	218
6.21.2 Typedef Documentation	218
6.21.3 Function Documentation	219
6.22 Buffer	222
6.22.1 Detailed Description	222
6.22.2 Typedef Documentation	222
6.22.3 Function Documentation	222
6.23 File Handling	225
6.23.1 Detailed Description	230
6.23.2 Typedef Documentation	230
6.23.3 Enumeration Type Documentation	230
6.23.4 Function Documentation	235
6.24 External trigger	255
6.24.1 Detailed Description	255
6.24.2 Function Documentation	255
6.25 Post Processing	259

6.25.1 Detailed Description	260
6.25.2 Enumeration Type Documentation	260
6.25.3 Function Documentation	262
6.26 Polarization	270
6.26.1 Detailed Description	272
6.26.2 Typedef Documentation	272
6.26.3 Enumeration Type Documentation	272
6.26.4 Function Documentation	274
6.27 Polarization Adjustment	281
6.27.1 Detailed Description	281
6.27.2 Typedef Documentation	281
6.27.3 Function Documentation	281
6.28 Reference Intensity Control	284
6.28.1 Detailed Description	284
6.28.2 Typedef Documentation	284
6.28.3 Function Documentation	284
6.29 Amplification Control	287
6.29.1 Detailed Description	287
6.29.2 Function Documentation	287
6.30 General Information	289
6.30.1 Detailed Description	289
6.30.2 Function Documentation	289
7 Data Structure Documentation	290
7.1 ComplexFloat Struct Reference	290
7.1.1 Detailed Description	290
7.1.2 Field Documentation	290
8 File Documentation	290
8.1 SpectralRadar.h File Reference	290
8.1.1 Detailed Description	319
8.1.2 Macro Definition Documentation	319
8.1.3 Function Documentation	319
8.2 SpectralRadar.h	326
8.3 SpectralRadar_Handles.h File Reference	343
8.3.1 Detailed Description	343
8.4 SpectralRadar_Handles.h	344
8.5 SpectralRadar_Properties.h File Reference	344
8.5.1 Detailed Description	349
8.5.2 Enumeration Type Documentation	349
8.6 SpectralRadar_Properties.h	350
8.7 SpectralRadar_Types.h File Reference	353
8.7.1 Detailed Description	362

8.7.2 Macro Definition Documentation	362
8.7.3 Typedef Documentation	362
8.7.4 Enumeration Type Documentation	363
8.8 SpectralRadar_Types.h	363

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 via [getDataPropertyInt\(\)](#), [getDataPropertyFloat\(\)](#), etc. These include sizes along their first, second and third axis, physical spacing between pixels, their total range, etc.

1.2.3 OCTDeviceHandle

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

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

1.2.4 ProcessingHandle

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

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

1.2.5 ProbeHandle

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

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

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

1.2.6 ScanPatternHandle

A scan pattern is used to specify 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 resources can be freed using [clearScanPattern\(\)](#). The [ScanPatternHandle](#) needs to be specified to all functions that need information on the resulting scan. For example:

- creating a pattern ([createBScanPattern\(\)](#), [createVolumePattern\(\)](#), etc.)
- starting a measurement ([startMeasurement\(\)](#))

1.2.7 Other Handles

Other Handles that are used in the Spectral Radar SDK are

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

1.3 First Steps

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

1.3.1 Initializing The Device

The easiest way to initialize the device is to use the [initDevice\(\)](#) function. It returns an appropriate [OCTDeviceHandle](#) that can be used to identify 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 using a Fast Fourier transform and other pre- and postprocessing algorithms. To get a [ProcessingHandle](#) on these algorithms the most convenient way is to use the [createProcessingForDevice\(\)](#) functionality which requires a valid [OCTDeviceHandle](#):

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

1.3.3 Creating A Scan Pattern

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

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

1.3.4 Acquisition

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

```
int i;
RawDataHandle Raw = createRawData();
DataHandle BScan = createData();
startMeasurement(Dev, Pattern, Acquisition_ASyncContinuous);
for(i=0; i<1000; ++i) // get 1000 B-scans
{
    getRawData(Raw);
    setProcessedDataOutput(Proc, BScan);
    executeProcessing(Proc, Raw);
    // data is now in BScan...
    // do something with the data...
}
stopMeasurement(Dev);
clearData(BScan);
clearRawData(Raw);
```

1.4 Error Handling

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

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

1.5 Examples

The examples are meant to illustrate the usage of the SDK, and not the way to structure a program. As such, the examples contain almost no error checking or error handling code.

The code snippets shown here are part of a Microsoft Visual Studio project, located in directory SpectralRadar← Demos, which may be useful as a guideline to compile and link the code.

The SDK functions in the code are links that can be clicked to access deeper information.

1.5.1 Simple

The purpose of the simple examples is to demonstrate the usage of the basic building blocks of the SDK. Each code snippet focusses on a single concept.

1.5.1.1 B-Scan measurement

Acquire a simple B-Scan measurement (1024 A-Scans distributed along a segment 2 millimeter long).

```
void BScanMeasurement( void )
{
    // Initialization of device, probe and processing handles.
    OCTDeviceHandle Dev = initDevice();
    ProbeHandle Probe = initCurrentProbe(Dev);
    ProcessingHandle Proc = createProcessingForDevice(Dev);
    // The raw data handle will contain the unprocessed data from the detector
    // (e.g. line scan camera is SD-OCT systems) without any modification.
    RawDataHandle Raw = createRawData();
    // The data handle will be used for the processed data and will contain the OCT image.
    DataHandle BScan = createData();
    // Define a horizontal B-scan pattern with 2mm range and 1024 A-scans.
    ScanPatternHandle Pattern = createBScanPattern(Probe, 2.0, 1024);
    // Start the measurement to acquire the specified scan pattern once.
    startMeasurement(Dev, Pattern, Acquisition_AsyncFinite);
    // Grabs the spectral data from the framegrabber and copies it to the
    // previously created raw data handle.
    getRawData(Dev, Raw);
    // Specifies the output of the processing routine and executes the processing.
    setProcessedDataOutput(Proc, BScan);
    executeProcessing(Proc, Raw);
    // Stops the measurement.
    stopMeasurement(Dev);
    // Exports the processed data to a csv-file to the specified folder. Refer to the
    // companion example "ExportDataAndImage()" for further options regarding data export.
    exportData(BScan, DataExportFormat::DataExport_CSV, "C:\\\\test_oct_data.csv");
    // Clean up everything.
    clearScanPattern(Pattern);
    clearData(BScan);
    clearRawData(Raw);
    clearProcessing(Proc);
    closeProbe(Probe);
    closeDevice(Dev);
    // Check if an error occurred and write it to the console.
    char errorMessage[1024];
    if (getError(errorMessage, 1024))
        std::cout << "ERROR: " << errorMessage << std::endl;
    else
        std::cout << "SUCCESS." << std::endl;
    // Awaits a keystroke to return.
    std::system("PAUSE");
}
```

1.5.1.2 Export data and images

After a simple B-Scan measurement has been acquired, data are stored (for future post-processing) and exported (for beautiful pictures).

```
void ExportDataAndImage()
{
    char errorMessage[1024];
    // Initialization of device, probe and processing handles.
    OCTDeviceHandle Dev = initDevice();
    ProbeHandle Probe = initCurrentProbe(Dev);
    ProcessingHandle Proc = createProcessingForDevice(Dev);
    // The raw data handle will contain the unprocessed data from the detector
    // (e.g. line scan camera is SD-OCT systems) without any modification.
    RawDataHandle Raw = createRawData();
    // The data handle will be used for the processed data and will contain the OCT image.
    DataHandle BScan = createData();
    if (getError(errorMessage, 1024))
    {
        std::cout << "ERROR: " << errorMessage << std::endl;
        std::system("PAUSE");
        return;
    }
    // Define a horizontal B-scan pattern with 2mm range and 1024 A-scans.
    ScanPatternHandle Pattern = createBScanPattern(Probe, 2.0, 1024);
    // Start the measurement to acquire the specified scan pattern once.
    startMeasurement(Dev, Pattern, Acquisition_AsyncFinite);
    // Grabs the spectral data from the framegrabber and copies it to the
    // previously created raw data handle.
    getRawData(Dev, Raw);
    // Specifies the output of the processing routine and executes the processing.
    setProcessedDataOutput(Proc, BScan);
    executeProcessing(Proc, Raw);
    // Stops the measurement.
    stopMeasurement(Dev);
    // Exports the processed data to a csv-file to the specified folder. Different export
    // formats are available, refer to DataExportFormat for further details.
    exportData(BScan, DataExportFormat::DataExport_CSV, "C:\\\\test_oct_data.csv");
    // The OCT image can be exported as an image in common image format as well. It needs
    // to be colored beforehand, i.e. the colormap and boundaries for the coloring need to
    // be defined. In this example we use a very simple black and white color scheme and
```

```
// the RGBA (red, green, blue, alpha) coloring byte order.
ColoringHandle Coloring = createColoring32bit(ColorScheme_BlackAndWhite, Coloring_RGBA);
// Set the boundaries for the colormap, 0.0 as lower and 70.0 as upper boundary are
// normally a good choice.
setColoringBoundaries(Coloring, 0.0, 70.0);
// Exports the processed data to an image. The result is always a 2D-image (or a sequence
// of 2D-images, aka "slices", if volume data has been acquired). The slices are specified
// through a normal direction.
// To get the B-scan in one image with depth and scan field as axes for a single B-scan,
// Direction_3 is chosen.
exportDataAsImage(BScan, Coloring, ColoredDataExport_JPG,
    Direction_3, // Normal to the depth axis and to the scan axis
    "C:\\test_oct_image.jpg",
    ExportOption_DrawScaleBar | ExportOption_DrawMarkers |
    ExportOption_UsePhysicalAspectRatio);
// The unprocessed data from the detector in the raw data handle can be exported as well,
// in this case to a binary raw/srm file.
exportRawData(Raw, RawDataExportFormat::RawDataExport_RAW, "C:\\\\test_raw_data.raw");
// TODO: warum nicht .srm?
// Clean up everything.
clearScanPattern(Pattern);
clearData(BScan);
clearRawData(Raw);
clearProcessing(Proc);
closeProbe(Probe);
closeDevice(Dev);
// Check if an error occurred and write it to the console.
char errorMessage[1024];
if (getError(errorMessage, 1024))
    std::cout << "ERROR: " << errorMessage << std::endl;
else
    std::cout << "SUCCESS." << std::endl;
// Awaits a keystroke to return.
std::system("PAUSE");
}
```

1.5.2 Advanced

The purpose of the advanced examples is to show

- practical combinations of the building blocks,
- auxiliary operations that complement the acquisition and processing routines.

1.5.2.1 Load and read an OCT file Load and read the content of previously exported data.

```
// This code snippet shows how to read an oct-file with the SDK which has
// been acquired and saved with ThorImageOCT.
// Notice that the file may have been acquired with settings different to
// the current ones. In order to ensure that the right parameters will be
// used in any processing of the loaded data, it is necessary to use the
// functions specified for an OCTFile as in this example. These functions
// ensure that the settings and auxiliary files are taken from the dataset
// and not from the current settings of ThorImageOCT.
void ReadOCTfile( void )
{
    // Create a handle, that will group all necessary elements in memory.
    OCTfileHandle OCTfile = createOCTfile(FileFormat_OCITY);
    // Please provide the filename of an .oct-file you want to load. Notice that
    // a full path may be needed.
    loadFile(OCTfile, "Example.oct");
    OCTDeviceHandle Dev = initDevice();
    ProbeHandle Probe = initProbeFromOCTfile(Dev, OCTfile);
    ProcessingHandle Proc = createProcessingForOCTfile(OCTfile);
    // The specific information of the loaded dataset can be accessed through
    // the metadata.
    double RangeX = getFileMetadataFloat(OCTfile, FileMetadata_RangeX);
    // Extract further parameters...
}
```

To find out other metadata of interest, click on the function and explore other functions in the same group.

1.5.2.2 Write an OCT file Write the results from a measurement to an OCT file.

```
// This example program shows how to write data acquired with the SDK to an OCT-file which
// can be viewed with ThorImageOCT.
void WriteOCTFile( void )
{
    // Initialization of device, probe and processing handles.
    OCTDeviceHandle Dev = initDevice();
    ProbeHandle Probe = initCurrentProbe(Dev);
    ProcessingHandle Proc = createProcessingForDevice(Dev);
    // The raw data handle will contain the unprocessed data from the detector
    // (e.g. line scan camera is SD-OCT systems) without any modification.
    RawDataHandle Raw = createRawData();
    // The data handle will be used for the processed data and will contain the OCT image.
    DataHandle BScan = createData();
    // The colored data handle will contain the picture acquired with the video camera.
    ColoredDataHandle VideoImg = createColoredData();
    // Creating the pattern with no range in the second direction puts all B-scans
    // at the same position.
    ScanPatternHandle Pattern = createBScanPattern(Probe, 2, 1024);
    // Start the measurement to acquire the specified scan pattern once.
    startMeasurement(Dev, Pattern, Acquisition_AsyncFinite);

    // Grabs the spectral data from the framegrabber and copies it to the
    // previously created raw data handle.
    getRawData(Dev, Raw);
    // Grabs the picture of the video camera (visible light).
    getCameraImage(Dev, VideoImg);
    // Specifies the output of the processing routine and executes the processing.
    setProcessedDataOutput(Proc, BScan);
    executeProcessing(Proc, Raw);
    // Stops the measurement.
    stopMeasurement(Dev);
    // Creating the file in the correct data format for ThorImageOCT
    OCTFileHandle OCTfile = createOCTFile(FileFormat_OCITY);
    // Adding the processed data to the file. Click on the third argument to
    // find out alternative data types that can be stored.
    addFileRealData(OCTfile, BScan, DataObjectName_OCTData);

    // Save video camera image.
    addFileColoredData(OCTfile, VideoImg, DataObjectName_VideoImage);
    // In order to be able to open the file in ThorImageOCT, a suitable mode has
    // to be defined. Click on the third argument to find out other acquisition modes,
    // or look in "SpectralRadar.h" for keywords starting with "AcquisitionMode_".
    // Please note that not every available acquisitions mode is supported by every hardware.
    setFileMetadataString(OCTfile, FileMetadata_AcquisitionMode, AcquisitionMode_2D);

    // The data from the device, probe, processing and scan pattern will be saved as well
    saveFileMetadata(OCTfile, Dev, Proc, Probe, Pattern);
    saveFile(OCTfile, "AcquiredOCTfileWithSDK.oct");
    // Clean up everything.
    clearOCTfile(OCTfile);
    clearScanPattern(Pattern);
    clearColoredData(VideoImg);
    clearData(BScan);
    clearRawData(Raw);
    clearProcessing(Proc);
    closeProbe(Probe);
    closeDevice(Dev);
    // Check if an error occurred and write it to the console.
    char errorMessage[1024];
    if (getError(errorMessage, 1024))
        std::cout << "ERROR: " << errorMessage << std::endl;
    else
        std::cout << "SUCCESS." << std::endl;
    // Awaits a keystroke to return.
    std::system("PAUSE");
}
```

2 Deprecated List

Global `setInternalDeviceValueByIndex` (`OCTDeviceHandle Dev, int Index, double Value`)

Please use [setOutputDeviceValueByIndex](#) instead.

3 Module Index

3.1 Modules

Here is a list of all modules:

Hardware	9
Speckle Variance Contrast Processing	36
Error Handling	37
Data Access	40
Data Creation and Clearing	72
Internal Values	75
Output Values (digital or analog)	78
Pattern Factory/Probe	81
Scan Pattern	102
Mathematical manipulations	123
Acquisition	127
Processing	135
Export and Import	167
Volume	175
ProbeCalibration	198
Doppler	199
Service	207
Settings	208
Coloring	211
Camera	217
Helper function	218
Buffer	222
File Handling	225
External trigger	255
Post Processing	259
Polarization	270
Polarization Adjustment	281
Reference Intensity Control	284

Amplification Control	287
General Information	289

4 Data Structure Index

4.1 Data Structures

Here are the data structures with brief descriptions:

ComplexFloat A standard complex data type that is used to access complex data	290
---	-----

5 File Index

5.1 File List

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

SpectralRadar.h Header containing all functions of the Spectral Radar SDK. This SDK can be used for Callisto, Ganymede, Hyperion, Telesto and Vega devices	290
SpectralRadar_Handles.h Header containing all handles of the SDK. When including SpectralRadar.h this file will automatically be included. 343	
SpectralRadar_Properties.h Header containing all property enums of the SDK. When including SpectralRadar.h this file will automatically be included. 344	
SpectralRadar_Types.h Header containing all types of the SDK. When including SpectralRadar.h this file will automatically be included. 353	

6 Module Documentation

6.1 Hardware

Functions providing direct access to OCT Hardware functionality.

Typedefs

- **typedef void(__stdcall * cbRefstageStatusChanged) (RefstageStatus)**
Defines the function prototype for the reference stage status callback (see also [setRefstageStatusCallback\(\)](#)). The argument contains the current status of the reference stage when called.
- **typedef void(__stdcall * cbRefstagePositionChanged) (double)**
Defines the function prototype for the reference stage position change callback (see also [setRefstagePosChangedCallback\(\)](#)). The argument contains the reference stage position in mm when called.
- **typedef void(__stdcall * genericProgressCallback) (double progress, const char *msg)**
Definition for a generic callback function indicating progress. The argument will be passed a value from 0.0 to 1.0 indicating progress in a respective process.
- **typedef void(__stdcall * lightSourceStateCallback) (LightSourceState)**
Defines the function prototype for the light source callback (see also [setLightSourceTimeoutCallback\(\)](#)). The argument contains the current state of the light source.
- **typedef struct C_OCTDevice * OCTDeviceHandle**
The OCTDeviceHandle type is used as Handle for using the SpectralRadar.

Enumerations

- **enum DevicePropertyFloat {**
Device_FullWellCapacity,
Device_zSpacing,
Device_zRange,
Device_SignalAmplitudeMin_dB,
Device_SignalAmplitudeLow_dB,
Device_SignalAmplitudeHigh_dB,
Device_SignalAmplitudeMax_dB,
Device_BinToElectronScaling,
Device_Temperature,
Device_SLD_OnTime_sec,
Device_CenterWavelength_nm,
Device_SpectralWidth_nm,
Device_MaxTriggerFrequency_Hz,
Device_LineRate_Hz }
Floating point properties of the device that can be retrieved with the function [getDevicePropertyFloat](#).
- **enum DevicePropertyInt {**
Device_SpectrumElements,
Device_BytesPerElement,
Device_MaxLiveVolumeRenderingScans,
Device_BitDepth,
Device_NumOfCameras,
Device_RevisionNumber,
Device_NumOfAnalogInputChannels,
Device_MinimumSpectraPerBuffer **}**
Integer properties of the device that can be retrieved with the function [getDevicePropertyInt](#).
- **enum DevicePropertyString {**
Device_Type,
Device_Series,
Device_SerialNumber,
Device_HardwareConfig **}**
String-properties of the device that can be retrieved with the function [getDevicePropertyString](#).
- **enum DeviceFlag {**
Device_On = 0,
Device_CameraAvailable = 1,
Device_SLDAvailable = 2,

```
Device_SLDStatus = 3,
Device_LaserDiodeStatus = 4,
Device_CameraShowScanPattern = 5,
Device_ProbeControllerAvailable = 6,
Device_DataIsSigned = 7,
Device_IsSweptSource = 8,
Device_AnalogInputAvailable = 9,
Device_HasMasterBoard = 10,
Device_HasControlBoard = 11,
Device_SLDStatusQuick = 12 }
```

Boolean properties of the device that can be retrieved with the function [getDeviceFlag](#).

- enum **StaticDeviceFlag** {
 Device_PowerControlAvailable = 0,
 Device_PowerOn = 1 }

Boolean properties of the device that can be queried before the device is initialized. Retrieved with the function [getStaticDeviceFlag](#).

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

Axis selection for the function [moveScanner](#).

- enum **DeviceState** {
 DeviceState_Unavailable = 0,
 DeviceState_Standby = 1,
 DeviceState_Enabled = 2,
 DeviceState_NoConfiguration = 3,
 DeviceState_Error = 4,
 DeviceState_Startup = 5 }

Results for function [getDeviceState](#).

- enum **DeviceTriggerType** {
 Trigger_FreeRunning,
 Trigger_TrigBoard_ExternalStart,
 Trigger_External_AScan }

Enum identifying trigger types for the OCT system.

- enum **DeviceTriggerIOType** {
 TriggerIO_Disabled,
 TriggerIO_Output,
 TriggerIO_Input }

Enum identifying trigger types for the secondary trigger IO channel.

- enum **RefstageStatus** {
 RefStage_Status_Idle = 0,
 RefStage_Status_Homing = 1,
 RefStage_Status_Moving = 2,
 RefStage_Status_MovingTo = 3,
 RefStage_Status_Stopping = 4,
 RefStage_Status_NotAvailable = 5,
 RefStage_Status_Undefined = -1 }

Defines the status of the motorized reference stage.

- enum **RefstageSpeed** {
 RefStage_Speed_Slow = 0,
 RefStage_Speed_Fast = 1,
 RefStage_Speed_VerySlow = 2,
 RefStage_Speed_VeryFast = 3 }

Defines the velocity of movement for the motorized reference stage.

- enum **RefstageWaitForMovement** {
 RefStage_Movement_Wait = 0,
 RefStage_Movement_Continue = 1 }

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

- enum RefstageMovementDirection {
 RefStage_MoveShorter = 0,
 RefStage_MoveLonger = 1 }

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

- enum WaitForCompletion {
 Wait = 0,
 Continue = 1 }

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

- enum SpectrumDirectionType {
 LongToShortWavelengths,
 ShortToLongWavelengths,
 UnknownSpectrumDirection }

Describes the orientation of the spectrometer, i.e., if the first pixels correspond to longer or shorter wavelengths.

- enum LightSourceState {
 Activating,
 On,
 Off }

Values that define the state of the light source.

Functions

- **SPECTRALRADAR_API OCTDeviceHandle initDevice (void)**
Initializes the installed device.
- **SPECTRALRADAR_API int getDevicePropertyInt (OCTDeviceHandle Dev, DevicePropertyInt Selection)**
Returns properties of the device belonging to the specified OCTDeviceHandle.
- **SPECTRALRADAR_API const char * getDevicePropertyString (OCTDeviceHandle Dev, DevicePropertyString Selection)**
Returns properties of the device belonging to the specified OCTDeviceHandle.
- **SPECTRALRADAR_API double getDevicePropertyFloat (OCTDeviceHandle Dev, DevicePropertyFloat Selection)**
Returns properties of the device belonging to the specified OCTDeviceHandle.
- **SPECTRALRADAR_API BOOL getDeviceFlag (OCTDeviceHandle Dev, DeviceFlag Selection)**
Returns properties of the device belonging to the specified OCTDeviceHandle.
- **SPECTRALRADAR_API void setDeviceFlag (OCTDeviceHandle Dev, DeviceFlag Selection, BOOL Value)**
Sets the selected flag of the device belonging to the specified OCTDeviceHandle.
- **SPECTRALRADAR_API BOOL getStaticDeviceFlag (StaticDeviceFlag Selection)**
Returns properties of the device available before device initialization.
- **SPECTRALRADAR_API void setStaticDeviceFlag (StaticDeviceFlag Selection, BOOL Value)**
Sets the selected flag of the device available before device initialization.
- **SPECTRALRADAR_API void closeDevice (OCTDeviceHandle Dev)**
Closes the device opened previously with initDevice.
- **SPECTRALRADAR_API void moveScanner (OCTDeviceHandle Dev, ProbeHandle Probe, ScanAxis Axis, double Position_mm)**
Manually moves the scanner to a given position.
- **SPECTRALRADAR_API void moveScannerToApoPosition (OCTDeviceHandle Dev, ProbeHandle Probe)**
Moves the scanner to the apodization position.
- **SPECTRALRADAR_API int getNumberOfDevicePresetCategories (OCTDeviceHandle Dev)**
If the hardware supports multiple presets, the function returns the number of categories in which presets can be set.
- **SPECTRALRADAR_API const char * getDevicePresetCategoryName (OCTDeviceHandle Dev, int Category)**

- Gets a descriptor/name for the respective preset category.
SPECTRALRADAR_API int getDevicePresetCategoryIndex (OCTDeviceHandle Dev, const char *Name)
Gets the index of a preset category from the name of the category.
- **SPECTRALRADAR_API** void setDevicePreset (OCTDeviceHandle Dev, int Category, ProbeHandle Probe, ProcessingHandle Proc, int Preset)
Sets the preset of the device. Using presets the sensitivity and acquisition speed of the device can be influenced.
- **SPECTRALRADAR_API** int getDevicePreset (OCTDeviceHandle Dev, int Category)
Gets the currently used device preset.
- **SPECTRALRADAR_API** const char * getDevicePresetDescription (OCTDeviceHandle Dev, int Category, int Preset)
Returns a description of the selected device preset. Using the description more information about sensitivity and acquisition speed of the respective set can be found.
- **SPECTRALRADAR_API** int getNumberOfDevicePresets (OCTDeviceHandle Dev, int Category)
Returns the number of available device presets.
- **SPECTRALRADAR_API** void setRequiredSLDOnTime_s (int Time_s)
Sets the time the SLD needs to be switched on before any measurement can be started. Default is 3 seconds.
- **SPECTRALRADAR_API** void resetCamera (void)
Resets the spectrometer camera.
- **SPECTRALRADAR_API** BOOL isDeviceAvailable (void)
Returns whether any supported Base-Unit is available.
- **SPECTRALRADAR_API** DeviceState getDeviceState (void)
Returns the state of supported base-unit.
- **SPECTRALRADAR_API** void getDeviceError (char *ErrorMsg, int StringSize)
Returns the error message given by the hardware.
- **SPECTRALRADAR_API** double QuantumEfficiency (OCTDeviceHandle Dev, double CenterWavelength_nm, double PowerIntoSpectrometer_W, DataHandle Spectrum_e)
Calculates the quantum efficiency from the processed input spectrum in the Data instance.
- **SPECTRALRADAR_API** BOOL isRefstageAvailable (OCTDeviceHandle Dev)
Returns whether a motorized reference stage is available or not for the specified device. Please note that a motorized reference stage is not included in all systems.
- **SPECTRALRADAR_API** RefstageStatus getRefstageStatus (OCTDeviceHandle Dev)
Returns the current status of the reference stage, e.g. if it is moving.
- **SPECTRALRADAR_API** double getRefstageLength_mm (OCTDeviceHandle Dev, ProbeHandle Probe)
Returns the total length in mm of the reference stage.
- **SPECTRALRADAR_API** double getRefstagePosition_mm (OCTDeviceHandle Dev, ProbeHandle Probe)
Returns the current position in mm of the reference stage.
- **SPECTRALRADAR_API** void homeRefstage (OCTDeviceHandle Dev, RefstageWaitForMovement WaitForMoving)
Homes the reference stage to calibrate the zero position.
- **SPECTRALRADAR_API** void moveRefstageToPosition_mm (OCTDeviceHandle Dev, ProbeHandle Probe, double Pos_mm, RefstageSpeed Speed, RefstageWaitForMovement WaitForMoving)
Moves the reference stage to the specified position in mm.
- **SPECTRALRADAR_API** void moveRefstage_mm (OCTDeviceHandle Dev, ProbeHandle Probe, double Length_mm, RefstageMovementDirection Direction, RefstageSpeed Speed, RefstageWaitForMovement WaitForMoving)
Moves the reference stage with the specified length in mm.
- **SPECTRALRADAR_API** void startRefstageMovement (OCTDeviceHandle Dev, RefstageMovementDirection Direction, RefstageSpeed Speed)
Starts the movement of the reference stage with the chosen speed. Please note that the movement does not stop until stopRefstageMovement is called.
- **SPECTRALRADAR_API** void stopRefstageMovement (OCTDeviceHandle Dev)
Stops the movement of the reference stage.

- **SPECTRALRADAR_API** void `setRefstageSpeed (OCTDeviceHandle Dev, RefstageSpeed Speed)`
Sets the velocity of the movement of the reference stage.
- **SPECTRALRADAR_API** void `setRefstageStatusCallback (OCTDeviceHandle Dev, cbRefstageStatusChanged Callback)`
Registers the callback to get notified if the reference stage status changed.
- **SPECTRALRADAR_API** void `setRefstagePosChangedCallback (OCTDeviceHandle Dev, cbRefstagePositionChanged Callback)`
Registers the callback to get notified if the reference stage position changed.
- **SPECTRALRADAR_API** double `getRefstageMinPosition_mm (OCTDeviceHandle Dev, ProbeHandle Probe)`
Returns the minimal position in mm the reference stage can move to.
- **SPECTRALRADAR_API** double `getRefstageMaxPosition_mm (OCTDeviceHandle Dev, ProbeHandle Probe)`
Returns the maximal position in mm the reference stage can move to.
- **SPECTRALRADAR_API** void `setLightSourceTimeoutCallback (OCTDeviceHandle Dev, lightSourceStateCallback Callback)`
Sets a callback function that will be invoked by the SDK whenever the state of the lightsource of the device changes.
- **SPECTRALRADAR_API** void `setLightSourceTimeout_s (OCTDeviceHandle Dev, double Timeout)`
Sets a the timeout in seconds, after which the OCT lightsource will be turned off if no scanning is performed.
- **SPECTRALRADAR_API** double `getLightSourceTimeout_s (OCTDeviceHandle Dev)`
Gets a the timeout in seconds, after which the OCT lightsource will be turned off if no scanning is performed.
- **SPECTRALRADAR_API** void `updateAfterPresetChange (OCTDeviceHandle Dev, ProbeHandle Probe, ProcessingHandle Proc, int CameralIndex)`
Updates the processing handle after preset change. Please use `setDevicePreset` first for the first camera (with index 0) and this function to update the corresponding `ProcessingHandle` for the second camera (with index 1).

6.1.1 Detailed Description

Functions providing direct access to OCT Hardware functionality.

6.1.2 Typedef Documentation

6.1.2.1 cbRefstagePositionChanged `typedef void(__stdcall* cbRefstagePositionChanged) (double)`

Defines the function prototype for the reference stage position change callback (see also `setRefstagePosChangedCallback()`). The argument contains the reference stage position in mm when called.

Parameters

<code>double</code>	Current position of the reference stage in mm
---------------------	---

Definition at line 1080 of file [SpectralRadar_Types.h](#).

6.1.2.2 cbRefstageStatusChanged `typedef void(__stdcall* cbRefstageStatusChanged) (RefstageStatus)`

Defines the function prototype for the reference stage status callback (see also `setRefstageStatusCallback()`). The argument contains the current status of the reference stage when called.

Parameters

<i>RefstageStatus</i>	Current status of the reference stage
-----------------------	---------------------------------------

Definition at line 1075 of file [SpectralRadar_Types.h](#).

6.1.2.3 genericProgressCallback `typedef void(__stdcall* genericProgressCallback) (double progress, const char *msg)`

Definition for a generic callback function indicating progress. The argument will be passed a value from 0.0 to 1.0 indicating progress in a respective process.

Parameters

<i>progress</i>	Current progress in a range from 0.0 to 1.0
<i>msg</i>	given an additional message for the current progress

Definition at line 1099 of file [SpectralRadar_Types.h](#).

6.1.2.4 lightSourceStateCallback `typedef void(__stdcall* lightSourceStateCallback) (LightSourceState)`

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

Parameters

<i>LightSourceState</i>	Current state of the light source
-------------------------	-----------------------------------

Definition at line 1117 of file [SpectralRadar_Types.h](#).

6.1.2.5 OCTDeviceHandle `OCTDeviceHandle`

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

Definition at line 74 of file [SpectralRadar_Handles.h](#).

6.1.3 Enumeration Type Documentation

6.1.3.1 DeviceFlag `enum DeviceFlag`

Boolean properties of the device that can be retrieved with the function [getDeviceFlag](#).

Enumerator

Device_On	The type name of the device.
Device_CameraAvailable	Specifies if there is a video camera available.
Device_SLDAvailable	Specifies if there is a SLD available.
Device_SLDStatus	Status of the SLD, either on (true) or off (false)
Device_LaserDiodeStatus	Status of the laser diode, either on (true) or off (false) Warning Not all devices are equipped with this feature.
Device_CameraShowScanPattern	Parameter for the overlay of the video camera which shows the scan pattern in red.
Device_ProbeControllerAvailable	Gives information whether a probe controller (with buttons) is available.
Device_DatasSigned	Flag indicating if the data is signed.
Device_IsSweptSource	Flag indicating whether system is a swept source (or spectral domain) system.
Device_AnalogInputAvailable	Flag indicating if the system offers analog input capabilities.
Device_HasMasterBoard	Flag indicating if the hardware contains a control board.
Device_HasControlBoard	Flag indicating if the hardware contains a control board. A system may contain either a master or a control board, but not both.
Device_SLDStatusQuick	SLD activation without delay.

Definition at line 168 of file [SpectralRadar_Properties.h](#).

6.1.3.2 DevicePropertyFloat enum DevicePropertyFloat

Floating point properties of the device that can be retrieved with the function [getDevicePropertyFloat](#).

Enumerator

Device_FullWellCapacity	The full well capacity of the device.
Device_zSpacing	The spacing between two pixels in an A-scan.
Device_zRange	The maximum measurement range for an A-scan.
Device_SignalAmplitudeMin_db	The minimum expected dB value for final data.
Device_SignalAmplitudeLow_db	The typical low dB value for final data.
Device_SignalAmplitudeHigh_db	The typical high dB value for final data.
Device_SignalAmplitudeMax_db	The maximum expected dB value for final data.
Device_BinToElectronScaling	Scaling factor between binary raw data and electrons/photons.
Device_Temperature	Internal device temperature in degrees C.
Device_SLD_OnTime_sec	Absolute power-on time of the SLD since first start in seconds.
Device_CenterWavelength_nm	The center wavelength of the device.
Device_SpectralWidth_nm	The approximate spectral width of the spectrometer.
Device_MaxTriggerFrequency_Hz	Maximal valid trigger frequency depending on the chosen camera preset.

Definition at line 96 of file [SpectralRadar_Properties.h](#).

6.1.3.3 DevicePropertyInt enum `DevicePropertyInt`

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

Enumerator

<code>Device_SpectrumElements</code>	The number of pixels provided by the spectrometer.
<code>Device_BytesPerElement</code>	The number of bytes one element of the spectrum occupies.
<code>Device_MaxLiveVolumeRenderingScans</code>	The maximum number of scans per dimension in the live volume rendering mode.
<code>Device_BitDepth</code>	Bit depth of the DAQ.
<code>Device_NumOfCameras</code>	Number of spectrometer cameras.
<code>Device_RevisionNumber</code>	Revision number of the device.
<code>Device_NumOfAnalogInputChannels</code>	Number of analog input channels available (may be zero)
<code>Device_MinimumSpectraPerBuffer</code>	Minimum number of spectra per buffer (only applies in continuous mode and when using <code>measureSpectraContinuousEx</code>)

Definition at line 131 of file `SpectralRadar_Properties.h`.

6.1.3.4 DevicePropertyString enum `DevicePropertyString`

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

Enumerator

<code>Device_Type</code>	The type name of the device.
<code>Device_Series</code>	The series of the device.
<code>Device_SerialNumber</code>	Serial number of the device.
<code>Device_HardwareConfig</code>	Hardware Config of the currently used device.

Definition at line 153 of file `SpectralRadar_Properties.h`.

6.1.3.5 DeviceState enum `DeviceState`

Results for function `getDeviceState`.

Enumerator

<code>DeviceState_Unavailable</code>	Device is not available (e.g. not connected, no power)
<code>DeviceState_Standby</code>	Device is available, but in standby mode (can be activated via software)
<code>DeviceState_Enabled</code>	Device is enabled.
<code>DeviceState_NoConfiguration</code>	No configuration files found and could not be restored from device.
<code>DeviceState_Error</code>	The device has an error that the user should solve.
<code>DeviceState_Startup</code>	The device starts working and it is not yet ready for normal operation.

Definition at line 154 of file [SpectralRadar_Types.h](#).

6.1.3.6 DeviceTriggerIOType `enum DeviceTriggerIOType`

Enum identifying trigger types for the secondary trigger IO channel.

Warning

Not all systems support trigger IO. To check if the system supports trigger IO, please use [isTriggerIOModeAvailable](#)

Enumerator

TriggerIO_Disabled	Do not use trigger IO.
TriggerIO_Output	Output mode.
TriggerIO_Input	Input mode (uses trigger IO to start individual scans)

Definition at line 867 of file [SpectralRadar_Types.h](#).

6.1.3.7 DeviceTriggerType `enum DeviceTriggerType`

Enum identifying trigger types for the OCT system.

Warning

Not all trigger types are available for all different systems. To check whether the specified trigger mode is available or not please use [isTriggerModeAvailable](#)

Enumerator

Trigger_FreeRunning	Standard mode.
Trigger_TrigBoard_ExternalStart	Used to trigger the start of an acquisition. Additional hardware is needed.
Trigger_External_AScan	Mode to trigger the acquisition of each A-scan. An external trigger signal is needed. Please see the software manual for detailed information.

Definition at line 854 of file [SpectralRadar_Types.h](#).

6.1.3.8 LightSourceState `enum LightSourceState`

Values that define the state of the light source.

Enumerator

Activating	Light source is currently activating; state should change to on in a couple of seconds.
On	Light source is on.
Off	Light source is off.

Definition at line 1104 of file [SpectralRadar_Types.h](#).

6.1.3.9 RefstageMovementDirection `enum RefstageMovementDirection`

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

Enumerator

RefStage_MoveShorter	Shortens reference arm length.
RefStage_MoveLonger	Extends reference arm length.

Definition at line 1014 of file [SpectralRadar_Types.h](#).

6.1.3.10 RefstageSpeed `enum RefstageSpeed`

Defines the velocity of movement for the motorized reference stage.

Enumerator

RefStage_Speed_Slow	Slow speed (~0.4mm/s)
RefStage_Speed_Fast	Fast speed (~1.8mm/s)
RefStage_Speed_VerySlow	Very slow speed.
RefStage_Speed_VeryFast	Very fast speed (~13mm/s)

Definition at line 988 of file [SpectralRadar_Types.h](#).

6.1.3.11 RefstageStatus `enum RefstageStatus`

Defines the status of the motorized reference stage.

Enumerator

RefStage_Status_Idle	The reference stage is not busy and available for a task.
RefStage_Status_Homing	The reference stage is in its homing process. Please wait until this process is finished.
RefStage_Status_Moving	The reference stage is moving, you can stop this movement with stopRefstageMovement .
RefStage_Status_MovingTo	The reference stage it moving to a certain position. Please wait until this process is finished.
RefStage_Status_Stopping	The reference stage is in the stopping process after stopRefstageMovement was called. Please wait until this process is finished.
RefStage_Status_NotAvailable	The reference stage is not available any more.
RefStage_Status_Undefined	The status of the reference stage is not defined.

Definition at line 967 of file [SpectralRadar_Types.h](#).

6.1.3.12 RefstageWaitForMovement enum RefstageWaitForMovement

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

Enumerator

RefStage_Movement_Wait	Function waits until the movement has stopped before it returns.
RefStage_Movement_Continue	The movement of the motorized reference stage will be started and runs in another thread. The function returns while the reference stage is still moving.

Definition at line 1003 of file [SpectralRadar_Types.h](#).

6.1.3.13 ScanAxis enum ScanAxis

Axis selection for the function [moveScanner](#).

Enumerator

ScanAxis↔_X	X-Axis of the scanner.
ScanAxis↔_Y	Y-Axis of the scanner.

Definition at line 144 of file [SpectralRadar_Types.h](#).

6.1.3.14 SpectrumDirectionType enum SpectrumDirectionType

Describes the orientation of the spectrometer, i.e., if the first pixels correspond to longer or shorter wavelengths.

Warning

Experimental: This enum may change in future versions of the SDK.

Enumerator

LongToShortWavelengths	Pixels with lower index have longer wavelengths; pixels with higher index have shorter wavelengths.
ShortToLongWavelengths	Pixels with lower index have shorter wavelengths; pixels with higher index have longer wavelengths.
UnknownSpectrumDirection	Spectrum orientation is not known.

Definition at line 1062 of file [SpectralRadar_Types.h](#).

6.1.3.15 StaticDeviceFlag enum [StaticDeviceFlag](#)

Boolean properties of the device that can be queried before the device is initialized. Retrieved with the function [getStaticDeviceFlag](#).

Enumerator

Device_PowerControlAvailable	Read-only flag indicating if the system can be turned on/off using setStaticDeviceFlag with the flag Device_PowerOn .
Device_PowerOn	Write-only flag to enable or disable the device power supply.

Definition at line 201 of file [SpectralRadar_Properties.h](#).

6.1.3.16 WaitForCompletion enum [WaitForCompletion](#)

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

Enumerator

Wait	Wait for completion.
Continue	Do not wait for completion.

Definition at line 1040 of file [SpectralRadar_Types.h](#).

6.1.4 Function Documentation

6.1.4.1 closeDevice() void [closeDevice](#) ([OCTDeviceHandle](#) Dev)

Closes the device opened previously with [initDevice](#).

Parameters

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

6.1.4.2 getDeviceError() void [getDeviceError](#) (

```
    char * ErrorMsg,  
    int StringSize )
```

Returns the error message given by the hardware.

Parameters

out	<i>ErrorMsg</i>	A (preallocated) place holder for the error message.
in	<i>StringSize</i>	The maximum space available to write the error description.

It is assumed, but not mandatory, that the user previously invoked the [getDeviceState\(\)](#) function. If there are no errors, nothing will be done (the user may want to write a default answer to catch this case). This function can be invoked as many times as desired (e.g. in a polling strategy) without side effects.

```
6.1.4.3 getDeviceFlag() BOOL getDeviceFlag (   
    OCTDeviceHandle Dev,  
    DeviceFlag Selection )
```

Returns properties of the device belonging to the specified [OCTDeviceHandle](#).

Parameters

in	<i>Selection</i>	The desired flag.
----	------------------	-------------------

Returns

The value of the desired flag.

Parameters

in	<i>Dev</i>	A valid (non null) OCT device handle (OCTDeviceHandle), previously generated with the function initDevice .
----	------------	---

```
6.1.4.4 getDevicePreset() int getDevicePreset (   
    OCTDeviceHandle Dev,  
    int Category )
```

Gets the currently used device preset.

Returns

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

A description of the device preset associated with a particular index can be obtained by invoking the function [getDevicePresetDescription](#). The total number of presets for the active device can be retrieved with the function [getNumberOfDevicePresets](#).

Parameters

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

```
6.1.4.5 getDevicePresetCategoryIndex() const char * getDevicePresetCategoryIndex (
    OCTDeviceHandle Dev,
    const char * Name )
```

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

Parameters

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

Returns

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

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

```
6.1.4.6 getDevicePresetCategoryName() const char * getDevicePresetCategoryName (
    OCTDeviceHandle Dev,
    int Category )
```

Gets a descriptor/name for the respective preset category.

Parameters

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

Returns

The name of the requested device preset category.

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

```
6.1.4.7 getDevicePresetDescription() const char * getDevicePresetDescription (
    OCTDeviceHandle Dev,
    int Category,
    int Preset )
```

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

Returns

A text describing the preset (speed, sensitivity). This pointer refers to memory owned by SpectralRadar.dll. The user should not attempt to free it. Different devices support different preset categories (gain, speed, etc). When getting or setting a preset, the right category must be provided. To get the number of supported categories, use the function [getNumberOfDevicePresetCategories](#). To get a name (i.e. a description of the category), use the function [getDevicePresetCategoryName](#). To get the index of a supported category, provided you know the name, use the function [getDevicePresetCategoryIndex](#) (this is the index need when getting or setting a preset of a given category).

The current device preset can be obtained by invoking the function [getDevicePreset](#). The total number of presets for the active device can be retrieved with the function [getNumberOfDevicePresets](#).

Parameters

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

```
6.1.4.8 getDevicePropertyFloat() double getDevicePropertyFloat (
    OCTDeviceHandle Dev,
    DevicePropertyFloat Selection )
```

Returns properties of the device belonging to the specified [OCTDeviceHandle](#).

Parameters

in	<i>Selection</i>	The desired property.
----	------------------	-----------------------

Returns

The value of the desired property.

Parameters

in	<i>Dev</i>	A valid (non null) OCT device handle (OCTDeviceHandle), previously generated with the function initDevice .
----	------------	---

```
6.1.4.9 getDevicePropertyInt() int getDevicePropertyInt (
    OCTDeviceHandle Dev,
    DevicePropertyInt Selection )
```

Returns properties of the device belonging to the specified [OCTDeviceHandle](#).

Parameters

in	<i>Selection</i>	The desired property.
----	------------------	-----------------------

Returns

The value of the desired property.

Parameters

in	<i>Dev</i>	A valid (non null) OCT device handle (OCTDeviceHandle), previously generated with the function initDevice .
----	------------	---

```
6.1.4.10 getDevicePropertyString() const char * getDevicePropertyString (
    OCTDeviceHandle Dev,
    DevicePropertyString Selection )
```

Returns properties of the device belonging to the specified [OCTDeviceHandle](#).

Parameters

in	<i>Selection</i>	The desired property.
----	------------------	-----------------------

Returns

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

Parameters

in	<i>Dev</i>	A valid (non null) OCT device handle (OCTDeviceHandle), previously generated with the function initDevice .
----	------------	---

```
6.1.4.11 getDeviceState() DeviceState getDeviceState (
    void )
```

Returns the state of supported base-unit.

This function attempts to communicate with the device, and returns the state of the base-unit if a minimum of working functionality can be guaranteed. If no device can be found or communication fails, this function will return `DeviceState_Unavailable`. This function can be invoked as many times as desired (e.g. in a polling strategy) without side effects.

```
6.1.4.12 getLightSourceTimeout_s() double getLightSourceTimeout_s (
    OCTDeviceHandle Dev )
```

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

Parameters

<code>Dev</code>	the <code>OCTDeviceHandle</code> that was initially provided by <code>initDevice</code> .
------------------	---

Returns

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

```
6.1.4.13 getNumberOfDevicePresetCategories() int getNumberOfDevicePresetCategories (
    OCTDeviceHandle Dev )
```

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

Parameters

in	<code>Dev</code>	A valid (non null) OCT device handle (<code>OCTDeviceHandle</code>), previously generated with the function <code>initDevice</code> .
----	------------------	---

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

```
6.1.4.14 getNumberOfDevicePresets() int getNumberOfDevicePresets (
    OCTDeviceHandle Dev,
    int Category )
```

Returns the number of available device presets.

Returns

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

The current device preset can be obtained by invoking the function [getDevicePreset](#). A description of the device preset associated with a particular index can be obtained by invoking the function [getDevicePresetDescription](#).

Parameters

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

6.1.4.15 [getRefstageLength_mm\(\)](#) double [getRefstageLength_mm](#) (
OCTDeviceHandle Dev,
ProbeHandle Probe)

Returns the total length in mm of the reference stage.

Parameters

<i>Dev</i>	the OCTDeviceHandle that was initially provided by initDevice .
<i>Probe</i>	the ProbeHandle that was initially provided by initProbe .

6.1.4.16 [getRefstageMaxPosition_mm\(\)](#) double [getRefstageMaxPosition_mm](#) (
OCTDeviceHandle Dev,
ProbeHandle Probe)

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

Parameters

<i>Dev</i>	the OCTDeviceHandle that was initially provided by initDevice .
<i>Probe</i>	the ProbeHandle that is currently.

6.1.4.17 [getRefstageMinPosition_mm\(\)](#) double [getRefstageMinPosition_mm](#) (
OCTDeviceHandle Dev,
ProbeHandle Probe)

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

Parameters

<i>Dev</i>	the OCTDeviceHandle that was initially provided by initDevice.
<i>Probe</i>	the ProbeHandle that is currently.

6.1.4.18 [getRefstagePosition_mm\(\)](#) `double getRefstagePosition_mm (OCTDeviceHandle Dev, ProbeHandle Probe)`

Returns the current position in mm of the reference stage.

Parameters

<i>Dev</i>	the OCTDeviceHandle that was initially provided by initDevice.
<i>Probe</i>	the ProbeHandle that was initially provided by initProbe.

6.1.4.19 [getRefstageStatus\(\)](#) `RefstageStatus getRefstageStatus (OCTDeviceHandle Dev)`

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

Parameters

<i>Dev</i>	the OCTDeviceHandle that was initially provided by initDevice.
------------	--

6.1.4.20 [getStaticDeviceFlag\(\)](#) `BOOL getStaticDeviceFlag (StaticDeviceFlag Selection)`

Returns properties of the device available before device initialization.

Parameters

<i>in</i>	<i>Selection</i>	The desired flag.
-----------	------------------	-------------------

Returns

The value of the desired flag.

```
6.1.4.21 homeRefstage() void homeRefstage (
    OCTDeviceHandle Dev,
    RefstageWaitForMovement WaitForMoving )
```

Homes the reference stage to calibrate the zero position.

Parameters

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

```
6.1.4.22 initDevice() OCTDeviceHandle initDevice (
    void )
```

Initializes the installed device.

Returns

Handle to the initialized OCT device ([OCTDeviceHandle](#)).

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

```
6.1.4.23 isDeviceAvailable() BOOL isDeviceAvailable (
    void )
```

Returns whether any supported Base-Unit is available.

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

```
6.1.4.24 isRefstageAvailable() SPECTRALRADAR_API BOOL isRefstageAvailable (
    OCTDeviceHandle Dev )
```

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

Parameters

<i>Dev</i>	the OCTDeviceHandle that was initially provided by initDevice.
------------	--

```
6.1.4.25 moveRefstage_mm() void moveRefstage_mm (
```

```
OCTDeviceHandle Dev,
ProbeHandle Probe,
double Length_mm,
RefstageMovementDirection Direction,
RefstageSpeed Speed,
RefstageWaitForMovement WaitForMoving )
```

Moves the reference stage with the specified length in mm.

Parameters

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

6.1.4.26 moveRefstageToPosition_mm() void moveRefstageToPosition_mm (

```
OCTDeviceHandle Dev,
ProbeHandle Probe,
double pos_mm,
RefstageSpeed Speed,
RefstageWaitForMovement WaitForMoving )
```

Moves the reference stage to the specified position in mm.

Parameters

<i>Dev</i>	the OCTDeviceHandle that was initially provided by initDevice.
<i>Probe</i>	the ProbeHandle that was initially provided by initProbe.
<i>pos_mm</i>	gives the desired position in mm
<i>Speed</i>	is the velocity of the reference stage movement specified with RefstageSpeed
<i>WaitForMoving</i>	defines whether the function should wait until the movement of the reference stage has stopped or not until it returns

6.1.4.27 moveScanner() void moveScanner (

```
OCTDeviceHandle Dev,
ProbeHandle Probe,
ScanAxis Axis,
double Position_mm )
```

Manually moves the scanner to a given position.

Parameters

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

6.1.4.28 moveScannerToApoPosition() `void moveScannerToApoPosition (`

```
OCTDeviceHandle Dev,
ProbeHandle Probe )
```

Moves the scanner to the apodization position.

Parameters

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

6.1.4.29 QuantumEfficiency() `double QuantumEfficiency (`

```
OCTDeviceHandle Dev,
double CenterWavelength_nm,
double PowerIntoSpectrometer_W,
DataHandle Spectrum_e )
```

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

6.1.4.30 resetCamera() `void resetCamera (`

```
void )
```

Resets the spectrometer camera.

6.1.4.31 setDeviceFlag() `void setDeviceFlag (`

```
OCTDeviceHandle Dev,
DeviceFlag Selection,
BOOL Value )
```

Sets the selected flag of the device belonging to the specified [OCTDeviceHandle](#).

Parameters

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

```
6.1.4.32 setDevicePreset() void setDevicePreset (
    OCTDeviceHandle Dev,
    int Category,
    ProbeHandle Probe,
    ProcessingHandle Proc,
    int Preset )
```

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

Parameters

in	<i>Dev</i>	A valid (non null) OCT device handle (OCTDeviceHandle), previously generated with the function initDevice .
in	<i>Category</i>	An index describing the preset category in the range between 0 and the number of preset categories minus 1, given by getNumberOfDevicePresetCategories
in	<i>Probe</i>	A handle to the probe (ProbeHandle).
in	<i>Proc</i>	A valid (non null) processing handle.
in	<i>Preset</i>	The index of the preset.

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

```
6.1.4.33 setLightSourceTimeout_s() void setLightSourceTimeout_s (
    OCTDeviceHandle Dev,
    double Timeout )
```

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

Parameters

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

```
6.1.4.34 setLightSourceTimeoutCallback() void setLightSourceTimeoutCallback (
```

```
OCTDeviceHandle Dev,
lightSourceStateCallback Callback )
```

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

Parameters

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

6.1.4.35 setRefstagePosChangedCallback() [void setRefstagePosChangedCallback \(](#)

```
OCTDeviceHandle Dev,
cbRefstagePositionChanged Callback )
```

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

Parameters

<i>Dev</i>	the OCTDeviceHandle that was initially provided by initDevice.
<i>Callback</i>	to register.

6.1.4.36 setRefstageSpeed() [void setRefstageSpeed \(](#)

```
OCTDeviceHandle Dev,
RefstageSpeed Speed )
```

Sets the velocity of the movement of the reference stage.

Parameters

<i>Dev</i>	the OCTDeviceHandle that was initially provided by initDevice.
<i>Speed</i>	the chosen velocity of the movement.

6.1.4.37 setRefstageStatusCallback() [void setRefstageStatusCallback \(](#)

```
OCTDeviceHandle Dev,
cbRefstageStatusChanged Callback )
```

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

Parameters

<i>Dev</i>	the OCTDeviceHandle that was initially provided by initDevice.
<i>Callback</i>	to register.

6.1.4.38 setRequiredSLDOnTime_s() void setRequiredSLDOnTime_s (int Time_s)

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

Parameters

in	Time_s	Minimum required on time in seconds.
----	--------	--------------------------------------

6.1.4.39 setStaticDeviceFlag() void setStaticDeviceFlag (StaticDeviceFlag Selection, BOOL Value)

Sets the selected flag of the device available before device initialization.

Parameters

in	Selection	The desired flag.
in	Value	The value of the desired flag.

6.1.4.40 startRefstageMovement() void startRefstageMovement (OCTDeviceHandle Dev, RefstageMovementDirection Direction, RefstageSpeed Speed)

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

Parameters

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

6.1.4.41 stopRefstageMovement() void stopRefstageMovement (OCTDeviceHandle Dev)

Stops the movement of the reference stage.

Parameters

<i>Dev</i>	the OCTDeviceHandle that was initially provided by initDevice .
------------	---

6.1.4.42 updateAfterPresetChange() void updateAfterPresetChange (
 [OCTDeviceHandle](#) *Dev*,
 [ProbeHandle](#) *Probe*,
 [ProcessingHandle](#) *Proc*,
 int *CameraIndex*)

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

Parameters

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

6.2 Speckle Variance Contrast Processing

Typedefs

- `typedef struct C_SpeckleVariance * SpeckleVarianceHandle`
Handle used for SpeckleVariance processing.

6.2.1 Detailed Description

6.2.2 Typedef Documentation

6.2.2.1 `SpeckleVarianceHandle` [SpeckleVarianceHandle](#)

Handle used for SpeckleVariance processing.

Definition at line 115 of file [SpectralRadar_Handles.h](#).

6.3 Error Handling

Error handling.

Enumerations

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

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

- enum `LogOutputType` {
 Standard,
 File,
 None }

Specifies where to write text output by the SDK.

Functions

- `SPECTRALRADAR_API void setErrorsPerThreadFlag (bool getErrorsPerThread)`

Sets the whether or not errors are returned per thread in `isError` and `getError`. By default, errors are collected globally and a call to `getError` will return the last error from any thread. If this flag is set to true, only errors from the current thread will be returned.

- `SPECTRALRADAR_API ErrorCode isError (void)`

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

- `SPECTRALRADAR_API ErrorCode getError (char *Message, int StringSize)`

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

6.3.1 Detailed Description

Error handling.

6.3.2 Enumeration Type Documentation

6.3.2.1 ErrorCode 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.

Definition at line 49 of file [SpectralRadar_Types.h](#).

6.3.2.2 LogOutputType enum [LogOutputType](#)

Specifies where to write text output by the SDK.

Enumerator

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

Definition at line 59 of file [SpectralRadar_Types.h](#).

6.3.3 Function Documentation

```
6.3.3.1 ErrorCode getError (
    char * Message,
    int StringSize )
```

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

Returns

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

See also

[ErrorCode](#).

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

Parameters

out	<i>Message</i>	Error message describing the error.
in	<i>StringSize</i>	Size of the string that was given to <i>Message</i> .

6.3.3.2 isError() ErrorCode isError (

```
    void  )
```

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

6.3.3.3 **setErrorsPerThreadFlag()** `void setErrorsPerThreadFlag (` `bool getErrorsPerThread)`

Sets the whether or not errors are returned per thread in [isError](#) and [getError](#). By default, errors are collected globally and a call to [getError](#) will return the last error from any thread. If this flag is set to true, only errors from the current thread will be returned.

6.4 Data Access

Functions for accessing the information stored in data objects.

Data Structures

- struct [ComplexFloat](#)
A standard complex data type that is used to access complex data.

Typedefs

- typedef struct C_RawData * [RawDataHandle](#)
Handle to an object holding the unprocessed raw data.
- typedef struct C_Data * [DataHandle](#)
Handle to an object holding 1-, 2- or 3-dimensional floating point data.
- typedef struct C_ColoredData * [ColoredDataHandle](#)
Handle to an object holding 1-, 2- or 3-dimensional colored data.
- typedef struct C_ComplexData * [ComplexDataHandle](#)
Handle to an object holding complex 1-, 2- or 3-dimensional complex floating point data.
- typedef struct C_ImageFieldCorrection * [ImageFieldHandle](#)
Handle to the image field description.
- typedef struct C_FileHandling * [OCTFileHandle](#)
Handle to the OCT file class.

Enumerations

- enum [RawDataPropertyInt](#) {
 RawData_Size1,
 RawData_Size2,
 RawData_Size3,
 RawData_NumberOfElements,
 RawData_SizeInBytes,
 RawData_BytesPerElement,
 RawData_LostFrames }

Integer properties of raw data ([RawDataHandle](#)) that can be retrieved with the function [getRawDataPropertyInt](#).
- enum [RawDataPropertyFloat](#) {
 RawData_Range1,
 RawData_Range2,
 RawData_Range3 }

*Floating point properties of raw data ([RawDataHandle](#)) that can be retrieved with the function [getRawDataPropertyFloat](#).
Some of these parameters will be directly copied from the [ScanPatternHandle](#) that was used to acquire the data.*
- enum [DataPropertyInt](#) {
 Data_Dimensions,
 Data_Size1,
 Data_Size2,
 Data_Size3,
 Data_NumberOfElements,
 Data_SizeInBytes,
 Data_BytesPerElement }

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

- enum `DataPropertyFloat` {
 `Data_Spacing1`,
 `Data_Spacing2`,
 `Data_Spacing3`,
 `Data_Range1`,
 `Data_Range2`,
 `Data_Range3` }

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

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

Analysis types accepted by the functions `analyzeData` and `computeDataProjection`.

- enum `AScanAnalyzation` {
 `Data_Noise_dB`,
 `Data_Noise_electrons`,
 `Data_PeakPos_Pixel`,
 `Data_PeakPos_PhysUnits`,
 `Data_PeakHeight_dB`,
 `Data_PeakWidth_6dB`,
 `Data_PeakWidth_20dB`,
 `Data_PeakWidth_40dB`,
 `Data_PeakPhase`,
 `Data_PeakRealPart`,
 `Data_PeakImagPart` }

Analysis types accepted by the functions `analyzeAScan` and `analyzeComplexAScan`.

- enum `DataOrientation` {
 `DataOrientation_ZXY`,
 `DataOrientation_ZYX`,
 `DataOrientation_XZY`,
 `DataOrientation_XYZ`,
 `DataOrientation_YXZ`,
 `DataOrientation_YZX`,
 `DataOrientation_ZTX`,
 `DataOrientation_ZXT` }

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

Functions

- **SPECTRALRADAR_API** void `absComplexData` (const `ComplexDataHandle` `ComplexData`, `DataHandle` `Abs`)
Converts the complex values from the `ComplexDataHandle` to its absolute values and writes them to `DataHandle`.
- **SPECTRALRADAR_API** void `logAbsComplexData` (const `ComplexDataHandle` `ComplexData`, `DataHandle` `dB`)
Converts the complex values from the `ComplexDataHandle` to its dB values and writes them to `DataHandle`.
- **SPECTRALRADAR_API** int `getDataPropertyInt` (`DataHandle` `Data`, `DataPropertyInt` `Selection`)
Returns the selected integer property of the specified data.
- **SPECTRALRADAR_API** double `getDataPropertyFloat` (`DataHandle` `Data`, `DataPropertyFloat` `Selection`)
Returns the selected floating point property of the specified data.
- **SPECTRALRADAR_API** void `copyData` (`DataHandle` `DataSource`, `DataHandle` `DataDestination`)
Copies the content of the specified source to the specified destination.
- **SPECTRALRADAR_API** void `copyDataContent` (`DataHandle` `DataSource`, float *`Destination`)
Copies the data in the specified data object (`DataHandle`) into the specified pointer.

- **SPECTRALRADAR_API** float * `getDataPtr (DataHandle Data)`
The returned pointer points to memory owned by SpectralRadar.dll. The user should not attempt to free it.
- **SPECTRALRADAR_API** void `reserveData (DataHandle Data, int Size1, int Size2, int Size3)`
Reserves the amount of data specified. This might improve performance if appending data to the DataHandle as no additional memory needs to be reserved then.
- **SPECTRALRADAR_API** void `resizeData (DataHandle Data, int Size1, int Size2, int Size3)`
Resizes the respective data object. In general the data will be 1-dimensional if Size2 and Size3 are equal to 1, 2-dimensional if Size3 is equal to 1 or 3-dimensional if all, Size1, Size2, Size3, are unequal to 1.
- **SPECTRALRADAR_API** void `setDataRange (DataHandle Data, double range1, double range2, double range3)`
Sets the range in mm in the 3 axes represented in the RealData buffer.
- **SPECTRALRADAR_API** void `setDataContent (DataHandle Data, float *NewContent)`
Sets the data content of the data object. The data chunk pointed to by NewContent needs to be of the size expected by the data object, i. e. $\text{Size1} \times \text{Size2} \times \text{Size3} \times \text{sizeof}(float)$.
- **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 orientation of the data object to the given orientation.
- **SPECTRALRADAR_API** int `getComplexDataPropertyInt (ComplexDataHandle Data, DataPropertyInt Selection)`
Returns the selected integer property of the specified data.
- **SPECTRALRADAR_API** double `getComplexDataPropertyFloat (ComplexDataHandle Data, DataPropertyFloat Selection)`
Returns the selected floating-point property of the specified data.
- **SPECTRALRADAR_API** void `copyComplexDataContent (ComplexDataHandle DataSource, ComplexFloat *Destination)`
Copies the content of the complex data to the pointer specified as destination.
- **SPECTRALRADAR_API** void `copyComplexData (ComplexDataHandle DataSource, ComplexDataHandle DataDestination)`
Copies the contents of the specified ComplexDataHandle to the specified destination ComplexDataHandle.
- **SPECTRALRADAR_API** ComplexFloat * `getComplexDataPtr (ComplexDataHandle Data)`
The returned pointer points to memory owned by SpectralRadar.dll. The user should not attempt to free it.
- **SPECTRALRADAR_API** void `setComplexDataContent (ComplexDataHandle Data, ComplexFloat *NewContent)`
Sets the data content of the ComplexDataHandle to the content specified by the pointer.
- **SPECTRALRADAR_API** void `reserveComplexData (ComplexDataHandle Data, int Size1, int Size2, int Size3)`
Reserves the amount of data specified. This might improve performance if appending data to the ComplexDataHandle as no additional memory needs to be reserved then.
- **SPECTRALRADAR_API** void `resizeComplexData (ComplexDataHandle Data, int Size1, int Size2, int Size3)`
Resizes the respective data object. In general the data will be 1-dimensional if Size2 and Size3 are equal to 1, 2-dimensional if Size3 is equal to 1 or 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 floating-point property of the specified colored data.
- **SPECTRALRADAR_API** void `copyColoredData (ColoredDataHandle ImageSource, ColoredDataHandle ImageDestination)`
Copies the contents of the specified ColoredDataHandle to the specified destination ColoredDataHandle.

- **SPECTRALRADAR_API** void `copyColoredDataContent` (`ColoredDataHandle` Source, `unsigned long` *Destination)
Copies the data in the specified colored data object (`ColoredDataHandle`) into the specified pointer.
- **SPECTRALRADAR_API** void `copyColoredDataContentAligned` (`ColoredDataHandle` ImageSource, `unsigned long` *Destination, `int` Stride)
Copies the data in the specified colored data object (`ColoredDataHandle`) into the specified pointer.
- **SPECTRALRADAR_API** `unsigned long *` `getColoredDataPtr` (`ColoredDataHandle` ColData)
The returned pointer points to memory owned by SpectralRadar.dll. The user should not attempt to free it.
- **SPECTRALRADAR_API** void `resizeColoredData` (`ColoredDataHandle` ColData, `int` Size1, `int` Size2, `int` Size3)
Resizes the respective colored data object. In general the data will be 1-dimensional if Size2 and Size3 are equal to 1, 2-dimensional if Size3 is equal to 1 dn 3-dimensional if all, Size1, Size2, Size3, are unequal to 1.
- **SPECTRALRADAR_API** void `reserveColoredData` (`ColoredDataHandle` ColData, `int` Size1, `int` Size2, `int` Size3)
Reserves the amount of colored data specified. This might improve performance if appending data to the `ColoredDataHandle` as no additional memory needs to be reserved then.
- **SPECTRALRADAR_API** void `setColoredDataContent` (`ColoredDataHandle` ColData, `unsigned long` *NewContent)
*Sets the data content of the colored data object. The data chunk pointed to by NewContent needs to be of the size expected by the data object, i. e. `Size1*Size2*Size*sizeof(unsigned long)`.*
- **SPECTRALRADAR_API** void `setColoredDataRange` (`ColoredDataHandle` Data, `double` range1, `double` range2, `double` range3)
Sets the range in mm in the 3 axes represented in the data object buffer.
- **SPECTRALRADAR_API** `DataOrientation` `getColoredDataOrientation` (`ColoredDataHandle` Data)
Returns the data orientation of the colored data object.
- **SPECTRALRADAR_API** void `setColoredDataOrientation` (`ColoredDataHandle` Data, `DataOrientation` Orientation)
Sets the data orientation of the colored data object to the given orientation.
- **SPECTRALRADAR_API** void `copyRawDataContent` (`RawDataHandle` RawDataSource, `void *` DataContent)
Copies the content of the raw data into the specified buffer.
- **SPECTRALRADAR_API** void `copyRawData` (`RawDataHandle` RawDataSource, `RawDataHandle` RawData, `RawData` Target)
Copies raw data content and metadata into the specified target handle.
- **SPECTRALRADAR_API** `void *` `getRawDataPtr` (`RawDataHandle` RawDataSource)
Notice that raw data refers to the spectra as acquired, without processing of any kind.
- **SPECTRALRADAR_API** `int` `getRawDataPropertyInt` (`RawDataHandle` RawData, `RawDataPropertyInt` Property)
Notice that raw data refers to the spectra as acquired, without processing of any kind.
- **SPECTRALRADAR_API** `double` `getRawDataPropertyFloat` (`RawDataHandle` Data, `RawDataPropertyFloat` 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 `reserveRawData` (`RawDataHandle` Raw, `int` Size1, `int` Size2, `int` Size3)
Reserves the amount of data specified. This might improve performance if appending data to the `RawDataHandle` as no additional memory needs to be reserved then.
- **SPECTRALRADAR_API** void `resizeRawData` (`RawDataHandle` Raw, `int` Size1, `int` Size2, `int` Size3)
Resizes the specified raw data buffer accordingly.
- **SPECTRALRADAR_API** void `setRawDataContent` (`RawDataHandle` RawData, `void *` NewContent)
Sets the content of the raw data buffer. The size of the `RawDataHandle` needs to be adjusted first, as otherwise not all data might be copied.
- **SPECTRALRADAR_API** void `setScanSpectra` (`RawDataHandle` RawData, `int` NumberOfScanRegions, `int` *ScanRegions)

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

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

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

- **SPECTRALRADAR_API** int `getNumberOfScanRegions (RawDataHandle Raw)`

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

- **SPECTRALRADAR_API** int `getNumberOfApodizationRegions (RawDataHandle Raw)`

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

- **SPECTRALRADAR_API** void `getScanSpectra (RawDataHandle Raw, int *SpectralIndex)`

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 *SpectralIndex)`

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

- **SPECTRALRADAR_API** void `determineSurface (DataHandle Volume, DataHandle Surface)`

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

- **SPECTRALRADAR_API** void `argComplexData (ComplexDataHandle ComplexData, DataHandle Arg)`

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

- **SPECTRALRADAR_API** void `realComplexData (ComplexDataHandle ComplexData, DataHandle Real)`

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

- **SPECTRALRADAR_API** void `imagComplexData (ComplexDataHandle ComplexData, DataHandle Imag)`

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

- **SPECTRALRADAR_API** void `crossCorrelatedProjection (DataHandle DataIn, DataHandle DataOut)`

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

- **SPECTRALRADAR_API** void `averagingResizeFilter (DataHandle DataIn, DataHandle DataOut, int Decimation1, int Decimation2, int Decimation3)`

Resizes a data object by averaging. The number of pixels that are averaged on each axis can be specified by the Decimation parameters. E.g. if Decimation2 is set to 3, Dimension2 of the resulting `DataOut` object will be 1/3 of Dimension2 of `DataIn`.

- **SPECTRALRADAR_API** void `thresholdDopplerData (DataHandle Phase, DataHandle Intensity, float intensityThreshold, float phaseTargetValue)`

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

- **SPECTRALRADAR_API** void `getCurrentIntensityStatistics (OCTDeviceHandle Dev, ProcessingHandle Proc, float *relToRefIntensity, float *relToProjAbsIntensity)`

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

- **SPECTRALRADAR_API** void `extractLine (DataHandle Data, DataHandle Res, double P1_1_mm, double P1_2_mm, double P2_1_mm, double P2_2_mm)`

Extract 1D data from 2D along a specified line.

- **SPECTRALRADAR_API** int `extractLocalMaxima (DataHandle Data1D, int N, double *dataPos_mm, double *dataHeight)`

Extract multiple local maxima from 1D data.

- **SPECTRALRADAR_API** int `extractLocalMaximaEx (DataHandle Data1D, int N, double minDist, double *dataPos_mm, double *dataHeight)`

Extract multiple local maxima from 1D data while maintaining a minimum distance between peaks.

- **SPECTRALRADAR_API** void `readData (DataHandle Data, const char *filename, int SizeZ, int SizeX, int SizeY)`

Read data object from raw data stream in file.

- **SPECTRALRADAR_API** void `extractAScan (DataHandle Data, DataHandle Res, int x0, int y0)`

Extracts an A-scan at a given x and y coordinates and stores it in a different data handle.

6.4.1 Detailed Description

Functions for accessing the information stored in data objects.

6.4.2 Typedef Documentation

6.4.2.1 ColoredDataHandle [ColoredDataHandle](#)

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

Colored data handles are used to obtain processed data in a format that can readily be exported into a graphics format file, using a user selected palette. Otherwise they are the same as processed data ([DataHandle](#)).

In order to specify the desired palette and its properties, users should refer to coloring handles ([ColoringHandle](#)) and associated functions.

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

Definition at line 52 of file [SpectralRadar_Handles.h](#).

6.4.2.2 ComplexDataHandle [ComplexDataHandle](#)

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

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

Definition at line 62 of file [SpectralRadar_Handles.h](#).

6.4.2.3 DataHandle [DataHandle](#)

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

This structure may hold data generated by processing raw data ([RawDataHandle](#)), and also more abstract data, such as point sequences intended to determine a scan pattern (see e.g. [getNumberOfScanPointsFromDataHandle](#) or [getScanPointsFromDataHandle](#) below). The associated properties of the data (dimensionality, sizes in pixels, spacings/ranges in millimeters) are also part of this structure.

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

Definition at line 38 of file [SpectralRadar_Handles.h](#).

6.4.2.4 **ImageFieldHandle** [ImageFieldHandle](#)

Handle to the image field description.

Definition at line 133 of file [SpectralRadar_Handles.h](#).

6.4.2.5 **OCTFileHandle** [OCTFileHandle](#)

Handle to the OCT file class.

Definition at line 151 of file [SpectralRadar_Handles.h](#).

6.4.2.6 **RawDataHandle** [RawDataHandle](#)

Handle to an object holding the unprocessed raw data.

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

Definition at line 23 of file [SpectralRadar_Handles.h](#).

6.4.3 Enumeration Type Documentation

6.4.3.1 **AScanAnalyzation** [enum AScanAnalyzation](#)

Analysis types accepted by the functions [analyzeAScan](#) and [analyzeComplexAScan](#).

Enumerator

Data_Noise_dB	Noise of the A-scan in dB. This assumes that no signal is present in the A-scan. The noise is computed by averaging all fourier channels larger than 50.
Data_Noise_electrons	Noise of the A-scan in electrons. This assumes that no signal is present in the A-scan. The noise is computed by averaging all fourier channels larger than 50.
Data_PeakPos_Pixel	Peak position of the highest peak in pixels. The peak position is determined by computing a parable going through the maximum value point and its surrounding pixels. The position of the maximum is used.
Data_PeakPos_PhysUnits	Peak position of the highest peak in physical units. The peak position is determined by computing a parable going through the maximum value point and its surrounding pixels. The position of the maximum is used. Physical coordinates are computed by using the calibrated zSpacing property of the device. The concrete physical units of the return value depends on the calibration.
Data_PeakHeight_dB	Peak height of the highest peak in dB. The peak height is determined by computing a parable going through the maximum value point and its surrounding pixels. The height of the resulting parable is returned.
Data_PeakWidth_6dB	Signal width at -6dB. This is the FWHM.
Data_PeakWidth_20dB	Signal width at -20dB.
Data_PeakWidth_40dB	Signal width at -40dB.
Data_PeakPhase	Phase of the highest peak in radians. This value is only accepted by the function analyzeComplexAScan .

Definition at line 89 of file [SpectralRadar_Types.h](#).

6.4.3.2 DataAnalyzation `enum DataAnalyzation`

Analysis types accepted by the functions [analyzeData](#) and [computeDataProjection](#).

Enumerator

<code>Data_Min</code>	Minimum of the values in the data.
<code>Data_Mean</code>	Arithmetic mean of all values in the data.
<code>Data_Max</code>	Maximum of the values in the data.
<code>Data_MaxDepth</code>	The depth of the maximum of the values in the data.
<code>Data_Median</code>	The median of the values in the data.

Definition at line 73 of file [SpectralRadar_Types.h](#).

6.4.3.3 DataOrientation `enum DataOrientation`

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

See also

[getDataOrientation](#), [setDataOrientation](#), [getColoredDataOrientation](#), [setColoredDataOrientation](#).

Definition at line 127 of file [SpectralRadar_Types.h](#).

6.4.3.4 DataPropertyFloat `enum DataPropertyFloat`

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

Enumerator

<code>Data_Spacing1</code>	Spacing between two subsequent data elements in direction of the first axis in physical units (millimeter).
<code>Data_Spacing2</code>	Spacing between two subsequent data elements in direction of the second axis in physical units (millimeter).
<code>Data_Spacing3</code>	Spacing between two subsequent data elements in direction of the third axis in physical units (millimeter).
<code>Data_Range1</code>	Total range of the data in direction of the first axis in physical units (millimeter).
<code>Data_Range2</code>	Total range of the data in direction of the second axis in physical units (millimeter).
<code>Data_Range3</code>	Total range of the data in direction of the third axis in physical units (millimeter).

Definition at line 78 of file [SpectralRadar_Properties.h](#).

6.4.3.5 DataPropertyInt enum [DataPropertyInt](#)

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

Enumerator

Data_Dimensions	Dimension of the data object. Usually 1, 2 or 3. 0 indicates empty data.
Data_Size1	Size of the first dimension. For OCT data this is usually the longitudinal axis (z)
Data_Size2	Size of the second dimension. This is a transversal axis (x).
Data_Size3	Size of the third dimension. This is 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.

Definition at line 58 of file [SpectralRadar_Properties.h](#).

6.4.3.6 RawDataPropertyFloat enum [RawDataPropertyFloat](#)

Floating point properties of raw data ([RawDataHandle](#)) that can be retrieved with the function [getRawDataPropertyFloat](#). Some of these parameters will be directly copied from the [ScanPatternHandle](#) that was used to acquire the data.

Enumerator

RawData_Range1	Range of the first dimension. This will be the spectral dimension, i. e. z-dimension prior to Fourier transformation.
RawData_Range2	Range of the second dimension. This is a transversal axis (x).
RawData_Range3	Range of the third dimension. This is a transversal axis (y).

Definition at line 46 of file [SpectralRadar_Properties.h](#).

6.4.3.7 RawDataPropertyInt enum [RawDataPropertyInt](#)

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

Enumerator

RawData_Size1	Size of the first dimension. This will be the spectral dimension, i. e. z-dimension prior to Fourier transformation.
RawData_Size2	Size of the second dimension. This is a transversal axis (x).
RawData_Size3	Size of the third dimension. This is a transversal axis (y).
RawData_NumberOfElements	The number of elements in the raw data object.
RawData_SizeInBytes	The size of the data object in bytes.
RawData_BytesPerElement	The number of bytes of a single element, i. e. the data type of the raw data.

Enumerator

RawData_LostFrames	The number of lost frames during data acquisition. Lost frames are the usual consequence of a scanning dynamics going too fast in comparison to the acquisition settings (or capabilities) of the camera. To remedy the situation, consider increasing the number of measurements along the x-axis (size 2), or increasing the A-scan averaging. A rule of thumb to check if the settings are safe would be $[2s_{Hz}f_s] < X_{px}A$, where s_{Hz} is the scan speed expressed in Herz, f_s is the flyback time expressed in seconds (as written in the probe configuration file, e.g. Probe.ini), X_{px} is the number of desired A-scans along the x axis (i.e. RawData_Size2), and A is the A-scan averaging.
--------------------	--

Definition at line 17 of file [SpectralRadar_Properties.h](#).

6.4.4 Function Documentation

6.4.4.1 absComplexData() `void absComplexData (`
`const ComplexDataHandle ComplexData,`
`DataHandle Abs)`

Converts the complex values from the [ComplexDataHandle](#) to its absolute values and writes them to [DataHandle](#).

6.4.4.2 argComplexData() `void argComplexData (`
`ComplexDataHandle ComplexData,`
`DataHandle Arg)`

Converts the complex values from the [ComplexDataHandle](#) to its phase angle values and writes them to [DataHandle](#).

6.4.4.3 averagingResizeFilter() `void averagingResizeFilter (`
`DataHandle DataIn,`
`DataHandle DataOut,`
`int Decimation1,`
`int Decimation2,`
`int Decimation3)`

Resizes a data object by averaging. The number of pixels that are averaged on each axis can be specified by the Decimation parameters. E.g. if Decimation2 is set to 3, Dimension2 of the resulting DataOut object will be 1/3 of Dimension2 of DataIn.

Parameters

<i>DataIn</i>	Input DataHandle
<i>DataOut</i>	Output DataHandle (needs to be initialized)
<i>Decimation1</i>	Number of pixels to average in Dimension1
<i>Decimation2</i>	Number of pixels to average in Dimension2
<i>Decimation3</i>	Number of pixels to average in Dimension3

```
6.4.4.4 copyColoredData() void copyColoredData (
    ColoredDataHandle ImageSource,
    ColoredDataHandle ImageDestination )
```

Copies the contents of the specified [ColoredDataHandle](#) to the specified destination [ColoredDataHandle](#).

Parameters

in	<i>ImageSource</i>	A valid (non null) colored data handle of the source (ColoredDataHandle).
in	<i>ImageDestination</i>	A valid (non null) colored data handle of the destination (ColoredDataHandle).

```
6.4.4.5 copyColoredDialogContent() void copyColoredDialogContent (
    ColoredDataHandle Source,
    unsigned long * Destination )
```

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

Parameters

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

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

```
6.4.4.6 copyColoredDialogContentAligned() void copyColoredDialogContentAligned (
    ColoredDataHandle ImageSource,
    unsigned long * Destination,
    int Stride )
```

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

Parameters

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

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

```
6.4.4.7 copyComplexData() void copyComplexData (
    ComplexDataHandle DataSource,
    ComplexDataHandle DataDestination )
```

Copies the contents of the specified [ComplexDataHandle](#) to the specified destination [ComplexDataHandle](#).

Parameters

in	<i>DataSource</i>	A valid (non null) complex data handle of the source (ComplexDataHandle).
in	<i>DataDestination</i>	A valid (non null) complex data handle of the destination (ComplexDataHandle).

```
6.4.4.8 copyComplexDataContent() void copyComplexDataContent (
    ComplexDataHandle DataSource,
    ComplexFloat * Destination )
```

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

Parameters

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

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

```
6.4.4.9 copyData() void copyData (
    DataHandle DataSource,
    DataHandle DataDestination )
```

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

Parameters

in	<i>DataSource</i>	A valid (non null) data handle of the source (DataHandle).
in	<i>DataDestination</i>	A valid (non null) data handle of the destination (DataHandle).

```
6.4.4.10 copyDataContent() void copyDataContent (
    DataHandle DataSource,
    float * Destination )
```

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

Parameters

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

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

```
6.4.4.11 copyRawData() void copyRawData (
    RawDataHandle RawDataSource,
    RawDataHandle RawDataTarget )
```

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

Parameters

in	<i>RawDataSource</i>	A valid (non null) raw data handle of the source (RawDataHandle).
in	<i>RawDataTarget</i>	A valid (non null) raw data handle of the target (RawDataHandle).

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

```
6.4.4.12 copyRawDataContent() void copyRawDataContent (
    RawDataHandle RawDataSource,
    void * DataContent )
```

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

Parameters

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

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

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

```
6.4.4.13 crossCorrelatedProjection() void crossCorrelatedProjection (
    DataHandle DataIn,
    DataHandle DataOut )
```

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

```
6.4.4.14 determineSurface() void determineSurface (
    DataHandle Volume,
    DataHandle Surface )
```

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

```
6.4.4.15 extractAScan() void extractAScan (
    DataHandle Data,
    DataHandle Res,
    int x0,
    int y0 )
```

Extracts an A-scan at a given x and y coordinates and stores it in a different data handle.

Warning

Experimental: This function may change in its interface and behaviour, or even disappear in future versions.

Parameters

in	<i>Data</i>	2D or 3D data of which an A-scan is to be extracted
out	<i>Res</i>	A-scan that was extracted at the location (x0, y0)
in	<i>x0</i>	x-index of the A-scan to extract
in	<i>y0</i>	y-index of the A-scan to extract

```
6.4.4.16 extractLine() void extractLine (
```

```
    DataHandle Data,
    DataHandle Res,
    double P1_1_mm,
    double P1_2_mm,
    double P2_1_mm,
    double P2_2_mm )
```

Extract 1D data from 2D along a specified line.

Warning

Experimental: This function may change in its interface and behaviour, or even disappear in future versions.

Parameters

in	<i>Data</i>	2D source data
out	<i>Res</i>	1D result data
in	<i>P1_1_mm</i>	Start of line on axis 1 in mm
in	<i>P1_2_mm</i>	Start of line on axis 2 in mm
in	<i>P2_1_mm</i>	End of line on axis 1 in mm
in	<i>P2_2_mm</i>	End of line on axis 2 in mm

```
6.4.4.17 extractLocalMaxima() int extractLocalMaxima (
```

```
    DataHandle Data1D,
    int N,
```

```
double * dataPos_mm,
double * dataHeight )
```

Extract multiple local maxima from 1D data.

Warning

Experimental: This function may change in its interface and behaviour, or even disappear in future versions.

Parameters

in	<i>Data1D</i>	Source data
in	<i>N</i>	Maximum number of maxima to find
out	<i>dataPos_mm</i>	Initialized array of double values to be filled with the locations of the maxima
out	<i>dataHeight</i>	Initialized array of double values to be filled with the values of the maxima

Returns

Actual number of maxima found

6.4.4.18 extractLocalMaximaEx()

```
int extractLocalMaximaEx (
    DataHandle Data1D,
    int N,
    double minDist,
    double * dataPos_mm,
    double * dataHeight )
```

Extract multiple local maxima from 1D data while maintaining a minimum distance between peaks.

Warning

Experimental: This function may change in its interface and behaviour, or even disappear in future versions.

Parameters

in	<i>Data1D</i>	Source data
in	<i>N</i>	Maximum number of maxima to find
in	<i>minDist</i>	Minimum distance between each maxima
out	<i>dataPos_mm</i>	Initialized array of double values to be filled with the locations of the maxima
out	<i>dataHeight</i>	Initialized array of double values to be filled with the values of the maxima

Returns

Actual number of maxima found

```
6.4.4.19 getApodizationSpectra() void getApodizationSpectra (
    RawDataHandle Raw,
    int * SpectraIndex )
```

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

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

Parameters

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

```
6.4.4.20 getColoredDataOrientation() DataOrientation getColoredDataOrientation (
    ColoredDataHandle Data )
```

Returns the data orientation of the colored data object.

Returns

The current orientation ([DataOrientation](#)).

Parameters

in	<i>Data</i>	A valid (non null) colored data handle (ColoredDataHandle).
----	-------------	---

```
6.4.4.21 getColoredDataPropertyFloat() int getColoredDataPropertyFloat (
    ColoredDataHandle ColData,
    DataPropertyFloat Selection )
```

Returns the selected integer property of the specified colored data.

Returns

The value of the desired property.

Parameters

in	<i>ColData</i>	A valid (non null) colored data handle (ColoredDataHandle).
in	<i>Selection</i>	The desired property.

```
6.4.4.22 getColoredDataPropertyInt() int getColoredDataPropertyInt (
    ColoredDataHandle ColData,
    DataPropertyInt Selection )
```

Returns the selected integer property of the specified colored data.

Returns

The value of the desired property.

Parameters

in	<i>ColData</i>	A valid (non null) colored data handle (ColoredDataHandle).
in	<i>Selection</i>	The desired property.

```
6.4.4.23 getColoredDataPtr() unsigned long * getColoredDataPtr (
    ColoredDataHandle ColData )
```

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

Returns a pointer to the content of the specified [ColoredDataHandle](#).

Returns

A pointer to the memory owned by the handle.

Parameters

in	<i>ColData</i>	A valid (non null) colored data handle (ColoredDataHandle).
----	----------------	---

```
6.4.4.24 getComplexDataPropertyFloat() double getComplexDataPropertyFloat (
    ComplexDataHandle Data,
    DataPropertyFloat Selection )
```

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

Parameters

in	<i>Data</i>	A valid (non null) complex data handle (ComplexDataHandle).
in	<i>Selection</i>	The desired property.

Returns

The value of the desired property.

6.4.4.25 getComplexDataPropertyInt() `int getComplexDataPropertyInt (`
`ComplexDataHandle Data,`
`DataPropertyInt Selection)`

Returns the selected integer property of the specified data.

Returns

The value of the desired property.

Parameters

in	<i>Data</i>	A valid (non null) complex data handle (ComplexDataHandle).
in	<i>Selection</i>	The desired property.

6.4.4.26 getComplexDataPtr() `ComplexFloat * getComplexDataPtr (`
`ComplexDataHandle Data)`

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

Returns a pointer to the data represented by the [ComplexDataHandle](#). The data is still managed by the [ComplexDataHandle](#) object.

Returns

A pointer to the memory owned by the handle.

Parameters

in	<i>Data</i>	A valid (non null) complex data handle (ComplexDataHandle).
----	-------------	---

6.4.4.27 getCurrentIntensityStatistics() `void getCurrentIntensityStatistics (`
`OCTDeviceHandle Dev,`
`ProcessingHandle Proc,`
`float * relToRefIntensity,`
`float * relToProjAbsIntensity)`

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

used in SR Service, do not remove

```
6.4.4.28 getDataOrientation() void DataOrientation getDataOrientation (
    DataHandle Data )
```

Returns the data orientation of the data object.

Parameters

in	<i>Data</i>	A valid (non null) data handle.
----	-------------	---------------------------------

```
6.4.4.29 getDataPropertyFloat() double getDataPropertyFloat (
    DataHandle Data,
    DataPropertyFloat Selection )
```

Returns the selected floating point property of the specified data.

Returns

The value of the desired property.

Parameters

in	<i>Data</i>	A valid (non null) data handle (DataHandle).
in	<i>Selection</i>	The desired property.

```
6.4.4.30 getDataPropertyInt() int getDataPropertyInt (
    DataHandle Data,
    DataPropertyInt Selection )
```

Returns the selected integer property of the specified data.

Returns

The value of the desired property.

Parameters

in	<i>Data</i>	A valid (non null) data handle (DataHandle).
in	<i>Selection</i>	The desired property.

```
6.4.4.31 getDataPtr() float * getDataPtr (
    DataHandle Data )
```

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

Returns a pointer to the content of the specified data.

Returns

A pointer to the memory owned by the handle.

Parameters

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

6.4.4.32 getNumberOfApodizationRegions() `int getNumberOfApodizationRegions (`
`RawDataHandle Raw)`

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

Returns

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

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

Parameters

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

6.4.4.33 getNumberOfScanRegions() `int getNumberOfScanRegions (`
`RawDataHandle Raw)`

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

Returns

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

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

Parameters

in	<i>Raw</i>	A valid (non null) raw data handle (RawDataHandle).
----	------------	---

6.4.4.34 getRawDataPropertyFloat() `int getRawDataPropertyFloat (`
`RawDataHandle RawData,`
`RawDataPropertyFloat Property)`

Returns a raw data property.

Parameters

in	<i>RawData</i>	A valid (non null) raw data handle (RawDataHandle).
in	<i>Property</i>	The desired property.

Returns

The value of the desired property.

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

6.4.4.35 getRawDataPropertyInt() `int getRawDataPropertyInt (`
`RawDataHandle RawData,`
`RawDataPropertyInt Property)`

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

Returns a raw data property.

Returns

The value of the desired property.

Parameters

in	<i>RawData</i>	A valid (non null) raw data handle (RawDataHandle).
in	<i>Property</i>	The desired property.

6.4.4.36 getRawDataPtr() `void * getRawDataPtr (`
`RawDataHandle RawDataSource)`

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

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

Returns

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

Parameters

in	<i>RawDataSource</i>	A valid (non null) raw data handle of the source (RawDataHandle).
----	----------------------	---

6.4.4.37 `getScanSpectra()` `void getScanSpectra (`
`RawDataHandle Raw,`
`int * SpectraIndex)`

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

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

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

Parameters

in	<i>Raw</i>	A valid (non null) raw data handle (RawDataHandle).
out	<i>SpectralIndex</i>	the array of indices delimiting the scan regions. The size of this array should be twice the number

6.4.4.38 `imagComplexData()` `void imagComplexData (`
`ComplexDataHandle ComplexData,`
`DataHandle Imag)`

Writes the imaginary part of the complex values from the [ComplexDataHandle](#) to [DataHandle](#).

6.4.4.39 `logAbsComplexData()` `void logAbsComplexData (`
`const ComplexDataHandle ComplexData,`
`DataHandle dB)`

Converts the complex values from the [ComplexDataHandle](#) to its dB values and writes them to [DataHandle](#).

```
6.4.4.40 readData() void readData (
    DataHandle Data,
    const char * filename,
    int SizeZ,
    int SizeX,
    int SizeY )
```

Read data object from raw data stream in file.

Warning

Experimental: This function may change in its interface and behaviour, or even disappear in future versions.

Parameters

out	<i>Data</i>	Resulting data
in	<i>filename</i>	File to read
in	<i>SizeZ</i>	SizeZ of dataset
in	<i>SizeX</i>	SizeX of dataset
in	<i>SizeY</i>	SizeY of dataset

```
6.4.4.41 realComplexData() void realComplexData (
    ComplexDataHandle ComplexData,
    DataHandle Real )
```

Writes the real part of the complex values from the [ComplexDataHandle](#) to [DataHandle](#).

```
6.4.4.42 reserveColoredData() void reserveColoredData (
    ColoredDataHandle ColData,
    int Size1,
    int Size2,
    int Size3 )
```

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

Parameters

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

```
6.4.4.43 reserveComplexData() void reserveComplexData (
```

```
ComplexDataHandle Data,
int Size1,
int Size2,
int Size3 )
```

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

Parameters

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

6.4.4.44 reserveData() void reserveData (

```
DataHandle Data,
int Size1,
int Size2,
int Size3 )
```

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

Parameters

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

6.4.4.45 reserveRawData() void reserveRawData (

```
RawDataHandle Raw,
int Size1,
int Size2,
int Size3 )
```

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

Parameters

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

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

```
6.4.4.46 resizeColoredData() void resizeColoredData (
    ColoredDataHandle ColData,
    int Size1,
    int Size2,
    int Size3 )
```

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

Parameters

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

```
6.4.4.47 resizeComplexData() void resizeComplexData (
    ComplexDataHandle Data,
    int Size1,
    int Size2,
    int Size3 )
```

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

Parameters

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

```
6.4.4.48 resizeData() void resizeData (
    DataHandle Data,
    int Size1,
    int Size2,
    int Size3 )
```

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

Parameters

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

Parameters

in	<i>Size1</i>	The desired number of data along the first axis ("z" in the default orientation).
in	<i>Size2</i>	The desired number of data along the second axis ("x" in the default orientation).
in	<i>Size3</i>	The desired number of data along the third axis ("y" in the default orientation).

6.4.4.49 resizeRawData() void resizeRawData (

```
    RawDataHandle Raw,
    int Size1,
    int Size2,
    int Size3 )
```

Resizes the specified raw data buffer accordingly.

Parameters

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

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

6.4.4.50 setApodizationSpectra() void setApodizationSpectra (

```
    RawDataHandle RawData,
    int NumberOfApoRegions,
    int * ApodizationRegions )
```

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

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

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

Parameters

in	<i>RawData</i>	A valid (non null) raw data handle (RawDataHandle).
in	<i>NumberOfApoRegions</i>	is the number of desired apodization regions.
in	<i>ApodizationRegions</i>	is an array containing $2 * \text{NumberOfApoRegions}$ elements that delimit the regions.

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

interesting positions.

```
6.4.4.51 setColoredDataContent() void setColoredDataContent (
    ColoredDataHandle ColData,
    unsigned long * NewContent )
```

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

Parameters

in	<i>ColData</i>	A valid (non null) colored data handle (ColoredDataHandle).
in	<i>NewContent</i>	A valid (non null) pointer to an integer array with the source data.

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

```
6.4.4.52 setColoredDataOrientation() void setColoredDataOrientation (
    ColoredDataHandle Data,
    DataOrientation Orientation )
```

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

Parameters

in	<i>Data</i>	A valid (non null) colored data handle (ColoredDataHandle).
in	<i>Orientation</i>	The desired orientation (DataOrientation).

```
6.4.4.53 setColoredDataRange() void setColoredDataRange (
    ColoredDataHandle Data,
    double range1,
    double range2,
    double range3 )
```

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

Parameters

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

```
6.4.4.54 setComplexDataContent() void setComplexDataContent (
```

```
ComplexDataHandle Data,
ComplexFloat * NewContent )
```

Sets the data content of the [ComplexDataHandle](#) to the content specified by the pointer.

Parameters

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

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

6.4.4.55 setComplexDataRange() void setComplexDataRange (

```
ComplexDataHandle Data,
double range1,
double range2,
double range3 )
```

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

Parameters

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

6.4.4.56 setDataContent() void setDataContent (

```
DataHandle Data,
float * NewContent )
```

Sets the data content of the data object. The data chunk pointed to by NewContent needs to be of the size expected by the data object, i. e. $\text{Size1} \times \text{Size2} \times \text{Size} \times \text{sizeof}(\text{float})$.

Parameters

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

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

6.4.4.57 setDataOrientation() void setDataOrientation (

```
DataHandle Data,
DataOrientation Orientation )
```

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

Parameters

in	<i>Data</i>	A valid (non null) data handle (DataHandle).
in	<i>Orientation</i>	The desired orientation.

6.4.4.58 setDataRange() `void setDataRange (`

```
    DataHandle Data,
    double range1,
    double range2,
    double range3 )
```

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

Parameters

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

6.4.4.59 setRawDataBytesPerPixel() `void setRawDataBytesPerPixel (`

```
    RawDataHandle Raw,
    int BytesPerPixel )
```

Sets the bytes per pixel for raw data.

Parameters

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

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

6.4.4.60 setRawDataContent() `void setRawDataContent (`

```
    RawDataHandle RawData,
    void * NewContent )
```

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

Parameters

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

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

```
6.4.4.61 setScanSpectra() void setScanSpectra (
    RawDataHandle RawData,
    int NumberOfScanRegions,
    int * ScanRegions )
```

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

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

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

Parameters

in	<i>RawData</i>	A valid (non null) raw data handle (RawDataHandle).
in	<i>NumberOfScanRegions</i>	is the number of desired scan regions.
in	<i>ScanRegions</i>	is an array containing $2*\text{NumberOfScanRegions}$ elements that delimit the regions.

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

```
6.4.4.62 thresholdDopplerData() void thresholdDopplerData (
    DataHandle Phase,
    DataHandle Intensity,
    float intensityThreshold,
    float phaseTargetValue )
```

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

6.5 Data Creation and Clearing

Functions to create and clear object containing data.

Functions

- **SPECTRALRADAR_API** RawDataHandle `createRawData (void)`
Notice that raw data refers to the spectra as acquired, without processing of any kind.
- **SPECTRALRADAR_API** void `clearRawData (RawDataHandle Raw)`
Notice that raw data refers to the spectra as acquired, without processing of any kind.
- **SPECTRALRADAR_API** DataHandle `createData (void)`
Creates a 1-dimensional data object, containing floating point data.
- **SPECTRALRADAR_API** DataHandle `createGradientData (int Size)`
Creates a 1-dimensional data object, containing floating point data with equidistant arranged values between [0, size-1] with distance 1/(size-1).
- **SPECTRALRADAR_API** void `clearData (DataHandle Data)`
Clears the specified `DataHandle` object.
- **SPECTRALRADAR_API** ColoredDataHandle `createColoredData (void)`
Creates a colored data object (`ColoredDataHandle`).
- **SPECTRALRADAR_API** void `clearColoredData (ColoredDataHandle Volume)`
Clears a colored volume object.
- **SPECTRALRADAR_API** ComplexDataHandle `createComplexData (void)`
Creates a data object holding complex data (`ComplexDataHandle`).
- **SPECTRALRADAR_API** void `clearComplexData (ComplexDataHandle Data)`
Clears a data object holding complex data (`ComplexDataHandle`).

6.5.1 Detailed Description

Functions to create and clear object containing data.

6.5.2 Function Documentation

6.5.2.1 `clearColoredData()` void `clearColoredData (ColoredDataHandle Volume)`

Clears a colored volume object.

Parameters

in	<code>Volume</code>	A colored data handle (<code>ColoredDataHandle</code>). If the handle is a nullptr, this function does nothing.
----	---------------------	---

6.5.2.2 `clearComplexData()` void `clearComplexData (ComplexDataHandle Data)`

Clears a data object holding complex data ([ComplexDataHandle](#)).

Parameters

in	<i>Data</i>	A complex data handle (ComplexDataHandle). If the handle is a nullptr, this function does nothing.
----	-------------	--

6.5.2.3 clearData() `void clearData (`
`DataHandle Data)`

Clears the specified [DataHandle](#) object.

Parameters

in	<i>Data</i>	A data handle (DataHandle). If the handle is a nullptr, this function does nothing.
----	-------------	---

6.5.2.4 clearRawData() `void clearRawData (`
`RawDataHandle Raw)`

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

Clears a raw data object ([RawDataHandle](#))

Parameters

in	<i>Raw</i>	A raw data handle. If the handle is a nullptr, this function does nothing.
----	------------	--

6.5.2.5 createColoredData() `ColoredDataHandle createColoredData (`
`void)`

Creates a colored data object ([ColoredDataHandle](#)).

Returns

A valid colored data handle ([ColoredDataHandle](#)).

6.5.2.6 createComplexData() `ComplexDataHandle createComplexData (`
`void)`

Creates a data object holding complex data ([ComplexDataHandle](#)).

Returns

A valid complex data handle ([ComplexDataHandle](#)).

6.5.2.7 createData() `DataHandle createData (`
 `void)`

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

Returns

A valid data handle ([DataHandle](#)).

6.5.2.8 createGradientData() `DataHandle createGradientData (`
 `int Size)`

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

Parameters

in	Size	Data number.
----	------	--------------

Returns

A valid data handle ([DataHandle](#)).

6.5.2.9 createRawData() `RawDataHandle createRawData (`
 `void)`

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

Creates a raw data object ([RawDataHandle](#)).

Returns

A valid raw data handle.

6.6 Internal Values

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

Functions

- **SPECTRALRADAR_API** int `getNumberOfInternalDeviceValues` (`OCTDeviceHandle Dev`)

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

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

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

Returns the value of the selected ADC.

6.6.1 Detailed Description

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

6.6.2 Function Documentation

6.6.2.1 `getInternalDeviceValueByIndex()` double `getInternalDeviceValueByIndex` (

```
OCTDeviceHandle Dev,
int Index )
```

Returns the value of the selected ADC.

Parameters

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

Returns

The internal device value.

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

6.6.2.2 `getInternalDeviceValueByName()` double `getInternalDeviceValueByName` (

```
OCTDeviceHandle Dev,
const char * Name )
```

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

Parameters

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

Returns

The internal device value.

The ADC is specified by the name returned by [getInternalDeviceValueName](#).

6.6.2.3 getInternalDeviceValueName() `void getInternalDeviceValueName (`

```
OCTDeviceHandle Dev,
int Index,
char * Name,
int NameStringSize,
char * Unit,
int UnitStringSize )
```

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

Parameters

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

The index is a running number, starting with 0, smaller than the number specified by [getNumberOfInternalDeviceValues](#).

6.6.2.4 getNumberOfInternalDeviceValues() `int getNumberOfInternalDeviceValues (`

```
OCTDeviceHandle Dev )
```

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

Returns

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

Parameters

in	<i>Dev</i>	A valid (non null) OCT device handle (OCTDeviceHandle), previously generated with the function initDevice .
----	------------	---

6.7 Output Values (digital or analog)

Functions to inquire, setup and generate output values. Whether this functionality is supported, and to what extent, depends on the hardware.

Functions

- **SPECTRALRADAR_API** int `getNumberOfOutputDeviceValues` (`OCTDeviceHandle Dev`)
Returns the number of output values.
- **SPECTRALRADAR_API** void `getOutputDeviceValueName` (`OCTDeviceHandle Dev, int Index, char *Name, int NameStringSize, char *Unit, int UnitStringSize`)
Returns names and units of the requested output values.
- **SPECTRALRADAR_API** BOOL `doesOutputDeviceValueExist` (`OCTDeviceHandle Dev, const char *Name`)
Returns whether the requested output device values exists or not.
- **SPECTRALRADAR_API** void `setOutputDeviceValueByName` (`OCTDeviceHandle Dev, const char *Name, double value`)
Sets the specified output value.
- **SPECTRALRADAR_API** void `setInternalDeviceValueByIndex` (`OCTDeviceHandle Dev, int Index, double Value`)
Sets the value of the selected ADC.
- **SPECTRALRADAR_API** void `setOutputDeviceValueByIndex` (`OCTDeviceHandle Dev, int Index, double Value`)
Sets the value of the selected ADC.
- **SPECTRALRADAR_API** void `getOutputDeviceValueRangeByName` (`OCTDeviceHandle Dev, const char *Name, double *Min, double *Max`)
Gives the range of the specified output value.
- **SPECTRALRADAR_API** void `getOutputDeviceValueRangeByIndex` (`OCTDeviceHandle Dev, int Index, double *Min, double *Max`)
Gives the range of the specified output value.

6.7.1 Detailed Description

Functions to inquire, setup and generate output values. Whether this functionality is supported, and to what extent, depends on the hardware.

6.7.2 Function Documentation

6.7.2.1 `doesOutputDeviceValueExist()`

```
BOOL doesOutputDeviceValueExist (
    OCTDeviceHandle Dev,
    const char * Name )
```

Returns whether the requested output device values exists or not.

```
6.7.2.2 getNumberOfOutputDeviceValues() int getNumberOfOutputDeviceValues (  
    OCTDeviceHandle Dev )
```

Returns the number of output values.

```
6.7.2.3 getOutputDeviceValueName() void getOutputDeviceValueName (   
    OCTDeviceHandle Dev,  
    int Index,  
    char * Name,  
    int NameStringSize,  
    char * Unit,  
    int UnitStringSize )
```

Returns names and units of the requested output values.

```
6.7.2.4 getOutputDeviceValueRangeByIndex() void getOutputDeviceValueRangeByIndex (   
    OCTDeviceHandle Dev,  
    int Index,  
    double * Min,  
    double * Max )
```

Gives the range of the specified output value.

```
6.7.2.5 getOutputDeviceValueRangeByName() void getOutputDeviceValueRangeByName (   
    OCTDeviceHandle Dev,  
    const char * Name,  
    double * Min,  
    double * Max )
```

Gives the range of the specified output value.

```
6.7.2.6 setInternalDeviceValueByIndex() void setInternalDeviceValueByIndex (   
    OCTDeviceHandle Dev,  
    int Index,  
    double Value )
```

Sets the value of the selected ADC.

Deprecated Please use `setOutputDeviceValueByIndex` instead.

Parameters

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

The index is running number, starting with 0, smaller than the number specified by [getNumberOfInternalDeviceValues](#).

6.7.2.7 setOutputDeviceValueByIndex() `void setOutputDeviceValueByIndex (`

```
OCTDeviceHandle Dev,  
int Index,  
double Value )
```

Sets the value of the selected ADC.

Parameters

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

The index is running number, starting with 0, smaller than the number specified by [getNumberOfInternalDeviceValues](#).

6.7.2.8 setOutputDeviceValueByName() `void setOutputDeviceValueByName (`

```
OCTDeviceHandle Dev,  
const char * Name,  
double value )
```

Sets the specified output value.

6.8 Pattern Factory/Probe

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

Typedefs

- `typedef void(__stdcall * cbProbeMessageReceived) (int)`

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

- `typedef 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_RangeMaxX,`
 `Probe_RangeMaxY,`
 `Probe_MaximumSlope_XY,`
 `Probe_SpeckleSize,`
 `Probe_ApoVoltageX,`
 `Probe_ApoVoltageY,`
 `Probe_ReferenceStageOffset,`
 `Probe_FiberOpticalPathLength_mm,`
 `Probe_ProbeOpticalPathLength_mm,`
 `Probe_ObjectiveOpticalPathLength_mm,`
 `Probe_ObjectiveFocalLength_mm }`

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

- `enum ProbeParameterString {`
 `Probe_Name,`
 `Probe_SerialNumber,`
 `Probe_Description,`
 `Probe_Objective }`

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

- `enum ProbeParameterInt {`
 `Probe_ApodizationCycles,`
 `Probe_Oversampling,`
 `Probe_Oversampling_SlowAxis,`
 `Probe_SpeckleReduction }`

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

- enum `ProbeFlag` {

 `Probe_Camerainverted_X`,

 `Probe_Camerainverted_Y`,

 `Probe_HasMEMSScanner`,

 `Probe_ApoOnlyX` }

Boolean parameters describing the behaviour of the Probe.

- enum `ObjectivePropertyString` {

 `Objective_DisplayName`,

 `Objective_Mount` }

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

- enum `ObjectivePropertyInt` {

 `Objective_RangeMaxX_mm`,

 `Objective_RangeMaxY_mm` }

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

- enum `ObjectivePropertyFloat` {

 `Objective_FocalLength_mm`,

 `Objective_OpticalPathLength` }

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

- enum `ProbeScanRangeShape` {

 `Probe_ScanRange_Rectangular`,

 `Probe_ScanRange_Round` }

The shape of the maximal valid scan range.

Functions

- **SPECTRALRADAR_API** `ProbeHandle initCurrentProbe (OCTDeviceHandle Dev)`

Initializes the probe that is currently selected in the GUI. If no probe is configured, an error is raised.
- **SPECTRALRADAR_API** `ProbeHandle initProbe (OCTDeviceHandle Dev, const char *ProbeFile)`

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

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

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

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

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

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

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

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

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

Sets a string property of the specified probe.

- **SPECTRALRADAR_API** const char * `getProbeParameterString` (`ProbeHandle` Probe, `ProbeParameterString` Selection)
Gets the desired string property of the specified probe.
- **SPECTRALRADAR_API** void `positionToProbeVoltage` (`OCTDeviceHandle` Handle, `ProbeHandle` Probe, double Position_X_mm, double Position_Y_mm, double *Volt_X, double *Volt_Y)
Convert scan position in mm to probe output voltage.
- **SPECTRALRADAR_API** const char * `getProbeType` (`ProbeHandle` Probe)
Gets the type of the specified probe.
- **SPECTRALRADAR_API** void `setProbeType` (`ProbeHandle` Probe, const char *Type)
Sets the type of the specified probe.
- **SPECTRALRADAR_API** void `closeProbe` (`ProbeHandle` Probe)
Closes the probe and frees all memory associated with it.
- **SPECTRALRADAR_API** void `CameraPixelToPosition` (`ProbeHandle` Probe, `ColoredDataHandle` Image, int PixelX, int PixelY, double *PosX, double *PosY)
Computes the physical position of a camera pixel of the video camera in the probe. It assumes a properly calibrated device.
- **SPECTRALRADAR_API** void `PositionToCameraPixel` (`ProbeHandle` Probe, `ColoredDataHandle` Image, double PosX, double PosY, int *PixelX, int *PixelY)
Computes the pixel of the video camera corresponding to a physical position. It needs to be assured that the device is properly calibrated.
- **SPECTRALRADAR_API** void `visualizeScanPatternOnDevice` (`OCTDeviceHandle` Dev, `ProbeHandle` Probe, `ScanPatternHandle` Pattern, `BOOL` ShowRawPattern)
Visualizes the scan pattern on top of the camera image; if appropriate hardware is used for visualization.
- **SPECTRALRADAR_API** void `visualizeScanPatternOnImage` (`ProbeHandle` Probe, `ScanPatternHandle` ScanPattern, `ColoredDataHandle` VideolImage)
Visualizes the scan pattern on top of the camera image; scan pattern data is written into the image.
- **SPECTRALRADAR_API** int `getNumberOfProbeConfigs` ()
Returns the number of available probe configuration files.
- **SPECTRALRADAR_API** void `getProbeConfigName` (int Index, char *ProbeName, int StringSize)
Returns the name of the specified probe configuration file.
- **SPECTRALRADAR_API** int `getNumberOfAvailableProbes` (void)
Returns the number of the available probe types.
- **SPECTRALRADAR_API** void `getAvailableProbe` (int Index, char *ProbeName, int StringSize)
Returns the name of the desired probe type.
- **SPECTRALRADAR_API** void `getProbeDisplayName` (const char *ProbeName, char *DisplayName, int StringSize)
Returns the display name for the probe name specified.
- **SPECTRALRADAR_API** void `getObjectiveDisplayName` (const char *ObjectiveName, char *DisplayName, int StringSize)
Returns the display name for the objective name specified.
- **SPECTRALRADAR_API** int `getNumberOfCompatibleObjectives` (const char *ProbeName)
Returns the number of objectives compatible with the specified objective mount.
- **SPECTRALRADAR_API** void `getCompatibleObjective` (int Index, const char *ProbeName, char *Objective, int StringSize)
Returns the name of the specified objective for the selected probe type.
- **SPECTRALRADAR_API** `ProbeScanRangeShape` `getProbeMaxScanRangeShape` (`ProbeHandle` Probe)
Returns the shape of the valid scan range for the `ProbeHandle`. All possible scan range are defined in `ProbeScanRangeShape`.
- **SPECTRALRADAR_API** void `setProbeMaxScanRangeShape` (`ProbeHandle` Probe, `ProbeScanRangeShape` Shape)
Sets the Shape of the valid scan range for the `ProbeHandle`. All possible scan-range shapes are defined in `ProbeScanRangeShape`.

- **SPECTRALRADAR_API** void `setQuadraticProbeFactors` (ProbeHandle Probe, double *QuadFactorsX, double *QuadFactorsY, int NumberOfFactors)
Sets the probe calibration factors.
- **SPECTRALRADAR_API** int `getObjectivePropertyInt` (const char *Objective, ObjectivePropertyInt Selection)
Returns the selected `ObjectivePropertyInt` for the chosen objective.
- **SPECTRALRADAR_API** double `getObjectivePropertyFloat` (const char *Objective, ObjectivePropertyFloat Selection)
Returns the selected `ObjectivePropertyFloat` for the chosen objective.
- **SPECTRALRADAR_API** const char * `getObjectivePropertyString` (const char *Objective, ObjectivePropertyString Selection)
Returns the selected `ObjectivePropertyString` for the chosen objective. Warning: The returned `const char` will only be valid until the next call to `getObjectivePropertyString`.*
- **SPECTRALRADAR_API** void `addProbeButtonCallback` (OCTDeviceHandle Dev, cbProbeMessageReceived Callback)
Registers a callback function to notify when a button on the probe has been pressed. The int parameter passed to the callback function will contain the pressed button's ID. Caution: Since the callbacks will not be called in separate threads but in the order of addition, make sure that the callback function returns as soon as possible.
- **SPECTRALRADAR_API** void `removeProbeButtonCallback` (OCTDeviceHandle Dev, cbProbeMessageReceived Callback)
Removes a previously registered probe button callback function.
- **SPECTRALRADAR_API** void `useProbeCalibration` (bool OnOff)
Enable or disable use of probe calibration. Needs to be called before `initProbe`.

6.8.1 Detailed Description

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

6.8.2 Typedef Documentation

6.8.2.1 `cbProbeMessageReceived` `typedef void(__stdcall* cbProbeMessageReceived) (int)`

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

Parameters

<code>int</code>	Zero-based ID of the pressed button
------------------	-------------------------------------

Definition at line 1085 of file [SpectralRadar_Types.h](#).

6.8.2.2 `ProbeHandle` `ProbeHandle`

Handle for controlling the galvo scanner.

Definition at line 80 of file [SpectralRadar_Handles.h](#).

6.8.3 Enumeration Type Documentation

6.8.3.1 ObjectivePropertyFloat enum `ObjectivePropertyFloat`

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

Enumerator

<code>Objective_FocalLength_mm</code>	Focal length in mm of the specified objective.
<code>Objective_OpticalPathLength</code>	Optical path length, in millimeter (without counting the focal length, multiplied by the equivalent refractive index).

Definition at line 947 of file [SpectralRadar_Types.h](#).

6.8.3.2 ObjectivePropertyInt enum `ObjectivePropertyInt`

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

Enumerator

<code>Objective_RangeMaxX_mm</code>	The maximum range in mm of the x-direction for the specified objective.
<code>Objective_RangeMaxY_mm</code>	The maximum range in mm of the y-direction for the specified objective.

Definition at line 510 of file [SpectralRadar_Properties.h](#).

6.8.3.3 ObjectivePropertyString enum `ObjectivePropertyString`

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

Enumerator

<code>Objective_DisplayName</code>	Human-readable name of the objective to display in calibration process, or as device info.
<code>Objective_Mount</code>	The mount specification is used to find the compatible probes and objectives (to be found in .ordf and .prdf files).

Definition at line 500 of file [SpectralRadar_Properties.h](#).

6.8.3.4 ProbeFlag 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.
Probe_ApoOnlyX	Boolean if the apodization position is used for the x-mirror (or x-axis) only or both axes (mirrors) are used. This parameter can be set via the Probe.ini or prdf-file only.

Definition at line 301 of file [SpectralRadar_Properties.h](#).

6.8.3.5 ProbeParameterFloat [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 $\text{PhysicalPosition} = \text{Factor} * \text{RawValue} + \text{Offset}$

Enumerator

Probe_FactorX	Factor for the x axis.
Probe_OffsetX	Offset for the x axis.
Probe_FactorY	Factor for the y axis.
Probe_OffsetY	Offset for the y axis.
Probe_FlybackTime_Sec	Flyback time of the system. This time is usually needed to get from an apodization position to scan position and vice versa.
Probe_ExpansionTime_Sec	The scanning range is extended by a number of A-scans equivalent to the expansion time.
Probe_RotationTime_Sec	The scan pattern is usually shifted by a number of A-scans equivalent to the rotation time.
Probe_ExpectedScanRate_Hz	The expected scan rate. Warning In general the expected scan rate is set during initialization of the probe with respect to the attached device. In most cases it should not be altered manually.
Probe_CameraScalingX	The px/mm ratio in X direction for the BScan overlay on the video image.
Probe_CameraOffsetX	The BScan overlay X offset in pixels.
Probe_CameraScalingY	The px/mm ratio in Y direction for the BScan overlay on the video image.
Probe_CameraOffsetY	The BScan overlay Y offset in pixels.
Probe_CameraAngle	Corrective rotation angle for the BScan overlay.
Probe_RangeMaxX	Maximum scan range in X direction.
Probe_RangeMaxY	Maximum scan range in Y direction.
Probe_MaximumSlope_XY	Maximum galvo slope (accounting for the distortion capabilities of different galvo types)

Enumerator

Probe_SpeckleSize	Speckle size to be used for scan pattern computation if speckle reduction is switched on.
Probe_ApoVoltageX	X-voltage used to acquire the apodization spectrum.
Probe_ApoVoltageY	Y-voltage used to acquire the apodization spectrum.
Probe_ReferenceStageOffset	Offset for reference stage marking the zero delay line.
Probe_FiberOpticalPathLength_mm	Optical path length, in millimeter (fiber length up to the scanner, multiplied by the refractive index)
Probe_ProbeOpticalPathLength_mm	Optical path length, in millimeter (from scanner input to objective mount, multiplied by the refractive index)
Probe_ObjectiveOpticalPathLength_mm	Optical path length, in millimeter (without counting the focal length, multiplied by the equivalent refractive index)
Probe_ObjectiveFocalLength_mm	Optical focal length, in millimeter.

Definition at line 214 of file [SpectralRadar_Properties.h](#).

6.8.3.6 ProbeParameterInt `enum ProbeParameterInt`

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

Enumerator

Probe_ApodizationCycles	The number of cycles used for apodization.
Probe_Oversampling	A factor used as oversampling.
Probe_Oversampling_SlowAxis	A factor used as oversampling of the slow scanner axis.
Probe_SpeckleReduction	Number of speckles that are scanned over for averaging. Requires Oversampling >= SpeckleReduction.

Definition at line 287 of file [SpectralRadar_Properties.h](#).

6.8.3.7 ProbeParameterString `enum ProbeParameterString`

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

Enumerator

Probe_Name	The filename. Just Probe.ini, or some other name.
Probe_SerialNumber	Serial number of the probe.
Probe_Description	Name of the probe. From this name it is possible to find out the probe definition file. A version suffix (e.g. "_V2") might be part of it. The termination ".prdf" is not part of the name.
Probe_Objective	Objective from the probe. From this string it is possible to find out the objective definition file. A version suffix (e.g. "_V2") might be part of it. The termination ".odf" is not part of the name.

Definition at line 270 of file [SpectralRadar_Properties.h](#).

6.8.3.8 ProbeScanRangeShape enum ProbeScanRangeShape

The shape of the maximal valid scan range.

Enumerator

Probe_ScanRange_Rectangular	The shape of the valid scan range for the specified objective.
Probe_ScanRange_Round	The maximum range in mm of the y-direction for the specified objective.

Definition at line 957 of file [SpectralRadar_Types.h](#).

6.8.4 Function Documentation

```
6.8.4.1 addProbeButtonCallback() void addProbeButtonCallback (
    OCTDeviceHandle Dev,
    cbProbeMessageReceived Callback )
```

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

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

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

Parameters

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

```
6.8.4.3 closeProbe() void closeProbe (
    ProbeHandle Probe )
```

Closes the probe and frees all memory associated with it.

Parameters

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

```
6.8.4.4 getAvailableProbe() void getAvailableProbe (
    int Index,
    char * ProbeName,
    int StringSize )
```

Returns the name of the desired probe type.

Parameters

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

```
6.8.4.5 getCompatibleObjective() void getCompatibleObjective (
    int Index,
    const char * ProbeName,
    char * Objective,
    int StringSize )
```

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

Parameters

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

6.8.4.6 `getNumberOfAvailableProbes()` `int getNumberOfAvailableProbes (`
 `void)`

Returns the number of the available probe types.

Returns

The number of the available probe types.

6.8.4.7 `getNumberOfCompatibleObjectives()` `int getNumberOfCompatibleObjectives (`
 `const char * ProbeName)`

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

Parameters

<code>in</code>	<code>ProbeName</code>	The name of the probe, as retrieved with the function getAvailableProbe .
-----------------	------------------------	---

Returns

The number of objectives compatible with the specified probe name.

6.8.4.8 `getNumberOfProbeConfigs()` `int getNumberOfProbeConfigs ()`

Returns the number of available probe configuration files.

6.8.4.9 `getObjectiveDisplayName()` `void getObjectiveDisplayName (`
 `const char * ObjectiveName,`
 `char * DisplayName,`
 `int StringSize)`

Returns the display name for the objective name specified.

Parameters

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

6.8.4.10 `getObjectivePropertyFloat()` `double getObjectivePropertyFloat (`

```
const char * Objective,
ObjectivePropertyFloat Selection )
```

Returns the selected [ObjectivePropertyFloat](#) for the chosen objective.

Parameters

<i>Objective</i>	Specifies the name of the objective.
<i>Selection</i>	Specifies the ObjectivePropertyFloat property.

6.8.4.11 getObjectivePropertyInt() int getObjectivePropertyInt (

```
const char * Objective,
ObjectivePropertyInt Selection )
```

Returns the selected [ObjectivePropertyInt](#) for the chosen objective.

Parameters

<i>Objective</i>	Specifies the name of the objective.
<i>Selection</i>	Specifies the ObjectivePropertyInt property.

6.8.4.12 getObjectivePropertyString() const char * getObjectivePropertyString (

```
const char * Objective,
ObjectivePropertyString Selection )
```

Returns the selected [ObjectivePropertyString](#) for the chosen objective. Warning: The returned const char* will only be valid until the next call to [getObjectivePropertyString](#).

6.8.4.13 getProbeConfigName() void getProbeConfigName (

```
int Index,
char * ProbeName,
int StringSize )
```

Returns the name of the specified probe configuration file.

Parameters

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

```
6.8.4.14 getProbeDisplayName() void getProbeDisplayName (
    const char * ProbeName,
    char * DisplayName,
    int StringSize )
```

Returns the display name for the probe name specified.

Parameters

in	<i>ProbeName</i>	Name of the probe. This string is essentially the name of the corresponding .prdf file, except that a version number and the termination should be added.
out	<i>DisplayName</i>	The string to be shown in OCTImage software.
in	<i>StringSize</i>	The length of the returned char*.

```
6.8.4.15 getProbeFlag() BOOL getProbeFlag (
```

```
    ProbeHandle Probe,
    ProbeFlag Selection )
```

Returns the selected boolean value of the specified probe.

Parameters

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

Returns

The current value of the flag.

```
6.8.4.16 getProbeMaxScanRangeShape() ProbeScanRangeShape getProbeMaxScanRangeShape (
    ProbeHandle Probe )
```

Returns the shape of the valid scan range for the [ProbeHandle](#). All possible scan range are defined in [ProbeScanRangeShape](#).

Parameters

in	<i>Probe</i>	Specified ProbeHandle .
----	--------------	---

```
6.8.4.17 getProbeParameterFloat() double getProbeParameterFloat (
    ProbeHandle Probe,
    ProbeParameterFloat Selection )
```

Gets floating point parameters of the specified probe.

Parameters

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

Returns

The current value of the parameter.

6.8.4.18 getProbeParameterInt() `int getProbeParameterInt (`

```
    ProbeHandle Probe,  
    ProbeParameterInt Selection )
```

Gets integer parameters of the specified probe.

Parameters

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

Returns

The current value of the parameter.

6.8.4.19 getProbeParameterString() `const char * getProbeParameterString (`

```
    ProbeHandle Probe,  
    ProbeParameterString Selection )
```

Gets the desired string property of the specified probe.

Parameters

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

Returns

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

6.8.4.20 `getProbeType()` `const char * getProbeType (`
`ProbeHandle Probe)`

Gets the type of the specified probe.

Returns

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

Parameters

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

6.8.4.21 `initCurrentProbe()` `ProbeHandle initCurrentProbe (`
`OCTDeviceHandle Dev)`

Initializes the probe that is currently selected in the GUI. If no probe is configured, an error is raised.

Parameters

in	<i>Dev</i>	The OCTDeviceHandle that was initially provided by initDevice . Can be NULL in case no device is initialized or available.
----	------------	--

Returns

A valid probe handle ([ProbeHandle](#)).

6.8.4.22 `initDefaultProbe()` `ProbeHandle initDefaultProbe (`
`OCTDeviceHandle Dev,`
`const char * Type,`
`const char * Objective)`

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

Parameters

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

Returns

A valid probe handle ([ProbeHandle](#)).

```
6.8.4.23 initProbe() ProbeHandle initProbe (
    OCTDeviceHandle Dev,
    const char * ProbeFile )
```

Initializes a probe specified by ProbeFile.

Parameters

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

Returns

A valid probe handle ([ProbeHandle](#)).

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

```
6.8.4.24 initProbeFromOCTFile() ProbeHandle initProbeFromOCTFile (
    OCTDeviceHandle Dev,
    OCTFileHandle File )
```

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

Parameters

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

Returns

A valid probe handle ([ProbeHandle](#)).

```
6.8.4.25 PositionToCameraPixel() void PositionToCameraPixel (
    ProbeHandle Probe,
```

```
ColoredDataHandle Image,
double PosX,
double PosY,
int * PixelX,
int * PixelY )
```

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

Parameters

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

6.8.4.26 `positionToProbeVoltage()`

```
void positionToProbeVoltage (
    OCTDeviceHandle Handle,
    ProbeHandle Probe,
    double Position_X_mm,
    double Position_Y_mm,
    double * Volt_X,
    double * Volt_Y )
```

Convert scan position in mm to probe output voltage.

Parameters

in	<i>Dev</i>	A valid (non null) OCT device handle (OCTDeviceHandle), previously generated with the function initDevice .
in	<i>Probe</i>	A handle to the probe (ProbeHandle), whose voltage is to be calculated.
in	<i>Position_X_mm</i>	X-Position in mm
in	<i>Position_Y_mm</i>	Y-Position in mm
out	<i>Volt_X</i>	X-Position in volt
out	<i>Volt_Y</i>	Y-Position in volt

6.8.4.27 `removeProbeButtonCallback()`

```
void removeProbeButtonCallback (
    OCTDeviceHandle Dev,
    cbProbeMessageReceived Callback )
```

Removes a previously registered probe button callback function.

```
6.8.4.28 saveProbe() void saveProbe (
    ProbeHandle Probe,
    const char * ProbeFile )
```

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

Parameters

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

```
6.8.4.29 setProbeMaxScanRangeShape() ProbeScanRangeShape setProbeMaxScanRangeShape (
    ProbeHandle Probe,
    ProbeScanRangeShape Shape )
```

Sets the [Shape](#) of the valid scan range for the [ProbeHandle](#). All possible scan-range shapes are defined in [ProbeScanRangeShape](#).

Parameters

in	<i>Probe</i>	Specified ProbeHandle .
in	<i>Shape</i>	the desired shape, which should be in the range defined by ProbeScanRangeShape .

```
6.8.4.30 setProbeParameterFloat() void setProbeParameterFloat (
    ProbeHandle Probe,
    ProbeParameterFloat Selection,
    double Value )
```

Sets floating point parameters of the specified probe.

Parameters

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

```
6.8.4.31 setProbeParameterInt() void setProbeParameterInt (
    ProbeHandle Probe,
```

```
    ProbeParameterInt Selection,
    int Value )
```

Sets integer parameter of the specified probe.

Parameters

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

6.8.4.32 setProbeParameterString() void setProbeParameterString (

```
    ProbeHandle Probe,
    ProbeParameterString Selection,
    const char * Value )
```

Sets a string property of the specified probe.

Parameters

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

6.8.4.33 setProbeType() void setProbeType (

```
    ProbeHandle Probe,
    const char * Type )
```

Sets the type of the specified probe.

Parameters

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

6.8.4.34 setQuadraticProbeFactors() void setQuadraticProbeFactors (

```
    ProbeHandle Probe,
    double * QuadFactorsX,
    double * QuadFactorsY,
    int NumberOfFactors )
```

Sets the probe calibration factors.

Parameters

in	<i>Probe</i>	Specified ProbeHandle .
in	<i>QuadFactorsX</i>	the specified quadratic factors for the x-axis.
in	<i>QuadFactorsY</i>	the specified quadratic factors for the y-axis.
in	<i>NumberOfFactors</i>	the number of quadratic factors and the length of the specified arrays QuadFactorsX and QuadFactorsY.

```
6.8.4.35 useProbeCalibration() void useProbeCalibration (
    bool OnOff )
```

Enable or disable use of probe calibration. Needs to be called before [initProbe](#).

Warning

Experimental: This function may change in its interface and behaviour, or even disappear in future versions.

Parameters

in	<i>OnOff</i>	True to enable probe calibration
----	--------------	----------------------------------

```
6.8.4.36 visualizeScanPatternOnDevice() void visualizeScanPatternOnDevice (
    OCTDeviceHandle Dev,
    ProbeHandle Probe,
    ScanPatternHandle Pattern,
    BOOL ShowRawPattern )
```

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

Parameters

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

```
6.8.4.37 visualizeScanPatternOnImage() void visualizeScanPatternOnImage (
    ProbeHandle Probe,
```

```
ScanPatternHandle ScanPattern,
ColoredDataHandle VideoImage )
```

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

Parameters

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

6.9 Scan Pattern

Functions that describe the movement of the Scanner during measurement.

Typedefs

- `typedef struct C_ScanPattern * ScanPatternHandle`
Handle for creating, manipulating, and discarding a scan pattern.

Enumerations

- `enum ScanPatternPropertyInt {`
`ScanPattern_SizeTotal,`
`ScanPattern_Cycles,`
`ScanPattern_SizeCycle,`
`ScanPattern_SizePreparationCycle,`
`ScanPattern_SizeImagingCycle,`
`ScanPattern_SizePreparationScan }`

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

Enum identifying different floating-type properties of the specified scan pattern.
- `enum ScanPatternAcquisitionOrder {`
`ScanPattern_AcqOrderFrameByFrame,`
`ScanPattern_AcqOrderAll }`

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

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

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

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

Functions

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

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

- **SPECTRALRADAR_API** `ScanPatternHandle createBScanPattern (ProbeHandle Probe, double Range_mm, int AScans)`

Creates a horizontal rectilinear-segment B-scan pattern that moves the galvo over a specified range.
- **SPECTRALRADAR_API** `ScanPatternHandle createBScanPatternManual (ProbeHandle Probe, double StartX_mm, double StartY_mm, double StopX_mm, double StopY_mm, int AScans)`

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

Creates an ideal B-scan pattern assuming scanners with infinite speed. No correction factors are taken into account. This is only used for internal purposes and not as a scan pattern designed to be output to the galvo drivers.
- **SPECTRALRADAR_API** `ScanPatternHandle createCirclePattern (ProbeHandle Probe, double Radius_mm, int AScans)`

Creates a circle scan pattern.
- **SPECTRALRADAR_API** `ScanPatternHandle createVolumePattern (ProbeHandle Probe, double RangeX_mm, int SizeX, double RangeY_mm, int SizeY, ScanPatternApodizationType ApoType, ScanPatternAcquisitionOrder AcqOrder)`

Creates a simple volume pattern.
- **SPECTRALRADAR_API** `ScanPatternHandle createVolumePatternEx (ProbeHandle Probe, double RangeX_mm, int SizeX, double RangeY_mm, int SizeY, double CenterX_mm, double CenterY_mm, double Angle_rad, ScanPatternApodizationType ApoType, ScanPatternAcquisitionOrder AcqOrder)`

Creates a simple volume pattern.
- **SPECTRALRADAR_API** `void updateScanPattern (ScanPatternHandle Pattern)`

Updates the specified pattern (`ScanPatternHandle`) and computes the full look-up-table.
- **SPECTRALRADAR_API** `void rotateScanPattern (ScanPatternHandle Pattern, double Angle_rad)`

Rotates the specified pattern (`ScanPatternHandle`), counter-clockwise. The rotation is relative to current angle, not to the horizontal. That is, after multiple invocations of this function the final rotation is the addition of all rotations.
- **SPECTRALRADAR_API** `void rotateScanPatternEx (ScanPatternHandle Pattern, double Angle_rad, int Index)`

Counter-clockwise rotates the scan Index (0-based, i.e. zero for the first, one for the second, and so on) of the specified volume scan pattern (`ScanPatternHandle`). The rotation is relative to current angle, not to the horizontal. That is, after multiple invocations of this function the final rotation is the addition of all rotations.
- **SPECTRALRADAR_API** `void shiftScanPattern (ScanPatternHandle Pattern, double ShiftX_mm, double ShiftY_mm)`

Shifts the specified pattern (`ScanPatternHandle`). The shift is relative to current position, not to (0,0). That is, after multiple invocations of this function the final shift is the addition of all shifts.
- **SPECTRALRADAR_API** `void shiftScanPatternEx (ScanPatternHandle Pattern, double ShiftX_mm, double ShiftY_mm, BOOL ShiftApo, int Index)`

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

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

Returns the number of points in the specified scan pattern (`ScanPatternHandle`), including apodization and flyback.
- **SPECTRALRADAR_API** `void getScanPatternLUT (ScanPatternHandle Pattern, double *VoltX, double *VoltY)`

Returns the voltages that will be applied to reach the positions to be scanned, in the specified scan pattern (`ScanPatternHandle`).
- **SPECTRALRADAR_API** `int getScanPointsSize (ScanPatternHandle Pattern)`

Returns the number of points in the specified scan pattern (`ScanPatternHandle`), including apodization and flyback.
- **SPECTRALRADAR_API** `void getScanPoints (ScanPatternHandle Pattern, double *PosX_mm, double *PosY_mm)`

Returns the position coordinates (in mm) of the points that in the specified scan pattern (`ScanPatternHandle`).
- **SPECTRALRADAR_API** `void clearScanPattern (ScanPatternHandle Pattern)`

- Clears the specified scan pattern (`ScanPatternHandle`).*
- **SPECTRALRADAR_API** `ScanPatternHandle createFreeformScanPattern2D` (`ProbeHandle` Probe, double *`PosX_mm`, double *`PosY_mm`, int `Size`, int `AScans`, `InterpolationMethod` `InterpolationMethod`, `BOOL` `CloseScanPattern`)
Creates a B-scan scan pattern of arbitrary form with equidistant sampled scan points.
 - **SPECTRALRADAR_API** `ScanPatternHandle createFreeformScanPattern2DFromLUT` (`ProbeHandle` Probe, double *`PosX_mm`, double *`PosY_mm`, int `Size`, `BOOL` `ClosedScanPattern`)
Creates a B-scan scan pattern of arbitrary form with the specified scan points. The voltages array is taken as-is, so care must be taken to use sensible values with regard to the capabilities of the utilized scanner system and to the resolution of the system resp. the desired resolution of your scan pattern.
 - **SPECTRALRADAR_API** `ScanPatternHandle createFreeformScanPattern3DFromLUT` (`ProbeHandle` Probe, double *`PosX_mm`, double *`PosY_mm`, int `AScansPerBScan`, int `NumberOfBScans`, `BOOL` `ClosedScanPattern`, `ScanPatternApodizationType` `ApoType`, `ScanPatternAcquisitionOrder` `AcqOrder`)
Creates a volume scan pattern of arbitrary form with the specified scan voltages. The voltages array is taken as-is, so care must be taken to use sensible values with regard to the capabilities of the utilized scanner system and to the resolution of the system resp. the desired resolution of your scan pattern. With this function the definition of each single scan point is required. In order to create a scan pattern specifying only the end coordinates, please consider `createFreeformScanPattern3D`.
 - **SPECTRALRADAR_API** `ScanPatternHandle createFreeformScanPattern3D` (`ProbeHandle` Probe, double *`PosX_mm`, double *`PosY_mm`, int *`ScanIndices`, int `Size`, int `NumberOfAScansPerBScan`, `InterpolationMethod` `InterpolationMethod`, `BOOL` `CloseScanPattern`, `ScanPatternApodizationType` `ApoType`, `ScanPatternAcquisitionOrder` `AcqOrder`)
Creates a volume scan pattern of arbitrary form with equidistant sampled scan points.
 - **SPECTRALRADAR_API** `void saveScanPointsToFile` (double *`ScanPosX_mm`, double *`ScanPosY_mm`, int *`ScanIndices`, int `Size`, const char *`Filename`, `ScanPointsDataFormat` `DataFormat`)
Saves the scan points and scan indices to a file with the specified `ScanPointsDataFormat`.
 - **SPECTRALRADAR_API** `int getSizeOfScanPointsFromFile` (const char *`Filename`, `ScanPointsDataFormat` `DataFormat`)
Returns the number of scan points in the specified file.
 - **SPECTRALRADAR_API** `void loadScanPointsFromFile` (double *`ScanPosX_mm`, double *`ScanPosY_mm`, int *`ScanIndices`, int `Size`, const char *`Filename`, `ScanPointsDataFormat` `DataFormat`)
Copies the scan points and scan indices from the file to the provided arrays.
 - **SPECTRALRADAR_API** `int getSizeOfScanPointsFromDataHandle` (`DataHandle` `ScanPoints`)
Returns the size of the scan points and scan indices in the `DataHandle`.
 - **SPECTRALRADAR_API** `void getScanPointsFromDataHandle` (`DataHandle` `ScanPoints`, double *`PosX_mm`, double *`PosY_mm`, int *`ScanIndices`, int `Length`)
Copies the scan points and scan indices from the `DataHandle` to the provided arrays.
 - **SPECTRALRADAR_API** `DataHandle createDataHandleFromScanPoints` (double *`PosX_mm`, double *`PosY_mm`, int *`ScanIndices`, int `Length`)
Creates a `DataHandle` from the specified scan points and corresponding indices.
 - **SPECTRALRADAR_API** `int getScanPatternPropertyInt` (`ScanPatternHandle` `ScanPattern`, `ScanPatternPropertyInt` `Property`)
Returns the specified property of the scan pattern.
 - **SPECTRALRADAR_API** `double getScanPatternPropertyFloat` (`ScanPatternHandle` `Pattern`, `ScanPatternPropertyFloat` `Selection`)
Returns the specified property of the scan pattern.
 - **SPECTRALRADAR_API** `double expectedAcquisitionTime_s` (`ScanPatternHandle` `ScanPattern`, `OCTDeviceHandle` `Dev`)
Returns the expected acquisition time of the scan pattern.
 - **SPECTRALRADAR_API** `ScanPatternAcquisitionOrder` `getScanPatternAcqOrder` (`ScanPatternHandle` `ScanPattern`)
Returns the acquisition order of the scan pattern. See definition of `ScanPatternAcquisitionOrder` for detailed information.

- **SPECTRALRADAR_API** **BOOL** `isAcqTypeForScanPatternAvailable` (`ScanPatternHandle` `ScanPattern`, `AcquisitionType` `AcqType`)

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

6.9.1 Detailed Description

Functions that describe the movement of the Scanner during measurement.

6.9.2 Typedef Documentation

6.9.2.1 `ScanPatternHandle` `ScanPatternHandle`

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

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

Definition at line 90 of file [SpectralRadar_Handles.h](#).

6.9.3 Enumeration Type Documentation

6.9.3.1 `InflationMethod` `enum InflationMethod`

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

Enumerator

<code>Inflation_NormalDirection</code>	Inflates the points to the outer normal direction.
--	--

Definition at line 198 of file [SpectralRadar_Types.h](#).

6.9.3.2 `ScanPatternAcquisitionOrder` `enum ScanPatternAcquisitionOrder`

Parameters describing the behaviour of the scan pattern.

Enumerator

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

Definition at line 174 of file [SpectralRadar_Types.h](#).

6.9.3.3 ScanPatternApodizationType [enum ScanPatternApodizationType](#)

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

Enumerator

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

Definition at line 187 of file [SpectralRadar_Types.h](#).

6.9.3.4 ScanPatternPropertyFloat [enum ScanPatternPropertyFloat](#)

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

Enumerator

ScanPattern_RangeX	The range of the scan pattern in mm for the x-direction.
ScanPattern_RangeY	The range of the scan pattern in mm for the y-direction.
ScanPattern_CenterX	the current x-center position in mm
ScanPattern_CenterY	the current y-center position in mm
ScanPattern_Angle	the current scan pattern angle in radians
ScanPattern_MeanLength_mm	the mean of the B-scan lengths of the scan pattern in mm

Definition at line 482 of file [SpectralRadar_Properties.h](#).

6.9.3.5 ScanPatternPropertyInt [enum ScanPatternPropertyInt](#)

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

Enumerator

ScanPattern_SizeTotal	Total count of trigger pulses needed for acquisition of the scan pattern once. The acquisition will start again after finishing for continuous acquisition mode.
ScanPattern_Cycles	Count of cycles for the scan pattern.
ScanPattern_SizeCycle	Count of trigger pulses needed to acquire one cycle, e.g. one B-scan in a volume scan.
ScanPattern_SizePreparationCycle	Count of trigger pulses needed before the scanning of the sample starts. The OCT beam needs to be positioned and the apodization scans used for processing need to be acquired. The flyback time is the time used to reach the position of apodization and start of scan pattern.
ScanPattern_SizeImagingCycle	Count of trigger pulses to acquire the sample depending on averaging and size-x of the scan pattern.
ScanPattern_SizePreparationScan	Count of trigger pulses needed before the first scanning of the sample starts. Some scan patterns implement a dedicated apodisation scan before the first actual scan.

Definition at line 462 of file [SpectralRadar.Properties.h](#).

6.9.3.6 ScanPointsDataFormat enum `ScanPointsDataFormat`

Selects format with the functions [loadScanPointsFromFile](#) or [saveScanPointsToFile](#) to import or export data points.

Enumerator

ScanPoints_DataFormat_TXT	Data format txt.
ScanPoints_DataFormat_RAWandSRM	Data format raw/srm pair.

Definition at line 229 of file [SpectralRadar.Types.h](#).

6.9.4 Function Documentation

6.9.4.1 clearScanPattern() `void clearScanPattern (` `ScanPatternHandle Pattern)`

Clears the specified scan pattern ([ScanPatternHandle](#)).

Parameters

in	<code>Pattern</code>	A handle of a scan pattern (ScanPatternHandle). If the handle is a nullptr, this function does nothing.
----	----------------------	---

6.9.4.2 createBScanPattern() `ScanPatternHandle createBScanPattern (`

```
ProbeHandle Probe,
double Range_mm,
int AScans )
```

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

Parameters

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

Returns

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

If a different center position is desired, one of the functions [shiftScanPattern\(\)](#), [shiftScanPatternEx\(\)](#) should be invoked afterwards, passing the scan pattern handle returned by this function.

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

6.9.4.3 createBScanPatternManual() `ScanPatternHandle createBScanPatternManual (`

```
ProbeHandle Probe,
double StartX_mm,
double StartY_mm,
double StopX_mm,
double StopY_mm,
int AScans )
```

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

Parameters

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

Returns

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

```
6.9.4.4 createCirclePattern() ScanPatternHandle createCirclePattern (
    ProbeHandle Probe,
    double Radius_mm,
    int AScans )
```

Creates a circle scan pattern.

Parameters

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

Warning

Circle patterns cannot be rotated properly.

Returns

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

```
6.9.4.5 createDataHandleFromScanPoints() DataHandle createDataHandleFromScanPoints (
    double * PosX_mm,
    double * PosY_mm,
    int * ScanIndices,
    int Length )
```

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

Parameters

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

Returns

A `DataHandle` containing the scan points and indices.

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

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

Parameters

in	<i>Probe</i>	A valid (non null) handle of a probe (ProbeHandle), previously generated by one of the functions initProbe , or initDefaultProbe .
in	<i>PosX_mm</i>	A pointer to the double array of x-positions (in mm) of the scan pattern with length <i>Size</i>
in	<i>PosY_mm</i>	A pointer to the double array of y-positions (in mm) of the scan pattern with length <i>Size</i>
in	<i>Size</i>	The length of the arrays <i>PosX_mm</i> and <i>PosY_mm</i> .
in	<i>AScans</i>	The number of A-scans in the scan pattern that will be created. The number of A-scans should be greater than <i>Size</i> .
in	<i>InterpolationMethod</i>	The interpolation method used to fill up the specified points by <i>PosX_mm</i> and <i>PosY_mm</i> to create a pattern with evenly spaced sampled points.
in	<i>CloseScanPattern</i>	Specifies whether the scan pattern should be closed (TRUE) or not (FALSE). Closing the scan pattern will lead to the same start and end point of each B-scan.

```
6.9.4.7 createFreeformScanPattern2DFromLUT() ScanPatternHandle createFreeformScanPattern2D<-
FromLUT (
    ProbeHandle Probe,
    double * PosX_mm,
    double * PosY_mm,
    int Size,
    BOOL ClosedScanPattern )
```

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

Parameters

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

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

6.9.4.8 `createFreeformScanPattern3D()` [ScanPatternHandle](#) `createFreeformScanPattern3D (`

```
    ProbeHandle Probe,
    double * PosX_mm,
    double * PosY_mm,
    int * ScanIndices,
    int Size,
    int NumberOfAScansPerBScan,
    InterpolationMethod InterpolationMethod,
    BOOL CloseScanPattern,
    ScanPatternApodizationType ApoType,
    ScanPatternAcquisitionOrder AcqOrder )
```

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

Parameters

in	<i>Probe</i>	A valid (non null) handle of a probe (ProbeHandle), previously generated by one of the functions initProbe , or initDefaultProbe .
in	<i>PosX_mm</i>	The pointer to the array of x-positions of the scan pattern with length <i>Size</i> .
in	<i>PosY_mm</i>	The pointer to the array of y-positions of the scan pattern with length <i>Size</i> .
in	<i>ScanIndices</i>	The array specifies the assignment of each point to its B-scan. It needs to have the length <i>Size</i> . The entries need to go from 0 to number of (B-scans - 1). The number of B-scans is defined with the entries of <i>ScanIndices</i> . For example, if the minimum entry is 0 (cannot be negative!) and the maximum entry is 2, there will be three B-scans in the pattern.
in	<i>Size</i>	The length of the arrays <i>PosX_mm</i> , <i>PosY_mm</i> , and <i>ScanIndices</i> .
in	<i>NumberOfAScansPerBScan</i>	The number of A-scans in each B-scan of the created scan pattern. The number of B-scans will be defined with the entries in the <i>ScanIndices</i> .
in	<i>InterpolationMethod</i>	The interpolation method used to fill up the specified points by <i>PositionsX</i> and <i>PositionsY</i> to create a pattern with evenly-spaced sampled points.
in	<i>CloseScanPattern</i>	Specifies whether the scan pattern should be closed or not. Closing the scan pattern means that each B-scan starts and stops at the same point.
in	<i>ApoType</i>	The specified method used for apodization in a volume pattern. Please see ScanPatternApodizationType for more information.
in	<i>AcqOrder</i>	The specified method used for the acquisition order in a volume pattern. Please see ScanPatternAcquisitionOrder for more information.

Returns

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

6.9.4.9 createFreeformScanPattern3DFromLUT() `ScanPatternHandle createFreeformScanPattern3D←
FromLUT (`

```
    ProbeHandle Probe,
    double * PosX_mm,
    double * PosY_mm,
    int AScansPerBScan,
    int NumberOfBScans,
    BOOL ClosedScanPattern,
    ScanPatternApodizationType ApoType,
    ScanPatternAcquisitionOrder AcqOrder )
```

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

Parameters

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

See also

[ScanPatternApodizationType](#).

Parameters

in	<i>AcqOrder</i>	The specified method used for the acquisition order in a volume pattern.
----	-----------------	--

See also

[ScanPatternAcquisitionOrder](#).

Returns

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

```
6.9.4.10 createIdealBScanPattern() ScanPatternHandle createIdealBScanPattern (
    ProbeHandle Probe,
    double Range_mm,
    int AScans )
```

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

Parameters

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

Returns

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

```
6.9.4.11 createNoScanPattern() ScanPatternHandle createNoScanPattern (
```

```
    ProbeHandle Probe,
    int AScans,
    int NumberOfScans )
```

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

Parameters

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

Returns

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

```
6.9.4.12 createVolumePattern() ScanPatternHandle createVolumePattern (
```

```
    ProbeHandle Probe,
    double RangeX_mm,
    int SizeX,
    double RangeY_mm,
    int SizeY,
```

```
ScanPatternApodizationType ApoType,
ScanPatternAcquisitionOrder AcqOrder )
```

Creates a simple volume pattern.

Parameters

in	<i>Probe</i>	A valid (non null) handle of a probe (ProbeHandle), previously generated by one of the functions initProbe , initDefaultProbe , or initProbeFromOCTFile .
in	<i>RangeX_mm</i>	The extension of the volume along the x-axis, expressed in mm, centered at the current position.
in	<i>SizeX</i>	The number of planes that cross the x-axis..
in	<i>RangeY_mm</i>	The extension of the volume along the y-axis, expressed in mm, centered at the current position.
in	<i>SizeY</i>	The number of planes that cross the y-axis.
in	<i>ApoType</i>	The apodization type decides whether one apodization suffices for the whole set of measurements in the volume, or one apodization will be measured for each B-Scan (each segment).
in	<i>AcqOrder</i>	Dictates the acquisition strategy, as explained below, which reflects the way the user wants to retrieve the acquired data.

Returns

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

A volume scan pattern is actually a stack of B-scan patterns. At creation time the stack fills a parallelepiped volume in space but the shape can be subsequently modified if the individual slices are rotated, translated, or both (see explanation of functions [rotateScanPatternEx\(\)](#), [shiftScanPatternEx\(\)](#) for more information). Notice that the individual B-scans (the slices) will always contain the segment of the laser beam that illuminates the sample (rotations and translations cannot change that).

This function creates a parallelepiped volume scan pattern and in the default orientation (first axis is the depth "z", second axis is "x", third axis is "y") the slices will be accommodated along the "y" axis. Hence, the number of slices in the stack is given by the parameter *SizeY*. Afterwards, this parameter may be retrieved by invoking the function [getScanPatternPropertyInt\(\)](#) with the argument [ScanPattern_Cycles](#) .

Depending on the setting for [ScanPatternApodizationType](#), there will be either one apodization for the entire volume ([ScanPattern_ApoOneForAll](#)) or a single apodization for each B-scan ([ScanPattern_ApoEachBScan](#)).

The volume pattern with [ScanPatternAcquisitionOrder](#) set to [ScanPattern_AcqOrderAll](#) consists of a single uninterrupted scan and all data is acquired in a single measurement. The complete volume will be returned in one raw data ([RawDataHandle](#)) by calling [getRawData\(\)](#).

Otherwise (i.e. if [ScanPatternAcquisitionOrder](#) is set to [ScanPattern_AcqOrderFrameByFrame](#)) the scan pattern consists of individual B-Scan measurements that get retrieved separately through separate invocations of [getRawData\(\)](#). In other words: The structure of the final dataset will be identical to the former case, but the stack will be returned slice-by-slice by calling [getRawData\(\)](#), once for each slice.

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

6.9.4.13 `createVolumePatternEx()` [ScanPatternHandle](#) `createVolumePatternEx (`

```
ProbeHandle Probe,
double RangeX_mm,
int SizeX,
double RangeY_mm,
int SizeY,
double CenterX_mm,
double CenterY_mm,
double Angle_rad,
```

```
ScanPatternApodizationType ApoType,
ScanPatternAcquisitionOrder AcqOrder )
```

Creates a simple volume pattern.

Parameters

in	<i>Probe</i>	A valid (non null) handle of a probe (ProbeHandle), previously generated by one of the functions initProbe , initDefaultProbe , or initProbeFromOCTFile .
in	<i>RangeX_mm</i>	The extension of the volume along the x-axis, expressed in mm, centered at the current position.
in	<i>SizeX</i>	The number of planes that cross the x-axis.
in	<i>RangeY_mm</i>	The extension of the volume along the y-axis, expressed in mm, centered at the current position.
in	<i>SizeY</i>	The number of planes that cross the y-axis.
in	<i>CenterX_mm</i>	Center of the volume pattern
in	<i>CenterY_mm</i>	Center of the volume pattern
in	<i>Angle_rad</i>	Rotation in radians of the entire scan pattern
in	<i>ApoType</i>	The apodization type decides whether one apodization suffices for the whole set of measurements in the volume, or one apodization will be measured for each B-Scan (each segment).
in	<i>AcqOrder</i>	Dictates the acquisition strategy, as explained below, which reflects the way the user wants to retrieve the acquired data.

Returns

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

A volume scan pattern is actually a stack of B-scan patterns. At creation time the stack fills a parallelepiped volume in space but the shape can be subsequently modified if the individual slices are rotated, translated, or both (see explanation of functions [rotateScanPatternEx\(\)](#), [shiftScanPatternEx\(\)](#) for more information). Notice that the individual B-scans (the slices) will always contain the segment of the laser beam that illuminates the sample (rotations and translations cannot change that).

This function creates a parallelepiped volume scan pattern and in the default orientation (first axis is the depth "z", second axis is "x", third axis is "y") the slices will be accommodated along the "y" axis. Hence, the number of slices in the stack is given by the parameter *SizeY*. Afterwards, this parameter may be retrieved by invoking the function [getScanPatternPropertyInt\(\)](#) with the argument [ScanPattern_Cycles](#).

Depending on the setting for [ScanPatternApodizationType](#), there will be either one apodization for the entire volume ([ScanPattern_ApoOneForAll](#)) or a single apodization for each B-scan ([ScanPattern_ApoEachBScan](#)).

The volume pattern with [ScanPatternAcquisitionOrder](#) set to [ScanPattern_AcqOrderAll](#) consists of a single uninterrupted scan and all data is acquired in a single measurement. The complete volume will be returned in one raw data ([RawDataHandle](#)) by calling [getRawData\(\)](#).

Otherwise (i.e. if [ScanPatternAcquisitionOrder](#) is set to [ScanPattern_AcqOrderFrameByFrame](#)) the scan pattern consists of individual B-Scan measurements that get retrieved separately through separate invocations of [getRawData\(\)](#). In other words: The structure of the final dataset will be identical to the former case, but the stack will be returned slice-by-slice by calling [getRawData\(\)](#), once for each slice.

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

```
6.9.4.14 expectedAcquisitionTime_s() double expectedAcquisitionTime_s (
    ScanPatternHandle ScanPattern,
    OCTDeviceHandle Dev )
```

Returns the expected acquisition time of the scan pattern.

6.9.4.15 getScanPatternAcqOrder() `ScanPatternAcquisitionOrder getScanPatternAcqOrder (`
`ScanPatternHandle ScanPattern)`

Returns the acquisition order of the scan pattern. See definition of [ScanPatternAcquisitionOrder](#) for detailed information.

6.9.4.16 getScanPatternLUT() `void getScanPatternLUT (`
`ScanPatternHandle Pattern,`
`double * VoltsX,`
`double * VoltsY)`

Returns the voltages that will be applied to reach the positions to be scanned, in the specified scan pattern ([ScanPatternHandle](#)).

Parameters

in	<i>Pattern</i>	A valid (non null) handle of a scan pattern.
out	<i>VoltsX</i>	A pointer to the array in which the voltage for the X-positions will be written. If a nullptr is passed, nothing will be written. Otherwise it should have space for at least the size returned by getScanPatternLUTSize() .
out	<i>VoltsY</i>	A pointer to the array in which the voltage for the Y-positions will be written. If a nullptr is passed, nothing will be written. Otherwise it should have space for at least the size returned by getScanPatternLUTSize() .

The look-up-table mentioned here is a table with the voltages that will be sent to the galvos. It is computed beforehand.

6.9.4.17 getScanPatternLUTSize() `int getScanPatternLUTSize (`
`ScanPatternHandle Pattern)`

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

Parameters

in	<i>Pattern</i>	A valid (non null) handle of a scan pattern.
----	----------------	--

Returns

The size of the look-up-table.

The look-up-table mentioned here is a table with the voltages that will be sent to the galvos. It is computed beforehand.

6.9.4.18 getScanPatternPropertyFloat() `double getScanPatternPropertyFloat (`
`ScanPatternHandle Pattern,`
`ScanPatternPropertyFloat Selection)`

Returns the specified property of the scan pattern.

```
6.9.4.19 getScanPatternPropertyInt() int getScanPatternPropertyInt (
    ScanPatternHandle ScanPattern,
    ScanPatternPropertyInt Property )
```

Returns the specified property of the scan pattern.

```
6.9.4.20 getScanPoints() int getScanPoints (
    ScanPatternHandle Pattern,
    double * PosX_mm,
    double * PosY_mm )
```

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

Parameters

in	<i>Pattern</i>	A valid (non null) handle of a scan pattern.
out	<i>PosX_mm</i>	A pointer to the array in which the X-positions (in mm) will be written. If a nullptr is passed, nothing will be written. Otherwise it should have space for at least the size returned by getScanPointsSize() .
out	<i>PosY_mm</i>	A pointer to the array in which the Y-positions (in mm) will be written. If a nullptr is passed, nothing will be written. Otherwise it should have space for at least the size returned by getScanPointsSize() .

```
6.9.4.21 getScanPointsFromDataHandle() void getScanPointsFromDataHandle (
    DataHandle ScanPoints,
    double * PosX_mm,
    double * PosY_mm,
    int * ScanIndices,
    int Length )
```

Copies the scan points and scan indices from the [DataHandle](#) to the provided arrays.

Parameters

in	<i>ScanPoints</i>	The created DataHandle containing the provided points and scan indices.
out	<i>PosX_mm</i>	The pointer to the array of X-coords of the scan pattern, with length <i>Size</i>
out	<i>PosY_mm</i>	The pointer to the array of Y-coords of the scan pattern, with length <i>Size</i>
out	<i>ScanIndices</i>	The array specifies the assignment of each point to its B-scan, with length <i>Size</i> . The entries will go from 0 to number of B-scans - 1). The number of B-scans is defined with the entries of <i>ScanIndices</i> . To save scan points for a 2D-pattern set all entries to zero.
in	<i>Length</i>	The length of the arrays <i>FreeformCoordsX</i> , <i>FreeformCoordsY</i> , and <i>ScanIndices</i> .

6.9.4.22 getScanPointsSize() `int getScanPointsSize (`
 `ScanPatternHandle Pattern)`

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

Parameters

in	<i>Pattern</i>	A valid (non null) handle of a scan pattern.
----	----------------	--

Returns

The number of points in the scan pattern, including apodization and flyback.

6.9.4.23 getSizeOfScanPointsFromDataHandle() `int getSizeOfScanPointsFromDataHandle (`
 `DataHandle ScanPoints)`

Returns the size of the scan points and scan indices in the [DataHandle](#).

Parameters

in	<i>ScanPoints</i>	The DataHandle containing the provided points and scan indices.
----	-------------------	---

Returns

The number of scan points.

Notice that in this case a data structure is used to hold data other than spectra or A-scans.

6.9.4.24 getSizeOfScanPointsFromFile() `int getSizeOfScanPointsFromFile (`
 `const char * Filename,`
 `ScanPointsDataFormat DataFormat)`

Returns the number of scan points in the specified file.

Parameters

in	<i>Filename</i>	(including path) of the file that contains the scan points and indices.
in	<i>DataFormat</i>	The desired ScanPointsDataFormat .

Returns

The number of scan points in the give file.

```
6.9.4.25 isAcqTypeForScanPatternAvailable() BOOL isAcqTypeForScanPatternAvailable (
    ScanPatternHandle ScanPattern,
    AcquisitionType AcqType )
```

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

```
6.9.4.26 loadScanPointsFromFile() void loadScanPointsFromFile (
```

```
double * ScanPosX_mm,
double * ScanPosY_mm,
int * ScanIndices,
int Size,
const char * Filename,
ScanPointsDataFormat DataFormat )
```

Copies the scan points and scan indices from the file to the provided arrays.

Parameters

out	<i>ScanPosX_mm</i>	The pointer to the double array of x-positions of the scan pattern with length <i>Size</i> in mm
out	<i>ScanPosY_mm</i>	The pointer to the double array of y-positions of the scan pattern with length <i>Size</i> in mm
out	<i>ScanIndices</i>	The array specifies the assignment of each point to its B-scan. It has the length <i>Size</i> with entries from 0 to number of (B-scans - 1). The number of B-scans is defined with the entries of <i>ScanIndices</i> . To save scan points for a 2D-pattern set all entries to zero.
in	<i>Size</i>	The length of the arrays <i>PositionsX</i> , <i>PositionsY</i> and <i>ScanIndices</i> .
in	<i>Filename</i>	Path and name of the file containing the scan points and indices.
in	<i>DataFormat</i>	The selected <i>ScanPointsDataFormat</i> .

```
6.9.4.27 rotateScanPattern() void rotateScanPattern (
```

```
ScanPatternHandle Pattern,
double Angle_rad )
```

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

Parameters

in	<i>Pattern</i>	A valid (non null) handle of a scan pattern.
in	<i>Angle_rad</i>	The angle (expressed in radians) of the rotation.

```
6.9.4.28 rotateScanPatternEx() void rotateScanPatternEx (
```

```
ScanPatternHandle Pattern,
```

```
    double Angle_rad,
    int Index )
```

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

Parameters

in	<i>Pattern</i>	A valid (non null) handle of a volume scan pattern.
in	<i>Angle_rad</i>	The angle (expressed in radians) of the counter-clockwise rotation.
in	<i>Index</i>	The slice of the stack that should be rotated.

This function is specific of volume scan patterns, although only a slice of it will be rotated. A volume scan pattern is actually a stack of B-scan patterns. In the default orientation (first axis is the depth "z", second axis is "x", third axis is "y"), the slices will be accommodated along the "y" axis. The number of slices in the stack may be retrieved by invoking the function [getScanPatternPropertyInt\(\)](#) with the argument [ScanPattern_Cycles](#).

6.9.4.29 saveScanPointsToFile() void saveScanPointsToFile (

```
    double * ScanPosX_mm,
    double * ScanPosY_mm,
    int * ScanIndices,
    int Size,
    const char * Filename,
    ScanPointsDataFormat DataFormat )
```

Saves the scan points and scan indices to a file with the specified [ScanPointsDataFormat](#).

Parameters

in	<i>ScanPosX_mm</i>	The pointer to the double array of x-positions of the scan pattern with length <i>Size</i> in mm
in	<i>ScanPosY_mm</i>	The pointer to the double array of y-positions of the scan pattern with length <i>Size</i> in mm
in	<i>ScanIndices</i>	The array specifies the assignment of each point to its B-scan. It needs to have the length <i>Size</i> with entries from 0 to number of (B-scans - 1). The number of B-scans is defined with the entries of <i>ScanIndices</i> . To save scan points for a 2D-pattern set all entries to zero.
in	<i>Size</i>	The length of the arrays <i>PositionsX</i> , <i>PositionsY</i> and <i>ScanIndices</i> .
in	<i>Filename</i>	Path and name of the file containing the scan points and indices.
in	<i>DataFormat</i>	The specified ScanPointsDataFormat .

6.9.4.30 shiftScanPattern() void shiftScanPattern (

```
    ScanPatternHandle Pattern,
    double ShiftX,
    double ShiftY )
```

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

Parameters

in	<i>Pattern</i>	A valid (non null) handle of a scan pattern.
in	<i>ShiftX</i>	The relative shift in the x-axis direction, expressed in mm.
in	<i>ShiftY</i>	The relative shift in the y-axis direction, expressed in mm.

6.9.4.31 shiftScanPatternEx() `void shiftScanPatternEx (`

```
    ScanPatternHandle Pattern,
    double ShiftX_mm,
    double ShiftY_mm,
    BOOL ShiftApo,
    int Index )
```

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

Parameters

in	<i>Pattern</i>	A valid (non null) handle of a scan pattern.
in	<i>ShiftX_mm</i>	The relative shift in the x-axis direction, expressed in mm.
in	<i>ShiftY_mm</i>	The relative shift in the y-axis direction, expressed in mm.
in	<i>ShiftApo</i>	TRUE if the apodization should also be shifted. FALSE otherwise.
in	<i>Index</i>	The slice of the stack that should be shifted.

This function is specific of volume scan patterns, although only a slice of it will be shifted. A volume scan pattern is actually a stack of B-scan patterns. In the default orientation (first axis is the depth "z", second axis is "x", third axis is "y"), the slices will be accommodated along the "y" axis. The number of slices in the stack may be retrieved by invoking the function [getScanPatternPropertyInt\(\)](#) with the argument [ScanPattern_Cycles](#).

6.9.4.32 updateScanPattern() `void updateScanPattern (`

```
    ScanPatternHandle Pattern )
```

Updates the specified pattern ([ScanPatternHandle](#)) and computes the full look-up-table.

Parameters

in	<i>Pattern</i>	A valid (non null) handle of a scan pattern.
----	----------------	--

6.9.4.33 zoomScanPattern() `void zoomScanPattern (`

```
    ScanPatternHandle Pattern,
    double Factor )
```

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

Parameters

in	<i>Pattern</i>	A valid (non null) handle of a scan pattern.
in	<i>Factor</i>	The zoom factor.

6.10 Mathematical manipulations

Functions for pure mathematical manipulations (i.e. no physics involved).

Enumerations

- enum `InterpolationMethod` {
 `Interpolation_Linear`,
 `Interpolation_Spline` }
Selects the interpolation method.
- enum `BoundaryCondition` {
 `BoundaryCondition_Standard`,
 `BoundaryCondition_Natural`,
 `BoundaryCondition_Periodic` }
Selects the boundary conditions for the interpolation.

Functions

- **SPECTRALRADAR_API** void `interpolatePoints2D` (double *OrigPosX, double *OrigPosY, int Size, double *InterpPosX, double *InterpPosY, int NewSize, `InterpolationMethod` InterpolationMet, `BoundaryCondition` BoundaryCond)
Interpolates the imaginary curve defined by the given sequence of points with the specified `InterpolationMethod`. The coordinates are abstract and this function has no sideeffects that could affect any physical property. The original and the interpolated coordinates have a meaning for the user, but no consequence for SpectralRadar.
- **SPECTRALRADAR_API** void `inflatePoints` (double *PosX, double *PosY, int Size, double *InflatedPosX, double *InflatedPosY, int NumberOfInflationLines, double RangeOfInflation, `InflationMethod` Method)
Inflates the provided curve in space with the specified `InflationMethod`. It can be used to create scan patterns of arbitrary forms with `createFreeformScanPattern3DFromLUT` if the used positions correspond to coordinates of the valid scan field in mm.
- **SPECTRALRADAR_API** void `polynomialFitAndEval1D` (int Size, const float *OrigPosX, const float *OrigY, int DegreePolynom, int EvalSize, const float *EvalPosX, float *EvalY)
Computes the polynomial fit of the given 1D data.
- **SPECTRALRADAR_API** float `calcParabolaMaximum` (float x0, float y0, float yLeft, float yRight, float *peakHeight)
Computes the x-position of the highest peak of the parabola given by the point x0, y0, yLeft, yRight. y0 needs to be the point with the highest value.

6.10.1 Detailed Description

Functions for pure mathematical manipulations (i.e. no physics involved).

6.10.2 Enumeration Type Documentation

6.10.2.1 `BoundaryCondition` enum `BoundaryCondition`

Selects the boundary conditions for the interpolation.

Enumerator

BoundaryCondition_Standard	Matches the slope of the interpolated function at starting/end point to the following/previous points.
BoundaryCondition_Natural	Natural boundary conditions used for interpolation which means the interpolated spline will turn into a straight line at the start/end.
BoundaryCondition_Periodic	Periodic boundary conditions used for interpolation which means that the interpolated function will interpret the points as a closed loop and use therefore the points from the start/end for interpolation of the end/start.

Definition at line 216 of file [SpectralRadar_Types.h](#).

6.10.2.2 InterpolationMethod enum [InterpolationMethod](#)

Selects the interpolation method.

Enumerator

Interpolation_Linear	Linear interpolation.
Interpolation_Spline	Cubic B-Spline interpolation.

Definition at line 206 of file [SpectralRadar_Types.h](#).

6.10.3 Function Documentation

6.10.3.1 calcParabolaMaximum() float calcParabolaMaximum (

```
    float x0,
    float y0,
    float yLeft,
    float yRight,
    float * peakHeight )
```

Computes the x-position of the highest peak of the parabola given by the point x0, y0, yLeft, yRight. y0 needs to be the point with the highest value.

Parameters

x0	The x-position of the point with the highest value y0.
y0	The value of x0.
yLeft	The y-value from the point left to x0. The distance (x0, xLeft) is assumed to be 1.
yRight	The y-value from the point right to x0. The distance (x0, xRight) is assumed to be 1.
peakHeight	The y-value of the highest peak of the parabola will be written to this parameter.

```
6.10.3.2 inflatePoints() void inflatePoints (
    double * PosX,
    double * PosY,
    int Size,
    double * InflatedPosX,
    double * InflatedPosY,
    int NumberOfInflationLines,
    double RangeOfInflation,
    InflationMethod Method )
```

Inflates the provided curve in space with the specified **InflationMethod**. It can be used to create scan patterns of arbitrary forms with [createFreeformScanPattern3DFromLUT](#) if the used positions correspond to coordinates of the valid scan field in mm.

Parameters

in	<i>PosX</i>	The pointer to the double array of x-positions of the scan pattern with length <i>Size</i> .
in	<i>PosY</i>	The pointer to the double array of y-positions of the scan pattern with length <i>Size</i> .
in	<i>Size</i>	The length of the arrays PositionsX, PositionsY and ScanIndices.
out	<i>InflatedPosX</i>	The pointer to the double array of x-positions of the scan pattern with length <i>Size</i> * <i>NumberOfInflationLines</i>
out	<i>InflatedPosY</i>	The pointer to the double array of y-positions of the scan pattern with length <i>Size</i> * <i>NumberOfInflationLines</i>
in	<i>NumberOfInflationLines</i>	The number of inflation lines. Please note that the length of the arrays InflatedPointsX and InflatedPointsY need to match.
in	<i>RangeOfInflation</i>	The range of inflation which results in the width of the created data object.
in	<i>Method</i>	The specified InflationMethod .

```
6.10.3.3 interpolatePoints2D() void interpolatePoints2D (
    double * OrigPosX,
    double * OrigPosY,
    int Size,
    double * InterpPosX,
    double * InterpPosY,
    int NewSize,
    InterpolationMethod InterpolationMet,
    BoundaryCondition BoundaryCond )
```

Interpolates the imaginary curve defined by the given sequence of points with the specified **InterpolationMethod**. The coordinates are abstract and this function has no sideeffects that could affect any physical property. The original and the interpolated coordinates have a meaning for the user, but no consequence for SpectralRadar.

Parameters

in	<i>OrigPosX</i>	A pointer to the array of x-coords with length <i>Size</i> .
in	<i>OrigPosY</i>	A pointer to the array of y-coords with length <i>Size</i> .
in	<i>Size</i>	The length of the arrays PositionsX, PositionsY and ScanIndices.
out	<i>InterpPosX</i>	A pointer to the array of x-coords whose length should be <i>NewSize</i> .
out	<i>InterpPosY</i>	A pointer to the array of y-coords whose length should be <i>NewSize</i> .
in	<i>NewSize</i>	The number of interpolated points.

Parameters

in	<i>InterpolationMet</i>	The desired InterpolationMethod .
in	<i>BoundaryCond</i>	The desired BoundaryCondition .

6.10.3.4 polynomialFitAndEval1D() `void polynomialFitAndEval1D (`

```
    int Size,
    const float * OrigPosX,
    const float * OrigY,
    int DegreePolynom,
    int EvalSize,
    const float * EvalPosX,
    float * EvalY )
```

Computes the polynomial fit of the given 1D data.

Parameters

<i>Size</i>	The size of the arrays OrigPosX and OrigY
<i>OrigPosX</i>	The x-positions of the OrigY of the given data.
<i>OrigY</i>	The y-values to the belonging OrigPosX of the given data.
<i>DegreePolynom</i>	The degree of the polynomial for the fit.
<i>EvalSize</i>	The size of the array EvalPosX.
<i>EvalPosX</i>	The x-positions for evaluation the polynomial fit.
<i>EvalY</i>	The resulting y-values belonging to the given positions EvalPosX.

6.11 Acquisition

Functions for acquisition.

Enumerations

- enum `AcquisitionType` {
 `Acquisition_AsyncContinuous`,
 `Acquisition_AsyncFinite`,
 `Acquisition_Sync` }

Determines the kind of acquisition process. The type of acquisition process affects e.g. whether consecutive B-scans are acquired or if it is possible to lose some data.

Functions

- `SPECTRALRADAR_API size_t projectMemoryRequirement (OCTDeviceHandle Handle, ScanPatternHandle Pattern, AcquisitionType type)`
Returns the size of the required memory, e.g. for a raw data object, in bytes to acquire the scan pattern once.
- `SPECTRALRADAR_API void startMeasurement (OCTDeviceHandle Dev, ScanPatternHandle Pattern, AcquisitionType Type)`
starts a continuous measurement BScans.
- `SPECTRALRADAR_API void getRawData (OCTDeviceHandle Dev, RawDataHandle RawData)`
Acquires data and stores the data unprocessed.
- `SPECTRALRADAR_API void getRawDataEx (OCTDeviceHandle Dev, RawDataHandle RawData, int Cameraldx)`
Acquires data with the specific camera given with camera index and stores the data unprocessed.
- `SPECTRALRADAR_API void getAnalogInputData (OCTDeviceHandle Dev, DataHandle Output)`
Acquires analog data and stores the result floating point voltages. Data from apodisation and flyback regions will be discarded.
- `SPECTRALRADAR_API void getAnalogInputDataEx (OCTDeviceHandle Dev, DataHandle Data, DataHandle ApoData)`
Acquires analog data and stores the result floating point voltages. Data from flyback regions will be discarded, data from apodisation regions will be stored in ApoData.
- `SPECTRALRADAR_API void setAnalogInputChannelEnabled (OCTDeviceHandle Dev, int ChannelIndex, bool Enable)`
Enables or disables an analog input channel. Use `getDevicePropertyInt` with flag `Device_NumOfAnalogInputChannels` to determine the number of available channels.
- `SPECTRALRADAR_API bool getAnalogInputChannelEnabled (OCTDeviceHandle Dev, int ChannelIndex)`
Returns the current status (enabled/disabled) of an analog input channel. Use `getDevicePropertyInt` with flag `Device_NumOfAnalogInputChannels` to determine the number of available channels.
- `SPECTRALRADAR_API const char * getAnalogInputChannelName (OCTDeviceHandle Dev, int ChannelIndex)`
Returns the name of an analog input channel. Use `getDevicePropertyInt` with flag `Device_NumOfAnalogInputChannels` to determine the number of available channels.
- `SPECTRALRADAR_API void stopMeasurement (OCTDeviceHandle Dev)`
stops the current measurement.
- `SPECTRALRADAR_API void measureSpectra (OCTDeviceHandle Dev, int NumberOfSpectra, RawDataHandle Raw)`
Acquires the desired number of spectra (raw data without processing) without moving galvo scanners.
- `SPECTRALRADAR_API void measureSpectraEx (OCTDeviceHandle Dev, int NumberOfSpectra, RawDataHandle Raw, int CameralIndex)`

Acquires the desired number of spectra (raw data without processing) without moving galvo scanners, for the desired camera.

- **SPECTRALRADAR_API** void [measureSpectraContinuousEx](#) (OCTDeviceHandle Dev, int NumberOfSpectra, RawDataHandle Raw, int CameraIndex)

Acquires the desired number of spectra (raw data without processing) without moving galvo scanners, for the desired camera. Starts a continuous acquisition in the background and uses ongoing acquisition if possible, to avoid latency from start/stop measurement.

6.11.1 Detailed Description

Functions for acquisition.

6.11.2 Enumeration Type Documentation

6.11.2.1 AcquisitionType [enum AcquisitionType](#)

Determines the kind of acquisition process. The type of acquisition process affects e.g. whether consecutive B-scans are acquired or if it is possible to lose some data.

Enumerator

Acquisition_AsyncContinuous	Specifies an asynchronous infinite/continuous measurement. With this acquisition type an infinite loop to acquire the specified scan pattern will be started and stopped with the call of stopMeasurement . Several buffers will be created internally to hold the data of the specified scan pattern several times. With this acquisition mode it is possible to lose data if the acquisition is faster than the copying from the framegrabber with getRawData . If you lose data you will always lose a whole frame, e.g. a whole B-scan. The acquisition thread runs independently from the thread for grabbing the data to acquire the data as fast as possible. To get the information whether the data of a whole scan pattern got lost please use getRawDataPropertyInt with RawData_LostFrames when grabbing the data.
Acquisition_AsyncFinite	Specifies an asynchronous finite measurement. With this acquisition type enough memory is created internally to hold the data for the whole scan pattern once. Therefore it is guaranteed to grab all the data and not losing frames. Please note that it is possible to acquire the scan pattern once only with this acquisition mode.
Acquisition_Sync	Specifies a synchronous measurement. With this acquisition mode the acquisition of the specified scan pattern will be started with the call of getRawData . You can interpret this acquisition type as a software trigger to start the measurement. To start the data acquisition externally please see the chapter in the software manual about external triggering.

Definition at line 240 of file [SpectralRadar_Types.h](#).

6.11.3 Function Documentation

```
6.11.3.1 getAnalogInputChannelEnabled() bool getAnalogInputChannelEnabled (
    OCTDeviceHandle Dev,
    int ChannelIndex )
```

Returns the current status (enabled/disabled) of an analog input channel. Use [getDevicePropertyInt](#) with flag Device_NumOfAnalogInputChannels to determine the number of available channels.

Parameters

in	<i>Dev</i>	A valid (non null) OCT device handle (OCTDeviceHandle), previously generated with the function initDevice .
in	<i>ChannelIndex</i>	Index of the analog input channel to query ///

Returns

True if channel is enabled, false otherwise

```
6.11.3.2 getAnalogInputChannelName() const char * getAnalogInputChannelName (
```

```
    OCTDeviceHandle Dev,
    int ChannelIndex )
```

Returns the name of an analog input channel. Use [getDevicePropertyInt](#) with flag Device_NumOfAnalogInputChannels to determine the number of available channels.

Parameters

in	<i>Dev</i>	A valid (non null) OCT device handle (OCTDeviceHandle), previously generated with the function initDevice .
in	<i>ChannelIndex</i>	Index of the analog input channel to query

Returns

String containing the descriptive name of the channel

```
6.11.3.3 getAnalogInputData() void getAnalogInputData (
```

```
    OCTDeviceHandle Dev,
    DataHandle Output )
```

Acquires analog data and stores the result floating point voltages. Data from apodisation and flyback regions will be discarded.

Parameters

in	<i>Dev</i>	A valid (non null) OCT device handle (OCTDeviceHandle), previously generated with the function initDevice .
in	<i>Output</i>	A valid (non null) data handle (DataHandle).

`getRawData` needs to be called before each call to `getAnalogInputData`

6.11.3.4 `getAnalogInputDataEx()` `void getAnalogInputDataEx (`
`OCTDeviceHandle Dev,`
`DataHandle Data,`
`DataHandle ApoData)`

Acquires analog data and stores the result floating point voltages. Data from flyback regions will be discarded, data from apodisation regions will be stored in `ApoData`.

Parameters

in	<code>Dev</code>	A valid (non null) OCT device handle (<code>OCTDeviceHandle</code>), previously generated with the function <code>initDevice</code> .
in	<code>Data</code>	A valid (non null) data handle (<code>DataHandle</code>).
in	<code>ApoData</code>	A valid (non null) data handle (<code>DataHandle</code>).

`getRawData` needs to be called before each call to `getAnalogInputData`

6.11.3.5 `getRawData()` `void getRawData (`
`OCTDeviceHandle Dev,`
`RawDataHandle RawData)`

Acquires data and stores the data unprocessed.

In case of a synchronic measurement, this function will trigger the data acquisition. Otherwise it will return the latest acquired data buffer. In any case, this function will block until a data buffer is available (asynchronous measurements may satisfy this requirement immediately if a previously acquired buffer has not already been consumed).

This function is equivalent to
`getRawDataEx(Dev, RawData, 0);`

. In other words, in systems with more than just one camera, this function retrieves the raw data of the first camera. Notice that raw data refers to the spectra as acquired, without processing of any kind.

Parameters

in	<code>Dev</code>	A valid (non null) OCT device handle (<code>OCTDeviceHandle</code>), previously generated with the function <code>initDevice</code> .
in	<code>RawData</code>	A valid (non null) raw data handle (<code>RawDataHandle</code>).

6.11.3.6 `getRawDataEx()` `void getRawDataEx (`
`OCTDeviceHandle Dev,`
`RawDataHandle RawData,`
`int CameraIdx)`

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

In case of a synchronic measurement, this function will trigger the data acquisition. Otherwise it will return the latest acquired data buffer. In any case, this function will block until a data buffer is available (asynchronous measurements

may satisfy this requirement immediately if a previously acquired buffer has not already been consumed). In systems with more than one camera, the hardware connections ensure that all cameras measure simultaneously. That is, they have a common trigger. The master camera (index 0) will actually trigger the measurement of all slaves. for this reason, this function should be invoked first for the master (index 0) and only afterwards for the slaves (index greater than 0). If a slave triggers first, it will wait for the master (that is, this function call will block the current execution thread). If the master triggers first, the buffer for the slave will be ready for pick up by the time the slave retrieves (without blocking).

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

Warning

{Unless the program divides the acquisition in different threads, this function should be invoked first for the master camera (`CameraIdx = 0`) and only then for the slaves. Otherwise it will block for ever.}

Parameters

in	<i>Dev</i>	A valid (non null) OCT device handle (OCTDeviceHandle), previously generated with the function initDevice .
in	<i>RawData</i>	A valid (non null) raw data handle (RawDataHandle).
in	<i>Cameraldx</i>	The camera index (0-based, i.e. zero for the first = master, one for the second, and so on).

6.11.3.7 `measureSpectra()` void measureSpectra (

```
OCTDeviceHandle Dev,
int NumberOfSpectra,
RawDataHandle Raw )
```

Acquires the desired number of spectra (raw data without processing) without moving galvo scanners.

Parameters

in	<i>Dev</i>	A valid (non null) OCT device handle (OCTDeviceHandle), previously generated with the function initDevice .
in	<i>NumberOfSpectra</i>	The desired number of spectra.
out	<i>Raw</i>	A valid (non null) handle of raw data (RawDataHandle), where the acquired spectra will be stored. The meta data (dimensions, sizes, bytes per pixel, etc.) will be adjusted automatically.

This procedure assumes that there is no any ongoing measurement process (started with the function [startMeasurement](#)). The indicated number of measurements will be carried out. The user should not stop the measurement (this function will block till the whole data is ready).

If the hardware contains more than one camera, all cameras will be triggered, because the hardware has been setup to do so. This function will return raw data only for the first camera (the master). The raw data for the slaves, acquired simultaneously, will be available for retrieval any time afterwards (the function [measureSpectraEx](#) should be used).

This function blocks till the desired number of spectra get written in the indicated buffer (`Raw`).

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

6.11.3.8 `measureSpectraContinuousEx()` void measureSpectraContinuousEx (

```
OCTDeviceHandle Dev,
```

```
int NumberOfSpectra,
RawDataHandle Raw,
int CameraIndex )
```

Acquires the desired number of spectra (raw data without processing) without moving galvo scanners, for the desired camera. Starts a continuous acquisition in the background and uses ongoing acquisition if possible, to avoid latency from start/stop measurement.

Parameters

in	<i>Dev</i>	A valid (non null) OCT device handle (OCTDeviceHandle), previously generated with the function initDevice .
in	<i>NumberOfSpectra</i>	The desired number of spectra.
out	<i>Raw</i>	A valid (non null) handle of raw data (RawDataHandle), where the acquired spectra will be stored. The meta data (dimensions, sizes, bytes per pixel, etc.) will be adjusted automatically.
in	<i>CameraIndex</i>	The camera index (0-based, i.e. zero for the first = master, one for the second, and so on).

Warning

{Unless the program divides the acquisition in different threads, this function should be invoked first for the master camera (*CameraIdx* = 0) and only then for the slaves. Otherwise it will block for ever.}

Note that calling this function will leave the device with a running acquisition. This means, that trigger signals will be generated continuously and any kind of device configuration that is not supported during an acquisition will fail. To restore the device to a default state, call [stopMeasurement\(\)](#). Additionally, this method may not support acquiring an arbitrary number of spectra, depending on the device. Use [getDevicePropertyInt](#) with [Device_MinimumSpectraPerBuffer](#) to query the minimum buffer size. If less than the minimum number of buffers are requested, the device will discard the additional spectra.

6.11.3.9 measureSpectraEx() void measureSpectraEx (

```
OCTDeviceHandle Dev,
int NumberOfSpectra,
RawDataHandle Raw,
int CameraIndex )
```

Acquires the desired number of spectra (raw data without processing) without moving galvo scanners, for the desired camera.

Parameters

in	<i>Dev</i>	A valid (non null) OCT device handle (OCTDeviceHandle), previously generated with the function initDevice .
in	<i>NumberOfSpectra</i>	The desired number of spectra.
out	<i>Raw</i>	A valid (non null) handle of raw data (RawDataHandle), where the acquired spectra will be stored. The meta data (dimensions, sizes, bytes per pixel, etc.) will be adjusted automatically.
in	<i>CameraIndex</i>	The camera index (0-based, i.e. zero for the first = master, one for the second, and so on).

Warning

{Unless the program divides the acquisition in different threads, this function should be invoked first for the master camera (`CameraIdx = 0`) and only then for the slaves. Otherwise it will block for ever.}

This procedure assumes that there is no any ongoing measurement process (started with the function [startMeasurement](#)). The indicated number of measurements will be carried out. The user should not stop the measurement (this function will block till the whole data is ready).

If the hardware contains more than one camera, all cameras will be triggered together with the first one (the master), because the hardware has been setup to do so. If `CameraIdx` is different from zero, i.e. a slave is meant, this function will retrieve the spectra measured together with the master. If those data happen to be already consumed, this function will block until the master triggers. Notice that in a single thread programming model, the program would stop execution for ever. For this reason, it is strongly advised to invoke this function first for the master (`CameraIdx = 0`) and only then for the slaves.

This function will retrieve raw data only for the selected camera. The user must invoke this function for each camera separately, but in judicious order, as explained before.

This function blocks till the desired number of spectra get written in the indicated buffer (Raw).

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

```
6.11.3.10 projectMemoryRequirement() size_t projectMemoryRequirement (
    OCTDeviceHandle Handle,
    ScanPatternHandle Pattern,
    AcquisitionType type )
```

Returns the size of the required memory, e.g. for a raw data object, in bytes to acquire the scan pattern once.

```
6.11.3.11 setAnalogInputChannelEnabled() void setAnalogInputChannelEnabled (
    OCTDeviceHandle Dev,
    int ChannelIndex,
    bool Enable )
```

Enables or disables an analog input channel. Use [getDevicePropertyInt](#) with flag `Device_NumOfAnalogInputChannels` to determine the number of available channels.

Parameters

in	<i>Dev</i>	A valid (non null) OCT device handle (OCTDeviceHandle), previously generated with the function initDevice .
in	<i>ChannelIndex</i>	Index of the analog input channel to activate or deactivate
in	<i>Enable</i>	True to enable channel, false to disable

```
6.11.3.12 startMeasurement() void startMeasurement (
    OCTDeviceHandle Dev,
    ScanPatternHandle Pattern,
    AcquisitionType Type )
```

starts a continuous measurement BScans.

Parameters

in	<i>Dev</i>	A valid (non null) OCT device handle (OCTDeviceHandle), previously generated with the function initDevice .
in	<i>Pattern</i>	A valid (non null) scan pattern handle (ScanPatternHandle).
in	<i>Type</i>	This parameter (AcquisitionType) decides whether the acquisition proceeds asynchronous (continuous or finite) or synchronic.

Scanning proceeds according to the specified scan pattern handle. In order to retrieve the acquired data, refer to the [getRawData\(\)](#) function. To stop the measuring process, invoke [stopMeasurement\(\)](#).

Synchronous measurements get triggered when the user invokes function that retrieves the data. Asynchronous measurements proceed in background, and the retrieving function returns the last available buffer that has been filled with fresh data. Asynchronous measurements can acquire a pre-specified number of buffers (finite) or continue indefinitely (continuous). If it is not possible to retrieve acquired data for a while, intermediate buffers might be skipped.

6.11.3.13 [stopMeasurement\(\)](#) void stopMeasurement (
 [OCTDeviceHandle](#) Dev)

stops the current measurement.

Parameters

in	<i>Dev</i>	A valid (non null) OCT device handle (OCTDeviceHandle), previously generated with the function initDevice .
----	------------	---

6.12 Processing

Standard Processing Routines.

Typedefs

- `typedef struct C_Processing * ProcessingHandle`
Handle for a processing routine.

Enumerations

- `enum ProcessingParameterInt {`
`Processing_SpectrumAveraging,`
`Processing_AScanAveraging,`
`Processing_BScanAveraging,`
`Processing_ZeroPadding,`
`Processing_NumberOfThreads,`
`Processing_FourierAveraging }`

Parameters that set the behaviour of the processing algorithms.

- `enum ProcessingParameterFloat {`
`Processing_ApodizationDamping,`
`Processing_MinElectrons,`
`Processing_FFTOversampling,`
`Processing_MaxSensorValue }`

Parameters that set the behaviour of the processing algorithms.

- `enum CalibrationData {`
`Calibration_OffsetErrors,`
`Calibration_ApodizationSpectrum,`
`Calibration_ApodizationVector,`
`Calibration_Dispersion,`
`Calibration_Chirp,`
`Calibration_ExtendedAdjust,`
`Calibration_FixedPattern }`

Data describing the calibration of the processing routines.

- `enum ProcessingFlag {`
`Processing_UseOffsetErrors,`
`Processing_RemoveDCSpectrum,`
`Processing_RemoveAdvancedDCSpectrum,`
`Processing_UseApodization,`
`Processing_UseScanForApodization,`
`Processing_UseUndersamplingFilter,`
`Processing_UseDispersionCompensation,`
`Processing_UseDechirp,`
`Processing_UseExtendedAdjust,`
`Processing_FullRangeOutput,`
`Processing_FilterDC,`
`Processing_UseAutocorrCompensation,`
`Processing_UseDEFR,`
`Processing_OnlyWindowing,`
`Processing_RemoveFixedPattern,`
`Processing_CalculateSaturation }`

Flags that set the behaviour of the processing algorithms.

- enum `Processing_FFTType` {
 `Processing_StandardFFT`,
 `Processing_StandardNDFT`,
 `Processing_iFFT`,
 `Processing_NFFT1`,
 `Processing_NFFT2`,
 `Processing_NFFT3`,
 `Processing_NFFT4` }

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

- enum `DispersionCorrectionType` {
 `Dispersion_None`,
 `Dispersion_QuadraticCoeff`,
 `Dispersion_Preset`,
 `Dispersion_Manual` }

To select the dispersion correction algorithm.

- enum `ApodizationWindow` {
 `Apodization_Hann` = 0,
 `Apodization_Hamming` = 1,
 `Apodization_Gauss` = 2,
 `Apodization_TaperedCosine` = 3,
 `Apodization_Blackman` = 4,
 `Apodization_BlackmanHarris` = 5,
 `Apodization_LightSourceBased` = 6,
 `Apodization_Unknown` = 999 }

To select the apodization window function.

- enum `ProcessingAveragingAlgorithm` {
 `Processing_Averaging_Min`,
 `Processing_Averaging_Mean`,
 `Processing_Averaging_Median`,
 `Processing_Averaging_Norm2`,
 `Processing_Averaging_Max`,
 `Processing_Averaging_Fourier_Min`,
 `Processing_Averaging_Fourier_Norm4`,
 `Processing_Averaging_Fourier_Max`,
 `Processing_Averaging_StandardDeviationAbs`,
 `Processing_Averaging_PhaseMatched` }

This sets the averaging algorithm to be used for processing.

- enum `ApodizationWindowParameter` {
 `ApodizationWindowParameter_Sigma`,
 `ApodizationWindowParameter_Ratio`,
 `ApodizationWindowParameter_Frequency` }

Sets certain parameters that are used by the window functions to be applied during apodization.

Functions

- `SPECTRALRADAR_API ProcessingHandle createProcessing` (int SpectrumSize, int BytesPerRawPixel, BOOL Signed, float ScalingFactor, float MinElectrons, `Processing_FFTType` Type, float FFTOversampling)
Creates processing routines with the desired properties.
- `SPECTRALRADAR_API ProcessingHandle createProcessingForDevice` (`OCTDeviceHandle` Dev)
Creates processing routines for the specified device (`OCTDeviceHandle`).
- `SPECTRALRADAR_API ProcessingHandle createProcessingForDeviceEx` (`OCTDeviceHandle` Dev, int CameralIndex)
Creates processing routines for the specified device (`OCTDeviceHandle`) with camera index.
- `SPECTRALRADAR_API ProcessingHandle createProcessingForOCTFile` (`OCTFileHandle` File)

- Creates processing routines for the specified OCT file ([OCTFileHandle](#)), such that the processing conditions are exactly the same as those when the file had been saved.*
- **SPECTRALRADAR_API** ProcessingHandle [createProcessingForOCTFileEx](#) (OCTFileHandle File, const int CameralIndex)

Creates processing routines for the specified OCT file ([OCTFileHandle](#)), such that the processing conditions are exactly the same as those when the file had been saved.

 - **SPECTRALRADAR_API** int [getInputSize](#) (ProcessingHandle Proc)

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

 - **SPECTRALRADAR_API** int [getAScanSize](#) (ProcessingHandle Proc)

Returns the number of pixels in an A-Scan that can be obtained (computed) with the given processing routines.

 - **SPECTRALRADAR_API** void [setApodizationWindow](#) (ProcessingHandle Proc, ApodizationWindow Window)

Sets the windowing function that will be used for apodization (this apodization has nothing to do with the reference spectra measured without a sample!). The selected windowing function will be used in all subsequent processings right before the fast Fourier transformation.

 - **SPECTRALRADAR_API** ApodizationWindow [getApodizationWindow](#) (ProcessingHandle Proc)

Returns the current windowing function that is being used for apodization, [ApodizationWindow](#) (this apodization is not the reference spectrum measured without a sample!).

 - **SPECTRALRADAR_API** void [setApodizationWindowParameter](#) (ProcessingHandle Proc, ApodizationWindowParameter Selection, double Value)

Sets the apodization window parameter, such as window width or ratio between constant and cosine part. Notice that this apodization is unrelated to the reference spectrum measured without a sample!

 - **SPECTRALRADAR_API** double [getApodizationWindowParameter](#) (ProcessingHandle Proc, ApodizationWindowParameter Selection)

Gets the apodization window parameter, such as window width or ratio between constant and cosine part. Notice that this apodization is unrelated to the reference spectrum measured without a sample!

 - **SPECTRALRADAR_API** void [getCurrentApodizationEdgeChannels](#) (ProcessingHandle Proc, int *LeftPix, int *RightPix)

Returns the pixel positions of the left/right edge channels of the current apodization. Here apodization refers to the reference spectra measured without sample.

 - **SPECTRALRADAR_API** void [setProcessingDechirpAlgorithm](#) (ProcessingHandle Proc, Processing_FFTType Type, float Oversampling)

Sets the algorithm to be used for dechirping the input spectra.

 - **SPECTRALRADAR_API** void [setProcessingParameterInt](#) (ProcessingHandle Proc, ProcessingParameterInt Selection, int Value)

Sets the specified integer value processing parameter.

 - **SPECTRALRADAR_API** int [getProcessingParameterInt](#) (ProcessingHandle Proc, ProcessingParameterInt Selection)

Returns the specified integer value processing parameter.

 - **SPECTRALRADAR_API** void [setProcessingParameterFloat](#) (ProcessingHandle Proc, ProcessingParameterFloat Selection, double Value)

Sets the specified floating point processing parameter.

 - **SPECTRALRADAR_API** double [getProcessingParameterFloat](#) (ProcessingHandle Proc, ProcessingParameterFloat Selection)

Gets the specified floating point processing parameter.

 - **SPECTRALRADAR_API** void [setProcessingFlag](#) (ProcessingHandle Proc, ProcessingFlag Flag, BOOL Value)

Sets the specified processing flag.

 - **SPECTRALRADAR_API** BOOL [getProcessingFlag](#) (ProcessingHandle Proc, ProcessingFlag Flag)

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

 - **SPECTRALRADAR_API** void [setProcessingAveragingAlgorithm](#) (ProcessingHandle Proc, ProcessingAveragingAlgorithm Algorithm)

Sets the algorithm that will be used for averaging during the processing.

 - **SPECTRALRADAR_API** void [setCalibration](#) (ProcessingHandle Proc, CalibrationData Selection, DataHandle Data)

- Sets the calibration data.*

 - **SPECTRALRADAR_API** void `getCalibration` (`ProcessingHandle` `Proc`, `CalibrationData` `Selection`, `DataHandle` `Data`)
Retrieves the desired calibration vector.
 - **SPECTRALRADAR_API** void `measureCalibration` (`OCTDeviceHandle` `Dev`, `ProcessingHandle` `Proc`, `CalibrationData` `Selection`)
Measures the specified calibration parameters and uses them in subsequent processing.
 - **SPECTRALRADAR_API** void `measureCalibrationEx` (`OCTDeviceHandle` `Dev`, `ProcessingHandle` `Proc`, `CalibrationData` `Selection`, int `CameraIndex`)
Measures the specified calibration parameters and uses them in subsequent processing with specified camera index.
 - **SPECTRALRADAR_API** void `measureApodizationSpectra` (`OCTDeviceHandle` `Dev`, `ProbeHandle` `Probe`, `ProcessingHandle` `Proc`)
Measures the apodization spectra in the defined apodization position and size and uses them in subsequent processing.
 - **SPECTRALRADAR_API** void `saveCalibrationDefault` (`ProcessingHandle` `Proc`, `CalibrationData` `Selection`)
Saves the selected calibration in its default path. This same default path will be used by SpectralRadar in subsequent executions to retrieve the calibration data.
 - **SPECTRALRADAR_API** void `saveCalibrationDefaultEx` (`ProcessingHandle` `Proc`, `CalibrationData` `Selection`, int `CameraIndex`)
Saves the selected calibration in its default path, for the selected camera. This same default path will be used by SpectralRadar in.
 - **SPECTRALRADAR_API** void `saveCalibration` (`ProcessingHandle` `Proc`, `CalibrationData` `Selection`, const char `*Path`)
Saves the selected calibration in the specified path.
 - **SPECTRALRADAR_API** void `loadCalibration` (`ProcessingHandle` `Proc`, `CalibrationData` `Selection`, const char `*Path`)
Will load a specified calibration file and its content will be used for subsequent processing.
- **SPECTRALRADAR_API** void `setSpectrumOutput` (`ProcessingHandle` `Proc`, `DataHandle` `Spectrum`)
Sets the location for the resulting spectral data.
- **SPECTRALRADAR_API** void `setOffsetCorrectedSpectrumOutput` (`ProcessingHandle` `Proc`, `DataHandle` `OffsetCorrectedSpectrum`)
Sets the location for the resulting offset corrected spectral data.
- **SPECTRALRADAR_API** void `setDCCorrectedSpectrumOutput` (`ProcessingHandle` `Proc`, `DataHandle` `DC←CorrectedSpectrum`)
Sets the location for the resulting DC removed spectral data.
- **SPECTRALRADAR_API** void `setApodizedSpectrumOutput` (`ProcessingHandle` `Proc`, `DataHandle` `ApodizedSpectrum`)
Sets the location for the resulting apodized spectral data.
- **SPECTRALRADAR_API** void `setComplexDataOutput` (`ProcessingHandle` `Proc`, `ComplexDataHandle` `ComplexScan`)
Sets the pointer to the resulting complex scans that will be written after subsequent processing executions.
- **SPECTRALRADAR_API** void `setProcessedDataOutput` (`ProcessingHandle` `Proc`, `DataHandle` `Scan`)
Sets the pointer to the resulting scans that will be written after subsequent processing executions.
- **SPECTRALRADAR_API** void `setColoredDataOutput` (`ProcessingHandle` `Proc`, `ColoredDataHandle` `Scan`, `ColoringHandle` `Color`)
Sets the pointer to the resulting colored scans that will be written after subsequent processing executions.
- **SPECTRALRADAR_API** void `setTransposedColoredDataOutput` (`ProcessingHandle` `Proc`, `ColoredDataHandle` `Scan`, `ColoringHandle` `Color`)
Sets the pointer to the resulting colored scans that will be written after subsequent processing executions. The orientation of the colored data will be transposed in such a way that the first axis (normally z-axis) will be the x-axis (the depth of each individual A-scan) and the second axis (normally x-axis) will be the z-axis.
- **SPECTRALRADAR_API** void `executeProcessing` (`ProcessingHandle` `Proc`, `RawDataHandle` `RawData`)

Executes the processing. The results will be stored as requested through the functions `setProcessedDataOutput()`, `setComplexDataOutput()`, `setColoredDataOutput()` (including coloring properties) and similar ones. In all cases, sizes and ranges will be adjusted automatically to the right values.

- **SPECTRALRADAR_API** void `finishProcessing` (`ProcessingHandle` Proc, `ComplexDataHandle` ComplexData)

Completes the processing. The results will be stored as requested through the functions `setProcessedDataOutput()`, `setComplexDataOutput()`, `setColoredDataOutput()` (including coloring properties) and similar ones. In all cases, sizes and ranges will be adjusted automatically to the right values.

- **SPECTRALRADAR_API** void `clearProcessing` (`ProcessingHandle` Proc)

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

- **SPECTRALRADAR_API** void `computeDispersion` (`DataHandle` Spectrum1, `DataHandle` Spectrum2, `DataHandle` Chirp, `DataHandle` Disp)

Computes the dispersion and chirp of the two provided spectra, where both spectra need to have been subjected to same dispersion mismatch. Both spectra need to have been acquired for different path length differences.

- **SPECTRALRADAR_API** void `computeDispersionByCoeff` (double Quadratic, `DataHandle` Chirp, `DataHandle` Disp)

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

- **SPECTRALRADAR_API** void `computeDispersionByImage` (`DataHandle` LinearKSpectra, `DataHandle` Chirp, `DataHandle` Disp)

Guesses the dispersion based on the spectral data specified. The spectral data needs to be linearized in wavenumber before using this function.

- **SPECTRALRADAR_API** int `getNumberOfDispersionPresets` (`ProcessingHandle` Proc)

Gets the number of dispersion presets.

- **SPECTRALRADAR_API** const char * `getDispersionPresetName` (`ProcessingHandle` Proc, int Index)

Gets the name of the dispersion preset specified with index.

- **SPECTRALRADAR_API** void `setDispersionPresetByName` (`ProcessingHandle` Proc, const char *Name)

Sets the dispersion preset specified with name.

- **SPECTRALRADAR_API** void `setDispersionPresetByIndex` (`ProcessingHandle` Proc, int Index)

Sets the dispersion preset specified with index.

- **SPECTRALRADAR_API** void `setDispersionPresets` (`ProcessingHandle` Proc, `ProbeHandle` Probe)

Sets the dispersion presets for the probe.

- **SPECTRALRADAR_API** `Processing_FFTType` `getProcessing_FFTType` (`ProcessingHandle` Proc)

Retrieve the active FFT Type.

- **SPECTRALRADAR_API** void `setDispersionCorrectionType` (`ProcessingHandle` Proc, `DispersionCorrectionType` Type)

Sets the active dispersion correction type.

- **SPECTRALRADAR_API** `DispersionCorrectionType` `getDispersionCorrectionType` (`ProcessingHandle` Proc)

Sets the active dispersion correction type.

- **SPECTRALRADAR_API** void `setDispersionQuadraticCoeff` (`ProcessingHandle` Proc, double Coeff)

Sets the coefficient for the quadratic correction of the dispersion.

- **SPECTRALRADAR_API** double `getDispersionQuadraticCoeff` (`ProcessingHandle` Proc)

Sets the coefficient for the quadratic correction of the dispersion.

- **SPECTRALRADAR_API** const char * `getCurrentDispersionPresetName` (`ProcessingHandle` Proc)

Gets the name of the active dispersion preset.

- **SPECTRALRADAR_API** void `computeLinearKRawData` (`ComplexDataHandle` ComplexDataAfterFFT, `DataHandle` LinearKData)

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

- **SPECTRALRADAR_API** void `linearizeSpectralData` (`DataHandle` SpectralIn, `DataHandle` SpectraOut, `DataHandle` Chirp)

Linearizes the spectral data using the given chirp vector.

6.12.1 Detailed Description

Standard Processing Routines.

6.12.2 Typedef Documentation

6.12.2.1 ProcessingHandle [ProcessingHandle](#)

Handle for a processing routine.

The purpose of the processing routines is to compute A-Scans (light intensity as a function of depth) from spectra (light intensity as a function of wavelength). The former is typically stored in different types of data ([DataHandle](#), [ComplexDataHandle](#), [ColoredDataHandle](#)) whereas the latter is raw data ([RawDataHandle](#)).
A handle of processing routines can be obtained with one of the functions [createProcessing](#), [createProcessingForDevice](#), [createProcessingForDeviceEx](#) or [createProcessingForOCTFile](#).

Definition at line 102 of file [SpectralRadar_Handles.h](#).

6.12.3 Enumeration Type Documentation

6.12.3.1 ApodizationWindow [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.

Definition at line 296 of file [SpectralRadar_Types.h](#).

6.12.3.2 ApodizationWindowParameter enum `ApodizationWindowParameter`

Sets certain parameters that are used by the window functions to be applied during apodization.

Enumerator

<code>ApodizationWindowParameter_Sigma</code>	Sets the width of a Gaussian apodization window.
<code>ApodizationWindowParameter_Ratio</code>	Sets the ratio of the constant to the cosine part when using a tapered cosine window.
<code>ApodizationWindowParameter_Frequency</code>	<p>Sets the corner frequency of the filter applied when using a light-source based apodization.</p> <p>Warning</p> <p>{Light source based apodization is still experimental and might contain bugs or decrease performance of the OCT system.}</p>

Definition at line 336 of file [SpectralRadar_Types.h](#).

6.12.3.3 CalibrationData enum `CalibrationData`

Data describing the calibration of the processing routines.

Enumerator

<code>Calibration_OffsetErrors</code>	Calibration vector used as offset.
<code>Calibration_ApodizationSpectrum</code>	Calibration data used as reference spectrum.
<code>Calibration_ApodizationVector</code>	Calibration data used as apodization multiplicators.
<code>Calibration_Dispersion</code>	Calibration data used to compensate for dispersion.
<code>Calibration_Chirp</code>	Calibration data used for dechirping spectral data.
<code>Calibration_ExtendedAdjust</code>	Calibration data used as extended adjust.
<code>Calibration_FixedPattern</code>	Calibration data used as fixed scan pattern data.

Definition at line 347 of file [SpectralRadar_Properties.h](#).

6.12.3.4 DispersionCorrectionType enum `DispersionCorrectionType`

To select the dispersion correction algorithm.

Enumerator

<code>Dispersion_None</code>	No software dispersion correction is used.
<code>Dispersion_QuadraticCoeff</code>	Quadratic dispersion correction is used with the specified factor in setDispersionQuadraticCoeff .
<code>Dispersion_Preset</code>	The specified dispersion preset from setDispersionPresets is used. For more information please see the documentation of setDispersionPresets .
<code>Dispersion_Manual</code>	Vector for dispersion correction needs to be supplied manually. Please use the setCalibration function.

Definition at line 282 of file [SpectralRadar_Types.h](#).

6.12.3.5 Processing_FFTType enum [Processing_FFTType](#)

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

Enumerator

Processing_StandardFFT	FFT with no dechirp algorithm applied.
Processing_StandardNDFT	Full matrix multiplication ("filter bank"). Mathematical precise dechirp, but rather slow.
Processing_iFFT	Linear interpolation prior to FFT.
Processing_NFFT1	NFFT algorithm with parameter m=1.
Processing_NFFT2	NFFT algorithm with parameter m=2.
Processing_NFFT3	NFFT algorithm with parameter m=3.
Processing_NFFT4	NFFT algorithm with parameter m=4.

Definition at line 262 of file [SpectralRadar_Types.h](#).

6.12.3.6 ProcessingAveragingAlgorithm enum [ProcessingAveragingAlgorithm](#)

This sets the averaging algorithm to be used for processing.

Warning

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

Enumerator

Processing_Averaging_Mean	Default.
---------------------------	----------

Definition at line 319 of file [SpectralRadar_Types.h](#).

6.12.3.7 ProcessingFlag 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.

Enumerator

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	Experimental: Toggles dispersion encoded full range processing mode, eliminating folding of the signal at the top. This flag is deactivated by default.
Processing_OnlyWindowing	Flag deactivating deconvolution in apodization processing, using windowing only. This flag is deactivated by default.
Processing_RemoveFixedPattern	Flag for removal of fixed pattern noise, used for swept source OCT systems. This flag is deactivated by default.
Processing_CalculateSaturation	Flag to calculate sensor saturation, used in swept source OCT systems. This flag is deactivated by default.

Definition at line 367 of file [SpectralRadar.Properties.h](#).

6.12.3.8 ProcessingParameterFloat enum [ProcessingParameterFloat](#)

Parameters that set the behaviour of the processing algorithms.

Enumerator

Processing_ApodizationDamping	Sets how much influence newly acquired apodizations have compared to older ones.
Processing_MinElectrons	Determines the minimum signal intensity on the edge channels of the spectra. Warning {Setting this value may seriously reduce performance of the system.}
Processing_FFTOversampling	FFT oversampling to applied for non-equispaced FFT processing algorithms. In general, higher values indicate better imaging quality but slower processing. If unsure, 2 is a good value in most cases.
Processing_MaxSensorValue	Largest (absolute) value that the processing will expect for raw samples.

Definition at line 333 of file [SpectralRadar.Properties.h](#).

6.12.3.9 ProcessingParameterInt enum [ProcessingParameterInt](#)

Parameters that set the behaviour of the processing algorithms.

Enumerator

Processing_SpectrumAveraging	Identifier for averaging of several subsequent spectra prior to Fourier transform.
Processing_AScanAveraging	Identifier 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 be 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.

Definition at line 315 of file [SpectralRadar.Properties.h](#).

6.12.4 Function Documentation

6.12.4.1 clearProcessing() void clearProcessing ([ProcessingHandle](#) Proc)

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

Parameters

in	Proc	A handle of the processing routines (ProcessingHandle). If the handle is a nullptr, this function does nothing. In most cases this handle will have been previously obtained through one of the functions createProcessing , createProcessingForDevice , createProcessingForDeviceEx or createProcessingForOCTFile .
----	------	--

6.12.4.2 computeDispersion() void computeDispersion ([DataHandle](#) Spectrum1, [DataHandle](#) Spectrum2, [DataHandle](#) Chirp, [DataHandle](#) Disp)

Computes the dispersion and chirp of the two provided spectra, where both spectra need to have been subjected to same dispersion mismatch. Both spectra need to have been acquired for different path length differences.

Parameters

in	<i>Spectrum1</i>	A valid (non null) handle of data (DataHandle) with an apodized spectrum, with the functions setApodizedSpectrumOutput() followed by executeProcessing() , measuring a test reflector positioned at a distance different from the one used for the second parameter.
in	<i>Spectrum2</i>	A valid (non null) handle of data (DataHandle) with an apodized spectrum, with the functions setApodizedSpectrumOutput() followed by executeProcessing() , measuring a test reflector positioned at a distance different from the one used for the first parameter.
out	<i>Chirp</i>	A valid (non null) handle of data (DataHandle) where the calculated chirp curve will be written.
out	<i>Disp</i>	A valid (non null) handle of data (DataHandle) where the calculated dispersion curve will be written.

For a detailed explanation, do please refer to the documentation of [setDispersionPresets](#).

6.12.4.3 computeDispersionByCoeff() `void computeDispersionByCoeff (`

```
    double Quadratic,
    DataHandle Chirp,
    DataHandle Disp )
```

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

Parameters

in	<i>Quadratic</i>	The leading coefficient of the second order polynomia that will define the dispersion curve.
in	<i>Chirp</i>	A valid (non null) handle of data (DataHandle) where a valid chirp curve has been stored.
out	<i>Disp</i>	A valid (non null) handle of data (DataHandle) where the calculated dispersion curve will be written.

For a detailed explanation, do please refer to the documentation of [setDispersionPresets](#).

6.12.4.4 computeDispersionByImage() `void computeDispersionByImage (`

```
    DataHandle LinearKSpectra,
    DataHandle Chirp,
    DataHandle Disp )
```

Guesses the dispersion based on the spectral data specified. The spectral data needs to be linearized in wavenumber before using this function.

Parameters

in	<i>LinearKSpectra</i>	A valid (non null) handle of data (DataHandle) where the input spectra is stored. The spectral data needs to be linearized in wavenumber (not wavelength) before using this function.
in	<i>Chirp</i>	A valid (non null) handle of data (DataHandle) where a valid chirp curve has been stored.
out	<i>Disp</i>	A valid (non null) handle of data (DataHandle) where the calculated dispersion curve will be written.

For a detailed explanation, do please refer to the documentation of [setDispersionPresets](#).

```
6.12.4.5 computeLinearKRawData() void computeLinearKRawData (
    ComplexDataHandle ComplexDataAfterFFT,
    DataHandle LinearKData )
```

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

```
6.12.4.6 createProcessing() ProcessingHandle createProcessing (
    int SpectrumSize,
    int BytesPerRawPixel,
    BOOL Signed,
    float ScalingFactor,
    float MinElectrons,
    Processing_FFTType Type,
    float FFTOversampling )
```

Creates processing routines with the desired properties.

Parameters

in	<i>SpectrumSize</i>	The number of pixels in each spectrum.
in	<i>BytesPerRawPixel</i>	The number of bytes in each pixel (e.g. two for a 12-bit resolution). Currently, 1, 2, and 4-bytes per pixel are supported. 1 and 2-bytes per pixel assume an integer representation, whereas 4-bytes per pixel assumes a single precision floating point representation.
in	<i>Signed</i>	Indicates whether the value of each pixel is signed or not. This parameter is ignored in case of floating point representations.
in	<i>ScalingFactor</i>	A multiplicative constant to transform digital levels into the number of electrons actually freed.
in	<i>MinElectrons</i>	A threshold. This value is used to identify the portions of the measured spectra (close to the edges) where the signal-to-noise ratio is too poor for any practical purposes. After the <i>ScalingFactor</i> has been applied to the digitized data (i.e. a spectrum has been measured), this threshold can be used to identify the portions near the edges that can be regarded as "near zero".
in	<i>Type</i>	Specifies the FFT algorithm (Processing_FFTType) that will combine the dechirping with the Fourier transform.
in	<i>FFTOversampling</i>	In case the selected FFT algorithm bases on oversampling, this parameter gives the factor.

Returns

A handle of the newly created processing routines ([ProcessingHandle](#)).

```
6.12.4.7 createProcessingForDevice() ProcessingHandle createProcessingForDevice (
    OCTDeviceHandle Dev )
```

Creates processing routines for the specified device ([OCTDeviceHandle](#)).

Parameters

in	<i>Dev</i>	A valid (non null) OCT device handle (OCTDeviceHandle), previously generated with the function initDevice .
----	------------	---

Returns

A handle of the newly created processing routines ([ProcessingHandle](#)).

In systems containing several cameras, there should be one set of processing routines for each camera. The reason is that each camera has its own calibration, and the calibration is an integral part of the computations. This function creates and returns a handle only for the first camera. Thus, this function is intended for systems containing a single camera. In case of systems containing several cameras, the function [createProcessingForDeviceEx](#) should be used instead.

6.12.4.8 createProcessingForDeviceEx() `ProcessingHandle createProcessingForDeviceEx (`
 `OCTDeviceHandle Dev,`
 `int CameraIndex)`

Creates processing routines for the specified device ([OCTDeviceHandle](#)) with camera index.

Parameters

in	<i>Dev</i>	A valid (non null) OCT device handle (OCTDeviceHandle), previously generated with the function initDevice .
in	<i>CameraIndex</i>	The camera index (0-based, i.e. zero for the first, one for the second, and so on).

Returns

A handle of the newly created processing routines ([ProcessingHandle](#)).

In systems containing several cameras, there should be one set of processing routines for each camera. The reason is that each camera has its own calibration, and the calibration is an integral part of the computations. This function creates and returns a handle only for the first camera. Thus, this function is intended for systems containing more than one camera. In case the second parameter (*CameraIndex*) is zero, this function is equivalent to [createProcessingForDevice](#).

6.12.4.9 createProcessingForOCTFile() `ProcessingHandle createProcessingForOCTFile (`
 `OCTFileHandle File)`

Creates processing routines for the specified OCT file ([OCTFileHandle](#)), such that the processing conditions are exactly the same as those when the file had been saved.

Parameters

in	<i>File</i>	A valid (non null) OCT file handle (OCTFileHandle).
----	-------------	---

Returns

A handle of the newly created processing routines ([ProcessingHandle](#)).

```
6.12.4.10 createProcessingForOCTFileEx() ProcessingHandle createProcessingForOCTFileEx (
    OCTFileHandle File,
    const int CameraIndex )
```

Creates processing routines for the specified OCT file ([OCTFileHandle](#)), such that the processing conditions are exactly the same as those when the file had been saved.

Parameters

in	<i>File</i>	A valid (non null) OCT file handle (OCTFileHandle).
in	<i>CameraIndex</i>	The detector index (first camera has zero index).

Returns

A handle of the newly created processing routines ([ProcessingHandle](#)).

For systems with one camera, this function fall backs to [createProcessingForOCTFile](#).

```
6.12.4.11 executeProcessing() void executeProcessing (
    ProcessingHandle Proc,
    RawDataHandle RawData )
```

Executes the processing. The results will be stored as requested through the functions [setProcessedDataOutput\(\)](#), [setComplexDataOutput\(\)](#), [setColoredDataOutput\(\)](#) (including coloring properties) and similar ones. In all cases, sizes and ranges will be adjusted automatically to the right values.

Parameters

in	<i>Proc</i>	A valid (non null) handle of the processing routines (ProcessingHandle), previously obtained through one of the functions createProcessing , createProcessingForDevice , createProcessingForDeviceEx or createProcessingForOCTFile .
in	<i>RawData</i>	A valid (non null) handle of raw data (RawDataHandle) with fresh measured data (e.g. acquired with the getRawData() function).

```
6.12.4.12 finishProcessing() void finishProcessing (
    ProcessingHandle Proc,
    ComplexDataHandle ComplexData )
```

Completes the processing. The results will be stored as requested through the functions [setProcessedDataOutput\(\)](#), [setComplexDataOutput\(\)](#), [setColoredDataOutput\(\)](#) (including coloring properties) and similar ones. In all cases, sizes and ranges will be adjusted automatically to the right values.

Parameters

in	<i>Proc</i>	A valid (non null) handle of the processing routines (ProcessingHandle), previously obtained through one of the functions createProcessing , createProcessingForDevice , createProcessingForDeviceEx or createProcessingForOCTFile .
in	<i>ComplexData</i>	A valid (non null) handle of complex data (ComplexDataHandle) with processed data (computed with executeProcessing , previously set to generate complex data).

Under normal circumstances the function `executeProcessing()` takes raw-data (a spectrum) and computes a depth profile in decibel. In some cases (e.g. PS-OCT) it is preferred to spare the computational time of the last few steps, and to compute only up to the complex data (using `setComplexDataOutput()` right before). This function may be used to transform the intermediate complex data to fully processed output at a later time, resuming from the point where the work was last stopped.

6.12.4.13 `getApodizationWindow()`

```
ApodizationWindow getApodizationWindow (
    ProcessingHandle Proc )
```

Returns the current windowing function that is being used for apodization, `ApodizationWindow` (this apodization is not the reference spectrum measured without a sample!).

Parameters

in	<i>Proc</i>	A valid (non null) handle of the processing routines (<code>ProcessingHandle</code>), previously obtained through one of the functions <code>createProcessing</code> , <code>createProcessingForDevice</code> , <code>createProcessingForDeviceEx</code> or <code>createProcessingForOCTFile</code> .
----	-------------	---

Returns

The current windowing function that is being used for apodization (`ApodizationWindow`) right before Fourier transformations.

6.12.4.14 `getApodizationWindowParameter()`

```
double getApodizationWindowParameter (
    ProcessingHandle Proc,
    ApodizationWindowParameter Selection )
```

Gets the apodization window parameter, such as window width or ratio between constant and cosine part. Notice that this apodization is unrelated to the reference spectrum measured without a sample!.

Parameters

in	<i>Proc</i>	A valid (non null) handle of the processing routines (<code>ProcessingHandle</code>), previously obtained through one of the functions <code>createProcessing</code> , <code>createProcessingForDevice</code> , <code>createProcessingForDeviceEx</code> or <code>createProcessingForOCTFile</code> .
in	<i>Selection</i>	The desired parameter whose value shall be retrieved (<code>ApodizationWindowParameter</code>).

Returns

The current value of the parameter.

6.12.4.15 `getAScanSize()`

```
int getAScanSize (
    ProcessingHandle Proc )
```

Returns the number of pixels in an A-Scan that can be obtained (computed) with the given processing routines.

Parameters

in	<i>Proc</i>	A valid (non null) handle of the processing routines (ProcessingHandle), previously obtained through one of the functions createProcessing , createProcessingForDevice , createProcessingForDeviceEx or createProcessingForOCTFile .
----	-------------	--

Returns

The number of pixels in an A-Scan that can be obtained (computed) with the given processing routines.

The returned number is identical to the number of rows in a finished B-Scan, that can also be retrieved (after the processing has been executed) by invoking one of the functions [getDataPropertyInt](#), [getComplexDataPropertyInt](#), or [getColoredDataPropertyInt](#), passing the enumeration item [Data_Size1](#) as the second parameter, and passing the respective data object ([DataHandle](#), [ComplexDataHandle](#), [ColoredDataHandle](#)) as the first parameter.

```
6.12.4.16 getCalibration() void getCalibration (
    ProcessingHandle Proc,
    CalibrationData Selection,
    DataHandle Data )
```

Retrieves the desired calibration vector.

Parameters

in	<i>Proc</i>	A valid (non null) handle of the processing routines (ProcessingHandle), previously obtained through one of the functions createProcessing , createProcessingForDevice , createProcessingForDeviceEx or createProcessingForOCTFile .
in	<i>Selection</i>	Indicates the calibration that will be set (CalibrationData).
out	<i>Data</i>	A valid handle (DataHandle) of the calibration data that will be retrieved. <i>Data</i> will be automatically resized for the data to fit in the structure.

```
6.12.4.17 getCurrentApodizationEdgeChannels() SPECTRALRADAR_API void getCurrentApodization←
EdgeChannels (
    ProcessingHandle Proc,
    int * LeftPix,
    int * RightPix )
```

Returns the pixel positions of the left/right edge channels of the current apodization. Here apodization refers to the reference spectra measured without sample.

Parameters

in	<i>Proc</i>	A valid (non null) handle of the processing routines (ProcessingHandle), previously obtained through one of the functions createProcessing , createProcessingForDevice , createProcessingForDeviceEx or createProcessingForOCTFile .
out	<i>LeftPix</i>	The address to store the position of the last pixel position, starting from the left, at which the intensity is too low for reliable computations. If a nullptr is given, nothing will be written on it.
out	<i>RightPix</i>	The address to store the position of the last pixel position, starting from the right, at which the intensity is too low for reliable computations. If a nullptr is given, nothing will be written on it.

The apodization spectra (i.e. the spectra measured without a sample) have regions, at their left and right edges, where the signal to noise ratio is too low for practical purposes. This function returns the position of the last pixel position (or channel) at which the measured intensity is insufficient for reliable computations.

Notice that the camera is upside down. Hence the right-most pixel refers to the shortest measured wavelength, and the left-most pixel refers to the longest measured wavelength.

The second and third pointers are addresses in memory managed by the user, not by SpectralRadar.

6.12.4.18 getCurrentDispersionPresetName() `const char * getCurrentDispersionPresetName (ProcessingHandle Proc)`

Gets the name of the active dispersion preset.

Parameters

in	<i>Proc</i>	A valid (non null) handle of the processing routines (ProcessingHandle), previously obtained through one of the functions createProcessing , createProcessingForDevice , createProcessingForDeviceEx or createProcessingForOCTFile .
----	-------------	--

Returns

A zero terminated string with the name of the active dispersion preset.

6.12.4.19 getDispersionCorrectionType() `DispersionCorrectionType getDispersionCorrectionType (ProcessingHandle Proc)`

Sets the active dispersion correction type.

Parameters

in	<i>Proc</i>	A valid (non null) handle of the processing routines (ProcessingHandle), previously obtained through one of the functions createProcessing , createProcessingForDevice , createProcessingForDeviceEx or createProcessingForOCTFile .
----	-------------	--

Returns

The currently active dispersion correction algorithm ([DispersionCorrectionType](#)).

6.12.4.20 getDispersionPresetName() `const char * getDispersionPresetName (ProcessingHandle Proc, int Index)`

Gets the name of the dispersion preset specified with index.

Parameters

in	<i>Proc</i>	A valid (non null) handle of the processing routines (ProcessingHandle), previously obtained through one of the functions createProcessing , createProcessingForDevice , createProcessingForDeviceEx or createProcessingForOCTFile .
in	<i>Index</i>	The index of the desired dispersion preset.

Returns

A zero terminated string with the name of the dispersion preset associated with the given index.

For a detailed explanation, do please refer to the documentation of [setDispersionPresets](#).

6.12.4.21 getDispersionQuadraticCoeff() `double getDispersionQuadraticCoeff (ProcessingHandle Proc)`

Sets the coefficient for the quadratic correction of the dispersion.

Parameters

in	<i>Proc</i>	A valid (non null) handle of the processing routines (ProcessingHandle), previously obtained through one of the functions createProcessing , createProcessingForDevice , createProcessingForDeviceEx or createProcessingForOCTFile .
----	-------------	--

Returns

The coefficient currently used for the quadratic correction of the dispersion.

6.12.4.22 getInputSize() `int getInputSize (ProcessingHandle Proc)`

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

Parameters

in	<i>Proc</i>	A valid (non null) handle of the processing routines (ProcessingHandle), previously obtained through one of the functions createProcessing , createProcessingForDevice , createProcessingForDeviceEx or createProcessingForOCTFile .
----	-------------	--

Returns

The number of pixels per spectrum.

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

6.12.4.23 `getNumberOfDispersionPresets()` int getNumberOfDispersionPresets (
 ProcessingHandle Proc)

Gets the number of dispersion presets.

Parameters

in	<i>Proc</i>	A valid (non null) handle of the processing routines (ProcessingHandle), previously obtained through one of the functions createProcessing , createProcessingForDevice , createProcessingForDeviceEx or createProcessingForOCTFile .
----	-------------	--

Returns

The number of dispersion presets.

For a detailed explanation, do please refer to the documentation of [setDispersionPresets](#).

6.12.4.24 getProcessing_FFTType() `Processing_FFTType getProcessing_FFTType (ProcessingHandle Proc)`

Retrieve the active FFT Type.

Parameters

in	<i>Proc</i>	A valid (non null) handle of the processing routines (ProcessingHandle), previously obtained through one of the functions createProcessing , createProcessingForDevice , createProcessingForDeviceEx or createProcessingForOCTFile .
----	-------------	--

Returns

the current FFT algorithm type ([Processing_FFTType](#)) that will combine the dechirping with the Fourier transform.

6.12.4.25 getProcessingFlag() `BOOL getProcessingFlag (ProcessingHandle Proc, ProcessingFlag Flag)`

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

Parameters

in	<i>Proc</i>	A valid (non null) handle of the processing routines (ProcessingHandle), previously obtained through one of the functions createProcessing , createProcessingForDevice , createProcessingForDeviceEx or createProcessingForOCTFile .
in	<i>Flag</i>	The flag whose value will be retrieved.

Returns

The current value of the flag.

```
6.12.4.26 getProcessingParameterFloat() double getProcessingParameterFloat (
    ProcessingHandle Proc,
    ProcessingParameterFloat Selection )
```

Gets the specified floating point processing parameter.

Parameters

in	<i>Proc</i>	A valid (non null) handle of the processing routines (ProcessingHandle), previously obtained through one of the functions createProcessing , createProcessingForDevice , createProcessingForDeviceEx or createProcessingForOCTFile .
in	<i>Selection</i>	The floating point parameter whose value will be retrieved.

Returns

The current value of the floating point parameter.

```
6.12.4.27 getProcessingParameterInt() int getProcessingParameterInt (
    ProcessingHandle Proc,
    ProcessingParameterInt Selection )
```

Returns the specified integer value processing parameter.

Parameters

in	<i>Proc</i>	A valid (non null) handle of the processing routines (ProcessingHandle), previously obtained through one of the functions createProcessing , createProcessingForDevice , createProcessingForDeviceEx or createProcessingForOCTFile .
in	<i>Selection</i>	The parameter whose value will be retrieved.

Returns

The current value of the integer parameter.

```
6.12.4.28 linearizeSpectralData() void linearizeSpectralData (
    DataHandle SpectraIn,
    DataHandle SpectraOut,
    DataHandle Chirp )
```

Linearizes the spectral data using the given chirp vector.

```
6.12.4.29 loadCalibration() void loadCalibration (
    ProcessingHandle Proc,
    CalibrationData Selection,
    const char * Path )
```

Will load a specified calibration file and its content will be used for subsequent processing.

Parameters

in	<i>Proc</i>	A valid (non null) handle of the processing routines (ProcessingHandle), previously obtained through one of the functions createProcessing , createProcessingForDevice , createProcessingForDeviceEx or createProcessingForOCTFile .
in	<i>Selection</i>	Indicates the calibration that will be saved (CalibrationData).
in	<i>Path</i>	A zero terminated string specifying the filename, including full path.

6.12.4.30 measureApodizationSpectra() void measureApodizationSpectra (

```
OCTDeviceHandle Dev,
ProbeHandle Probe,
ProcessingHandle Proc )
```

Measures the apodization spectra in the defined apodization position and size and uses them in subsequent processing.

Parameters

in	<i>Dev</i>	A valid (non null) OCT device handle (OCTDeviceHandle), previously generated with the function initDevice .
in	<i>Probe</i>	A valid (non null) probe handle (ProbeHandle), previously generated with the function initProbe .
in	<i>Proc</i>	A valid (non null) handle of the processing routines (ProcessingHandle), previously obtained through one of the functions createProcessing , createProcessingForDevice , createProcessingForDeviceEx or

If the hardware contains more than one camera, all cameras will be triggered, because the hardware has been setup to do so. This function will return raw data only for the first camera (the master). The raw data for the slaves, acquired simultaneously, will be available for retrieval any time afterwards (the function [measureCalibrationEx](#) should be used).

6.12.4.31 measureCalibration() void measureCalibration (

```
OCTDeviceHandle Dev,
ProcessingHandle Proc,
CalibrationData Selection )
```

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

Parameters

in	<i>Dev</i>	A valid (non null) OCT device handle (OCTDeviceHandle), previously generated with the function initDevice .
in	<i>Proc</i>	A valid (non null) handle of the processing routines (ProcessingHandle), previously obtained through one of the functions createProcessing , createProcessingForDevice , createProcessingForDeviceEx or createProcessingForOCTFile .
in	<i>Selection</i>	Indicates the calibration that will be measured (CalibrationData).

If the hardware contains more than one camera, all cameras will be triggered, because the hardware has been setup

to do so. This function will return raw data only for the first camera (the master). The raw data for the slaves, acquired simultaneously, will be available for retrieval any time afterwards (the function [measureCalibrationEx](#) should be used).

Using the parameters [Calibration_ApodizationSpectrum](#) or [Calibration_ApodizationVector](#) will acquire the apodization spectra without moving the mirrors to the apodization position. To acquire the spectra used for the processing in the apodization position use [measureApodizationSpectra](#). Please note that the apodization spectra will not be acquired in the specified apodization position from the [ProbeHandle](#).

```
6.12.4.32 measureCalibrationEx() void measureCalibrationEx (
    OCTDeviceHandle Dev,
    ProcessingHandle Proc,
    CalibrationData Selection,
    int CameraIndex )
```

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

Parameters

in	<i>Dev</i>	A valid (non null) OCT device handle (OCTDeviceHandle), previously generated with the function initDevice .
in	<i>Proc</i>	A valid (non null) handle of the processing routines (ProcessingHandle), previously obtained through one of the functions createProcessing , createProcessingForDevice , createProcessingForDeviceEx or createProcessingForOCTFile .
in	<i>Selection</i>	Indicates the calibration that will be measured (CalibrationData).
in	<i>CameraIndex</i>	The camera index (0-based, i.e. zero for the first = master, one for the second, and so on).

Warning

{Unless the program divides the acquisition in different threads, this function should be invoked first for the master camera (*CameraIdx* = 0) and only then for the slaves. Otherwise it will block for ever.}

If the hardware contains more than one camera, all cameras will be triggered together with the first one (the master), because the hardware has been setup to do so. If *CameraIdx* is different from zero, i.e. a slave is meant, this function will retrieve the spectra measured together with the master. If those data happen to be already consumed, this function will block until the master triggers. Notice that in a single thread programming model, the program would stop execution for ever. For this reason, it is strongly advised to invoke this function first for the master (*CameraIdx* = 0) and only then for the slaves.

Using the parameters [Calibration_ApodizationSpectrum](#) or [Calibration_ApodizationVector](#) will acquire the apodization spectra without moving the mirrors to the apodization position. To acquire the spectra used for the processing in the apodization position use [measureApodizationSpectra](#).

```
6.12.4.33 saveCalibration() 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 specify the default path.

Parameters

in	<i>Proc</i>	A valid (non null) handle of the processing routines (ProcessingHandle), previously obtained through one of the functions createProcessing , createProcessingForDevice , createProcessingForDeviceEx or createProcessingForOCTFile .
in	<i>Selection</i>	Indicates the calibration that will be saved (CalibrationData).
in	<i>Path</i>	A zero terminated string specifying the filename, including full path.

```
6.12.4.34 saveCalibrationDefault() void saveCalibrationDefault (
    ProcessingHandle Proc,
    CalibrationData Selection )
```

Saves the selected calibration in its default path. This same default path will be used by SpectralRadar in subsequent executions to retrieve the calibration data.

Warning

This will override your default calibration of the device.

Parameters

in	<i>Proc</i>	A valid (non null) handle of the processing routines (ProcessingHandle), previously obtained through one of the functions createProcessing , createProcessingForDevice , createProcessingForDeviceEx or createProcessingForOCTFile .
in	<i>Selection</i>	Indicates the calibration that will be saved (CalibrationData).

In systems with more than one camera, this function will only save calibration data pertaining to the first camera. For the other cameras use function [saveCalibrationDefaultEx](#).

```
6.12.4.35 saveCalibrationDefaultEx() void saveCalibrationDefaultEx (
    ProcessingHandle Proc,
    CalibrationData Selection,
    int CameraIndex )
```

Saves the selected calibration in its default path, for the selected camera. This same default path will be used by SpectralRadar in.

subsequent executions to retrieve the calibration data.

Warning

This will override your default calibration of the device.

Parameters

in	<i>Proc</i>	A valid (non null) handle of the processing routines (ProcessingHandle), previously obtained through one of the functions createProcessing , createProcessingForDevice , createProcessingForDeviceEx or createProcessingForOCTFile .
in	<i>Selection</i>	Indicates the calibration that will be saved (CalibrationData).
in	<i>CameraIndex</i>	The camera index (0-based, i.e. zero for the first, one for the second, and so on).

This function will only save calibration data pertaining to the selected camera. To save the calibration of all cameras, multiple invocations are needed. the order plays no role.

```
6.12.4.36 setApodizationWindow() void setApodizationWindow (
    ProcessingHandle Proc,
    ApodizationWindow Window )
```

Sets the windowing function that will be used for apodization (this apodization has nothing to do with the reference spectra measured without a sample!). The selected windowing function will be used in all subsequent processings right before the fast Fourier transformation.

Parameters

in	<i>Proc</i>	A valid (non null) handle of the processing routines (ProcessingHandle), previously obtained through one of the functions createProcessing , createProcessingForDevice , createProcessingForDeviceEx or createProcessingForOCTFile .
in	<i>Window</i>	The desired apodization window to be used for apodizations right before Fourier transformations.

The selection of a windowing function is a balance between the acceptable width of the main lobe (that is, how many "frequency bins" does it take the response to reach half maximum power) and the attenuation of the side lobes (that is, what level of artifacts caused by the spectral leakage can be tolerated). As such, it depends on the particular experiment. The default selection (Hann windowing) cannot be expected to fit everyone's needs. If this function is not explicitly called, a Hann window will be assumed ([Apodization_Hann](#)).

```
6.12.4.37 setApodizationWindowParameter() void setApodizationWindowParameter (
    ProcessingHandle Proc,
    ApodizationWindowParameter Selection,
    double Value )
```

Sets the apodization window parameter, such as window width or ratio between constant and cosine part. Notice that this apodization is unrelated to the reference spectrum measured without a sample!.

Parameters

in	<i>Proc</i>	A valid (non null) handle of the processing routines (ProcessingHandle), previously obtained through one of the functions createProcessing , createProcessingForDevice , createProcessingForDeviceEx or createProcessingForOCTFile .
in	<i>Selection</i>	The desired parameter whose value will be changed (ApodizationWindowParameter).
in	<i>Value</i>	The desired value for the parameter.

```
6.12.4.38 setApodizedSpectrumOutput() void setApodizedSpectrumOutput (
    ProcessingHandle Proc,
    DataHandle ApodizedSpectrum )
```

Sets the location for the resulting apodized spectral data.

Parameters

in	<i>Proc</i>	A valid (non null) handle of the processing routines (ProcessingHandle), previously obtained through one of the functions createProcessing , createProcessingForDevice , createProcessingForDeviceEx or createProcessingForOCTFile .
in	<i>ApodizedSpectrum</i>	A valid (non null) data handle (DataHandle). Suitable sizes and ranges will be automatically set during the processing (executeProcessing).

```
6.12.4.39 setCalibration() void setCalibration (
    ProcessingHandle Proc,
    CalibrationData Selection,
    DataHandle Data )
```

Sets the calibration data.

Parameters

in	<i>Proc</i>	A valid (non null) handle of the processing routines (ProcessingHandle), previously obtained through one of the functions createProcessing , createProcessingForDevice , createProcessingForDeviceEx or createProcessingForOCTFile .
in	<i>Selection</i>	Indicates the calibration that will be set (CalibrationData).
in	<i>Data</i>	A valid handle (DataHandle) of the calibration data that will be set.

```
6.12.4.40 setColoredDataOutput() void setColoredDataOutput (
    ProcessingHandle Proc,
    ColoredDataHandle Scan,
    ColoringHandle Color )
```

Sets the pointer to the resulting colored scans that will be written after subsequent processing executions.

After the next completion of the function [executeProcessing\(\)](#), this data object will contain the colored amplitude of the scans.

If set to nullptr no colored data will be written in the subsequent processing executions.

Parameters

in	<i>Proc</i>	A valid (non null) handle of the processing routines (ProcessingHandle), previously obtained through one of the functions createProcessing , createProcessingForDevice , createProcessingForDeviceEx or createProcessingForOCTFile .
in	<i>Scan</i>	A valid (non null) colored data handle (ColoredDataHandle). Suitable sizes and ranges will be automatically set during the processing (executeProcessing).
in	<i>Color</i>	A valid (non null) coloring handle (ColoringHandle) as created, for example, with the functions createColoring32Bit() or createCustomColoring32Bit() .

```
6.12.4.41 setComplexDataOutput() void setComplexDataOutput (
    ProcessingHandle Proc,
    ComplexDataHandle ComplexScan )
```

Sets the pointer to the resulting complex scans that will be written after subsequent processing executions.

Parameters

in	<i>Proc</i>	A valid (non null) handle of the processing routines (ProcessingHandle), previously obtained through one of the functions createProcessing , createProcessingForDevice , createProcessingForDeviceEx or createProcessingForOCTFile .
in	<i>ComplexScan</i>	A valid (non null) complex data handle (ComplexDataHandle). Suitable sizes and ranges will be automatically set during the processing (executeProcessing).

After the next completion of the function [executeProcessing\(\)](#), this complex data object will contain the real and imaginary parts of the scans.

If set to a nullptr, no complex data result will be written in the subsequent processing executions.

```
6.12.4.42 setDCCorrectedSpectrumOutput() void setDCCorrectedSpectrumOutput (
    ProcessingHandle Proc,
    DataHandle DCCorrectedSpectrum )
```

Sets the location for the resulting DC removed spectral data.

Parameters

in	<i>Proc</i>	A valid (non null) handle of the processing routines (ProcessingHandle), previously obtained through one of the functions createProcessing , createProcessingForDevice , createProcessingForDeviceEx or createProcessingForOCTFile .
in	<i>DCCorrectedSpectrum</i>	A valid (non null) data handle (DataHandle). Suitable sizes and ranges will be automatically set during the processing (executeProcessing).

```
6.12.4.43 setDispersionCorrectionType() void setDispersionCorrectionType (
    ProcessingHandle Proc,
    DispersionCorrectionType Type )
```

Sets the active dispersion correction type.

Parameters

in	<i>Proc</i>	A valid (non null) handle of the processing routines (ProcessingHandle), previously obtained through one of the functions createProcessing , createProcessingForDevice , createProcessingForDeviceEx or createProcessingForOCTFile .
in	<i>Type</i>	The specification of the dispersion correction algorithm (DispersionCorrectionType).

6.12.4.44 setDispersionPresetByIndex() void setDispersionPresetByIndex (

```
ProcessingHandle Proc,
int Index )
```

Sets the dispersion preset specified with index.

Parameters

in	<i>Proc</i>	A valid (non null) handle of the processing routines (ProcessingHandle), previously obtained through one of the functions createProcessing , createProcessingForDevice , createProcessingForDeviceEx or createProcessingForOCTFile .
in	<i>Index</i>	An index specifying the desired dispersion preset that has to be set.

For a detailed explanation, do please refer to the documentation of [setDispersionPresets](#).

6.12.4.45 setDispersionPresetByName() void setDispersionPresetByName (

```
ProcessingHandle Proc,
const char * Name )
```

Sets the dispersion preset specified with name.

Parameters

in	<i>Proc</i>	A valid (non null) handle of the processing routines (ProcessingHandle), previously obtained through one of the functions createProcessing , createProcessingForDevice , createProcessingForDeviceEx or createProcessingForOCTFile .
in	<i>Name</i>	A zero terminated string with the name of the desired dispersion preset that has to be set.

For a detailed explanation, do please refer to the documentation of [setDispersionPresets](#).

6.12.4.46 setDispersionPresets() void setDispersionPresets (

```
ProcessingHandle Proc,
ProbeHandle Probe )
```

Sets the dispersion presets for the probe.

Hence it is suggested to use image quality as of criterion to set the coefficients, because this criterion usually works quite well. The quadratic coefficient can be easily found by using the ThorImage OCT software, and using the built-in quadratic slider and simultaneously looking for image quality and axial sharpness. Usually, the quadratic parameter alone gives rather good quality and dispersion correction and only for very broadband sources and strong dispersion higher coefficients are required.

To set higher coefficients, either this SDK is required or an entry "Dispersion_NameOfPreset" has to be added to the respective probe.ini file being used (see Settings Dialog of ThorImage OCT). This file is default located in C:\Program Files\Thorlabs\SpectralRadar\Config (or equivalent, if the software has been installed to another location). In probe.ini, the relevant entry looks like:

Dispersion_Probe = 10.0, 2.0, -1.0

and this particular example would set three dispersion factors, the quadratic one being 10, the third degree 2, and fourth degree -1. Again, unfortunately, there is no option to set these automatically. The user has to experiment with different parameters and iteratively optimize for a sharp signal. After this entry has been added to the file probe.ini, the Preset dispersions menu will contain a new entry on the next software start of ThorImageOCT.

Of course, the presets added to probe.ini can also be used by the functions in this SDK. Each line should give a different preset-name, and after this function is invoked, all of them will be available through indeces ([setDispersionPresetByIndex](#), [getNumberOfDispersionPresets](#)) or names ([getDispersionPresetName](#), [setDispersionPresetByName](#)).

Parameters

in	<i>Proc</i>	A valid (non null) handle of the processing routines (ProcessingHandle), previously obtained through one of the functions createProcessing , createProcessingForDevice , createProcessingForDeviceEx or createProcessingForOCTFile .
in	<i>Probe</i>	A valid (non null) probe handle (ProbeHandle), previously generated with the function initProbe .

Unfortunately no really good and easy method to predict the dispersion coefficient(s) is offered here. The coefficients (currently) used in the software do not correspond to physically meaningful parameters, but are rather given in arbitrary units.

6.12.4.47 `setDispersionQuadraticCoeff()` `void setDispersionQuadraticCoeff (`
`ProcessingHandle Proc,`
`double Coeff)`

Sets the coefficient for the quadratic correction of the dispersion.

Parameters

in	<i>Proc</i>	A valid (non null) handle of the processing routines (ProcessingHandle), previously obtained through one of the functions createProcessing , createProcessingForDevice , createProcessingForDeviceEx or createProcessingForOCTFile .
in	<i>Coeff</i>	The desired coefficient.

6.12.4.48 `setOffsetCorrectedSpectrumOutput()` `void setOffsetCorrectedSpectrumOutput (`
`ProcessingHandle Proc,`
`DataHandle OffsetCorrectedSpectrum)`

Sets the location for the resulting offset corrected spectral data.

Parameters

in	<i>Proc</i>	A valid (non null) handle of the processing routines (ProcessingHandle), previously obtained through one of the functions createProcessing , createProcessingForDevice , createProcessingForDeviceEx or createProcessingForOCTFile .
in	<i>OffsetCorrectedSpectrum</i>	A valid (non null) data handle (DataHandle). Suitable sizes and ranges will be automatically set during the processing (executeProcessing).

6.12.4.49 `setProcessedDataOutput()` `void setProcessedDataOutput (`
`ProcessingHandle Proc,`
`DataHandle Scan)`

Sets the pointer to the resulting scans that will be written after subsequent processing executions.

Parameters

in	<i>Proc</i>	A valid (non null) handle of the processing routines (ProcessingHandle), previously obtained through one of the functions createProcessing , createProcessingForDevice , createProcessingForDeviceEx or createProcessingForOCTFile .
in	<i>Scan</i>	A valid (non null) data handle (DataHandle). Suitable sizes and ranges will be automatically set during the processing (executeProcessing).

After the next completion of the function [executeProcessing\(\)](#), this data object will contain the amplitude (in dB) of the scans.

If set to `nullptr` no processed floating point data in dB will be written in the subsequent processing executions.

6.12.4.50 setProcessingAveragingAlgorithm() `void setProcessingAveragingAlgorithm (`

```
    ProcessingHandle Proc,
    ProcessingAveragingAlgorithm Algorithm )
```

Sets the algorithm that will be used for averaging during the processing.

Parameters

in	<i>Proc</i>	A valid (non null) handle of the processing routines (ProcessingHandle), previously obtained through one of the functions createProcessing , createProcessingForDevice , createProcessingForDeviceEx or createProcessingForOCTFile .
in	<i>Algorithm</i>	The averaging algorithm (ProcessingAveragingAlgorithm). If this function is not explicitly invoked, the value Processing_Averaging_Mean can be assumed.

6.12.4.51 setProcessingDechirpAlgorithm() `void setProcessingDechirpAlgorithm (`

```
    ProcessingHandle Proc,
    Processing_FFTType Type,
    float Oversampling )
```

Sets the algorithm to be used for dechirping the input spectra.

Parameters

in	<i>Proc</i>	A valid (non null) handle of the processing routines (ProcessingHandle), previously obtained through one of the functions createProcessing , createProcessingForDevice , createProcessingForDeviceEx or createProcessingForOCTFile .
in	<i>Type</i>	Specifies the FFT algorithm (Processing_FFTType) that will combine the dechirping with the Fourier transform.
in	<i>Oversampling</i>	In case the selected FFT algorithm bases on oversampling, this parameter gives the factor.

6.12.4.52 setProcessingFlag() `void setProcessingFlag (`

```
    ProcessingHandle Proc,
```

```
ProcessingFlag Flag,
BOOL Value )
```

Sets the specified processing flag.

Parameters

in	<i>Proc</i>	A valid (non null) handle of the processing routines (ProcessingHandle), previously obtained through one of the functions createProcessing , createProcessingForDevice , createProcessingForDeviceEx or createProcessingForOCTFile .
in	<i>Flag</i>	The flag whose value will be modified.
in	<i>Value</i>	The desired value for the flag.

6.12.4.53 [setProcessingParameterFloat\(\)](#) void setProcessingParameterFloat (

```
ProcessingHandle Proc,
ProcessingParameterFloat Selection,
double Value )
```

Sets the specified floating point processing parameter.

Parameters

in	<i>Proc</i>	A valid (non null) handle of the processing routines (ProcessingHandle), previously obtained through one of the functions createProcessing , createProcessingForDevice , createProcessingForDeviceEx or createProcessingForOCTFile .
in	<i>Selection</i>	The floating point parameter whose value will be modified.
in	<i>Value</i>	The desired value for the floating point parameter.

6.12.4.54 [setProcessingParameterInt\(\)](#) void setProcessingParameterInt (

```
ProcessingHandle Proc,
ProcessingParameterInt Selection,
int Value )
```

Sets the specified integer value processing parameter.

Parameters

in	<i>Proc</i>	A valid (non null) handle of the processing routines (ProcessingHandle), previously obtained through one of the functions createProcessing , createProcessingForDevice , createProcessingForDeviceEx or createProcessingForOCTFile .
in	<i>Selection</i>	The parameter whose value will be modified.
in	<i>Value</i>	The desired value for the integer parameter.

6.12.4.55 [setSpectrumOutput\(\)](#) void setSpectrumOutput (

```
ProcessingHandle Proc,  
DataHandle Spectrum )
```

Sets the location for the resulting spectral data.

Parameters

in	<i>Proc</i>	A valid (non null) handle of the processing routines (ProcessingHandle), previously obtained through one of the functions createProcessing , createProcessingForDevice , createProcessingForDeviceEx or createProcessingForOCTFile .
in	<i>Spectrum</i>	A valid (non null) data handle (DataHandle). Suitable sizes and ranges will be automatically set during the processing (executeProcessing).

6.12.4.56 `setTransposedColoredDataOutput()` void setTransposedColoredDataOutput (

```
ProcessingHandle Proc,  
ColoredDataHandle Scan,  
ColoringHandle Color )
```

Sets the pointer to the resulting colored scans that will be written after subsequent processing executions. The orientation of the colored data will be transposed in such a way that the first axis (normally z-axis) will be the x-axis (the depth of each individual A-scan) and the second axis (normally x-axis) will be the z-axis.

After the next completion of the function [executeProcessing\(\)](#), this data object will contain the transposed colored amplitude of the scans.

If set to nullptr no colored data will be written in the subsequent processing executions.

Parameters

in	<i>Proc</i>	A valid (non null) handle of the processing routines (ProcessingHandle), previously obtained through one of the functions createProcessing , createProcessingForDevice , createProcessingForDeviceEx or createProcessingForOCTFile .
in	<i>Scan</i>	A valid (non null) colored data handle (ColoredDataHandle). Suitable sizes and ranges will be automatically set during the processing (executeProcessing).
in	<i>Color</i>	A valid (non null) coloring handle (ColoringHandle) as created, for example, with the functions createColoring32Bit() or createCustomColoring32Bit() .

6.13 Export and Import

Functionality to store data to disk and load it from there.

Enumerations

- enum `DataExportFormat` {
 `DataExport_SRM`,
 `DataExport_RAW`,
 `DataExport_CSV`,
 `DataExport_TXT`,
 `DataExport_TableTXT`,
 `DataExport_Fits`,
 `DataExport_VFF`,
 `DataExport_VTK`,
 `DataExport_TIFF` }

 Export format for any data represented by a `DataHandle`.
- enum `ComplexDataExportFormat` { `ComplexDataExport_RAW` }

 Export format for complex data.
- enum `ColoredDataExportFormat` {
 `ColoredDataExport_SRM`,
 `ColoredDataExport_RAW`,
 `ColoredDataExport_BMP`,
 `ColoredDataExport_PNG`,
 `ColoredDataExport_JPG`,
 `ColoredDataExport_PDF`,
 `ColoredDataExport_TIFF` }

 Export format for images (`ColoredDataHandle`).
- enum `DataImportFormat` { `DataImport_SRM` }

 Supported import format to load data from disk.
- enum `RawDataExportFormat` {
 `RawDataExport_RAW`,
 `RawDataExport_SRR` }

 Supported raw data export formats to store data to disk.
- enum `RawDataImportFormat` { `RawDataImport_SRR` }

 Supported raw data import formats to load data from disk.

Functions

- **SPECTRALRADAR_API** void `exportData` (`DataHandle` Data, `DataExportFormat` Format, const char *FileName)

 Exports data (`DataHandle`) to a file. The number of dimensions is handled automatically upon analysis of the first argument.
- **SPECTRALRADAR_API** void `exportDataAsImage` (`DataHandle` Data, `ColoringHandle` Color, `ColoredDataExportFormat` Format, `Direction` SliceNormalDirection, const char *FileName, int ExportOptionMask)

 Exports 2-dimensional and 3-dimensional data (`DataHandle`) as image data (such as BMP, PNG, JPEG, ...).
- **SPECTRALRADAR_API** void `exportComplexData` (`ComplexDataHandle` Data, `ComplexDataExportFormat` Format, const char *FileName)

 Exports 1-, 2- and 3-dimensional complex data (`ComplexDataHandle`)
- **SPECTRALRADAR_API** void `exportColoredData` (`ColoredDataHandle` Data, `ColoredDataExportFormat` Format, `Direction` SliceNormalDirection, const char *FileName, int ExportOptionMask)

 Exports colored data (`ColoredDataHandle`).

- **SPECTRALRADAR_API** void [importColoredData](#) ([ColoredDataHandle](#) ColoredData, [DataImportFormat](#) Format, const char *FileName)
Imports colored data ([ColoredDataHandle](#)) with the specified format and copied it into a data object ([ColoredDataHandle](#)).
- **SPECTRALRADAR_API** void [importData](#) ([DataHandle](#) Data, [DataImportFormat](#) Format, const char *FileName)
Imports data with the specified format and copies it into a data object ([DataHandle](#)).
- **SPECTRALRADAR_API** void [exportRawData](#) ([RawDataHandle](#) Raw, [RawDataExportFormat](#) Format, const char *FileName)
Exports the specified data to disk.
- **SPECTRALRADAR_API** void [importRawData](#) ([RawDataHandle](#) Raw, [RawDataImportFormat](#) Format, const char *FileName)
Imports the specified data from disk.

ExportOptions

Specifies additional export options to be used with functions such as [exportDataAsImage\(\)](#). Multiple options can be combined by bit-wise or ("|"). Different options can be used for different export format. If an option is not supported by an export format, it is ignored.

- const int [ExportOption_None](#) = 0x00000000
- const int [ExportOption_DrawScaleBar](#) = 0x00000001
Draw scale bar on exported image.
- const int [ExportOption_DrawMarkers](#) = 0x00000002
Draw markers on exported image.
- const int [ExportOption_UsePhysicalAspectRatio](#) = 0x00000004
Honor physical aspect ratio when exporting data (width and height of each pixel will have the same physical dimensions).
- const int [ExportOption_Flip_X_Axis](#) = 0x00000008
Flip X-axis.
- const int [ExportOption_Flip_Y_Axis](#) = 0x00000010
Flip Y-axis.
- const int [ExportOption_Flip_Z_Axis](#) = 0x00000020
Flip Z-axis.

6.13.1 Detailed Description

Functionality to store data to disk and load it from there.

6.13.2 Enumeration Type Documentation

6.13.2.1 [ColoredDataExportFormat](#) 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.

Definition at line 382 of file [SpectralRadar_Types.h](#).

6.13.2.2 ComplexDataExportFormat [enum ComplexDataExportFormat](#)

Export format for complex data.

Enumerator

ComplexDataExport_RAW	RAW data format containg binary data.
-----------------------	---------------------------------------

Definition at line 374 of file [SpectralRadar_Types.h](#).

6.13.2.3 DataExportFormat [enum DataExportFormat](#)

Export format for any data represented by a [DataHandle](#).

Enumerator

DataExport_SRM	Spectral Radar Metaformat, containing no data but many additinal parameters, such as spacing, size, etc.
DataExport_RAW	RAW data format containing the data of the object as binary, single precision floating point values, little endian.
DataExport_CSV	CSV (Comma Separated Values) is a text file having all values stored, comma seperated and human readable.
DataExport_TXT	TXT is a text file having all values stored space seperated and human readable.
DataExport_TableTXT	TableTXT is a human readable text-file in a table like format, having the physical 1- and 2-axis as first two columns and the data value as third. Currently only works for 1D- and 2D-Data.
DataExport_Fits	FITS Data format.
DataExport_VFF	VFF data format.
DataExport_VTK	VTK data format.
DataExport_TIFF	TIFF Data format as 32-bit floating point numbers.

Definition at line 350 of file [SpectralRadar_Types.h](#).

6.13.2.4 DataImportFormat enum [DataImportFormat](#)

Supported import format to load data from disk.

Enumerator

DataImport_SRM	Spectral Radar Metaformat, containing no data but all additional parameters, such as spacing, size, etc. It is searched for an appropriate file with same name but different extension containing the according data.
----------------	---

Definition at line 416 of file [SpectralRadar_Types.h](#).

6.13.2.5 RawDataExportFormat 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.

Definition at line 425 of file [SpectralRadar_Types.h](#).

6.13.2.6 RawDataImportFormat 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.
-------------------	--

Definition at line 435 of file [SpectralRadar_Types.h](#).

6.13.3 Function Documentation

```
6.13.3.1 exportColoredData() void exportColoredData (
    ColoredDataHandle Data,
    ColoredDataExportFormat Format,
    Direction SliceNormalDirection,
    const char * FileName,
    int ExportOptionMask )
```

Exports colored data ([ColoredDataHandle](#)).

Parameters

in	<i>Data</i>	A valid (non null) colored-data handle of the data (ColoredDataHandle). These data may be multi-dimensional.
in	<i>Format</i>	The desired data-format, to be selected among those in ColoredDataExportFormat .
in	<i>SliceNormalDirection</i>	Specifies the direction normal to the generated pictures (to be chosen among those elements in Direction).
in	<i>FileName</i>	A zero-terminated string specifying the full pathname to the file to be written. Notice that backslashes should be escaped with an additional backslash.
in	<i>ExportOptionMask</i>	An OR-ed combination of the flags ExportOption_None , ExportOption_DrawScaleBar , ExportOption_DrawMarkers , and ExportOption_UsePhysicalAspectRatio .

```
6.13.3.2 exportComplexData() void exportComplexData (
    ComplexDataHandle Data,
    ComplexDataExportFormat Format,
    const char * FileName )
```

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

Parameters

in	<i>Data</i>	A valid (non null) complex-data handle of the data (ComplexDataHandle). These data may be multi-dimensional.
in	<i>Format</i>	The desired data-format, to be selected among those in ComplexDataExportFormat .
in	<i>FileName</i>	A zero-terminated string specifying the full pathname to the file to be written. Notice that backslashes should be escaped with an additional backslash.

```
6.13.3.3 exportData() void exportData (
    DataHandle Data,
    DataExportFormat Format,
    const char * FileName )
```

Exports data ([DataHandle](#)) to a file. The number of dimensions is handled automatically upon analysis of the first argument.

Parameters

in	<i>Data</i>	A valid (non null) data handle of the data (DataHandle). These data may be multi-dimensional.
in	<i>Format</i>	The desired data-format, to be selected among those in DataExportFormat .
in	<i>FileName</i>	A zero-terminated string specifying the full pathname to the file to be written. Notice that backslashes should be escaped with an additional backslash.

6.13.3.4 `exportDataAsImage()` `void exportDataAsImage (`

```
    DataHandle Data,
    ColoringHandle Color,
    ColoredDataExportFormat Format,
    Direction SliceNormalDirection,
    const char * FileName,
    int ExportOptionMask )
```

Exports 2-dimensional and 3-dimensional data ([DataHandle](#)) as image data (such as BMP, PNG, JPEG, ...).

Parameters

in	<i>Data</i>	A valid (non null) data handle of the data (DataHandle). These data may be multi-dimensional.
in	<i>Color</i>	A valid (non null) coloring handle (ColoringHandle) as created, for example, with the functions createColoring32Bit() or createCustomColoring32Bit() .
in	<i>Format</i>	The desired data-format, to be selected among those in ColoredDataExportFormat .
in	<i>SliceNormalDirection</i>	Specifies the direction normal to the generated pictures (to be chosen among those elements in Direction).
in	<i>FileName</i>	A zero-terminated string specifying the full pathname to the file to be written. Notice that backslashes should be escaped with an additional backslash.
in	<i>ExportOptionMask</i>	An OR-ed combination of the flags ExportOption_None , ExportOption_DrawScaleBar , ExportOption_DrawMarkers , and ExportOption_UsePhysicalAspectRatio .

6.13.3.5 `exportRawData()` `void exportRawData (`

```
    RawDataHandle Raw,
    RawDataExportFormat Format,
    const char * FileName )
```

Exports the specified data to disk.

Parameters

in	<i>Raw</i>	A valid (non null) raw-data handle of the data (RawDataHandle).
in	<i>Format</i>	The desired data-format to be stored in the file (the supported ones are the items of RawDataExportFormat).
in	<i>FileName</i>	A zero-terminated string specifying the full pathname to the file to be written. Notice that backslashes should be escaped with an additional backslash.

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

```
6.13.3.6 importColoredData() void importColoredData (
    ColoredDataHandle ColoredData,
    DataImportFormat Format,
    const char * FileName )
```

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

Parameters

out	<i>ColoredData</i>	A valid (non null) colored-data handle of the data (ColoredDataHandle). These data may be multi-dimensional.
in	<i>Format</i>	The data-format stored in the file (the supported ones are the items of DataImportFormat).
in	<i>FileName</i>	A zero-terminated string specifying the full pathname to the file to be written. Notice that backslashes should be escaped with an additional backslash.

```
6.13.3.7 importData() void importData (
    DataHandle Data,
    DataImportFormat Format,
    const char * FileName )
```

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

Parameters

out	<i>Data</i>	A valid (non null) data handle of the data (DataHandle). These data may be multi-dimensional.
in	<i>Format</i>	The data-format stored in the file (the supported ones are the items of DataImportFormat).
in	<i>FileName</i>	A zero-terminated string specifying the full pathname to the file to be written. Notice that backslashes should be escaped with an additional backslash.

```
6.13.3.8 importRawData() void importRawData (
    RawDataHandle Raw,
    RawDataImportFormat Format,
    const char * FileName )
```

Imports the specified data from disk.

Parameters

out	<i>Raw</i>	A valid (non null) raw-data handle of the data (RawDataHandle).
in	<i>Format</i>	The data-format stored in the file (the supported ones are the items of RawDataImportFormat).
in	<i>FileName</i>	A zero-terminated string specifying the full pathname to the file to be written. Notice that backslashes should be escaped with an additional backslash.
<small>Generated by Doxygen</small>		

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

6.13.4 Variable Documentation

6.13.4.1 ExportOption_DrawMarkers const int ExportOption_DrawMarkers = 0x00000002

Draw markers on exported image.

Definition at line [2729](#) of file [SpectralRadar.h](#).

6.13.4.2 ExportOption_DrawScaleBar const int ExportOption_DrawScaleBar = 0x00000001

Draw scale bar on exported image.

Definition at line [2727](#) of file [SpectralRadar.h](#).

6.13.4.3 ExportOption_Flip_X_Axis const int ExportOption_Flip_X_Axis = 0x00000008

Flip X-axis.

Definition at line [2733](#) of file [SpectralRadar.h](#).

6.13.4.4 ExportOption_Flip_Y_Axis const int ExportOption_Flip_Y_Axis = 0x000000010

Flip Y-axis.

Definition at line [2735](#) of file [SpectralRadar.h](#).

6.13.4.5 ExportOption_Flip_Z_Axis const int ExportOption_Flip_Z_Axis = 0x000000020

Flip Z-axis.

Definition at line [2737](#) of file [SpectralRadar.h](#).

6.13.4.6 ExportOption_None const int ExportOption_None = 0x000000000

For default or no specific export options.

Definition at line [2725](#) of file [SpectralRadar.h](#).

6.13.4.7 ExportOption_UsePhysicalAspectRatio const int ExportOption_UsePhysicalAspectRatio = 0x000000004

Honor physical aspect ratio when exporting data (width and height of each pixel will have the same physical dimensions).

Definition at line [2731](#) of file [SpectralRadar.h](#).

6.14 Volume

Functionality to store and access volume data.

Enumerations

- enum **Direction** {

Direction_1,

Direction_2,

Direction_3 }

Specifies a direction. In the default orientation, the first orientation is the Z-axis (parallel to the illumination-ray during the measurement), the second is the X-axis, and the third is the Y-axis.

- enum **Plane2D** {

Plane2D_12,

Plane2D_23,

Plane2D_13 }

Planes for slices of the volume data.

Functions

- **SPECTRALRADAR_API** void **appendRawData** (**RawDataHandle** Raw, **RawDataHandle** DataToAppend, **Direction** Dir)

Appends the new raw data to the old raw data perpendicular to the specified direction.
- **SPECTRALRADAR_API** void **getRawDataSliceAtIndex** (**RawDataHandle** Raw, **RawDataHandle** Slice, **Direction** SliceNormalDirection, int Index)

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

Analyzes the given data, extracts the selected feature, and returns the computed value.
- **SPECTRALRADAR_API** double **analyzeAScan** (**DataHandle** Data, **AScanAnalyzation** Selection)

Analyzes the given A-scan data, extracts the selected feature, and returns the computed value.
- **SPECTRALRADAR_API** void **analyzePeaksInAScan** (**DataHandle** Data, **AScanAnalyzation** Selection, int NumberOfPeaksToAnalyze, int MinDistBetweenPeaks, double *Result)

Analyzes the given A-scan data, extracts the selected feature, and returns the computed value. It returns the result of multiple peaks compared to the function `analyzeAScan`.
- **SPECTRALRADAR_API** void **transposeData** (**DataHandle** DataIn, **DataHandle** DataOut)

Transposes the given data and writes the result to `DataOut`. First and second axes will be swapped.
- **SPECTRALRADAR_API** void **transposeDataInplace** (**DataHandle** Data)

Transposes the given `Data`. First and second axes will be swapped.
- **SPECTRALRADAR_API** void **transposeAndScaleData** (**DataHandle** DataIn, **DataHandle** DataOut, float Min, float Max)

Transposes the given data and writes the result to `DataOut`. First and second axes will be swapped, and the range of the entries will be scaled in such a way, that the range [Min,Max] will be mapped onto the range [0,1].
- **SPECTRALRADAR_API** void **normalizeData** (**DataHandle** Data, float Min, float Max)

Scales the given data in such a way, that the range [Min, Max] is mapped onto the range [0,1].
- **SPECTRALRADAR_API** void **getDataSliceAtPos** (**DataHandle** Data, **DataHandle** Slice, **Direction** SliceNormalDirection, double Pos_mm)

Returns a slice of data perpendicular to the specified direction at the specified position.
- **SPECTRALRADAR_API** void **getComplexDataSlicePos** (**ComplexDataHandle** Data, **ComplexDataHandle** Slice, **Direction** SliceNormalDirection, double Pos_mm)

Returns a slice of complex data perpendicular to the specified direction at the specified position.
- **SPECTRALRADAR_API** void **getColoredDataSlicePos** (**ColoredDataHandle** Data, **ColoredDataHandle** Slice, **Direction** SliceNormalDirection, double Pos_mm)

- Returns a slice of colored data perpendicular to the specified direction at the specified position.*
- **SPECTRALRADAR_API** void `getDataSliceAtIndex` (`DataHandle` Data, `DataHandle` Slice, `Direction` Slice←NormalDirection, int Index)
Returns a slice of data perpendicular to the specified direction at the specified index.
 - **SPECTRALRADAR_API** void `getComplexDataSliceIndex` (`ComplexDataHandle` Data, `ComplexDataHandle` Slice, `Direction` SliceNormalDirection, int Index)
Returns a slice of complex data perpendicular to the specified direction at the specified index.
 - **SPECTRALRADAR_API** void `getColoredDataSliceIndex` (`ColoredDataHandle` Data, `ColoredDataHandle` Slice, `Direction` SliceNormalDirection, int Index)
Returns a slice of colored data perpendicular to the specified direction at the specified index.
 - **SPECTRALRADAR_API** void `computeDataProjection` (`DataHandle` Data, `DataHandle` Slice, `Direction` ProjectionDirection, `DataAnalyzation` Selection)
Returns a single slice of data, in which each pixel value is the feature extracted through an analysis along the specified direction.
 - **SPECTRALRADAR_API** void `appendData` (`DataHandle` Data, `DataHandle` DataToAppend, `Direction` Dir)
Appends the new data to the provided data, perpendicular to the specified direction.
 - **SPECTRALRADAR_API** void `appendComplexData` (`ComplexDataHandle` Data, `ComplexDataHandle` Data←ToAppend, `Direction` Dir)
Appends the new data to the provided data, perpendicular to the specified direction.
 - **SPECTRALRADAR_API** void `appendColoredData` (`ColoredDataHandle` Data, `ColoredDataHandle` DataTo←Append, `Direction` Dir)
Appends the new data to the provided data, perpendicular to the specified direction.
 - **SPECTRALRADAR_API** void `cropData` (`DataHandle` Data, `Direction` Dir, int IndexMax, int IndexMin)
Crops the data along the desired direction at the given indices. Upon return the data will only contain those slices whose indices where in the interval [IndexMin, IndexMax], counted along the cropping direction.
 - **SPECTRALRADAR_API** void `cropComplexData` (`ComplexDataHandle` Data, `Direction` Dir, int IndexMax, int IndexMin)
Crops the complex data along the desired direction at the given indices. Upon return the data will only contain those slices whose indices where in the interval [IndexMin, IndexMax], counted along the cropping direction.
 - **SPECTRALRADAR_API** void `cropColoredData` (`ColoredDataHandle` Data, `Direction` Dir, int IndexMax, int IndexMin)
Crops the colored data along the desired direction at the given indices. Upon return the data will only contain those slices whose indices where in the interval [IndexMin, IndexMax], counted along the cropping direction.
 - **SPECTRALRADAR_API** void `separateData` (`DataHandle` Data1, `DataHandle` Data2, int SeparationIndex, `Direction` Dir)
Separates the data at the given index at specific separation direction. The first part of the separated data will remain in Data1, the second separated in Data2.
 - **SPECTRALRADAR_API** void `separateComplexData` (`ComplexDataHandle` Data1, `ComplexDataHandle` Data2, int SeparationIndex, `Direction` Dir)
Separates the data at the given index at specific separation direction. The first part of the separated data will remain in Data1, the second separated in Data2.
 - **SPECTRALRADAR_API** void `separateColoredData` (`ColoredDataHandle` Data1, `ColoredDataHandle` Data2, int SeparationIndex, `Direction` Dir)
Separates the data at the given index at specific separation direction. The first part of the separated data will remain in Data1, the second separated in Data2.
 - **SPECTRALRADAR_API** void `flipData` (`DataHandle` Data, `Direction` FlippingDir)
Mirrors the data across a plane perpendicular to the given direction.
 - **SPECTRALRADAR_API** void `flipComplexData` (`ComplexDataHandle` Data, `Direction` FlippingDir)
Mirrors the data across a plane perpendicular to the given direction.
 - **SPECTRALRADAR_API** void `flipColoredData` (`ColoredDataHandle` Data, `Direction` FlippingDir)
Mirrors the data across a plane perpendicular to the given direction.
 - **SPECTRALRADAR_API** `ImageFieldHandle` `createImageField` (void)
Creates an object holding image field data.
 - **SPECTRALRADAR_API** `ImageFieldHandle` `createImageFieldFromProbe` (`ProbeHandle` Probe)

Creates an object holding image field data from the specified Probe Handle.

- **SPECTRALRADAR_API** void `clearImageField (ImageFieldHandle ImageField)`

Frees an object holding image field data.

- **SPECTRALRADAR_API** void `savelImageField (ImageFieldHandle ImageField, const char *Path)`

Saves data containing image field data.

- **SPECTRALRADAR_API** void `loadImageField (ImageFieldHandle ImageField, const char *Path)`

Loads data containing image field data.

- **SPECTRALRADAR_API** void `determineImageField (ImageFieldHandle ImageField, ScanPatternHandle Pattern, DataHandle Surface)`

Determines the image field correction for the given surface data, previously measured with the given scan pattern.

- **SPECTRALRADAR_API** void `determineImageFieldWithMask (ImageFieldHandle ImageField, ScanPatternHandle Pattern, DataHandle Surface, DataHandle Mask)`

Determines the image field correction for the given surface data, previously measured with the given scan pattern.

The positive entries of the mask determine the points that actually enter in the computation.

- **SPECTRALRADAR_API** void `correctImageField (ImageFieldHandle ImageField, ScanPatternHandle Pattern, DataHandle Data)`

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

- **SPECTRALRADAR_API** void `correctImageFieldComplex (ImageFieldHandle ImageField, ScanPatternHandle Pattern, ComplexDataHandle Data)`

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

- **SPECTRALRADAR_API** void `correctSurface (ImageFieldHandle ImageField, ScanPatternHandle Pattern, DataHandle Surface)`

Applies the image field correction to the given Surface. Surface must contain depth values as a function of x/y coordinates.

- **SPECTRALRADAR_API** void `setImageFieldInProbe (ImageFieldHandle ImageField, ProbeHandle Probe)`

Sets the specified image field to the specified Probe handle. Notice that no probe file will be automatically saved.

- **SPECTRALRADAR_API** double `analyzeComplexAScan (ComplexDataHandle AScanIn, AScanAnalysis Selection)`

Analyzes the given complex A-scan data, extracts the selected feature, and returns the computed value.

6.14.1 Detailed Description

Functionality to store and access volume data.

6.14.2 Enumeration Type Documentation

6.14.2.1 Direction enum `Direction`

Specifies a direction. In the default orientation, the first orientation is the Z-axis (parallel to the illumination-ray during the measurement), the second is the X-axis, and the third is the Y-axis.

Enumerator

Direction ₁	The 1-axis direction.
Direction ₂	The 2-axis direction.
Direction ₃	The 3-axis direction.

Definition at line 404 of file [SpectralRadar_Types.h](#).

6.14.2.2 **Plane2D** `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.

Definition at line 444 of file [SpectralRadar_Types.h](#).

6.14.3 Function Documentation

6.14.3.1 **analyzeAScan()** `double analyzeAScan (` `DataHandle Data,` `AScanAnalyzation Selection)`

Analyzes the given A-scan data, extracts the selected feature, and returns the computed value.

Parameters

in	<i>Data</i>	A valid (non null) data handle of the A-scan (DataHandle).
in	<i>Selection</i>	The desired feature that should be computed (AScanAnalyzation).

Returns

The computed feature.

If the given data is multi-dimensional, only the first A-scan will be analyzed.

6.14.3.2 **analyzeComplexAScan()** `double analyzeComplexAScan (` `ComplexDataHandle AScanIn,` `AScanAnalyzation Selection)`

Analyzes the given complex A-scan data, extracts the selected feature, and returns the computed value.

Parameters

in	<i>AScanIn</i>	A valid (non null) complex data handle of the A-scan (ComplexDataHandle).
in	<i>Selection</i>	The desired feature that should be computed (AScanAnalyzation).

Returns

The computed feature.

If the given data is multi-dimensional, only the first A-scan will be analyzed.

```
6.14.3.3 analyzeData() double analyzeData (
    DataHandle Data,
    DataAnalyzation Selection )
```

Analyzes the given data, extracts the selected feature, and returns the computed value.

Parameters

in	<i>Data</i>	A valid (non null) data handle of the data (DataHandle). These data may be multi-dimensional.
in	<i>Selection</i>	The desired feature that should be computed (DataAnalyzation).

Returns

The value of the desired feature.

```
6.14.3.4 analyzePeaksInAScan() void analyzePeaksInAScan (
    DataHandle Data,
    AScanAnalyzation Selection,
    int NumberOfPeaksToAnalyze,
    int MinDistBetweenPeaks,
    double * Result )
```

Analyzes the given A-scan data, extracts the selected feature, and returns the computed value. It returns the result of multiple peaks compared to the function [analyzeAScan](#).

Parameters

in	<i>Data</i>	A valid (non null) data handle of the A-scan (DataHandle).
in	<i>Selection</i>	The desired feature that should be computed (AScanAnalyzation).
in	<i>NumberOfPeaksToAnalyze</i>	The number of peaks which should be analyzed.
in	<i>MinDistBetweenPeaks</i>	The minimum distance between the peaks in pixel.
out	<i>Result</i>	An array of size <i>NumberOfPeaksToAnalyze</i> which contains the computed result.

If the given data is multi-dimensional, only the first A-scan will be analyzed.

```
6.14.3.5 appendColoredData() void appendColoredData (
    ColoredDataHandle Data,
    ColoredDataHandle DataToAppend,
    Direction Dir )
```

Appends the new data to the provided data, perpendicular to the specified direction.

Parameters

in, out	<i>Data</i>	A valid (non null) colored data handle of the existing data (ColoredDataHandle), that will be expanded.
in	<i>DataToAppend</i>	A valid (non null) colored data handle of the new data (ColoredDataHandle). These data will not be modified.
in	<i>Dir</i>	The physical direction (Direction) along which the existing data will be expanded in order to accomodate the new data. Currently the Direction_1 (usually the Z-axis) is not supported and should not be specified.

Appending data implies expanding the number of data and also their physical range. These expansions are carried out automatically before the function returns.

```
6.14.3.6 appendComplexData() void appendComplexData (
    ComplexDataHandle Data,
    ComplexDataHandle DataToAppend,
    Direction Dir )
```

Appends the new data to the provided data, perpendicular to the specified direction.

Parameters

in, out	<i>Data</i>	A valid (non null) complex data handle of the existing data (ComplexDataHandle), that will be expanded.
in	<i>DataToAppend</i>	A valid (non null) complex data handle of the new data (ComplexDataHandle). These data will not be modified.
in	<i>Dir</i>	The physical direction (Direction) along which the existing data will be expanded in order to accomodate the new data. Currently the Direction_1 (usually the Z-axis) is not supported and should not be specified.

Appending data implies expanding the number of data and also their physical range. These expansions are carried out automatically before the function returns.

```
6.14.3.7 appendData() void appendData (
    DataHandle Data,
    DataHandle DataToAppend,
    Direction Dir )
```

Appends the new data to the provided data, perpendicular to the specified direction.

Parameters

in, out	<i>Data</i>	A valid (non null) data handle of the existing data (DataHandle), that will be expanded.
in	<i>DataToAppend</i>	A valid (non null) data handle of the new data (DataHandle). These data will not be modified.
in	<i>Dir</i>	The physical direction (Direction) along which the existing data will be expanded in order to accomodate the new data. Currently the Direction_1 (usually the Z-axis) is not supported and should not be specified.

Appending data implies expanding the number of data and also their physical range. These expansions are carried

out automatically before the function returns.

```
6.14.3.8 appendRawData() void appendRawData (
    RawDataHandle Raw,
    RawDataHandle DataToAppend,
    Direction Dir )
```

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

Parameters

in, out	<i>Raw</i>	A valid (non null) raw-data handle of the existing data (RawDataHandle), that will be expanded.
in	<i>DataToAppend</i>	A valid (non null) raw-data handle of the new data (RawDataHandle). These raw-data will not be modified.
in	<i>Dir</i>	The physical direction (Direction) in which the new data will be appended. Currently the Direction_1 (usually the Z-axis) is not supported and should not be specified.

Appending data implies expanding the number of data and also their physical range. These expansions are carried out automatically before the function returns.

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

```
6.14.3.9 clearImageField() void clearImageField (
    ImageFieldHandle ImageField )
```

Frees an object holding image field data.

Parameters

in	<i>ImageField</i>	A handle of the image field (ImageFieldHandle). If the handle is a nullptr, this function does nothing.
----	-------------------	---

```
6.14.3.10 computeDataProjection() void computeDataProjection (
    DataHandle Data,
    DataHandle Slice,
    Direction ProjectionDirection,
    DataAnalyzation Selection )
```

Returns a single slice of data, in which each pixel value is the feature extracted through an analysis along the specified direction.

Parameters

in	<i>Data</i>	A valid (non null) data handle of the existing, three-dimensional data (DataHandle). These data will not be modified.
out	<i>Slice</i>	A valid (non null) data handle (DataHandle), where the data of the slice will be written. Geometrically, this slice is situated perpendicular to the specified Direction .

Parameters

in	<i>ProjectionDirection</i>	The physical direction (Direction) along which the provided data will be analyzed.
in	<i>Selection</i>	The desired feature that should be extracted from the data along the specified direction.

```
6.14.3.11 correctImageField() void correctImageField (
    ImageFieldHandle ImageField,
    ScanPatternHandle Pattern,
    DataHandle Data )
```

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

Parameters

in	<i>ImageField</i>	A valid (non null) handle of the image field (ImageFieldHandle), previously created with one of the functions createImageField or createImageFieldFromProbe . Besides, the image field has already been determined with the help of the function determineImageField or determineImageFieldWithMask .
in	<i>Pattern</i>	A valid (non null) handle of a scan pattern (ScanPatternHandle). This scan pattern should be the one used to acquire the <i>Data</i> (third parameter), because the correction depends on the measurement coordinates. The scan pattern enables the conversion between index coordinates (i,j) and physical coordinates (in millimeter). Hence it should be a scan pattern that covers the coordinates of the <i>Data</i> (third parameter).
in,out	<i>Data</i>	A valid (non null) handle of data (DataHandle) pointing to data measured (acquired and processed) in a B-scan or in a volume scan.

```
6.14.3.12 correctImageFieldComplex() void correctImageFieldComplex (
    ImageFieldHandle ImageField,
    ScanPatternHandle Pattern,
    ComplexDataHandle Data )
```

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

Parameters

in	<i>ImageField</i>	A valid (non null) handle of the image field (ImageFieldHandle), previously created with one of the functions createImageField or createImageFieldFromProbe . Besides, the image field has already been determined with the help of the function determineImageField or determineImageFieldWithMask .
in	<i>Pattern</i>	A valid (non null) handle of a scan pattern (ScanPatternHandle). This scan pattern should be the one used to acquire the <i>Data</i> (third parameter), because the correction depends on the measurement coordinates. The scan pattern enables the conversion between index coordinates (i,j) and physical coordinates (in millimeter). Hence it should be a scan pattern that covers the coordinates of the <i>Data</i> (third parameter).
in,out	<i>Data</i>	A valid (non null) handle of complex data (ComplexDataHandle) pointing to data measured (acquired and processed) in a B-scan or in a volume scan.

```
6.14.3.13 correctSurface() void correctSurface (
    ImageFieldHandle ImageField,
    ScanPatternHandle Pattern,
    DataHandle Surface )
```

Applies the image field correction to the given Surface. Surface must contain depth values as a function of x/y coordinates.

Parameters

in	<i>ImageField</i>	A valid (non null) handle of the image field (ImageFieldHandle), previously created with one of the functions createlImageField or createlImageFieldFromProbe . Besides, the image field has already been determined with the help of the function determinelImageField or determinelImageFieldWithMask .
in	<i>Pattern</i>	A valid (non null) handle of a scan pattern (ScanPatternHandle). The scan pattern enables the conversion between index coordinates (i,j) and physical coordinates (in millimeter). Hence it should be a scan pattern that covers the coordinates of the <i>Surface</i> (third parameter).
in,out	<i>Surface</i>	A 2D data array, in a DataHandle structure, whose entries are the depth of the surface at each (x,y) coordinate, expressed in millimeter. This surface will be corrected. Notice that, unlike scans, the first coordinate is the x-axis and the second coordinate is the y-axis.

```
6.14.3.14 createlImageField() ImageFieldHandle createImageField (
    void )
```

Creates an object holding image field data.

Returns

A valid handle of the newly created image field ([ImageFieldHandle](#)).

```
6.14.3.15 createlImageFieldFromProbe() ImageFieldHandle createImageFieldFromProbe (
    ProbeHandle Probe )
```

Creates an object holding image field data from the specified Probe Handle.

Parameters

in	<i>Probe</i>	A valid (non null) probe handle (ProbeHandle), previously generated with the function initProbe .
----	--------------	---

Returns

A valid handle of the newly created image field ([ImageFieldHandle](#)).

```
6.14.3.16 cropColoredData() void cropColoredData (
    ColoredDataHandle Data,
    Direction Dir,
    int IndexMax,
    int IndexMin )
```

Crops the colored data along the desired direction at the given indices. Upon return the data will only contain those slices whose indices where in the interval [IndexMin, IndexMax), counted along the cropping direction.

Parameters

in,out	<i>Data</i>	A valid (non null) colored data handle of the data (ColoredDataHandle). These data will be cropped.
in	<i>Dir</i>	The physical direction (Direction) along which the existing data will be cropped.
in	<i>IndexMax</i>	One-past-the-last slice that will be kept. This index is zero-based.
in	<i>IndexMin</i>	The first slice that will be kept. This index is zero-based.

```
6.14.3.17 cropComplexData() void cropComplexData (
    ComplexDataHandle Data,
    Direction Dir,
    int IndexMax,
    int IndexMin )
```

Crops the complex data along the desired direction at the given indices. Upon return the data will only contain those slices whose indices where in the interval [IndexMin, IndexMax), counted along the cropping direction.

Parameters

in,out	<i>Data</i>	A valid (non null) complex data handle of the data (ComplexDataHandle). These data will be cropped.
in	<i>Dir</i>	The physical direction (Direction) along which the existing data will be cropped.
in	<i>IndexMax</i>	One-past-the-last slice that will be kept. This index is zero-based.
in	<i>IndexMin</i>	The first slice that will be kept. This index is zero-based.

```
6.14.3.18 cropData() void cropData (
    DataHandle Data,
    Direction Dir,
    int IndexMax,
    int IndexMin )
```

Crops the data along the desired direction at the given indices. Upon return the data will only contain those slices whose indices where in the interval [IndexMin, IndexMax), counted along the cropping direction.

Parameters

in, out	<i>Data</i>	A valid (non null) data handle of the data (DataHandle). These data will be cropped.
in	<i>Dir</i>	The physical direction (Direction) along which the existing data will be cropped.
in	<i>IndexMax</i>	One-past-the-last slice that will be kept. This index is zero-based.
in	<i>IndexMin</i>	The first slice that will be kept. This index is zero-based.

6.14.3.19 determineImageField() void determineImageField (

```
    ImageFieldHandle ImageField,
    ScanPatternHandle Pattern,
    DataHandle Surface )
```

Determines the image field correction for the given surface data, previously measured with the given scan pattern.

Parameters

out	<i>ImageField</i>	A valid (non null) handle of the image field (ImageFieldHandle), previously created with one of the functions createImageField or createImageFieldFromProbe .
in	<i>Pattern</i>	A valid (non null) handle of a volume scan pattern (ScanPatternHandle), created with one of the functions createVolumePattern , createFreeformScanPattern3DFromLUT , or createFreeformScanPattern3D . The scan pattern should uniformly cover the whole field of view.

Warning

If the scan pattern is non uniform, or fails to cover some areas, the resulting image field corrections will be impaired. The scan pattern enables the conversion between index coordinates (i,j) and physical coordinates (in millimeter). Hence it should be a scan pattern that covers the coordinates of the *Surface* (third parameter).

Parameters

in	<i>Surface</i>	A 2D data array, in a DataHandle structure, whose entries are the depth of the surface at each (x,y) coordinate, expressed in millimeter. The surface can be calculated from a volume scan using the function determineSurface . Notice that, unlike scans, the first coordinate is the x-axis and the second coordinate is the y-axis.
----	----------------	---

The purpose of the image field is to compensate the deformations introduced by the optical elements (e.g. lenses). To that end, a measurement of the substrate surface is carried out, and the geometric correction is computed. The default calibration of an instrument needs not be re-computed, unless a new objective is installed, or the objective is the same but the desired reference surface is non planar (the user must supply the desired surface, which should actually be measured).

6.14.3.20 determineImageFieldWithMask() void determineImageFieldWithMask (

```
    ImageFieldHandle ImageField,
    ScanPatternHandle Pattern,
    DataHandle Surface,
    DataHandle Mask )
```

Determines the image field correction for the given surface data, previously measured with the given scan pattern. The positive entries of the mask determine the points that actually enter in the computation.

Parameters

out	<i>ImageField</i>	A valid (non null) handle of the image field (ImageFieldHandle), previously created with one of the functions createImageField or createImageFieldFromProbe .
in	<i>Pattern</i>	A valid (non null) handle of a volume scan pattern (ScanPatternHandle), created with one of the functions createVolumePattern , createFreeformScanPattern3DFromLUT , or createFreeformScanPattern3D . The scan pattern should uniformly cover the whole field of view.

Warning

If the scan pattern is non uniform, or fails to cover some areas, the resulting image field corrections will be impaired. The scan pattern enables the conversion between index coordinates (i,j) and physical coordinates (in millimeter). Hence it should be a scan pattern that covers the coordinates of the Surface (third parameter).

Parameters

in	<i>Surface</i>	A 2D data array, in a DataHandle structure, whose entries are the depth of the surface at each (x,y) coordinate, expressed in millimeter. The surface can be calculated from a volume scan using the function determineSurface . Notice that, unlike scans, the first coordinate is the x-axis and the second coordinate is the y-axis.
in	<i>Mask</i>	A 2D array, in stored a DataHandle structure, indicating which points of the Surface should be taken into account (positive entries in Mask). Negative entries in Mask identify points of the Surface which should not be considered in the computation. Notice that the entries are single-precision floating-point numbers. In case a 3D data structure is passed, only the first slice will be used (index zero along the third Direction).

The purpose of the image field is to compensate the deformations introduced by the optical elements (e.g. lenses). To that end, a measurement of the substrate surface is carried out, and the geometric correction is computed. The default calibration of an instrument needs not be re-computed, unless a new objective is installed, or the objective is the same but the desired reference surface is non planar (the user must supply the desired surface, which should actually be measured).

This function checks that the first two dimensions (in pixels) of *Surface* and *Mask* match each other.

```
6.14.3.21 flipColoredData() void flipColoredData (
    ColoredDataHandle Data,
    Direction FlippingDir )
```

Mirrors the data across a plane perpendicular to the given direction.

Parameters

in,out	<i>Data</i>	A valid (non null) complex data handle of the data (ComplexDataHandle). These data will be flipped.
in	<i>FlippingDir</i>	The physical direction (Direction) along which the existing data will be flipped.

```
6.14.3.22 flipComplexData() void flipComplexData (
    ComplexDataHandle Data,
    Direction FlippingDir )
```

Mirrors the data across a plane perpendicular to the given direction.

Parameters

in, out	<i>Data</i>	A valid (non null) complex data handle of the data (ComplexDataHandle). These data will be flipped.
in	<i>FlippingDir</i>	The physical direction (Direction) along which the existing data will be flipped.

```
6.14.3.23 flipData() void flipData (
    DataHandle Data,
    Direction FlippingDir )
```

Mirrors the data across a plane perpendicular to the given direction.

Parameters

in, out	<i>Data</i>	A valid (non null) data handle of the data (DataHandle). These data will be flipped.
in	<i>FlippingDir</i>	The physical direction (Direction) along which the existing data will be flipped.

```
6.14.3.24 getColoredDataSliceIndex() void getColoredDataSliceIndex (
    ColoredDataHandle Data,
    ColoredDataHandle Slice,
    Direction SliceNormalDirection,
    int Index )
```

Returns a slice of colored data perpendicular to the specified direction at the specified index.

Parameters

in	<i>Data</i>	A valid (non null) colored data handle of the existing, three-dimensional data (ColoredDataHandle). These data will not be modified.
out	<i>Slice</i>	A valid (non null) complex data handle (ColoredDataHandle), where the data of the slice will be written.
in	<i>SliceNormalDirection</i>	The physical direction (Direction) in which the existing data will be sliced. Currently only the Direction_3 (usually the Y-axis) is supported.
in	<i>Index</i>	The index of the desired slice along the direction <i>Dir</i> (zero-based, that is, the first slice is 0). The total number of slices can be obtained with the function getDataPropertyInt , and specifying the property Data_Size2 or Data_Size3 (depending on the <i>Dir</i> specified).

The colored data that will be sliced (*data*) should be three-dimensional, e.g., a sequence of B-scans. A slice is one of the B-scans, perpendicular to the specified direction *Dir*.

```
6.14.3.25 getColoredDataSlicePos() void getColoredDataSlicePos (
    ColoredDataHandle Data,
```

```
ColoredDataHandle Slice,
Direction SliceNormalDirection,
double Pos_mm )
```

Returns a slice of colored data perpendicular to the specified direction at the specified position.

Parameters

in	<i>Data</i>	A valid (non null) colored data handle of the existing, three-dimensional data (ColoredDataHandle). These data will not be modified.
out	<i>Slice</i>	A valid (non null) complex data handle (ComplexDataHandle), where the data of the slice will be written.
in	<i>SliceNormalDirection</i>	The physical direction (Direction) in which the existing data will be sliced. Currently only the Direction_3 (usually the Y-axis) is supported.
in	<i>Pos_mm</i>	The position of the desired slice along the direction <i>Dir</i> , expressed in millimeter. The total range of positions can be inquired with the function getDataPropertyFloat , specifying the property Data_Range2 or Data_Range3 (depending on the <i>Dir</i> specified). If the scan pattern has not been manipulated (e.g. shifted), the center position is 0 mm.

The colored data that will be sliced (*Data*) should be three-dimensional, e.g., a sequence of B-scans. A slice is one of the B-scans, perpendicular to the specified direction *Dir*. If a position intermediate between two measured B-scans is given, this function will pick the closest; no interpolation will take place.

6.14.3.26 [getComplexDataSliceIndex\(\)](#)

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

Returns a slice of complex data perpendicular to the specified direction at the specified index.

Parameters

in	<i>Data</i>	A valid (non null) complex data handle of the existing, three-dimensional data (ComplexDataHandle). These data will not be modified.
out	<i>Slice</i>	A valid (non null) complex data handle (ComplexDataHandle), where the data of the slice will be written.
in	<i>SliceNormalDirection</i>	The physical direction (Direction) in which the existing data will be sliced. Currently only the Direction_3 (usually the Y-axis) is supported.
in	<i>Index</i>	The index of the desired slice along the direction <i>Dir</i> (zero-based, that is, the first slice is 0). The total number of slices can be obtained with the function getDataPropertyInt , and specifying the property Data_Size2 or Data_Size3 (depending on the <i>Dir</i> specified).

The complex data that will be sliced (*Data*) should be three-dimensional, e.g., a sequence of B-scans. A slice is one of the B-scans, perpendicular to the specified direction *Dir*.

6.14.3.27 [getComplexDataSlicePos\(\)](#)

```
void getComplexDataSlicePos (
    ComplexDataHandle Data,
    ComplexDataHandle Slice,
```

```
Direction SliceNormalDirection,
double Pos_mm )
```

Returns a slice of complex data perpendicular to the specified direction at the specified position.

Parameters

in	<i>Data</i>	A valid (non null) complex data handle of the existing, three-dimensional data (ComplexDataHandle). These data will not be modified.
out	<i>Slice</i>	A valid (non null) complex data handle (ComplexDataHandle), where the data of the slice will be written.
in	<i>SliceNormalDirection</i>	The physical direction (Direction) in which the existing data will be sliced. Currently only the Direction_3 (usually the Y-axis) is supported.
in	<i>Pos_mm</i>	The position of the desired slice along the direction <i>Dir</i> , expressed in millimeter. The total range of positions can be inquired with the function getDataPropertyFloat , specifying the property Data_Range2 or Data_Range3 (depending on the <i>Dir</i> specified). If the scan pattern has not been manipulated (e.g. shifted), the center position is 0 mm.

The complex data that will be sliced (*Data*) should be three-dimensional, e.g., a sequence of B-scans. A slice is one of the B-scans, perpendicular to the specified direction *Dir*. If a position intermediate between two measured B-scans is given, this function will pick the closest; no interpolation will take place.

```
6.14.3.28 void getDataSliceAtIndex (
    DataHandle Data,
    DataHandle Slice,
    Direction SliceNormalDirection,
    int Index )
```

Returns a slice of data perpendicular to the specified direction at the specified index.

Parameters

in	<i>Data</i>	A valid (non null) data handle of the existing, three-dimensional data (DataHandle). These data will not be modified.
out	<i>Slice</i>	A valid (non null) data handle (DataHandle), where the data of the slice will be written.
in	<i>SliceNormalDirection</i>	The physical direction (Direction) in which the existing data will be sliced. Currently only the Direction_3 (usually the Y-axis) is supported.
in	<i>Index</i>	The index of the desired slice along the direction <i>Dir</i> (zero-based, that is, the first slice is 0). The total number of slices can be obtained with the function getDataPropertyInt , and specifying the property Data_Size2 or Data_Size3 (depending on the <i>Dir</i> specified).

The data that will be sliced (*Data*) should be three-dimensional, e.g., a sequence of B-scans. A slice is one of the B-scans, perpendicular to the specified direction *Dir*.

```
6.14.3.29 void getDataSliceAtPos (
    DataHandle Data,
    DataHandle Slice,
    Direction SliceNormalDirection,
    double Pos_mm )
```

Returns a slice of data perpendicular to the specified direction at the specified position.

Parameters

in	<i>Data</i>	A valid (non null) data handle of the existing, three-dimensional data (DataHandle). These data will not be modified.
out	<i>Slice</i>	A valid (non null) data handle (DataHandle), where the data of the slice will be written.
in	<i>SliceNormalDirection</i>	The physical direction (Direction) in which the existing data will be sliced. Currently only the Direction_3 (usually the Y-axis) is supported.
in	<i>Pos_mm</i>	The position of the desired slice along the direction <i>Dir</i> , expressed in millimeter. The total range of positions can be inquired with the function getDataPropertyFloat , specifying the property Data_Range2 or Data_Range3 (depending on the <i>Dir</i> specified). If the scan pattern has not been manipulated (e.g. shifted), the center position is 0 mm.

The data that will be sliced (*Data*) should be three-dimensional, e.g., a sequence of B-scans. A slice is one of the B-scans, perpendicular to the specified direction *Dir*. If a position intermediate between two measured B-scans is given, this function will pick the closest; no interpolation will take place.

```
6.14.3.30 getRawDataSliceAtIndex() void getRawDataSliceAtIndex (
    RawDataHandle Raw,
    RawDataHandle Slice,
    Direction SliceNormalDirection,
    int Index )
```

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

Parameters

in	<i>Raw</i>	A valid (non null) raw-data handle of the existing, three-dimensional raw data (RawDataHandle). These data will not be modified.
out	<i>Slice</i>	A valid (non null) raw-data handle (RawDataHandle), where the raw data of the slice will be written.
in	<i>SliceNormalDirection</i>	The physical direction (Direction) in which the existing data will be sliced. Currently only the Direction_3 (usually the Y-axis) is supported.
in	<i>Index</i>	The desired slice number in the direction <i>Dir</i> .

The raw data that will be sliced (*Raw*) should be three-dimensional, that is, a sequence of B-scans. A slice is one of the B-scans, perpendicular to the third direction (usually the Y-axis).

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

```
6.14.3.31 loadImageField() void loadImageField (
    ImageFieldHandle ImageField,
    const char * Path )
```

Loads data containing image field data.

Parameters

out	<i>ImageField</i>	A valid (non null) handle of the image field (ImageFieldHandle), previously created with one of the functions createImageField or createImageFieldFromProbe .
in	<i>Path</i>	Filename (including path), where the data will be read from.

```
6.14.3.32 normalizeData() void normalizeData (
    DataHandle Data,
    float Min,
    float Max )
```

Scales the given data in such a way, that the range [Min, Max] is mapped onto the range [0,1].

Parameters

in, out	<i>Data</i>	A valid (non null) data handle of the data (DataHandle). These data will be scaled.
in	<i>Min</i>	The lower bound of the data that will be mapped to 0 in <i>DataOut</i> .
in	<i>Max</i>	The upper bound of the data that will be mapped to 1 in <i>DataOut</i> .

```
6.14.3.33 saveImageField() void saveImageField (
    ImageFieldHandle ImageField,
    const char * Path )
```

Saves data containing image field data.

Parameters

in	<i>ImageField</i>	A valid (non null) handle of the image field (ImageFieldHandle), previously created with one of the functions createImageField or createImageFieldFromProbe .
in	<i>Path</i>	Filename (including path), where the data will be saved. If the file exists, it will be (merciless) overwritten.

```
6.14.3.34 separateColoredData() void separateColoredData (
    ColoredDataHandle Data1,
    ColoredDataHandle Data2,
    int SeparationIndex,
    Direction Dir )
```

Separates the data at the given index at specific separation direction. The first part of the separated data will remain in *Data1*, the second separated in *Data2*.

Parameters

in, out	<i>Data1</i>	A valid (non null) colored data handle of the data (ColoredDataHandle). Upon return, only the first part will remain in this container.
out	<i>Data2</i>	A valid (non null) colored data handle to the second part of the data (ColoredDataHandle).
in	<i>SeparationIndex</i>	The first slice of the second part, or one-past-the-last slice kept in the first part.
in	<i>Dir</i>	The physical direction (Direction) along which the separation will take place.

```
6.14.3.35 separateComplexData() void separateComplexData (
    ComplexDataHandle Data1,
    ComplexDataHandle Data2,
    int SeparationIndex,
    Direction Dir )
```

Separates the data at the given index at specific separation direction. The first part of the separated data will remain in Data1, the second separated in Data2.

Parameters

in,out	<i>Data1</i>	A valid (non null) complex data handle of the data (ComplexDataHandle). Upon return, only the first part will remain in this container.
out	<i>Data2</i>	A valid (non null) complex data handle to the second part of the data (ComplexDataHandle).
in	<i>SeparationIndex</i>	The first slice of the second part, or one-past-the-last slice kept in the first part.
in	<i>Dir</i>	The physical direction (Direction) along which the separation will take place.

```
6.14.3.36 separateData() void separateData (
    DataHandle Data1,
    DataHandle Data2,
    int SeparationIndex,
    Direction Dir )
```

Separates the data at the given index at specific separation direction. The first part of the separated data will remain in Data1, the second separated in Data2.

Parameters

in,out	<i>Data1</i>	A valid (non null) data handle of the data (DataHandle). Upon return, only the first part will remain in this container.
out	<i>Data2</i>	A valid (non null) data handle to the second part of the data (DataHandle).
in	<i>SeparationIndex</i>	The first slice of the second part, or one-past-the-last slice kept in the first part.
in	<i>Dir</i>	The physical direction (Direction) along which the separation will take place.

```
6.14.3.37 setImageFieldInProbe() void setImageFieldInProbe (
    ImageFieldHandle ImageField,
    ProbeHandle Probe )
```

Sets the specified image field to the specified Probe handle. Notice that no probe file will be automatically saved.

Parameters

in	<i>ImageField</i>	A valid (non null) handle of the image field (ImageFieldHandle), previously created with one of the functions createImageField or createImageFieldFromProbe . Besides, the image field has already been determined with the help of the function determineImageField or determineImageFieldWithMask .
in	<i>Probe</i>	A valid (non null) probe handle (ProbeHandle), previously generated with the function initProbe .

```
6.14.3.38 transposeAndScaleData() void transposeAndScaleData (
    DataHandle DataIn,
    DataHandle DataOut,
    float Min,
    float Max )
```

Transposes the given data and writes the result to DataOut. First and second axes will be swaped, and the range of the entries will be scaled in such a way, that the range [Min,Max] will be mapped onto the range [0,1].

Parameters

in	<i>DataIn</i>	A valid (non null) data handle of the input data (DataHandle). These data should be multi-dimensional. These data will not be modified.
out	<i>DataOut</i>	A valid (non null) data handle of the output data (DataHandle). These data will be a scaled and transposed copy of the input data, that is, the first and the second axes will be swaped (usually Z- and X-axes).
in	<i>Min</i>	The lower bound of the data that will be mapped to 0 in DataOut.
in	<i>Max</i>	The upper bound of the data that will be mapped to 1 in DataOut.

```
6.14.3.39 transposeData() void transposeData (
    DataHandle DataIn,
    DataHandle DataOut )
```

Transposes the given data and writes the result to DataOut. First and second axes will be swaped.

Parameters

in	<i>DataIn</i>	A valid (non null) data handle of the input data (DataHandle). These data should be multi-dimensional. These data will not be modified.
out	<i>DataOut</i>	A valid (non null) data handle of the output data (DataHandle). These data will be a copy of the input data, except that the first and the second axes will be swaped (usually Z- and X-axes).

```
6.14.3.40 transposeDataInplace() void transposeDataInplace (
    DataHandle Data )
```

Transposes the given Data. First and second axes will be swaped.

Parameters

in, out	<i>Data</i>	A valid (non null) data handle of the data (DataHandle). These data will be modified: the first and the second axes will be swaped (usually Z- and X-axes).
---------	-------------	---

6.15 ProbeCalibration

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

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

Warning

ThorImageOCT uses these functions to calibrate the galvo, assuming a very specific sequence of actions and conditions, as explained in the ThorImageOCT. For these functions to properly work, the user need to re-create the same sequence of actions and conditions.

The galvo offset/factor / Draw & Scan overlay calibration assumes a sample with a triangular dot pattern with a fixed edge length which must be aligned parallel to the video image edges.

6.16 Doppler

Doppler Processing Routines.

Typedefs

- `typedef struct C_DopplerProcessing * DopplerProcessingHandle`
Handle used for Doppler processing.

Enumerations

- `enum DopplerPropertyInt {`
`Doppler_Averaging_1,`
`Doppler_Averaging_2,`
`Doppler_Stride_1,`
`Doppler_Stride_2 }`
Values that determine the behaviour of the Doppler processing routines.
- `enum DopplerPropertyFloat {`
`Doppler_RefractiveIndex,`
`Doppler_ScanRate_Hz,`
`Doppler_CenterWavelength_nm,`
`Doppler_DopplerAngle_Deg }`
Values that determine the behaviour of the Doppler processing routines.
- `enum DopplerFlag { Doppler_VelocityScaling }`
Flags that determine the behaviour of the Doppler processing routines.

Functions

- `SPECTRALRADAR_API DopplerProcessingHandle createDopplerProcessing (void)`
Returns a handle for the use of Doppler-computation routines.
- `SPECTRALRADAR_API DopplerProcessingHandle createDopplerProcessingForFile (OCTFileHandle File)`
Returns a handle for the use of Doppler-computation routines. The handle is created based on a saved OCT file.
- `SPECTRALRADAR_API int getDopplerPropertyInt (DopplerProcessingHandle Handle, DopplerPropertyInt Property)`
Gets the value of the given Doppler processing property.
- `SPECTRALRADAR_API void setDopplerPropertyInt (DopplerProcessingHandle Handle, DopplerPropertyInt Property, int Value)`
Sets the value of the given Doppler processing property.
- `SPECTRALRADAR_API double getDopplerPropertyFloat (DopplerProcessingHandle Handle, DopplerPropertyFloat Property)`
Gets the value of the given Doppler processing property.
- `SPECTRALRADAR_API void setDopplerPropertyFloat (DopplerProcessingHandle Handle, DopplerPropertyFloat Property, float Value)`
Sets the value of the given Doppler processing property.
- `SPECTRALRADAR_API BOOL getDopplerFlag (DopplerProcessingHandle Handle, DopplerFlag Flag)`
Gets the given Doppler processing flag.
- `SPECTRALRADAR_API void setDopplerFlag (DopplerProcessingHandle Handle, DopplerFlag Flag, BOOL OnOff)`
Sets the given Doppler processing flag.
- `SPECTRALRADAR_API void setDopplerAmplitudeOutput (DopplerProcessingHandle Handle, DataHandle AmpOut)`

- Sets the location of the resulting Doppler amplitude output.
- **SPECTRALRADAR_API** void `setDopplerPhaseOutput` (`DopplerProcessingHandle` Handle, `DataHandle` PhasesOut)
 - Sets the location of the resulting Doppler phase output.
- **SPECTRALRADAR_API** void `executeDopplerProcessing` (`DopplerProcessingHandle` Handle, `ComplexDataHandle` Input)
 - Executes the Doppler processing of the input data and returns phases and amplitudes.
- **SPECTRALRADAR_API** void `dopplerVelocityToPhase` (`DopplerProcessingHandle` Doppler, `DataHandle` In←Out)
 - Scales flow velocities computed by Doppler OCT back to original phase differences.
- **SPECTRALRADAR_API** void `clearDopplerProcessing` (`DopplerProcessingHandle` Handle)
 - Closes the Doppler processing routines and frees the memory that has been allocated for these to work properly.
- **SPECTRALRADAR_API** void `getDopplerOutputSize` (`DopplerProcessingHandle` Handle, int Size1In, int Size2In, int *Size1Out, int *Size2Out)
 - Returns the final size of the Doppler output if `executeDopplerProcessing` is executed using data of the specified input size.

6.16.1 Detailed Description

Doppler Processing Routines.

6.16.2 Typedef Documentation

6.16.2.1 **DopplerProcessingHandle** `DopplerProcessingHandle`

Handle used for Doppler processing.

Definition at line 108 of file [SpectralRadar_Handles.h](#).

6.16.3 Enumeration Type Documentation

6.16.3.1 **DopplerFlag** `enum DopplerFlag`

Flags that determine the behaviour of the Doppler processing routines.

Enumerator

<code>Doppler_VelocityScaling</code>	Averaging along the first axis, usually the longitudinal axis (z)
--------------------------------------	---

Definition at line 434 of file [SpectralRadar_Properties.h](#).

6.16.3.2 DopplerPropertyFloat enum DopplerPropertyFloat

Values that determine the behaviour of the Doppler processing routines.

Enumerator

Doppler_RefractiveIndex	Averaging along the first axis, usually the longitudinal axis (z)
Doppler_ScanRate_Hz	Scan Rate (in Hz) that was used to acquire the Doppler data to be processed. This is only required for computing the actual velocity scaling.
Doppler_CenterWavelength_nm	Center Wavelength (in nanometers) that was used to acquire the Doppler data to be processed. This is only required for computing the actual velocity scaling.
Doppler_DopplerAngle_Deg	Angle of the Doppler detection beam to the normal. This is only required for computing the actual velocity scaling.

Definition at line 420 of file [SpectralRadar_Properties.h](#).

6.16.3.3 DopplerPropertyInt enum DopplerPropertyInt

Values that determine the behaviour of the Doppler processing routines.

Enumerator

Doppler_Averaging_1	Averaging along the first axis, usually the longitudinal axis (z)
Doppler_Averaging_2	Averaging along the first axis, usually the first transversal axis (x)
Doppler_Stride_1	Step size for calculating the doppler processing in the longitudinal axis (z). Stride needs to be smaller or equal to Doppler_Averaging_1 and larger or equal to 1.
Doppler_Stride_2	Step size for calculating the doppler processing in the transversal axis (x). Stride needs to be smaller or equal to Doppler_Averaging_2 and larger or equal to 1.

Definition at line 406 of file [SpectralRadar_Properties.h](#).

6.16.4 Function Documentation

6.16.4.1 clearDopplerProcessing() void clearDopplerProcessing (DopplerProcessingHandle Handle)

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

Parameters

in	Handle	A handle of Doppler processing routines (DopplerProcessingHandle). If the handle is a nullptr, this function does nothing. In most cases this handle will have been previously obtained with the function createDopplerProcessing .
----	--------	---

6.16.4.2 createDopplerProcessing() `DopplerProcessingHandle createDopplerProcessing (`
`void)`

Returns a handle for the use of Doppler-computation routines.

Returns

`DopplerProcessingHandle` to the created Doppler routines.

6.16.4.3 createDopplerProcessingForFile() `DopplerProcessingHandle createDopplerProcessingForFile (`
`OCTFileHandle File)`

Returns a handle for the use of Doppler-computation routines. The handle is created based on a saved OCT file.

Returns

`DopplerProcessingHandle` to the created Doppler routines.

6.16.4.4 dopplerVelocityToPhase() `void dopplerVelocityToPhase (`
`DopplerProcessingHandle Handle,`
`DataHandle InOut)`

Scales flow velocities computed by Doppler OCT back to original phase differences.

Parameters

in	<code>Handle</code>	A valid (non null) handle of Doppler processing routines (<code>DopplerProcessingHandle</code>), obtained with the function <code>createDopplerProcessing</code> .
in, out	<code>InOut</code>	A handle of data representing first velocity data that will then be modified to contain velocity data.

This requires the Doppler scan rate, Doppler angle and center velocity of the Doppler object to be set correctly.

6.16.4.5 executeDopplerProcessing() `void executeDopplerProcessing (`
`DopplerProcessingHandle Handle,`
`ComplexDataHandle Input)`

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

Parameters

in	<i>Handle</i>	A valid (non null) handle of Doppler processing routines (DopplerProcessingHandle), obtained with the function createDopplerProcessing .
in	<i>Input</i>	A valid (non null) handle of complex data (ComplexDataHandle). These data should have previously obtained by invoking the functions createComplexData , setComplexDataOutput and executeProcessing .

Doppler processing takes place after the standard processing. It takes as input complex data computed by the standard processing, and during execution it writes amplitudes and phases, provided either 8 or both) of the function [setDopplerAmplitudeOutput](#) or [setDopplerPhaseOutput](#) have previously been invoked.

```
6.16.4.6 getDopplerFlag() void getDopplerFlag (
    DopplerProcessingHandle Handle,
    DopplerFlag Flag )
```

Gets the given Doppler processing flag.

Parameters

in	<i>Handle</i>	A valid (non null) handle of Doppler processing routines (DopplerProcessingHandle), obtained with the function createDopplerProcessing .
in	<i>Flag</i>	The desired boolean flag (DopplerFlag).

Returns

The boolean value of the selected flag.

```
6.16.4.7 getDopplerOutputSize() void getDopplerOutputSize (
    DopplerProcessingHandle Handle,
    int Size1In,
    int Size2In,
    int * Size1Out,
    int * Size2Out )
```

Returns the final size of the Doppler output if [executeDopplerProcessing](#) is executed using data of the specified input size.

Parameters

in	<i>Handle</i>	A valid (non null) handle of Doppler processing routines (DopplerProcessingHandle), obtained with the function createDopplerProcessing .
in	<i>Size1In</i>	The value of the Data_Size1 property (DataPropertyInt) of the complex-data that will be used as input. In the default orientation, this is the number of pixels along the z-axis.
in	<i>Size2In</i>	The value of the Data_Size2 property (DataPropertyInt) of the complex-data that will be used as input. In the default orientation, this is the number of pixels along the x-axis.
out	<i>Size1Out</i>	The value of the Data_Size1 property (DataPropertyInt) of the amplitude/phase data that will result upon invocation of the function executeDopplerProcessing . In the default orientation, this is the number of pixels along the z-axis.
out	<i>Size2Out</i>	The value of the Data_Size2 property (DataPropertyInt) of the amplitude/phase data that will result upon invocation of the function executeDopplerProcessing . In the default orientation, this is the number of pixels along the x-axis.

6.16.4.8 getDopplerPropertyFloat() double getDopplerPropertyFloat (
 DopplerProcessingHandle Handle,
 DopplerPropertyFloat Property)

Gets the value of the given Doppler processing property.

Parameters

in	<i>Handle</i>	A valid (non null) handle of Doppler processing routines (DopplerProcessingHandle), obtained with the function createDopplerProcessing .
in	<i>Property</i>	The desired floating-point property (DopplerPropertyFloat).

Returns

The value of the desired property.

6.16.4.9 getDopplerPropertyInt() int getDopplerPropertyInt (
 DopplerProcessingHandle Handle,
 DopplerPropertyInt Property)

Gets the value of the given Doppler processing property.

Parameters

in	<i>Handle</i>	A valid (non null) handle of Doppler processing routines (DopplerProcessingHandle), obtained with the function createDopplerProcessing .
in	<i>Property</i>	The desired integer property (DopplerPropertyInt).

Returns

The value of the desired property.

6.16.4.10 setDopplerAmplitudeOutput() void setDopplerAmplitudeOutput (
 DopplerProcessingHandle Handle,
 DataHandle AmpOut)

Sets the location of the resulting Doppler amplitude output.

Parameters

in	<i>Handle</i>	A valid (non null) handle of Doppler processing routines (DopplerProcessingHandle), obtained with the function createDopplerProcessing .
in	<i>AmpOut</i>	A valid (non null) handle of data (DataHandle), where the resulting amplitudes of the Doppler computation will be written. The right number of dimensions, sizes, and ranges will be automatically adjusted by the function executeDopplerProcessing .

```
6.16.4.11 setDopplerFlag() void setDopplerFlag (
    DopplerProcessingHandle Handle,
    DopplerFlag Flag,
    BOOL OnOff )
```

Sets the given Doppler processing flag.

Parameters

in	<i>Handle</i>	A valid (non null) handle of Doppler processing routines (DopplerProcessingHandle), obtained with the function createDopplerProcessing .
in	<i>Flag</i>	The selected boolean flag (DopplerFlag).
in	<i>OnOff</i>	The desired boolean value for the selected flag.

```
6.16.4.12 setDopplerPhaseOutput() void setDopplerPhaseOutput (
    DopplerProcessingHandle Handle,
    DataHandle PhasesOut )
```

Sets the location of the resulting Doppler phase output.

Parameters

in	<i>Handle</i>	A valid (non null) handle of Doppler processing routines (DopplerProcessingHandle), obtained with the function createDopplerProcessing .
in	<i>PhasesOut</i>	A valid (non null) handle of data (DataHandle), where the resulting phases of the Doppler computation will be written. The right number of dimensions, sizes, and ranges will be automatically adjusted by the function executeDopplerProcessing .

```
6.16.4.13 setDopplerPropertyFloat() void setDopplerPropertyFloat (
    DopplerProcessingHandle Handle,
    DopplerPropertyFloat Property,
    float Value )
```

Sets the value of the given Doppler processing property.

Parameters

in	<i>Handle</i>	A valid (non null) handle of Doppler processing routines (DopplerProcessingHandle), obtained with the function createDopplerProcessing .
in	<i>Property</i>	The selected floating-point property (DopplerPropertyFloat).
in	<i>Value</i>	The desired value for the selected property.

```
6.16.4.14 setDopplerPropertyInt() void setDopplerPropertyInt (
    DopplerProcessingHandle Handle,
    DopplerPropertyInt Property,
    int Value )
```

Sets the value of the given Doppler processing property.

Parameters

in	<i>Handle</i>	A valid (non null) handle of Doppler processing routines (DopplerProcessingHandle), obtained with the function createDopplerProcessing .
in	<i>Property</i>	The selected integer property (DopplerPropertyInt).
in	<i>Value</i>	The desired value for the selected property.

6.17 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.

6.17.1 Detailed Description

Service functions for additional analyzing of OCT functionality.

6.17.2 Function Documentation

6.17.2.1 calcContrast() void calcContrast (
 DataHandle ApodizedSpectrum,
 DataHandle Contrast)

Computes the contrast for the specified (apodized) spectrum.

Parameters

in	<i>ApodizedSpectrum</i>	The spectrum after offset subtraction and apodization. This spectrum can be obtained using the functions setApodizedSpectrumOutput and executeProcessing in succession.
out	<i>Contrast</i>	A valid (non null) data handle (DataHandle). Its dimensions will be automatically be adjusted.

The contrast is a measure of the amount of information in the interference pattern as a fraction of the total signal. The computed values are expressed as percentage of the measured amplitudes, for each camera pixel.

6.18 Settings

Direct access to INI files and settings.

Typedefs

- **typedef struct C_Settings *** [SettingsHandle](#)
Handle for saving settings on disk.

Functions

- **SPECTRALRADAR_API** [SettingsHandle](#) [initSettingsFile](#) (const char *Path)
*Loads a settings file (usually *.ini); and prepares its properties to be read.*
- **SPECTRALRADAR_API** int [getSettingsEntryInt](#) ([SettingsHandle](#) SettingsFile, const char *Node, int Default←Value)
Gets an integer number from the specified ini file (see [SettingsHandle](#) and [initSettingsFile](#));.
- **SPECTRALRADAR_API** double [getSettingsEntryFloat](#) ([SettingsHandle](#) SettingsFile, const char *Node, double DefaultValue)
Gets an floating point number from the specified ini file (see [SettingsHandle](#) and [initSettingsFile](#));.
- **SPECTRALRADAR_API** void [getSettingsEntryFloatArray](#) ([SettingsHandle](#) SettingsFile, const char *Node, const double *DefaultValues, double *Values, int *Size)
Gets an array of floating point numbers from the specified ini file (see [SettingsHandle](#) and [initSettingsFile](#));.
- **SPECTRALRADAR_API** const char * [getSettingsEntryString](#) ([SettingsHandle](#) SettingsFile, const char *Node, const char *Default)
Gets a string from the specified ini file (see [SettingsHandle](#) and [initSettingsFile](#));. The resulting const char ptr will be valid until the settings file is closed by [closeSettingsFile](#)).*
- **SPECTRALRADAR_API** void [setSettingsEntryInt](#) ([SettingsHandle](#) SettingsFile, const char *Node, int Value)
Sets an integer entry in the specified ini file (see [SettingsHandle](#) and [initSettingsFile](#));.
- **SPECTRALRADAR_API** void [setSettingsEntryFloat](#) ([SettingsHandle](#) SettingsFile, const char *Node, double Value)
Sets a floating point entry in the specified ini file (see [SettingsHandle](#) and [initSettingsFile](#));.
- **SPECTRALRADAR_API** void [setSettingsEntryString](#) ([SettingsHandle](#) SettingsFile, const char *Node, const char *Value)
Sets a string in the specified ini file (see [SettingsHandle](#) and [initSettingsFile](#));.
- **SPECTRALRADAR_API** void [saveSettings](#) ([SettingsHandle](#) SettingsFile)
Saves the changes to the specified Settings file.
- **SPECTRALRADAR_API** void [closeSettingsFile](#) ([SettingsHandle](#) Handle)
Closes the specified ini file and stores the set entries .

6.18.1 Detailed Description

Direct access to INI files and settings.

6.18.2 Typedef Documentation

6.18.2.1 `SettingsHandle` [SettingsHandle](#)

Handle for saving settings on disk.

Definition at line 158 of file [SpectralRadar_Handles.h](#).

6.18.3 Function Documentation

6.18.3.1 `closeSettingsFile()` [closeSettingsFile](#) (

`SettingsHandle Handle`)

Closes the specified ini file and stores the set entries (.

See also

[SettingsHandle](#), [initSettingsFile](#)).

Parameters

in	<code>Handle</code>	A handle of settings (SettingsHandle). If the handle is a nullptr, this function does nothing. In most cases this handle will have been previously obtained with the function initSettingsFile .
----	---------------------	--

6.18.3.2 `getSettingsEntryFloat()` [getSettingsEntryFloat](#) (

`SettingsHandle SettingsFile,`
`const char * Node,`
`double DefaultValue`)

Gets an floating point number from the specified ini file (see [SettingsHandle](#) and [initSettingsFile](#)):.

6.18.3.3 `getSettingsEntryFloatArray()` [getSettingsEntryFloatArray](#) (

`SettingsHandle SettingsFile,`
`const char * Node,`
`const double * DefaultValues,`
`double * Values,`
`int * Size`)

Gets an array of floating point numbers from the specified ini file (see [SettingsHandle](#) and [initSettingsFile](#)):.

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

Gets an integer number from the specified ini file (see [SettingsHandle](#) and [initSettingsFile](#));.

```
6.18.3.5 getSettingsEntryString() const char * getSettingsEntryString (
    SettingsHandle SettingsFile,
    const char * Node,
    const char * Default )
```

Gets a string from the specified ini file (see [SettingsHandle](#) and [initSettingsFile](#));. The resulting const char* ptr will be valid until the settings file is closed by [closeSettingsFile](#)).

```
6.18.3.6 initSettingsFile() SettingsHandle initSettingsFile (
    const char * Path )
```

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

```
6.18.3.7 saveSettings() void saveSettings (
    SettingsHandle SettingsFile )
```

Saves the changes to the specified Settings file.

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

Sets a floating point entry in the specified ini file (see [SettingsHandle](#) and [initSettingsFile](#));.

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

Sets an integer entry in the specified ini file (see [SettingsHandle](#) and [initSettingsFile](#));.

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

Sets a string in the specified ini file (see [SettingsHandle](#) and [initSettingsFile](#));.

6.19 Coloring

Functions used for coloring of floating point data.

TypeDefs

- `typedef struct C_Coloring32Bit * ColoringHandle`
Handle for routines that color available scans for displaying.

Enumerations

- `enum ColorScheme {`
`ColorScheme_BlackAndWhite = 0,`
`ColorScheme_Inverted = 1,`
`ColorScheme_Color = 2,`
`ColorScheme_BlackAndOrange = 3,`
`ColorScheme_BlackAndRed = 4,`
`ColorScheme_BlackRedAndYellow = 5,`
`ColorScheme_DopplerPhase = 6,`
`ColorScheme_BlueAndBlack = 7,`
`ColorScheme_PolarizationRetardation = 8,`
`ColorScheme_GreenBlueAndBlack = 9,`
`ColorScheme_BlackAndRedYellow = 10,`
`ColorScheme_TransparentAndWhite = 11,`
`ColorScheme_GreenBlueWhiteRedYellow = 12,`
`ColorScheme_BlueGreenBlackYellowRed = 13,`
`ColorScheme_RedGreenBlue = 14,`
`ColorScheme_GreenBlueRed = 15,`
`ColorScheme_BlueRedGreen = 16,`
`ColorScheme_GreenBlueRedGreen = 17,`
`ColorScheme_BlueRedGreenBlue = 18,`
`ColorScheme_Inverse_RedGreenBlue = 19,`
`ColorScheme_Inverse_GreenBlueRed = 20,`
`ColorScheme_Inverse_BlueRedGreen = 21,`
`ColorScheme_Inverse_GreenBlueRedGreen = 22,`
`ColorScheme_Inverse_BlueRedGreenBlue = 23,`
`ColorScheme_RedYellowGreenBlueRed = 24,`
`ColorScheme_RedGreenBlueRed = 25,`
`ColorScheme_Inverse_RedGreenBlueRed = 26,`
`ColorScheme_RedYellowBlue = 27,`
`ColorScheme_Inverse_RedYellowBlue = 28,`
`ColorScheme_DEM_Normal = 29,`
`ColorScheme_Inverse_DEM_Normal = 30,`
`ColorScheme_DEM_Blind = 31,`
`ColorScheme_Inverse_DEM_Blind = 32,`
`ColorScheme_WhiteBlackWhite = 33,`
`ColorScheme_BlackWhiteBlack = 34 }`
selects the ColorScheme of the data to transform real data to colored data.

- `enum ColoringByteOrder {`
`Coloring_RGBA = 0,`
`Coloring_BGRA = 1,`
`Coloring_ARGB = 2 }`

Selects the byte order of the coloring to be applied.

- enum `ColorEnhancement` {
 `ColorEnhancement_None` = 0,
 `ColorEnhancement_Sine` = 1,
 `ColorEnhancement_Parable` = 2,
 `ColorEnhancement_Cubic` = 3,
 `ColorEnhancement_Sqrt` = 4 }

Selects the byte order of the coloring to be applied.

Functions

- **SPECTRALRADAR_API** `ColoringHandle createColoring32Bit` (`ColorScheme Color`, `ColoringByteOrder ByteOrder`)
Creates processing that can be used to color given floating point B-scans to 32 bit colored images.
- **SPECTRALRADAR_API** `ColoringHandle createCustomColoring32Bit` (`int LUTSize`, `unsigned long *LUT`)
Create custom coloring using the specified color look-up-table.
- **SPECTRALRADAR_API** `void setColoringBoundaries` (`ColoringHandle Coloring`, `float Min_dB`, `float Max_dB`)
Sets the boundaries in dB which are used by the coloring algorithm to map colors to floating point values in dB.
- **SPECTRALRADAR_API** `void setColoringEnhancement` (`ColoringHandle Coloring`, `ColorEnhancement Enhancement`)
Selects a function for non-linear coloring to enhance (subjective) image impression.
- **SPECTRALRADAR_API** `void colorizeData` (`ColoringHandle Coloring`, `DataHandle Data`, `ColoredDataHandle ColoredData`, `BOOL Transpose`)
Colors a given data object (`DataHandle`) into a given colored object (`ColoredDataHandle`).
- **SPECTRALRADAR_API** `void colorizeDopplerData` (`ColoringHandle AmpColoring`, `ColoringHandle Phase←Coloring`, `DataHandle AmpData`, `DataHandle PhaseData`, `ColoredDataHandle Output`, `double MinSignal_dB`, `BOOL Transpose`)
Colors a two given data object (`DataHandle`) using overlay and intensity to represent phase and amplitude data. Used for Doppler imaging.
- **SPECTRALRADAR_API** `void colorizeDopplerDataEx` (`ColoringHandle AmpColoring`, `ColoringHandle PhaseColoring[2]`, `DataHandle AmpData`, `DataHandle PhaseData`, `ColoredDataHandle Output`, `double MinSignal_dB`, `BOOL Transpose`)
Colors a two given data object (`DataHandle`) using overlay and intensity to represent phase and amplitude data. Used for Doppler imaging. In the extended version, two `ColoringHandle`s can be specified, two provide different coloring for increasing and decreasing phase, for example.
- **SPECTRALRADAR_API** `void clearColoring` (`ColoringHandle Handle`)
Clears the coloring previously created by `createColoring32Bit`.

6.19.1 Detailed Description

Functions used for coloring of floating point data.

6.19.2 Typedef Documentation

6.19.2.1 `ColoringHandle` `ColoringHandle`

Handle for routines that color available scans for displaying.

Definition at line 127 of file `SpectralRadar_Handles.h`.

6.19.3 Enumeration Type Documentation

6.19.3.1 ColorEnhancement enum `ColorEnhancement`

Selects the byte order of the coloring to be applied.

Enumerator

<code>ColorEnhancement_None</code>	Use no color enhancement.
<code>ColorEnhancement_Sine</code>	Apply a sine function as enhancement.
<code>ColorEnhancement_Parable</code>	Apply a parable as enhancement.
<code>ColorEnhancement_Cubic</code>	Apply a cubic function as enhancement.
<code>ColorEnhancement_Sqrt</code>	Apply a sqrt function as enhancement.

Definition at line 543 of file [SpectralRadar_Types.h](#).

6.19.3.2 ColoringByteOrder enum `ColoringByteOrder`

Selects the byte order of the coloring to be applied.

Enumerator

<code>Coloring_RGB</code>	Byte order RGBA.
<code>Coloring_BGRA</code>	Byte order BGRA.
<code>Coloring_ARGB</code>	Byte order ARGB.

Definition at line 531 of file [SpectralRadar_Types.h](#).

6.19.3.3 ColorScheme enum `ColorScheme`

selects the ColorScheme of the data to transform real data to colored data.

Enumerator

<code>ColorScheme_BlackAndWhite</code>	Black and white (monochrome) coloring.
<code>ColorScheme_Inverted</code>	Black and white inverted (monochrome inverted) coloring.
<code>ColorScheme_Color</code>	colored
<code>ColorScheme_BlackAndOrange</code>	orange and black coloring
<code>ColorScheme_BlackAndRed</code>	red and black coloring
<code>ColorScheme_BlackRedAndYellow</code>	black, red and yellow coloring
<code>ColorScheme_DopplerPhase</code>	Doppler phase data coloring. Red and blue always colored in a range from -pi to +pi. Setting the boundaries for this color scheme is only allowed inbetween +pi and -pi

Enumerator

ColorScheme_BlueAndBlack	blue and black coloring
ColorScheme_PolarizationRetardation	colorful colorscheme
ColorScheme_GreenBlueAndBlack	Green, blue and black is used as one half of a Doppler color scheme.
ColorScheme_BlackAndRedYellow	Black, red, and yellow is used as one half of a Doppler color scheme.
ColorScheme_TransparentAndWhite	Transparent and white coloring for overlay and 3D volume rendering purposes.
ColorScheme_GreenBlueWhiteRedYellow	Green, blue, White, Red, and Yellow for polarization sensitive measurements.
ColorScheme_BlueGreenBlackYellowRed	Blue, green, black, yellow, and red for polarization sensitive measurements.
ColorScheme_RedGreenBlue	Red, green, and blue for polarization sensitive measurements.
ColorScheme_GreenBlueRed	Green, blue and red for polarization sensitive measurements.
ColorScheme_BlueRedGreen	Blue, red, and green for polarization sensitive measurements.
ColorScheme_GreenBlueRedGreen	Green, blue and red for polarization sensitive measurements.
ColorScheme_BlueRedGreenBlue	Blue, red, and green for polarization sensitive measurements.
ColorScheme_Inverse_RedGreenBlue	Red, green, and blue for polarization sensitive measurements.
ColorScheme_Inverse_GreenBlueRed	Green, blue and red for polarization sensitive measurements.
ColorScheme_Inverse_BlueRedGreen	Blue, red, and green for polarization sensitive measurements.
ColorScheme_Inverse_GreenBlueRedGreen	Green, blue and red for polarization sensitive measurements.
ColorScheme_Inverse_BlueRedGreenBlue	Blue, red, and green for polarization sensitive measurements.
ColorScheme_RedYellowGreenBlueRed	Red, yellow, green, blue, and red for polarization sensitive measurements.
ColorScheme_RedGreenBlueRed	Red, green, blue, and red for polarization sensitive measurements.
ColorScheme_Inverse_RedGreenBlueRed	Red, green, blue, and red for polarization sensitive measurements.
ColorScheme_RedYellowBlue	Red, yellow, and blue.
ColorScheme_Inverse_RedYellowBlue	Red, yellow, and blue.
ColorScheme_DEM_Normal	DEM.
ColorScheme_DEM_Blind	DEM.

Definition at line 458 of file [SpectralRadar_Types.h](#).

6.19.4 Function Documentation

6.19.4.1 clearColoring() `void clearColoring (`
`ColoringHandle Handle)`

Clears the coloring previously created by [createColoring32Bit](#).

Parameters

in	<code>Handle</code>	A handle of a coloring (ColoringHandle). If the handle is a nullptr, this function does nothing. In most cases this handle will have been previously obtained with the function createColoring32Bit .
----	---------------------	--

```
6.19.4.2 colorizeData() void colorizeData (
    ColoringHandle Coloring,
    DataHandle Data,
    ColoredDataHandle ColoredData,
    BOOL Transpose )
```

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

```
6.19.4.3 colorizeDopplerData() oid colorizeDopplerData (
    ColoringHandle AmpColoring,
    ColoringHandle PhaseColoring,
    DataHandle AmpData,
    DataHandle PhaseData,
    ColoredDataHandle Output,
    double MinSignal_dB,
    BOOL Transpose )
```

Colors a two given data object ([DataHandle](#)) using overlay and intensity to represent phase and amplitude data.
Used for Doppler imaging.

```
6.19.4.4 colorizeDopplerDataEx() oid colorizeDopplerDataEx (
    ColoringHandle AmpColoring,
    ColoringHandle PhaseColoring[2],
    DataHandle AmpData,
    DataHandle PhaseData,
    ColoredDataHandle Output,
    double MinSignal_dB,
    BOOL Transpose )
```

Colors a two given data object ([DataHandle](#)) using overlay and intensity to represent phase and amplitude data.
Used for Doppler imaging. In the extended version, two ColoringHandles can be specified, two provide different coloring for increasing and decreasing phase, for example.

```
6.19.4.5 createColoring32Bit() ColoringHandle createColoring32Bit (
    ColorScheme Color,
    ColoringByteOrder ByteOrder )
```

Creates processing that can be used to color given floating point B-scans to 32 bit colored images.

Parameters

<i>Color</i>	The color-table to be used
<i>ByteOrder</i>	The byte order the coloring is supposed to use.

Returns

The handle ([ColoringHandle](#)) to the coloring algorithm.

6.19.4.6 `createCustomColoring32Bit()` `ColoringHandle createCustomColoring32Bit (`

```
    int LUTSize,  
    unsigned long * LUT )
```

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

6.19.4.7 `setColoringBoundaries()` `void setColoringBoundaries (`

```
    ColoringHandle Colorng,  
    float Min_dB,  
    float Max_dB )
```

Sets the boundaries in dB which are used by the coloring algorithm to map colors to floating point values in dB.

6.19.4.8 `setColoringEnhancement()` `void setColoringEnhancement (`

```
    ColoringHandle Coloring,  
    ColorEnhancement Enhancement )
```

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

6.20 Camera

Functions for acquiring camera video images.

Functions

- **SPECTRALRADAR_API** void `getMaxCameralImageSize` (`OCTDeviceHandle Dev, int *SizeX, int *SizeY`)
Returns the maximum possible camera image size for the current device.
- **SPECTRALRADAR_API** void `getCameralImage` (`OCTDeviceHandle Dev, ColoredDataHandle Image`)
Gets a camera image.

6.20.1 Detailed Description

Functions for acquiring camera video images.

6.20.2 Function Documentation

```
6.20.2.1 getCameralImage() void getCameraImage (
    OCTDeviceHandle Dev,
    ColoredDataHandle Image )
```

Gets a camera image.

```
6.20.2.2 getMaxCameralImageSize() void getMaxCameraImageSize (
    OCTDeviceHandle Dev,
    int * SizeX,
    int * SizeY )
```

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

6.21 Helper function

Functions for chores common to many categories and scenarios.

Typedefs

- **SPECTRALRADAR_API** `typedef struct C_VisualCalibration * VisualCalibrationHandle`
Handle to the visual galvo calibration class.

Functions

- **SPECTRALRADAR_API** `unsigned long InterpretReferenceIntensity (float intensity)`
interprets the reference intensity and gives a color code that reflects its state.
- **SPECTRALRADAR_API** `unsigned long InterpretReferenceIntensitySingleValue (float DesiredIntensity, float Tolerance, float CurrentIntensity)`
interprets the reference intensity and gives green color code when the reference intensity is the DesiredIntensity plus/minus the Tolerance, otherwise red is returned.
- **SPECTRALRADAR_API** `void getConfigPath (char *Path, int StrSize)`
Returns the path that hold the config files.
- **SPECTRALRADAR_API** `void getPluginPath (char *Path, int StrSize)`
Returns the path that hold the plugins.
- **SPECTRALRADAR_API** `void getInstallationPath (char *Path, int StrSize)`
Returns the installation path.
- **SPECTRALRADAR_API** `double getReferenceIntensity (ProcessingHandle Proc)`
Returns an absolute value that indicates the reference intensity that was present when the currently used apodization was determined.
- **SPECTRALRADAR_API** `double getRelativeReferenceIntensity (OCTDeviceHandle Dev, ProcessingHandle Proc)`
Returns a value larger than 0.0 and smaller than 1.0 that indicates the reference intensity (relative to saturation) that was present when the currently used apodization was determined.
- **SPECTRALRADAR_API** `double getRelativeSaturation (ProcessingHandle Proc)`
Returns a value larger than 0.0 and smaller than 1.0 that indicates the saturation of the sensor that was present during the last processing cycle.

6.21.1 Detailed Description

Functions for chores common to many categories and scenarios.

6.21.2 Typedef Documentation

6.21.2.1 **VisualCalibrationHandle** `VisualCalibrationHandle`

Handle to the visual galvo calibration class.

Definition at line 139 of file [SpectralRadar_Handles.h](#).

6.21.3 Function Documentation

6.21.3.1 **getConfigPath()** void getConfigPath (

```
    char * Path,  
    int StrSize )
```

Returns the path that hold the config files.

6.21.3.2 **getInstallationPath()** void getInstallationPath (

```
    char * Path,  
    int StrSize )
```

Returns the installation path.

6.21.3.3 **getPluginPath()** void getPluginPath (

```
    char * Path,  
    int StrSize )
```

Returns the path that hold the plugins.

6.21.3.4 **getReferenceIntensity()** double getReferenceIntensity (

```
    ProcessingHandle Proc )
```

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

The reference intensity is the maximum value of an apodization spectrum after subtracting the offset.
This value gets computed automatically along the measurement. The function just returns the latest value computed.

6.21.3.5 **getRelativeReferenceIntensity()** double getRelativeReferenceIntensity (

```
    OCTDeviceHandle Dev,  
    ProcessingHandle Proc )
```

Returns a value larger than 0.0 and smaller than 1.0 that indicates the reference intensity (relative to saturation) that was present when the currently used apodization was determined.

The value is determined as the quotient between the reference intensity (function [getReferenceIntensity\(\)](#)) and the full well capacity of the detector.

6.21.3.6 `getRelativeSaturation()` double double getRelativeSaturation (ProcessingHandle Proc)

Returns a value larger than 0.0 and smaller than 1.0 that indicates the saturation of the sensor that was present during the last processing cycle.

6.21.3.7 `InterpretReferenceIntensity()` 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

<i>intensity</i>	the current reference intensity as a value between 0.0 and 1.0
------------------	--

Returns

the color code reflecting the state of the reference intensity

6.21.3.8 `InterpretReferenceIntensitySingleValue()` unsigned long InterpretReferenceIntensitySingleValue (

 float DesiredIntensity,
 float Tolerance,
 float CurrentIntensity)

interprets the reference intensity and gives green color code when the reference intensity is the DesiredIntensity plus/minus the Tolerance, otherwise red is returned.

Possible colors include:

- red = 0x00FF0000 (bad intensity);
- green = 0x0000FF00 (good intensity);

Parameters

in	<i>DesiredIntensity</i>	the desired reference intensity as a value between 0.0 and 1.0
in	<i>Tolerance</i>	the specified reference intensity tolerance as a value between 0.0 and 1.0
in	<i>CurrentIntensity</i>	the current reference intensity as a value between 0.0 and 1.0

Returns

the color code reflecting the state of the reference intensity

6.22 Buffer

Functions for acquiring camera video images.

Typedefs

- `typedef struct C_Buffer * BufferHandle`
The BufferHandle identifies a data buffer.

Functions

- `SPECTRALRADAR_API BufferHandle createMemoryBuffer (void)`
Creates a buffer holding data and colored data.
- `SPECTRALRADAR_API void appendToBuffer (BufferHandle, DataHandle, ColoredDataHandle)`
Appends specified data and colored data to the requested buffer.
- `SPECTRALRADAR_API void purgeBuffer (BufferHandle)`
Discards all data.
- `SPECTRALRADAR_API int getBufferSize (BufferHandle)`
Returns the currently available data sets in the buffer.
- `SPECTRALRADAR_API int getBufferFirstIndex (BufferHandle)`
Returns the index of the first data sets available in the buffer.
- `SPECTRALRADAR_API int getBufferLastIndex (BufferHandle)`
Returns the index of one past the last data sets available in the buffer.
- `SPECTRALRADAR_API DataHandle getData (BufferHandle, int Index)`
Returns the data in the buffer.
- `SPECTRALRADAR_API ColoredDataHandle getColoredBufferData (BufferHandle, int Index)`
Returns the colored data in the buffer.
- `SPECTRALRADAR_API void clearBuffer (BufferHandle BufferHandle)`
Clears the buffer and frees all data and colored data objects in it.

6.22.1 Detailed Description

Functions for acquiring camera video images.

6.22.2 Typedef Documentation

6.22.2.1 BufferHandle `BufferHandle`

The BufferHandle identifies a data buffer.

Definition at line 68 of file [SpectralRadar_Handles.h](#).

6.22.3 Function Documentation

```
6.22.3.1 appendToBuffer() void appendToBuffer (
    BufferHandle ,
    DataHandle ,
    ColoredDataHandle )
```

Appends specified data and colored data to the requested buffer.

If insufficient memory is available the oldest items in the buffer will be freed automatically.

```
6.22.3.2 clearBuffer() void clearBuffer (
    BufferHandle BufferHandle )
```

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

Parameters

in	<i>BufferHandle</i>	A handle of a buffer (BufferHandle). If the handle is a nullptr, this function does nothing.
----	---------------------	--

```
6.22.3.3 createMemoryBuffer() BufferHandle createMemoryBuffer (
    void )
```

Creates a buffer holding data and colored data.

```
6.22.3.4 getBufferData() DataHandle getBufferData (
    BufferHandle ,
    int Index )
```

Returns the data in the buffer.

```
6.22.3.5 getBufferFirstIndex() int getBufferFirstIndex (
    BufferHandle )
```

Returns the index of the first data sets available in the buffer.

```
6.22.3.6 getBufferLastIndex() int getBufferLastIndex (
    BufferHandle )
```

Returns the index of one past the last data sets available in the buffer.

6.22.3.7 `getBufferSize()` `int bufferSize (`
`BufferHandle)`

Returns the currently available data sets in the buffer.

6.22.3.8 `getColoredBufferData()` `ColoredDataHandle getColoredBufferData (`
`BufferHandle ,`
`int Index)`

Returns the colored data in the buffer.

6.22.3.9 `purgeBuffer()` `void purgeBuffer (`
`BufferHandle)`

Discards all data.

6.23 File Handling

Typedefs

- `typedef struct C_MarkerList * MarkerListHandle`
Handle to the marker list class.

Enumerations

- `enum OCTFileFormat {
 FileFormat_OCITY,
 FileFormat_IMG,
 FileFormat_SDR,
 FileFormat_SRM,
 FileFormat_TIFF32 }`
Enum identifying possible file formats.
- `enum DataObjectType {
 DataObjectType_Real,
 DataObjectType_Colored,
 DataObjectType_Complex,
 DataObjectType_Raw,
 DataObjectType_Binary,
 DataObjectType_Text,
 DataObjectType_Unknown = 999 }`
Enum identifying.
- `enum FileMetadataFloat {
 FileMetadata_RefRACTiveIndex,
 FileMetadata_RangeX,
 FileMetadata_RangeY,
 FileMetadata_RangeZ,
 FileMetadata_CenterX,
 FileMetadata_CenterY,
 FileMetadata_Angle,
 FileMetadata_BinToElectronScaling,
 FileMetadata_CentralWavelength_nm,
 FileMetadata_SourceBandwidth_nm,
 FileMetadata_MinElectrons,
 FileMetadata_QuadraticDispersionCorrectionFactor,
 FileMetadata_SpeckleVarianceThreshold,
 FileMetadata_ScanTime_Sec,
 FileMetadata_ReferenceIntensity,
 FileMetadata_ScanPause_Sec,
 FileMetadata_Zoom,
 FileMetadata_MinPointDistance,
 FileMetadata_MaxPointDistance,
 FileMetadata_FFTOversampling,
 FileMetadata_FullWellCapacity,
 FileMetadata_Saturation,
 FileMetadata_CameraLineRate_Hz,
 FileMetadata_PMDCorrectionAngle_rad,
 FileMetadata_OpticalAxisOffset_rad,
 FileMetadata_ReferenceLength_mm,
 FileMetadata_ReferenceIntensityControl_Value,
 FileMetadata_PolarizationAdjustment_QuarterWave,
 FileMetadata_PolarizationAdjustment_HalfWave }`

Enum identifying file metadata fields of floating point type.

- enum `FileMetadataInt` {
 `FileMetadata_ProcessState`,
 `FileMetadata_SizeX`,
 `FileMetadata_SizeY`,
 `FileMetadata_SizeZ`,
 `FileMetadata_Oversampling`,
 `FileMetadata_IntensityAveragedSpectra`,
 `FileMetadata_IntensityAveragedAScans`,
 `FileMetadata_IntensityAveragedBScans`,
 `FileMetadata_DopplerAverageX`,
 `FileMetadata_DopplerAverageZ`,
 `FileMetadata_ApoWindow`,
 `FileMetadata_DeviceBitDepth`,
 `FileMetadata_SpectrometerElements`,
 `FileMetadata_ExperimentNumber`,
 `FileMetadata_DeviceBytesPerPixel`,
 `FileMetadata_SpeckleAveragingFastAxis`,
 `FileMetadata_SpeckleAveragingSlowAxis`,
 `FileMetadata_Processing_FFTType`,
 `FileMetadata_NumOfCameras`,
 `FileMetadata_SelectedCamera`,
 `FileMetadata_ApodizationType`,
 `FileMetadata_AcquisitionOrder`,
 `FileMetadata_DOPUFilter`,
 `FileMetadata_DOPUAverageZ`,
 `FileMetadata_DOPUAverageX`,
 `FileMetadata_DOPUAverageY`,
 `FileMetadata_PolarizationAverageZ`,
 `FileMetadata_PolarizationAverageX`,
 `FileMetadata_PolarizationAverageY`,
 `FileMetadata_SamplingAmplification`,
 `FileMetadata_SamplingAmplificationSteps`,
 `FileMetadata_AnalogInputNumOfChannels` }

Enum identifying file metadata fields of integral type.

- enum `FileMetadataString` {
 `FileMetadata_DeviceSeries`,
 `FileMetadata_DeviceName`,
 `FileMetadata_Serial`,
 `FileMetadata_Comment`,
 `FileMetadata_CustomInfo`,
 `FileMetadata_AcquisitionMode`,
 `FileMetadata_Study`,
 `FileMetadata_DispersionPreset`,
 `FileMetadata_ProbeName`,
 `FileMetadata_FreeformScanPatternInterpolation`,
 `FileMetadata_HardwareConfig`,
 `FileMetadata_OrigVersion`,
 `FileMetadata_LastModVersion`,
 `FileMetadata_AnalogInputActiveChannels`,
 `FileMetadata_AnalogInputChannelNames` }

Enum identifying file metadata fields of character string type.

- enum `FileMetadataFlag` {
 `FileMetadata_OffsetApplied`,
 `FileMetadata_DCSubtracted`,
 `FileMetadata_ApoApplied`,
 `FileMetadata_DechirpApplied`,
 `FileMetadata_UndersamplingFilterApplied`,

```
FileMetadata_DispersionCompensationApplied,
FileMetadata_QuadraticDispersionCorrectionUsed,
FileMetadata_ImageFieldCorrectionApplied,
FileMetadata_ScanLineShown,
FileMetadata_AutoCorrCompensationUsed,
FileMetadata_BScanCrossCorrelation,
FileMetadata_DCSubtractedAdvanced,
FileMetadata_OnlyWindowing,
FileMetadata_RawDataIsSigned,
FileMetadata_FreeformScanPatternIsActive,
FileMetadata_FreeformScanPatternCloseLoop,
FileMetadata_IsSweptSource,
FileMetadata_DopplerOversampling }
```

Enum identifying file metadata fields of bool type.

- enum `FileMetadata_ProcessingState` {
 `RawSpectra`,
 `ProcessedIntensity`,
 `RawSpectraAndProcessedIntensity`,
 `ProcessedIntensityAndPhase`,
 `RawSpectraAndProcessedIntensityAndPhase`,
 `psSpeckleVariance`,
 `psRawSpectraAndSpeckleVariance`,
 `psColored`,
 `psUnknown = 999` }

Enum to specify the processing state of the stored data.

Functions

- `SPECTRALRADAR_API const char * DataObjectName_SpectralData (int index)`

Returns the filename of the spectral-data object with the specified index.
- `SPECTRALRADAR_API OCTFileHandle createOCTFile (OCTFormat format)`

Creates a handle to an OCT file of the given format.
- `SPECTRALRADAR_API void clearOCTFile (OCTFileHandle Handle)`

Clears the given OCT file handle and frees its resources.
- `SPECTRALRADAR_API int getFileDataObjectCount (OCTFileHandle Handle)`

Returns the number of data objects in the OCT file. This number will vary depending on the file's format and contents (Files with the .oct extension may contain multiple OCT data objects depending on their internal structure).
- `SPECTRALRADAR_API void loadFile (OCTFileHandle Handle, const char *Filename)`

Loads the actual OCT data file from a file system. The file must have the format given in `createOCTFile()`.
- `SPECTRALRADAR_API void saveFile (OCTFileHandle Handle, const char *Filename)`

Saves the OCT data file in the given fully qualified path name.
- `SPECTRALRADAR_API void saveChangesToFile (OCTFileHandle Handle)`

Saves the OCT data file in the file previously opened with `loadFile()`. Only changes will be saved.
- `SPECTRALRADAR_API void copyFileMetadata (OCTFileHandle SrcHandle, OCTFileHandle DstHandle)`

Copies metadata from one OCT file to another.
- `SPECTRALRADAR_API bool containsFileMetadataFloat (OCTFileHandle Handle, FileMetadataFloat Floatfield)`

Returns true if the given metadata field is present in the file.
- `SPECTRALRADAR_API double getFileMetadataFloat (OCTFileHandle Handle, FileMetadataFloat Floatfield)`

Returns the value of the given file metadata field as a floating point number if found.
- `SPECTRALRADAR_API void setFileMetadataFloat (OCTFileHandle Handle, FileMetadataFloat Floatfield, double Value)`

Sets the value of the given file metadata field as a floating point number.

- **SPECTRALRADAR_API** bool `containsFileMetadataInt` (`OCTFileHandle` Handle, `FileMetadataInt` Intfield)
Returns true if the given metadata field is present in the file.
- **SPECTRALRADAR_API** int `getFileMetadataInt` (`OCTFileHandle` Handle, `FileMetadataInt` Intfield)
Returns the value of the given file metadata field as an integer if found.
- **SPECTRALRADAR_API** void `setFileMetadataInt` (`OCTFileHandle` Handle, `FileMetadataInt` Intfield, int Value)
Sets the value of the given file metadata field as an integer.
- **SPECTRALRADAR_API** bool `containsFileMetadataString` (`OCTFileHandle` Handle, `FileMetadataString` StringField)
Returns true if the given metadata field is present in the file.
- **SPECTRALRADAR_API** const char * `getFileMetadataString` (`OCTFileHandle` Handle, `FileMetadataString` StringField)
Returns the value of the given file metadata field as a string if found.
- **SPECTRALRADAR_API** void `setFileMetadataString` (`OCTFileHandle` Handle, `FileMetadataString` StringField, const char *Content)
Sets the value of the given file metadata field as a string.
- **SPECTRALRADAR_API** bool `containsFileMetadataFlag` (`OCTFileHandle` Handle, `FileMetadataFlag` Boolfield)
Returns true if the given metadata field is present in the file.
- **SPECTRALRADAR_API** BOOL `getFileMetadataFlag` (`OCTFileHandle` Handle, `FileMetadataFlag` Boolfield)
Gets the boolean value of the given file metadata field.
- **SPECTRALRADAR_API** void `setFileMetadataFlag` (`OCTFileHandle` Handle, `FileMetadataFlag` Boolfield, BOOL Value)
Sets the boolean value of the given file metadata field.
- **SPECTRALRADAR_API** void `saveFileMetadata` (`OCTFileHandle` Handle, `OCTDeviceHandle` Dev, ProcessingHandle Proc, ProbeHandle Probe, ScanPatternHandle Pattern)
Saves meta information from the given device, processing, probe and scan pattern instances in the metadata block of the given file handle. This information will be available in files of type `FileFormat_OCITY`; mileage on other formats may vary according to their description.
- **SPECTRALRADAR_API** void `setFileMetadataTimestamp` (`OCTFileHandle` File, time_t Timestamp)
Saves provided timestamp to meta information to the given file handle. This information will be available in files of type `FileFormat_OCITY`; mileage on other formats may vary according to their description.
- **SPECTRALRADAR_API** time_t `getFileMetadataTimestamp` (`OCTFileHandle` File)
Returns the specified timestamp from the meta information of the given file handle. This information will be available in files of type `FileFormat_OCITY`; mileage on other formats may vary according to their description.
- **SPECTRALRADAR_API** void `saveFileMetadataDoppler` (`OCTFileHandle` Handle, DopplerProcessingHandle DopplerProc)
Saves meta information from the given DopplerProcessingHandle. A corresponding DopplerProcessingHandle can then be recreated using `createDopplerProcessingForFile`.
- **SPECTRALRADAR_API** void `saveFileMetadataSpeckle` (`OCTFileHandle` Handle, SpeckleVarianceHandle SpeckleVarianceProc)
Saves meta information from the given SpeckleVarianceHandle. A corresponding SpeckleVarianceHandle can then be recreated using `initSpeckleVarianceForFile`.
- **SPECTRALRADAR_API** void `loadCalibrationFromFile` (`OCTFileHandle` Handle, ProcessingHandle Proc)
Loads Chirp, Offset, and Apodization vectors from the given OCT file into the given processing object.
- **SPECTRALRADAR_API** void `loadCalibrationFromFileEx` (`OCTFileHandle` Handle, ProcessingHandle Proc, const int CameralIndex)
Loads Chirp, Offset, and Apodization vectors from the given OCT file into the given processing object.
- **SPECTRALRADAR_API** void `saveCalibrationToFile` (`OCTFileHandle` Handle, ProcessingHandle Proc)
Saves Chirp, Offset, and Apodization vectors from the given processing object into the given OCT file.
- **SPECTRALRADAR_API** void `saveCalibrationToFileEx` (`OCTFileHandle` Handle, ProcessingHandle Proc, int CameralIndex)
Saves Chirp, Offset, and Apodization vectors from the given processing object into the given OCT file.
- **SPECTRALRADAR_API** void `getFileRealData` (`OCTFileHandle` Handle, DataHandle Data, int Index)

- Retrieves a RealData object from the OCT file at the given index with $0 \leq index < \text{getFileDialogObjectCount(OCTFileHandle handle)}$. Users must ensure that the data handle is properly prepared and destroyed.*
- **SPECTRALRADAR_API** void `getFileColoredData (OCTFileHandle Handle, ColoredDataHandle Data, size_t Index)`
Retrieves a ColoredData object from the OCT file at the given index with $0 \leq index < \text{getFileDialogObjectCount(OCTFileHandle handle)}$. Users must ensure that the data handle is properly prepared and destroyed.
 - **SPECTRALRADAR_API** void `getFileComplexData (OCTFileHandle Handle, ComplexDataHandle Data, size_t Index)`
Retrieves a ComplexData object from the OCT file at the given index with $0 \leq index < \text{getFileDialogObjectCount(OCTFileHandle handle)}$. Users must ensure that the data handle is properly prepared and destroyed.
 - **SPECTRALRADAR_API** void `getFileRawData (OCTFileHandle Handle, RawDataHandle Data, size_t Index)`
Retrieves a RawData object from the OCT file at the given index with $0 \leq index < \text{getFileDialogObjectCount(OCTFileHandle handle)}$. Users must ensure that the data handle is properly prepared and destroyed.
 - **SPECTRALRADAR_API** void `getFile (OCTFileHandle Handle, size_t Index, const char *FilenameOnDisk)`
Retrieves a data object of arbitrary type from the OCT file at the given index with $0 \leq index < \text{getFileDialogObjectCount(OCTFileHandle handle)}$ and stores it at the given fully qualified path.
 - **SPECTRALRADAR_API** int `findFileDialogObject (OCTFileHandle Handle, const char *Search)`
Searches for a data object the name of which contains the given string and returns its index, -1 if not found.
 - **SPECTRALRADAR_API** BOOL `containsFileDialogObject (OCTFileHandle Handle, const char *Search)`
Searches for a data object the name of which contains the given string and returns TRUE if at least one data object name matches.
 - **SPECTRALRADAR_API** BOOL `containsFileRawData (OCTFileHandle Handle)`
Returns TRUE if the file contains raw data objects.
 - **SPECTRALRADAR_API** void `addFileRealData (OCTFileHandle Handle, DataHandle Data, const char *DataObjectName)`
Adds a RealData object to the OCT file; DataObjectName will be its name inside the OCT file if applicable. The object that the DataHandle refers to must live until after `saveFile()` has been called.
 - **SPECTRALRADAR_API** void `addFileColoredData (OCTFileHandle Handle, ColoredDataHandle Data, const char *DataObjectName)`
Adds a ColoredData object to the OCT file; DataObjectName will be its name inside the OCT file if applicable. The object that the ColoredDataHandle refers to must live until after `saveFile()` has been called.
 - **SPECTRALRADAR_API** void `addFileComplexData (OCTFileHandle Handle, ComplexDataHandle Data, const char *DataObjectName)`
Adds a ComplexData object to the OCT file; DataObjectName will be its name inside the OCT file if applicable. The object that the ComplexDataHandle refers to must live until after `saveFile()` has been called.
 - **SPECTRALRADAR_API** void `addFileRawData (OCTFileHandle Handle, RawDataHandle Data, const char *DataObjectName)`
Adds raw Data object to the OCT file; DataObjectName will be its name inside the OCT file if applicable. The object that the RawDataHandle refers to must live until after `saveFile` has been called.
 - **SPECTRALRADAR_API** void `addFileDialogText (OCTFileHandle Handle, const char *FilenameOnDisk, const char *DataObjectName)`
Adds a text object read from FilenameOnDisk to the OCT file; DataObjectName will be its name inside the OCT file if applicable. The file identified by filenameOnDisk must exist until after `saveFile()` has been called.
 - **SPECTRALRADAR_API** DataObjectType `getFileDataObjectType (OCTFileHandle Handle, int Index)`
Returns the type of the data object at the given Index in the OCT file.
 - **SPECTRALRADAR_API** void `getFileDataObjectName (OCTFileHandle Handle, int Index, char *Filename, int Length)`
Returns the name of the data object at the given Index in the OCT file.
 - **SPECTRALRADAR_API** int `getFileFileSizeX (OCTFileHandle Handle, size_t Index)`
Returns the pixel count in X of the data object at the given Index in the OCT file.
 - **SPECTRALRADAR_API** int `getFileFileSizeY (OCTFileHandle Handle, size_t Index)`
Returns the pixel count in Y of the data object at the given Index in the OCT file.
 - **SPECTRALRADAR_API** int `getFileFileSizeZ (OCTFileHandle Handle, size_t Index)`
Returns the pixel count in Z of the data object at the given Index in the OCT file.

- **SPECTRALRADAR_API** float `getFileDataRangeX` (`OCTFileHandle` Handle, `size_t` Index)
Returns the range (usually in mm) in X of the data object at the given Index in the OCT file.
- **SPECTRALRADAR_API** float `getFileDataRangeY` (`OCTFileHandle` Handle, `size_t` Index)
Returns the range (usually in mm) in Y of the data object at the given Index in the OCT file.
- **SPECTRALRADAR_API** float `getFileDataRangeZ` (`OCTFileHandle` Handle, `size_t` Index)
Returns the range (usually in mm) in Z of the data object at the given Index in the OCT file.
- **SPECTRALRADAR_API** void `clearMarkerList` (`OCTFileHandle` Handle)
Clears the marker list of a given OCT file. This removes all line and point markers from the file.
- **SPECTRALRADAR_API** void `copyMarkerListFromRealData` (`OCTFileHandle` Handle, `DataHandle` Data)
coordinates, so re-use is possible.
- **SPECTRALRADAR_API** void `copyMarkerListToRealData` (`OCTFileHandle` Handle, `DataHandle` Data)
coordinates, so re-use is possible.
- **SPECTRALRADAR_API** void `addFileMetadataPreset` (`OCTFileHandle` Handle, const char *Category, const char *PresetDescription)
Adds one of the presets set during acquisition for the `OCTFileHandle`.
- **SPECTRALRADAR_API** int `getFileMetadataNumberOfPresets` (`OCTFileHandle` Handle)
Gets the number of presets that were set during the acquisition.
- **SPECTRALRADAR_API** const char * `getFileMetadataPresetCategory` (`OCTFileHandle` Handle, int Index)
Gets the preset category belonging to the preset with given Index.
- **SPECTRALRADAR_API** const char * `getFileMetadataPresetDescription` (`OCTFileHandle` Handle, int Index)
Gets the preset description belonging to the preset with given Index.

6.23.1 Detailed Description

6.23.2 Typedef Documentation

6.23.2.1 **MarkerListHandle** `MarkerListHandle`

Handle to the marker list class.

Definition at line 145 of file [SpectralRadar_Handles.h](#).

6.23.3 Enumeration Type Documentation

6.23.3.1 **DataObjectType** `enum DataObjectType`

Enum identifying.

Definition at line 571 of file [SpectralRadar_Types.h](#).

6.23.3.2 **FileMetadata_ProcessingState** `enum FileMetadata_ProcessingState`

Enum to specify the processing state of the stored data.

Deprecated The future SDK will deduce the processing state out fo the available data in the OCT file.

Enumerator

RawSpectra	Just the spectra, without any further processing. See also saveCalibrationToFile and addFileRawData .
ProcessedIntensity	Just the computed intensity. See also addFileRealData .
RawSpectraAndProcessedIntensity	Both the spectra and the computed intensity get stored. In the case of polarization-sensitive instruments, the complex data C_0 and C_1 are saved.
ProcessedIntensityAndPhase	The computed intensity and the computed phase are stored in the OCT file.
RawSpectraAndProcessedIntensityAndPhase	The spectra, the computed intensity and the computed phase are stored in the OCT file.
psSpeckleVariance	Speckle variance data, i.e., averaged intensity and speckle variance contrast is stored in the OCT file.
psRawSpectraAndSpeckleVariance	The spectra and speckle variance data, i.e., averaged intensity and speckle variance contrast is stored in the OCT file.
psColored	Colored data is stored in the OCT file.
psUnknown	It is unknown what kind of data is stored in the file.

Definition at line 808 of file [SpectralRadar_Types.h](#).

6.23.3.3 FileMetadataFlag [enum FileMetadataFlag](#)

Enum identifying file metadata fields of bool type.

Enumerator

FileMetadata_OffsetApplied	This field is the flag that can be accessed with the functions getProcessingFlag / setProcessingFlag and the constant Processing_UseOffsetErrors .
FileMetadata_DCSubtracted	This field is the flag that can be accessed with the functions getProcessingFlag / setProcessingFlag and the constant Processing_RemoveDCSpectrum .
FileMetadata_ApoApplied	Field indicates/sets whether an apodization was performed during processing of the respective OCT data file.
FileMetadata_DechirpApplied	Field indicates/sets whether dechirp was applied during processing of the respective OCT data file.
FileMetadata_UndersamplingFilterApplied	This field is the flag that can be accessed with the functions getProcessingFlag / setProcessingFlag and the constant Processing_UseUndersamplingFilter .
FileMetadata_DispersionCompensationApplied	Field indicates/sets whether dispersion compensations was applied when acquiring the file.
FileMetadata_QuadraticDispersionCorrectionUsed	Field indicates/sets whether quadratic dispersion compensations was applied when acquiring the file.
FileMetadata_ImageFieldCorrectionApplied	Field indicates/sets whether image field correction was applied when acquiring the file.
FileMetadata_ScanLineShown	Field indicates/sets whether the scan line was visualized on the camera image when the file was acquired.

Enumerator

FileMetadata_AutoCorrCompensationUsed	This field is the flag that can be accessed with the functions getProcessingFlag / setProcessingFlag and the constant Processing_UseAutocorrCompensation .
FileMetadata_BScanCrossCorrelation	Field indicates/sets whether B-scans were correlated during averaging when the file was acquired.
FileMetadata_DCSubtractedAdvanced	This field is the flag that can be accessed with the functions getProcessingFlag / setProcessingFlag and the constant Processing_RemoveAdvancedDCSpectrum .
FileMetadata_OnlyWindowing	This field is the flag that can be accessed with the functions getProcessingFlag / setProcessingFlag and the constant Processing_OnlyWindowing .
FileMetadata_RawDataIsSigned	This field is the flag that can be retrieved with the function getDevicePropertyInt and the constant Device_DataIsSigned .
FileMetadata_FreeformScanPatternIsActive	Field indicates/sets whether a freeform scan pattern was active during acquisition.
FileMetadata_FreeformScanPatternCloseLoop	Field indicates/sets whether the used freeform pattern was a closed.
FileMetadata_IsSweptSource	Field indicates/sets whether data was acquired with a swept-source system.
FileMetadata_DopplerOversampling	Field indicates/sets whether data was acquired with oversampling parameter checker or not.

Definition at line 758 of file [SpectralRadar_Types.h](#).

6.23.3.4 FileMetadataFloat enum [FileMetadataFloat](#)

Enum identifying file metadata fields of floating point type.

Enumerator

FileMetadata_RefractiveIndex	The refractive index applied to the whole image.
FileMetadata_RangeX	The FOV in axial direction (x) in mm.
FileMetadata_RangeY	The FOV in axial direction (y) in mm.
FileMetadata_RangeZ	The FOV in longitudinal axis (z) in mm.
FileMetadata_CenterX	The center of the scan pattern in axial direction (x) in mm.
FileMetadata_CenterY	The center of the scan pattern in axial direction (y) in mm.
FileMetadata_Angle	The angle between the scanner and the video camera image.
FileMetadata_BinToElectronScaling	Ratio between the binary value from the camera to the count of electrons.
FileMetadata_CentralWavelength_nm	Central wavelength of the device.
FileMetadata_SourceBandwidth_nm	Bandwidth of the light source.
FileMetadata_MinElectrons	Electron cut-off parameter used for processing.
FileMetadata_QuadraticDispersionCorrectionFactor	Quadratic dispersion factor used for dispersion correction.

Enumerator

FileMetadata_SpeckleVarianceThreshold	Threshold for speckle variance mode.
FileMetadata_ScanTime_Sec	Time needed for data acquisition. The processing and saving time is not included.
FileMetadata_ReferenceIntensity	Value for the reference intensity.
FileMetadata_ScanPause_Sec	Scan pause in between scans.
FileMetadata_Zoom	Zooms the scan pattern.
FileMetadata_MinPointDistance	Minimum distance between two points of the scan pattern used for freeform scan patterns.
FileMetadata_MaxPointDistance	Maximum distance between two points of the scan pattern used for freeform scan patterns.
FileMetadata_FFTOversampling	FFT oversampling use for processing and chirp correction.
FileMetadata_FullWellCapacity	Sets/indicates the full-well capacity of the device used for acquiring the data.
FileMetadata_Saturation	Sets/indicates the saturation value that was present when acquiring the data. Warning Not all devices evaluate the saturation during acquisition
FileMetadata_CameraLineRate_Hz	Sets/indicates the line rate during acquisition.
FileMetadata_PMDCorrectionAngle_rad	Polarization mode correction. This angle (expressed in radians) is used to compute a phasor ($\exp(i\alpha)$), that will be applied to the complex reflectivities vector associated with camera 0.
FileMetadata_OpticalAxisOffset_rad	In birefringent samples, this offset allows referring the angle of the fast axis to an axis in the sample holder.
FileMetadata_ReferenceLength_mm	Reference stage position, if adjustable reference stage is available.
FileMetadata_ReferenceIntensityControl_Value	Position of reference intensity control, if available.
FileMetadata_PolarizationAdjustment_QuarterWave	Position of quarter wave polarization adjustment, if available.
FileMetadata_PolarizationAdjustment_HalfWave	Position of half wave polarization adjustment, if available.

Definition at line 585 of file [SpectralRadar_Types.h](#).

6.23.3.5 FileMetadataInt enum FileMetadataInt

Enum identifying file metadata fields of integral type.

Enumerator

FileMetadata_ProcessState	Contains the specific data format.
FileMetadata_SizeX	Number of pixels in x.
FileMetadata_SizeY	Number of pixels in y.
FileMetadata_SizeZ	Number of pixels in z.

Enumerator

FileMetadata_Oversampling	Oversampling parameter.
FileMetadata_IntensityAveragedSpectra	Spectrum averaging.
FileMetadata_IntensityAveragedAScans	A-scan averaging.
FileMetadata_IntensityAveragedBScans	B-scan averaging.
FileMetadata_DopplerAverageX	Averaging for doppler processing in x-direction.
FileMetadata_DopplerAverageZ	Averaging for doppler processing in z-direction.
FileMetadata_ApoWindow	Type of window used for apodization.
FileMetadata_DeviceBitDepth	Bits per pixel of the camera.
FileMetadata_SpectrometerElements	Number of elements of the spectrometer.
FileMetadata_ExperimentNumber	Serial number of the dataset.
FileMetadata_DeviceBytesPerPixel	Bytes per pixel of the camera.
FileMetadata_SpeckleAveragingFastAxis	Averaging parameter of the fast scan axis in speckle variance mode.
FileMetadata_SpeckleAveragingSlowAxis	Averaging parameter of the slow scan axis in speckle variance mode.
FileMetadata_Processing_FFTType	FFT algorithm used.
FileMetadata_NumOfCameras	Number of cameras, or sensors, stored in the file. In case of legacy files, this property takes the default value "1".
FileMetadata_SelectedCamera	In devices with more than one camera, some modi need to know which camera is active, because they do not support work with multiple cameras. In case of legacy files, this property takes the default value "0".
FileMetadata_ApodizationType	This field sets/indicates the apodization type set when the data was acquired (see ScanPatternApodizationType).
FileMetadata_AcquisitionOrder	This field sets/indicates the acquisition order set when the data was acquired (see ScanPatternAcquisitionOrder).
FileMetadata_DOPUFilter	DOPU filter specification. See PolarizationDOPUFilterType .
FileMetadata_DOPUAverageZ	Number of pixels for DOPU averaging in the z-direction.
FileMetadata_DOPUAverageX	Number of pixels for DOPU averaging in the x-direction.
FileMetadata_DOPUAverageY	Number of pixels for DOPU averaging in the y-direction.
FileMetadata_PolarizationAverageZ	Number of pixels for averaging along the z axis.
FileMetadata_PolarizationAverageX	Number of pixels for averaging along the x axis.
FileMetadata_PolarizationAverageY	Number of pixels for averaging along the y axis.
FileMetadata_SamplingAmplification	Sampling amplification for VEGA systems.
FileMetadata_AnalogInputNumOfChannels	Number of channels provided by analog input.

Definition at line 653 of file [SpectralRadar_Types.h](#).

6.23.3.6 FileMetadataString enum [FileMetadataString](#)

Enum identifying file metadata fields of character string type.

Enumerator

FileMetadata_DeviceSeries	Name of the OCT device series.
FileMetadata_DeviceName	Name of the OCT device.

Enumerator

FileMetadata_Serial	Serial number of the OCT device.
FileMetadata_Comment	Comment of the OCT data file.
FileMetadata_CustomInfo	Additional, custom info.
FileMetadata_AcquisitionMode	Acquisition mode of the OCT data file.
FileMetadata_Study	Study of the OCT data file.
FileMetadata_DispersionPreset	Dispersion Preset of the OCT data file.
FileMetadata_ProbeName	Name of the probe.
FileMetadata_AnalogInputActiveChannels	Comma separated list of active channels.
FileMetadata_AnalogInputChannelNames	Comma separated list of input channel descriptions.

Definition at line 725 of file [SpectralRadar_Types.h](#).

6.23.3.7 OCTFormat `enum OCTFormat`

Enum identifying possible file formats.

Definition at line 559 of file [SpectralRadar_Types.h](#).

6.23.4 Function Documentation

```
6.23.4.1 addFileColoredData() void addFileColoredData (
    OCTHandle Handle,
    ColoredDataHandle Data,
    const char * DataObjectName )
```

Adds a ColoredData object to the OCT file; `DataObjectName` will be its name inside the OCT file if applicable. The object that the `ColoredDataHandle` refers to must live until after `saveFile()` has been called.

Parameters

in	<code>Handle</code>	A valid (non null) handle of OCTFile (<code>OCTHandle</code>), obtained with the function createOCTFile .
in	<code>Data</code>	A valid (non null) handle to the ColoredData object (<code>ColoredDataHandle</code>) to add.
in	<code>DataObjectName</code>	Name that will be assigned to the object in the OCT file.

```
6.23.4.2 addFileComplexData() void addFileComplexData (
    OCTHandle Handle,
    ComplexDataHandle Data,
    const char * DataObjectName )
```

Adds a ComplexData object to the OCT file; dataObjectName will be its name inside the OCT file if applicable. The object that the ComplexDataHandle refers to must live until after [saveFile\(\)](#) has been called.

Parameters

in	<i>Handle</i>	A valid (non null) handle of OCTFile (OCTFileHandle), obtained with the function createOCTFile .
in	<i>Data</i>	A valid (non null) handle to the ComplexData object (ComplexDataHandle) to add.
in	<i>DataObjectName</i>	Name that will be assigned to the object in the OCT file.

6.23.4.3 addFileMetadataPreset() void addFileMetadataPreset (

```
OCTFileHandle Handle,
const char * Category,
const char * PresetDescription )
```

Adds one of the presets set during acquisition for the [OCTFileHandle](#).

Parameters

in	<i>Handle</i>	A valid (non null) handle of OCTFile (OCTFileHandle), obtained with the function createOCTFile .
in	<i>Category</i>	Name of the category of the added preset.
in	<i>PresetDescription</i>	Description for the added preset.

6.23.4.4 addFileRawData() void addFileRawData (

```
OCTFileHandle Handle,
RawDataHandle Data,
const char * DataObjectName )
```

Adds raw *Data* object to the OCT file; *DataObjectName* will be its name inside the OCT file if applicable. The object that the [RawDataHandle](#) refers to must live until after [saveFile](#) has been called.

Parameters

in	<i>Handle</i>	A valid (non null) handle of OCTFile (OCTFileHandle), obtained with the function createOCTFile .
in	<i>Data</i>	A valid (non null) raw data handle of the existing data (RawDataHandle), previously obtained with the function createRawData . It is assumed that these data have already been filled in with an appropriate data acquisition procedure.
in	<i>DataObjectName</i>	Name that will be assigned to the object in the OCT file. Notice that raw data refers to the spectra as acquired, without processing of any kind.

6.23.4.5 addFileRealData() void addFileRealData (

```
OCTFileHandle Handle,
DataHandle Data,
const char * DataObjectName )
```

Adds a RealData object to the OCT file; *dataObjectName* will be its name inside the OCT file if applicable. The object that the DataHandle refers to must live until after [saveFile\(\)](#) has been called.

Parameters

in	<i>Handle</i>	A valid (non null) handle of OCTFile (OCTFileHandle), obtained with the function createOCTFile .
in	<i>Data</i>	A valid (non null) handle to the RealData object (DataHandle) to add.
in	<i>DataObjectName</i>	Name that will be assigned to the object in the OCT file.

```
6.23.4.6 addFileText() void addFileText (
    OCTFileHandle Handle,
    const char * FilenameOnDisk,
    const char * DataObjectName )
```

Adds a text object read from *FilenameOnDisk* to the OCT file; *DataObjectName* will be its name inside the OCT file if applicable. The file identified by filenameOnDisk must exist until after [saveFile\(\)](#) has been called.

Parameters

in	<i>Handle</i>	A valid (non null) handle of OCTFile (OCTFileHandle), obtained with the function createOCTFile .
in	<i>FilenameOnDisk</i>	Filename from which text file will be read.
in	<i>DataObjectName</i>	Name that will be assigned to the object in the OCT file.

```
6.23.4.7 clearMarkerList() void clearMarkerList (
    OCTFileHandle Handle )
```

Clears the marker list of a given OCT file. This removes all line and point markers from the file.

Parameters

in	<i>Handle</i>	A valid (non null) handle of OCTFile (OCTFileHandle), obtained with the function createOCTFile .
----	---------------	--

```
6.23.4.8 clearOCTFile() void clearOCTFile (
    OCTFileHandle Handle )
```

Clears the given OCT file handle and frees its resources.

Parameters

in	<i>Handle</i>	A valid (non null) handle of OCTFile (OCTFileHandle), obtained with the function createOCTFile .
----	---------------	--

```
6.23.4.9 containsFileDataObject() BOOL containsFileDataObject (
OCTFileHandle Handle,
const char * Search )
```

Searches for a data object the name of which contains the given string and returns TRUE if at least one data object name matches.

Parameters

in	<i>Handle</i>	A valid (non null) handle of OCTFile (<a>OCTFileHandle), obtained with the function <a>createOCTFile .
in	<i>Search</i>	Data object name to find in OCT file.

```
6.23.4.10 containsFileMetadataFlag() bool containsFileMetadataFlag (
OCTFileHandle Handle,
FileMetadataFlag Boolfield )
```

Returns true if the given metadata field is present in the file.

Parameters

in	<i>Handle</i>	A valid (non null) handle of OCTFile (<a>OCTFileHandle), obtained with the function <a>createOCTFile .
in	<i>Boolfield</i>	Metadata field to test.

```
6.23.4.11 containsFileMetadataFloat() bool containsFileMetadataFloat (
OCTFileHandle Handle,
FileMetadataFloat Floatfield )
```

Returns true if the given metadata field is present in the file.

Parameters

in	<i>Handle</i>	A valid (non null) handle of OCTFile (<a>OCTFileHandle), obtained with the function <a>createOCTFile .
in	<i>Floatfield</i>	Metadata field to test.

```
6.23.4.12 containsFileMetadataInt() bool containsFileMetadataInt (
OCTFileHandle Handle,
FileMetadataInt Intfield )
```

Returns true if the given metadata field is present in the file.

Parameters

in	<i>Handle</i>	A valid (non null) handle of OCTFile (OCTFileHandle), obtained with the function createOCTFile .
in	<i>Intfield</i>	Metadata field to test.

6.23.4.13 containsFileMetadataString() `bool containsFileMetadataString (`

```
OCTFileHandle Handle,
FileMetadataString StringField )
```

Returns true if the given metadata field is present in the file.

Parameters

in	<i>Handle</i>	A valid (non null) handle of OCTFile (OCTFileHandle), obtained with the function createOCTFile .
in	<i>Stringfield</i>	Metadata field to test.

6.23.4.14 containsFileRawData() `BOOL containsFileRawData (`

```
OCTFileHandle Handle )
```

Returns TRUE if the file contains raw data objects.

Parameters

in	<i>Handle</i>	A valid (non null) handle of OCTFile (OCTFileHandle), obtained with the function createOCTFile . Notice that raw data refers to the spectra as acquired, without processing of any kind.
----	---------------	--

6.23.4.15 copyFileMetadata() `void copyFileMetadata (`

```
OCTFileHandle SrcHandle,
OCTFileHandle DstHandle )
```

Copies metadata from one OCT file to another.

Parameters

in	<i>SrcHandle</i>	A valid (non null) handle of OCTFile (OCTFileHandle), obtained with the function createOCTFile . This is the source and will not be altered by this function in any way.
out	<i>DstHandle</i>	A valid (non null) handle of OCTFile (OCTFileHandle), obtained with the function createOCTFile . This is the destination and will be filled in using the information in the source.

6.23.4.16 copyMarkerListFromRealData() `void copyMarkerListFromRealData (`
`OCTFileHandle Handle,`
`DataHandle Data)`

coordinates, so re-use is possible.

Copies the marker list from the given data handle into the metadata block of the given OCT file handle.

Markers are a visual help, that can be created or manipulated by ThorImage-OCT. Markers are always expressed in physical

Parameters

in	<i>Handle</i>	A valid (non null) handle of OCTFile (OCTFileHandle), obtained with the function createOCTFile .
in	<i>Data</i>	A valid (non null) data handle of the existing data (DataHandle), previously obtained with the function createData . It is assumed that this structure has already been filled with processed data. If no markers are present, this function does nothing.

6.23.4.17 copyMarkerListToRealData() `void copyMarkerListToRealData (`
`OCTFileHandle Handle,`
`DataHandle Data)`

coordinates, so re-use is possible.

Copies the marker list from the metadata block of the given file handle to the given data handle.

Markers are a visual help, that can be created or manipulated by ThorImage-OCT. Markers are always expressed in physical

Parameters

in	<i>Handle</i>	A valid (non null) handle of OCTFile (OCTFileHandle), obtained with the function createOCTFile .
out	<i>Data</i>	A valid (non null) data handle of the existing data (DataHandle), previously obtained with the function createData . If no markers are present, this function does nothing.

6.23.4.18 createOCTFile() `OCTFileHandle createOCTFile (`
`OCTFileFormat format)`

Creates a handle to an OCT file of the given format.

6.23.4.19 DataObjectName_SpectralData() `const char * DataObjectName_SpectralData (`
`int index)`

Returns the filename of the spectral-data object with the specified index.

Parameters

<i>index</i>	Index of spectral-data object to return
--------------	---

Returns

Filename of the specified data object

```
6.23.4.20 findFileDataObject() int findFileDataObject (
    OCTFileHandle Handle,
    const char * Search )
```

Searches for a data object the name of which contains the given string and returns its index, -1 if not found.

Parameters

in	<i>Handle</i>	A valid (non null) handle of OCTFile (OCTFileHandle), obtained with the function createOCTFile .
in	<i>Search</i>	Data object name to find in OCT file.

```
6.23.4.21 getFile() void getFile (
    OCTFileHandle Handle,
    size_t Index,
    const char * FilenameOnDisk )
```

Retrieves a data object of arbitrary type from the OCT file at the given index with $0 \leq \text{index} < \text{getFileDataObjectCount(OCTFileHandle handle)}$ and stores it at the given fully qualified path.

Parameters

in	<i>Handle</i>	A valid (non null) handle of OCTFile (OCTFileHandle), obtained with the function createOCTFile .
in	<i>Index</i>	Index of the file inside the OCT file, e.g. returned by findFileDataObject .
in	<i>FilenameOnDisk</i>	Filename to which requested file will be written.

```
6.23.4.22 getFileColoredData() void getFileColoredData (
    OCTFileHandle Handle,
    ColoredDataHandle Data,
    size_t Index )
```

Retrieves a ColoredData object from the OCT file at the given index with $0 \leq \text{index} < \text{getFileDataObjectCount(OCTFileHandle handle)}$. Users must ensure that the data handle is properly prepared and destroyed.

Parameters

in	<i>Handle</i>	A valid (non null) handle of OCTFile (OCTFileHandle), obtained with the function createOCTFile .
out	<i>Data</i>	A valid (non null) colored data handle of the existing data (ColoredDataHandle), previously obtained with the function createColoredData . It will be filled in with the data read from the OCT file at the given <i>Index</i> .
in	<i>Index</i>	Index of the data inside the OCT file, e.g. returned by findFileDataObject .

6.23.4.23 getFileComplexData() `void getFileComplexData (`

```
    OCTFileHandle Handle,
    ComplexDataHandle Data,
    size_t Index )
```

Retrieves a ComplexData object from the OCT file at the given index with $0 \leq \text{index} < \text{getFileDataObjectCount}(\text{OCTFileHandle handle})$. Users must ensure that the data handle is properly prepared and destroyed.

Parameters

in	<i>Handle</i>	A valid (non null) handle of OCTFile (OCTFileHandle), obtained with the function createOCTFile .
out	<i>Data</i>	A valid (non null) complex data handle of the existing data (ComplexDataHandle), previously obtained with the function createComplexData . It will be filled in with the data read from the OCT file at the given <i>Index</i> .
in	<i>Index</i>	Index of the data inside the OCT file, e.g. returned by findFileDataObject .

6.23.4.24 getFileDataObjectCount() `int getFileDataObjectCount (`

```
    OCTFileHandle Handle )
```

Returns the number of data objects in the OCT file. This number will vary depending on the file's format and contents (Files with the .oct extension may contain multiple OCT data objects depending on their internal structure).

Parameters

in	<i>Handle</i>	A valid (non null) handle of OCTFile (OCTFileHandle), obtained with the function createOCTFile .
----	---------------	--

6.23.4.25 getFileDataObjectName() `void getFileDataObjectName (`

```
    OCTFileHandle Handle,
    int Index,
    char * Filename,
    int Length )
```

Returns the name of the data object at the given *Index* in the OCT file.

Parameters

in	<i>Handle</i>	A valid (non null) handle of OCTFile (OCTFileHandle), obtained with the function createOCTFile .
in	<i>Index</i>	Index of the data inside the OCT file, $0 \leq \text{Index} < \text{getFileDataObjectCount}()$
out	<i>Filename</i>	Name of the requested file
in	<i>Length</i>	Length of the user-provided buffer at <i>Filename</i>

6.23.4.26 getFileDataObjectType() `DataObjectType getFileDataObjectType (`

```
    OCTFileHandle Handle,
    int Index )
```

Returns the type of the data object at the given *Index* in the OCT file.

Parameters

in	<i>Handle</i>	A valid (non null) handle of OCTFile (OCTFileHandle), obtained with the function createOCTFile .
in	<i>Index</i>	Index of the data inside the OCT file, e.g. returned by findFileDataObject .

Returns

The type of the selected data object or `DataObjectType_Unknown` in case of an error.

6.23.4.27 getFileDataRangeX() `float getFileDataRangeX (`

```
    OCTFileHandle Handle,
    size_t Index )
```

Returns the range (usually in mm) in X of the data object at the given *Index* in the OCT file.

Parameters

in	<i>Handle</i>	A valid (non null) handle of OCTFile (OCTFileHandle), obtained with the function createOCTFile .
in	<i>Index</i>	Index of the data inside the OCT file, e.g. returned by findFileDataObject .

Returns

Range in X of the data object or 0.0f in case of an error

6.23.4.28 getFileDataRangeY() `float getFileDataRangeY (`

```
    OCTFileHandle Handle,
    size_t Index )
```

Returns the range (usually in mm) in Y of the data object at the given *Index* in the OCT file.

Parameters

in	<i>Handle</i>	A valid (non null) handle of OCTFile (<a>OCTFileHandle), obtained with the function <a>createOCTFile .
in	<i>Index</i>	Index of the data inside the OCT file, e.g. returned by <a>findFileDataObject .

Returns

Range in Y of the data object or 0.0f in case of an error

6.23.4.29 getFileDataRangeZ() float getFileDataRangeZ (
 OCTFileHandle Handle,
 size_t Index)

Returns the range (usually in mm) in Z of the data object at the given *Index* in the OCT file.

Parameters

in	<i>Handle</i>	A valid (non null) handle of OCTFile (<a>OCTFileHandle), obtained with the function <a>createOCTFile .
in	<i>Index</i>	Index of the data inside the OCT file, e.g. returned by <a>findFileDataObject .

Returns

Range in Z of the data object or 0.0f in case of an error

6.23.4.30 getFileDataSizeX() int getFileDataSizeX (
 OCTFileHandle Handle,
 size_t Index)

Returns the pixel count in X of the data object at the given *Index* in the OCT file.

Parameters

in	<i>Handle</i>	A valid (non null) handle of OCTFile (<a>OCTFileHandle), obtained with the function <a>createOCTFile .
in	<i>Index</i>	Index of the data inside the OCT file, e.g. returned by <a>findFileDataObject .

Returns

Pixel count in X of the data object or 0 in case of an error

6.23.4.31 getFileDataSizeY() int getFileDataSizeY (
 OCTFileHandle Handle,
 size_t Index)

Returns the pixel count in Y of the data object at the given *Index* in the OCT file.

Parameters

in	<i>Handle</i>	A valid (non null) handle of OCTFile (<a>OCTFileHandle), obtained with the function <a>createOCTFile .
in	<i>Index</i>	Index of the data inside the OCT file, e.g. returned by <a>findFileDataObject .

Returns

Pixel count in Y of the data object or 0 in case of an error

6.23.4.32 getFileDataSizeZ() `int getFileDataSizeZ (`
 OCTFileHandle *Handle*,
 size_t *Index* `)`

Returns the pixel count in Z of the data object at the given *Index* in the OCT file.

Parameters

in	<i>Handle</i>	A valid (non null) handle of OCTFile (<a>OCTFileHandle), obtained with the function <a>createOCTFile .
in	<i>Index</i>	Index of the data inside the OCT file, e.g. returned by <a>findFileDataObject .

Returns

Pixel count in Z of the data object or 0 in case of an error

6.23.4.33 getFileMetadataFlag() `BOOL getFileMetadataFlag (`
 OCTFileHandle *Handle*,
 FileMetadataFlag *Boolfield* `)`

Gets the boolean value of the given file metadata field.

Parameters

in	<i>Handle</i>	A valid (non null) handle of OCTFile (<a>OCTFileHandle), obtained with the function <a>createOCTFile .
in	<i>Boolfield</i>	Metadata field to read.

6.23.4.34 getFileMetadataFloat() `double getFileMetadataFloat (`
 OCTFileHandle *Handle*,
 FileMetadataFloat *Floatfield* `)`

Returns the value of the given file metadata field as a floating point number if found.

Parameters

in	<i>Handle</i>	A valid (non null) handle of OCTFile (OCTFileHandle), obtained with the function createOCTFile .
in	<i>Floatfield</i>	Metadata field to read.

```
6.23.4.35 getFileMetadataInt() int getFileMetadataInt (
    OCTFileHandle Handle,
    FileMetadataInt Intfield )
```

Returns the value of the given file metadata field as an integer if found.

Parameters

in	<i>Handle</i>	A valid (non null) handle of OCTFile (OCTFileHandle), obtained with the function createOCTFile .
in	<i>Intfield</i>	Metadata field to read.

```
6.23.4.36 getFileMetadataNumberOfPresets() int getFileMetadataNumberOfPresets (
    OCTFileHandle Handle )
```

Gets the number of presets that were set during the acquisition.

Parameters

in	<i>Handle</i>	A valid (non null) handle of OCTFile (OCTFileHandle), obtained with the function createOCTFile .
----	---------------	--

```
6.23.4.37 getFileMetadataPresetCategory() const char * getFileMetadataPresetCategory (
    OCTFileHandle Handle,
    int Index )
```

Gets the preset category belonging to the preset with given *Index*.

Parameters

in	<i>Handle</i>	A valid (non null) handle of OCTFile (OCTFileHandle), obtained with the function createOCTFile .
in	<i>Index</i>	Index of the preset inside the OCT file, $0 \leq \text{Index} < \text{getFileMetadataNumberOfPresets}$

```
6.23.4.38 getFileMetadataPresetDescription() const char * getFileMetadataPresetDescription (
    OCTFileHandle Handle,
    int Index )
```

Gets the preset description belonging to the preset with given *Index*.

Parameters

in	<i>Handle</i>	A valid (non null) handle of OCTFile (OCTFileHandle), obtained with the function createOCTFile .
in	<i>Index</i>	Index of the preset inside the OCT file, $0 \leq \text{Index} < \text{getFileMetadataNumberOfPresets}$

```
6.23.4.39 getFileMetadataString() const char * getFileMetadataString (
    OCTFileHandle Handle,
    FileMetadataString StringField )
```

Returns the value of the given file metadata field as a string if found.

Parameters

in	<i>Handle</i>	A valid (non null) handle of OCTFile (OCTFileHandle), obtained with the function createOCTFile .
in	<i>Stringfield</i>	Metadata field to read.

```
6.23.4.40 getFileMetadataTimestamp() time_t getFileMetadataTimestamp (
    OCTFileHandle File )
```

Returns the specified timestamp from the meta information of the given file handle. This information will be available in files of type FileFormat_OCITY; mileage on other formats may vary according to their description.

Parameters

in	<i>File</i>	A valid (non null) handle of OCTFile (OCTFileHandle), obtained with the function createOCTFile .
----	-------------	--

```
6.23.4.41 getFileRawData() void getFileRawData (
    OCTFileHandle Handle,
    RawDataHandle Data,
    size_t Index )
```

Retrieves a RawData object from the OCT file at the given index with $0 \leq \text{index} < \text{getFileDataObjectCount(OCTFileHandle handle)}$. Users must ensure that the data handle is properly prepared and destroyed.

Parameters

in	<i>Handle</i>	A valid (non null) handle of OCTFile (OCTFileHandle), obtained with the function createOCTFile .
out	<i>Data</i>	A valid (non null) raw data handle of the existing data (RawDataHandle), previously obtained with the function createRawData . It will be filled in with the data read from the OCT file at the given <i>Index</i> .
in	<i>Index</i>	Index of the data inside the OCT file, e.g. returned by findFileDataObject . Notice that raw data refers to the spectra as acquired, without processing of any kind.

```
6.23.4.42 getFileRealData() void getFileRealData (
    OCTFileHandle Handle,
    DataHandle Data,
    int Index )
```

Retrieves a RealData object from the OCT file at the given index with $0 \leq \text{index} < \text{getFileDataObjectCount(OCTFileHandle handle)}$. Users must ensure that the data handle is properly prepared and destroyed.

Parameters

in	<i>Handle</i>	A valid (non null) handle of OCTFile (OCTFileHandle), obtained with the function createOCTFile .
out	<i>Data</i>	A valid (non null) data handle of the existing data (DataHandle), previously obtained with the function createData . It will be filled in with the data read from the OCT file at the given <i>Index</i> .
in	<i>Index</i>	Index of the data inside the OCT file, e.g. returned by findFileDataObject .

```
6.23.4.43 loadCalibrationFromFile() void loadCalibrationFromFile (
    OCTFileHandle Handle,
    ProcessingHandle Proc )
```

Loads Chirp, Offset, and Apodization vectors from the given OCT file into the given processing object.

Parameters

in	<i>Handle</i>	A valid (non null) handle of OCTFile (OCTFileHandle), obtained with the function createOCTFile .
out	<i>Proc</i>	A valid (non null) handle of the processing routines (ProcessingHandle), previously obtained through one of the functions createProcessing , createProcessingForDevice , createProcessingForDeviceEx or createProcessingForOCTFile .

```
6.23.4.44 loadCalibrationFromFileEx() void loadCalibrationFromFileEx (
    OCTFileHandle Handle,
    ProcessingHandle Proc,
    const int CameraIndex )
```

Loads Chirp, Offset, and Apodization vectors from the given OCT file into the given processing object.

Parameters

in	<i>Handle</i>	A valid (non null) handle of OCTFile (OCTFileHandle), obtained with the function createOCTFile .
in	<i>Proc</i>	A valid (non null) handle of the processing routines (ProcessingHandle), previously obtained through one of the functions createProcessing , createProcessingForDevice , createProcessingForDeviceEx or createProcessingForOCTFile .
in	<i>CameraIndex</i>	The camera index (0-based, i.e. zero for the first, one for the second, and so on).

```
6.23.4.45 loadFile() void loadFile (
    OCTFileHandle Handle,
    const char * Filename )
```

Loads the actual OCT data file from a file system. The file must have the format given in [createOCTFile\(\)](#).

Parameters

in	<i>Handle</i>	A valid (non null) handle of OCTFile (OCTFileHandle), obtained with the function createOCTFile .
in	<i>Filename</i>	Name of the data file to load.

```
6.23.4.46 saveCalibrationToFile() void saveCalibrationToFile (
    OCTFileHandle Handle,
    ProcessingHandle Proc )
```

Saves Chirp, Offset, and Apodization vectors from the given processing object into the given OCT file.

Parameters

in	<i>Handle</i>	A valid (non null) handle of OCTFile (OCTFileHandle), obtained with the function createOCTFile .
in	<i>Proc</i>	A valid (non null) handle of the processing routines (ProcessingHandle), previously obtained through one of the functions createProcessing , createProcessingForDevice , createProcessingForDeviceEx or createProcessingForOCTFile .

```
6.23.4.47 saveCalibrationToFileEx() void saveCalibrationToFileEx (
    OCTFileHandle Handle,
    ProcessingHandle Proc,
    int CameraIndex )
```

Saves Chirp, Offset, and Apodization vectors from the given processing object into the given OCT file.

Parameters

in	<i>Handle</i>	A valid (non null) handle of OCTFile (OCTFileHandle), obtained with the function createOCTFile .
in	<i>Proc</i>	A valid (non null) handle of the processing routines (ProcessingHandle), previously obtained through one of the functions createProcessing , createProcessingForDevice , createProcessingForDeviceEx or createProcessingForOCTFile .
in	<i>CameraIndex</i>	The camera index (0-based, i.e. zero for the first, one for the second, and so on).

```
6.23.4.48 saveChangesToFile() void saveChangesToFile (
    OCTFileHandle Handle )
```

Saves the OCT data file in the file previously opened with [loadFile\(\)](#). Only changes will be saved.

Parameters

in	<i>Handle</i>	A valid (non null) handle of OCTFile (OCTFileHandle), obtained with the function createOCTFile .
----	---------------	--

```
6.23.4.49 saveFile() void saveFile (
    OCTFileHandle Handle,
    const char * Filename )
```

Saves the OCT data file in the given fully qualified path name.

Parameters

in	<i>Handle</i>	A valid (non null) handle of OCTFile (OCTFileHandle), obtained with the function createOCTFile .
in	<i>Filename</i>	Name to which the OCT data file will be written.

```
6.23.4.50 saveFileMetadata() void saveFileMetadata (
    OCTFileHandle Handle,
    OCTDeviceHandle Dev,
    ProcessingHandle Proc,
    ProbeHandle Probe,
    ScanPatternHandle Pattern )
```

Saves meta information from the given device, processing, probe and scan pattern instances in the metadata block of the given file handle. This information will be available in files of type FileFormat_OCITY; mileage on other formats may vary according to their description.

Parameters

in	<i>Handle</i>	A valid (non null) handle of OCTFile (OCTFileHandle), obtained with the function createOCTFile .
in	<i>Dev</i>	A valid (non null) OCT device handle (OCTDeviceHandle), previously generated with the function initDevice .
in	<i>Proc</i>	A valid (non null) handle of the processing routines (ProcessingHandle), previously obtained through one of the functions createProcessing , createProcessingForDevice , createProcessingForDeviceEx or createProcessingForOCTFile .
in	<i>Probe</i>	A valid (non null) handle of an initialized probem, obtained through initProbe .
in	<i>Pattern</i>	A valid (non null) handle of a scan pattern.

```
6.23.4.51 saveFileMetadataDoppler() void saveFileMetadataDoppler (
```

```
OCTFileHandle Handle,
DopplerProcessingHandle DopplerProc )
```

Saves meta information from the given DopplerProcessingHandle. A corresponding DopplerProcessingHandle can then be recreated using `createDopplerProcessingForFile`.

Parameters

in	<i>Handle</i>	A valid (non null) handle of OCTFile (OCTFileHandle), obtained with the function createOCTFile . This describes the files then handle data is stored to.
in	<i>DopplerProc</i>	A valid (non null) handle of Doppler processing obtained by createDopplerProcessing . This is the handle whose data is stored.

6.23.4.52 `saveFileMetadataSpeckle()` void saveFileMetadataSpeckle (

```
OCTFileHandle Handle,
SpeckleVarianceHandle SpeckleVarianceProc )
```

Saves meta information from the given SpeckleVarianceHandle. A corresponding SpeckleVarianceHandle can then be recreated using `initSpeckleVarianceForFile`.

Parameters

in	<i>Handle</i>	A valid (non null) handle of OCTFile (OCTFileHandle), obtained with the function createOCTFile . This describes the files then handle data is stored to.
in	<i>SpeckleVarianceProc</i>	A valid (non null) handle of speckle variance processing obtained by initSpeckleVariance . This is the handle whose data is stored.

6.23.4.53 `setFileMetadataFlag()` void setFileMetadataFlag (

```
OCTFileHandle Handle,
FileMetadataFlag Boolfield,
BOOL Value )
```

Sets the boolean value of the given file metadata field.

Parameters

in	<i>Handle</i>	A valid (non null) handle of OCTFile (OCTFileHandle), obtained with the function createOCTFile .
in	<i>Boolfield</i>	Metadata field to set.
in	<i>Value</i>	Boolean value to set on the field.

6.23.4.54 `setFileMetadataFloat()` void setFileMetadataFloat (

```
OCTFileHandle Handle,
```

```
FileMetadataFloat Floatfield,
double Value )
```

Sets the value of the given file metadata field as a floating point number.

Parameters

in	<i>Handle</i>	A valid (non null) handle of OCTFile (<a>OCTFileHandle), obtained with the function <a>createOCTFile .
in	<i>Floatfield</i>	Metadata field to set.
in	<i>Value</i>	Double value to set on the field.

6.23.4.55 setFileMetadataInt() void setFileMetadataInt (

```
OCTFileHandle Handle,
FileMetadataInt Intfield,
int Value )
```

Sets the value of the given file metadata field as an integer.

Parameters

in	<i>Handle</i>	A valid (non null) handle of OCTFile (<a>OCTFileHandle), obtained with the function <a>createOCTFile .
in	<i>Intfield</i>	Metadata field to set.
in	<i>Value</i>	Integer value to set on the field. If <i>Intfield</i> is <a>FileMetadata_ProcessState , this argument should be one of <a>FileMetadata_ProcessingState .

6.23.4.56 setFileMetadataString() void setFileMetadataString (

```
OCTFileHandle Handle,
FileMetadataString StringField,
const char * Content )
```

Sets the value of the given file metadata field as a string.

Parameters

in	<i>Handle</i>	A valid (non null) handle of OCTFile (<a>OCTFileHandle), obtained with the function <a>createOCTFile .
in	<i>Stringfield</i>	Metadata field to set.
in	<i>Content</i>	String value to set on the field.

6.23.4.57 setFileMetadataTimestamp() void setFileMetadataTimestamp (

```
OCTFileHandle File,
time_t Timestamp )
```

Saves provided timestamp to meta information to the given file handle. This information will be available in files of type FileFormat_OCITY; mileage on other formats may vary according to their description.

Parameters

in	<i>File</i>	A valid (non null) handle of OCTFile (OCTFileHandle), obtained with the function createOCTFile .
in	<i>Timestamp</i>	A valid (non null) timestamp.

6.24 External trigger

Functions to inquire, setup, and deal with an external trigger. Whether this functionality is supported, and to what extent, depends on the hardware.

Functions

- **SPECTRALRADAR_API void setTriggerMode (OCTDeviceHandle Dev, DeviceTriggerType TriggerMode)**
Sets the trigger mode for the OCT device used for acquisition. Additional hardware may be needed.
- **SPECTRALRADAR_API DeviceTriggerType getTriggerMode (OCTDeviceHandle Dev)**
Returns the trigger mode used for acquisition.
- **SPECTRALRADAR_API BOOL isTriggerModeAvailable (OCTDeviceHandle Dev, DeviceTriggerType TriggerMode)**
Returns whether the specified trigger mode is possible or not for the used device.
- **SPECTRALRADAR_API BOOL isTriggerIOModeAvailable (OCTDeviceHandle Dev, DeviceTriggerIOType TriggerMode)**
Returns whether the specified trigger IO mode is possible or not for the used device.
- **SPECTRALRADAR_API void setTriggerIOMode (OCTDeviceHandle Dev, DeviceTriggerIOType TriggerMode)**
Sets the trigger mode for the trigger IO channel of the OCT device.
- **SPECTRALRADAR_API DeviceTriggerIOType getTriggerIOMode (OCTDeviceHandle Dev)**
Returns the trigger IO mode used for acquisition.
- **SPECTRALRADAR_API void setTriggerIOConfiguration (OCTDeviceHandle Dev, size_t Offset, size_t Divider)**
Configures the parameters for trigger mode for the trigger IO channel of the OCT device. If the trigger IO channel is set to output, it will start generating a pulses after n=Offset A-Scans. It will then generate a pulse every after d=Divider A-Scans. If the trigger IO channel is set to input, Offset is ignored. Whenever a trigger pulse is received on the trigger IO channel, the system will perform d=Divider A-Scans and wait for the next trigger.
- **SPECTRALRADAR_API void setTriggerTimeout_s (OCTDeviceHandle Dev, int Timeout_s)**
Sets the timeout of the camera in seconds (useful in external trigger mode).
- **SPECTRALRADAR_API int getTriggerTimeout_s (OCTDeviceHandle Dev)**
Returns the timeout of the camera in seconds (not used in trigger mode Trigger_FreeRunning).

6.24.1 Detailed Description

Functions to inquire, setup, and deal with an external trigger. Whether this functionality is supported, and to what extent, depends on the hardware.

6.24.2 Function Documentation

6.24.2.1 `getTriggerIOMode()` `DeviceTriggerIOType getTriggerIOMode (` `OCTDeviceHandle Dev)`

Returns the trigger IO mode used for acquisition.

Parameters

in	Dev	Valid OCT device handle
----	-----	-------------------------

Returns

Current trigger mode

6.24.2.2 getTriggerMode() `DeviceTriggerType` `getTriggerMode` (
 `OCTDeviceHandle Dev`)

Returns the trigger mode used for acquisition.

Parameters

in	<i>Dev</i>	Valid OCT device handle
----	------------	-------------------------

Returns

Current trigger mode

6.24.2.3 getTriggerTimeout_s() `int` `getTriggerTimeout_s` (
 `OCTDeviceHandle Dev`)

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

Parameters

in	<i>Dev</i>	Valid OCT device handle
----	------------	-------------------------

Returns

Timeout in external trigger mode in seconds

6.24.2.4 isTriggerIOModeAvailable() `BOOL` `isTriggerIOModeAvailable` (
 `OCTDeviceHandle Dev,`
 `DeviceTriggerIOType TriggerMode`)

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

Parameters

in	<i>Dev</i>	Valid OCT device handle
in	<i>TriggerMode</i>	Mode to query

```
6.24.2.5 isTriggerModeAvailable() BOOL isTriggerModeAvailable (
    OCTDeviceHandle Dev,
    DeviceTriggerType TriggerMode )
```

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

Parameters

in	<i>Dev</i>	Valid OCT device handle
in	<i>TriggerMode</i>	Mode to query

```
6.24.2.6 setTriggerIOConfiguration() void setTriggerIOConfiguration (
```

```
OCTDeviceHandle Dev,
size_t Offset,
size_t Divider )
```

Configures the parameters for trigger mode for the trigger IO channel of the OCT device. If the trigger IO channel is set to output, it will start generating a pulses after n=Offset A-Scans. It will then generate a pulse every after d=Divider A-Scans. If the trigger IO channel is set to input, Offset is ignored. Whenever a trigger pulse is received on the trigger IO channel, the system will perform d=Divider A-Scans and wait for the next trigger.

Parameters

in	<i>Dev</i>	Valid OCT device handle
in	<i>Offset</i>	Number of A-Scans to wait before first trigger IO pulse
in	<i>Divider</i>	Number of A-Scans to wait before each subsequent trigger IO pulse

```
6.24.2.7 setTriggerIOMode() void setTriggerIOMode (
```

```
OCTDeviceHandle Dev,
DeviceTriggerIOType TriggerMode )
```

Sets the trigger mode for the trigger IO channel of the OCT device.

Warning

Not all systems support trigger IO. To check if the system supports trigger IO, please use [isTriggerIOModeAvailable](#)

Parameters

in	<i>Dev</i>	Valid OCT device handle
in	<i>TriggerMode</i>	Which trigger mode to use

```
6.24.2.8 setTriggerMode() void setTriggerMode (
```

```
OCTDeviceHandle Dev,  
DeviceTriggerType TriggerMode )
```

Sets the trigger mode for the OCT device used for acquisition. Additional hardware may be needed.

Parameters

in	<i>Dev</i>	Valid OCT device handle
in	<i>TriggerMode</i>	Which trigger mode to use

6.24.2.9 setTriggerTimeout_s()

```
void setTriggerTimeout_s (
```

```
    OCTDeviceHandle Dev,
```

```
    int Timeout_s )
```

Sets the timeout of the camera in seconds (useful in external trigger mode).

Parameters

in	<i>Dev</i>	Valid OCT device handle
in	<i>Timeout_s</i>	Timeout in external trigger mode in seconds

6.25 Post Processing

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

Enumerations

- enum PepperFilterType {
 PepperFilter_Horizontal,
 PepperFilter_Vertical,
 PepperFilter_Star,
 PepperFilter_Block }

 Specifies the type of pepper filter to be applied.
- enum ComplexFilterType2D { FilterComplex2D_PhaseContrast }

 Specifies the type of filter to be applied to complex data.
- enum FilterType1D { Filter1D_Gaussian_5 }

 Specifies the type of 1D-filter to be applied. All filters are normalized.
- enum FilterType2D {
 Filter2D_Gaussian_3x3,
 Filter2D_Gaussian_5x5,
 Filter2D_Prewitt_Horizontal_3x3,
 Filter2D_Prewitt_Vertical_3x3,
 Filter2D_NonlinearPrewitt_3x3,
 Filter2D_Sobel_Horizontal_3x3,
 Filter2D_Sobel_Vertical_3x3,
 Filter2D_NonlinearSobel_3x3,
 Filter2D_Laplacian_NoDiagonal_3x3,
 Filter2D_Laplacian_3x3 }

 Specifies the type of 2D-filter to be applied. All filters are normalized.
- enum FilterType3D { Filter3D_Gaussian_3x3x3 }

 Specifies the type of 3D-filter to be applied. All filters are normalized.

Functions

- **SPECTRALRADAR_API** void `determineDynamicRange_dB` (`DataHandle` Data, double *MinRange_dB, double *MaxRange_dB)

 Gives a rough estimation of the dynamic range of the specified data object.
- **SPECTRALRADAR_API** void `determineDynamicRangeWithMinRange_dB` (`DataHandle` Data, double *MinRange_dB, double *MaxRange_dB, double MinDynamicRange_dB)

 Gives a rough estimation of the dynamic range of the specified data object.
- **SPECTRALRADAR_API** void `medianFilter1D` (`DataHandle` Data, int Rank, `Direction` FilterDirection)

 Computes a 1D-median filter on the specified data.
- **SPECTRALRADAR_API** void `medianFilter2D` (`DataHandle` Data, int Rank, `Direction` FilterNormalDirection)

 Computes a 2D-median filter on the specified data.
- **SPECTRALRADAR_API** void `pepperFilter2D` (`DataHandle` Data, `PepperFilterType` Type, float Threshold, `Direction` FilterNormalDirection)

 Removes pepper-noise (very low values, i. e. dark spots in the data). This enhances the visual (colored) representation of the data.
- **SPECTRALRADAR_API** void `convolutionFilter1D` (`DataHandle` Data, int FilterSize, float *FilterKernel, `Direction` FilterDirection)

 Calculates a mathematical convolution of the Data and the 1D-FilterKernel.
- **SPECTRALRADAR_API** void `convolutionFilter2D` (`DataHandle` Data, int FilterSize1, int FilterSize2, float *FilterKernel, `Direction` FilterNormalDirection)

- Calculates a mathematical convolution of the Data and the 2D-FilterKernel.*
- **SPECTRALRADAR_API** void `convolutionFilter3D` (`DataHandle` Data, int FilterSize1, int FilterSize2, int FilterSize3, float *FilterKernel)
Calculates a mathematical convolution of the Data and the 3D-FilterKernel.
 - **SPECTRALRADAR_API** void `predefinedFilter1D` (`DataHandle` Data, `FilterType1D` Filter, `Direction` FilterDirection)
Applies the predefined 1D-Filter to the Data.
 - **SPECTRALRADAR_API** void `predefinedFilter2D` (`DataHandle` Data, `FilterType2D` Filter, `Direction` FilterNormalDirection)
Applies the predefined 2D-Filter to the Data.
 - **SPECTRALRADAR_API** void `predefinedFilter3D` (`DataHandle` Data, `FilterType3D` FilterType)
Applies the predefined 3D-Filter to the Data.
 - **SPECTRALRADAR_API** void `predefinedComplexFilter2D` (`ComplexDataHandle` ComplexData, `ComplexFilterType2D` Type, `Direction` FilterNormalDirection)
Applies the predefined 2D-Filter to the ComplexData.
 - **SPECTRALRADAR_API** void `darkFieldComplexFilter2D` (`ComplexDataHandle` ComplexData, double Radius, `Direction` FilterNormalDirection)
Filters the image such that the image contrast comes from light scattered by the sample.
 - **SPECTRALRADAR_API** void `brightFieldComplexFilter2D` (`ComplexDataHandle` ComplexData, double Radius, `Direction` FilterNormalDirection)
Filters the image such that the image contrast comes from absorbance of light in the sample.
 - **SPECTRALRADAR_API** double `getExpectedHannFWHM` (`OCTDeviceHandle` Dev, `ProcessingHandle` Proc)
Returns the theoretical width of a hann point-spread function (PSF) given the current spectral width and edge channels.
 - **SPECTRALRADAR_API** void `calcContrastEx` (`ProcessingHandle` Proc, `DataHandle` ApodizedSpectrum, `DataHandle` Contrast)
Calculates the spectrometer's contrast given a modulated spectrum, a properly calibrated processing (containing an apodization spectrum) as a function of the camera pixel.
 - **SPECTRALRADAR_API** double `calcSpectrometerImagingQualityIndex` (`DataHandle` Spectrum)
Given a maximally modulated spectrum, this function calculates a quality indicator that describes the image quality of the spectrum onto the line camera.
 - **SPECTRALRADAR_API** void `analyzeMaxPeak` (`DataHandle` Data, float *PeakHeight_dB, float *PeakFWHM_Pixel)
This function analyzes the highest peak in the given A-scan data and yields its height (in dB) and full-width at half maximum (FWHM) in pixels.
 - **SPECTRALRADAR_API** void `analyzeMaxPeakEx` (`DataHandle` Data, float *PeakHeight_dB, float *PeakFWHM_Pixel, float *Points, float *Spline, int *SplineSize)
This function analyzes the highest peak in the given A-scan data and yields its height (in dB) and full-width at half maximum (FWHM) in pixels. In addition to `analyzeMaxPeak` this function gives additional data that was used for the determination, such as the relevant points for maximum, and FWHM computations, as well as the interpolated spline.

6.25.1 Detailed Description

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

6.25.2 Enumeration Type Documentation

6.25.2.1 ComplexFilterType2D `enum ComplexFilterType2D`

Specifies the type of filter to be applied to complex data.

Enumerator

FilterComplex2D_PhaseContrast	A filter applied to complex data to get a phase contrast image.
-------------------------------	---

Definition at line 895 of file [SpectralRadar_Types.h](#).

6.25.2.2 FilterType1D enum FilterType1D

Specifies the type of 1D-filter to be applied. All filters are normalized.

Enumerator

Filter1D_Gaussian_5	A gaussian 1D-filter of size 5 to smooth the data.
---------------------	--

Definition at line 903 of file [SpectralRadar_Types.h](#).

6.25.2.3 FilterType2D enum FilterType2D

Specifies the type of 2D-filter to be applied. All filters are normalized.

Enumerator

Filter2D_Gaussian_3x3	A gaussian filter of size 3x3 to smooth the data.
Filter2D_Gaussian_5x5	A gaussian filter of size 5x5 to smooth the data.
Filter2D_Prewitt_Horizontal_3x3	Horizontal prewitt filter of size 3x3 to detect edges in horizontal direction.
Filter2D_Prewitt_Vertical_3x3	Vertical prewitt filter of size 3x3 to detect edges in vertical direction.
Filter2D_NonlinearPrewitt_3x3	Maximum of horizontal and vertical prewitt filter each of size 3x3 to detect edges.
Filter2D_Sobel_Horizontal_3x3	Horizontal sobel filter of size 3x3 to detect edges in horizontal direction while smoothing in vertical direction.
Filter2D_Sobel_Vertical_3x3	Vertical prewitt filter of size 3x3 to detect edges in vertical direction while smoothing in horizontal direction.
Filter2D_NonlinearSobel_3x3	Maximum of horizontal and vertical sobel filter each of size 3x3 to detect edges while smoothing the data simultaneously.
Filter2D_Laplacian_NoDiagonal_3x3	Laplacian filter of size 3x3 to detect horizontal and vertical edges, no diagonal edges.
Filter2D_Laplacian_3x3	Laplacian filter of size 3x3 to detect horizontal, vertical and diagonal edges.

Definition at line 911 of file [SpectralRadar_Types.h](#).

6.25.2.4 FilterType3D enum `FilterType3D`

Specifies the type of 3D-filter to be applied. All filters are normalized.

Enumerator

<code>Filter3D_Gaussian_3x3x3</code>	A gaussian filter of size 3x3 to smooth the data.
--------------------------------------	---

Definition at line 937 of file [SpectralRadar_Types.h](#).

6.25.2.5 PepperFilterType enum `PepperFilterType`

Specifies the type of pepper filter to be applied.

Enumerator

<code>PepperFilter_Horizontal</code>	Values along the horizontal axis are taken into account for the pepper filter.
<code>PepperFilter_Vertical</code>	Values along the vertical axis are taken into account for the pepper filter.
<code>PepperFilter_Star</code>	Values along the vertical and horizontal axis (star shape) are taken into account for the pepper filter.
<code>PepperFilter_Block</code>	Values in a block surrounding the destination pixel are taken into account.

Definition at line 881 of file [SpectralRadar_Types.h](#).

6.25.3 Function Documentation

```
6.25.3.1 analyzeMaxPeak() void analyzeMaxPeak (
    DataHandle Data,
    float * PeakHeight_dB,
    float * PeakFWHM_Pixel )
```

This function analyzes the highest peak in the given A-scan data and yields its height (in dB) and full-width at half maximum (FWHM) in pixels.

Warning

Experimental: This function may change in its interface and behaviour, or even disappear in future versions.

Parameters

in	<code>Data</code>	A-scan data in dB containing the peak to be analyzed
out	<code>PeakHeight_dB</code>	Computed peak height (usually in dB)
out	<code>PeakFWHM_Pixel</code>	FWHM (-6 dB) of the highest peak

```
6.25.3.2 analyzeMaxPeakEx() void analyzeMaxPeakEx (
    DataHandle Data,
    float * PeakHeight_dB,
    float * PeakFWHM_Pixel,
    float * Points,
    float * Spline,
    int * SplineSize )
```

This function analyzes the highest peak in the given A-scan data and yields its height (in dB) and full-width at half maximum (FWHM) in pixels. In addition to [analyzeMaxPeak](#) this function gives additional data that was used for the determination, such as the relevant points for maximum, and FWHM computations, as well as the interpolated spline.

Warning

Experimental: This function may change in its interface and behaviour, or even disappear in future versions.

Parameters

in	<i>Data</i>	A-scan data in dB containing the peak to be analyzed
out	<i>PeakHeight_dB</i>	Computed peak height (usually in dB)
out	<i>PeakFWHM_Pixel</i>	FWHM (-6 dB) of the highest peak
out	<i>Points</i>	Contains the six point coordinates of the three points there were relevant for the determination of the peak height and the FWHM.
out	<i>Spline</i>	This array contains a spline interpolated from the A-scan data that was used for a more precise determination of peak height and FWHM. The array must be at least of the size <i>SplineSize</i> provided to the function.
	<i>SplineSize</i>	This pointer serves as in- and output. As input it gives the maximum length of the spline array; as output it gives the actual number of points in the spline.

```
6.25.3.3 brightFieldComplexFilter2D() void brightFieldComplexFilter2D (
```

```
    ComplexDataHandle ComplexData,
    double Radius,
    Direction FilterNormalDirection )
```

Filters the image such that the image contrast comes from absorbance of light in the sample.

Parameters

<i>ComplexData</i>	The ComplexDataHandle the filter will be applied to.
<i>Radius</i>	Parameter to adjust the image contrast.
<i>FilterNormalDirection</i>	The normal of the direction the 2D-filter will be applied to the complex data, e.g. <i>Direction_3</i> for filtering each single B-scan

```
6.25.3.4 calcContrastEx() void calcContrastEx (
    ProcessingHandle Proc,
    DataHandle ApodizedSpectrum,
    DataHandle Contrast )
```

Calculates the spectrometer's contrast given a modulated spectrum, a properly calibrated processing (containing an apodization spectrum) as a function of the camera pixel.

Warning

Experimental: This function may change in its interface and behaviour, or even disappear in future versions.

Parameters

in	<i>Proc</i>	Processing handle holding valid calibration data
in	<i>ApodizedSpectrum</i>	Modulated spectrum that was properly apodized
out	<i>Contrast</i>	Result contrast

```
6.25.3.5 calcSpectrometerImagingQualityIndex() double calcSpectrometerImagingQualityIndex (
    DataHandle Spectrum )
```

Given a maximally modulated spectrum, this function calculates a quality indicator that describes the image quality of the spectrum onto the line camera.

Warning

Experimental: This function may change in its interface and behaviour, or even disappear in future versions.

Parameters

in	<i>Spectrum</i>	The maximally modulated spectrum that is used for quality estimation
----	-----------------	--

Returns

Quality value in the range of 0 to 100

```
6.25.3.6 convolutionFilter1D() void convolutionFilter1D (
    DataHandle Data,
    int Size,
    float * FilterKernel,
    Direction FilterDirection )
```

Calculates a mathematical convolution of the Data and the 1D-FilterKernel.

Parameters

<i>Data</i>	The DataHandle the filter will be applied to
<i>Size</i>	Size of the filter
<i>FilterKernel</i>	Pointer to the array containing the filter kernel
<i>FilterDirection</i>	The filter direction the 1D-filter will be applied to the data, e.g. <code>Direction_1</code> for filtering each single A-scan

6.25.3.7 convolutionFilter2D() `void convolutionFilter2D (`

```
    DataHandle Data,
    int FilterSize1,
    int FilterSize2,
    float * FilterKernel,
    Direction FilterNormalDirection )
```

Calculates a mathematical convolution of the Data and the 2D-FilterKernel.

Parameters

<i>Data</i>	The DataHandle the filter will be applied to
<i>FilterSize1</i>	Size of the first dimension of the filter
<i>FilterSize2</i>	Size of the second dimension of the filter
<i>FilterKernel</i>	Pointer to the array containing the filter kernel
<i>FilterNormalDirection</i>	The normal of the direction the 2D-filter will be applied to the data, e.g. <code>Direction_3</code> for filtering each single B-scan

6.25.3.8 convolutionFilter3D() `void convolutionFilter3D (`

```
    DataHandle Data,
    int FilterSize1,
    int FilterSize2,
    int FilterSize3,
    float * FilterKernel )
```

Calculates a mathematical convolution of the Data and the 3D-FilterKernel.

Parameters

<i>Data</i>	The DataHandle the filter will be applied to
<i>FilterSize1</i>	Size of the first dimension of the filter
<i>FilterSize2</i>	Size of the second dimension of the filter
<i>FilterSize3</i>	Size of the third dimension of the filter
<i>FilterKernel</i>	Pointer to the array containing the filter kernel

6.25.3.9 darkFieldComplexFilter2D() `void darkFieldComplexFilter2D (`

```
ComplexDataHandle ComplexData,
double Radius,
Direction FilterNormalDirection )
```

Filters the image such that the image contrast comes from light scattered by the sample.

Parameters

<i>ComplexData</i>	The ComplexDataHandle the filter will be applied to.
<i>Radius</i>	Parameter to adjust the image contrast.
<i>FilterNormalDirection</i>	The normal of the direction the 2D-filter will be applied to the complex data, e.g. <i>Direction_3</i> for filtering each single B-scan

6.25.3.10 **determineDynamicRange_dB()** void determineDynamicRange_dB (

```
DataHandle Data,
double * MinRange_dB,
double * MaxRange_dB )
```

Gives a rough estimation of the dynamic range of the specified data object.

This functions assumes that the data contains an A-scan and performs A-scan specific analysis on it.

Parameters

<i>Data</i>	The DataHandle the filter will be applied to
<i>MinRange_dB</i>	Used to return the lower bound of the dynamic range
<i>MaxRange_dB</i>	Used to return the upper bound of the dynamic range

6.25.3.11 **determineDynamicRangeWithMinRange_dB()** void determineDynamicRangeWithMinRange_dB (

```
DataHandle Data,
double * MinRange_dB,
double * MaxRange_dB,
double MinDynamicRange_dB )
```

Gives a rough estimation of the dynamic range of the specified data object.

Parameters

<i>Data</i>	The DataHandle the filter will be applied to
<i>MinRange_dB</i>	Used to return the lower bound of the dynamic range
<i>MaxRange_dB</i>	Used to return the upper bound of the dynamic range
<i>MinDynamicRange_dB</i>	Minimal size of the returned dynamic range interval in dB

6.25.3.12 **getExpectedHannFWHM()** double getExpectedHannFWHM (

```
OCTDeviceHandle Dev,
ProcessingHandle Proc )
```

Returns the theoretical width of a hann point-spread function (PSF) given the current spectral width and edge channels.

Warning

Experimental: This function may change in its interface and behaviour, or even disappear in future versions.

Parameters

in	<i>Dev</i>	Current device handle
in	<i>Proc</i>	Processing handle holding reference calibrations for current device

Returns

PSF-Width in pixels

6.25.3.13 medianFilter1D() void medianFilter1D (

```
DataHandle Data,
int Rank,
Direction FilterDirection )
```

Computes a 1D-median filter on the specified data.

Parameters

<i>Data</i>	The DataHandle the filter will be applied to
<i>Rank</i>	The size of the filter
<i>FilterDirection</i>	The direction the 1D-filter will be applied to the data.

6.25.3.14 medianFilter2D() void medianFilter2D (

```
DataHandle Data,
int Rank,
Direction FilterNormalDirection )
```

Computes a 2D-median filter on the specified data.

Parameters

<i>Data</i>	The DataHandle the filter will be applied to
<i>Rank</i>	The size of the filter
<i>FilterNormalDirection</i>	The normal of the direction the 2D-filter will be applied to the data.

```
6.25.3.15 pepperFilter2D() void pepperFilter2D (
    DataHandle Data,
    PepperFilterType Type,
    float Threshold,
    Direction FilterNormalDirection )
```

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

Parameters

<i>Data</i>	The DataHandle the filter will be applied to
<i>Type</i>	The type of the pepper filter chosen from PepperFilterType
<i>Threshold</i>	If the value is lower than the given value it will be replaced by the mean
<i>FilterNormalDirection</i>	The normal of the direction the 2D-filter will be applied to the data

The pepper filter compares all pixels to a mean of surrounding pixels. The surrounding pixels taking into account are specified by [PepperFilterType](#). If the pixels is lower than specified by the Threshold the pixel will be replaced by the mean.

```
6.25.3.16 predefinedComplexFilter2D() void predefinedComplexFilter2D (
    ComplexDataHandle ComplexData,
    ComplexFilterType2D Type,
    Direction FilterNormalDirection )
```

Applies the predefined 2D-Filter to the ComplexData.

Parameters

<i>ComplexData</i>	The ComplexDataHandle the filter will be applied to
<i>Type</i>	Chosen predefined filter for complex data. See ComplexFilterType2D for selection.
<i>FilterNormalDirection</i>	The normal of the direction the 2D-filter will be applied to the complex data, e.g. Direction_3 for filtering each single B-scan

```
6.25.3.17 predefinedFilter1D() void predefinedFilter1D (
    DataHandle Data,
    FilterType1D Filter,
    Direction FilterDirection )
```

Applies the predefined 1D-Filter to the Data.

Parameters

<i>Data</i>	The DataHandle the filter will be applied to
<i>Filter</i>	Selection of a predefined filter FilterType1D
<i>FilterDirection</i>	The filter direction the 1D-filter will be applied to the data, e.g. Direction_1 for filtering each single A-scan

```
6.25.3.18 predefinedFilter2D() void predefinedFilter2D (
    DataHandle Data,
    FilterType2D Filter,
    Direction FilterNormalDirection )
```

Applies the predefined 2D-Filter to the Data.

Parameters

<i>Data</i>	The DataHandle the filter will be applied to
<i>Filter</i>	Selection of a predefined filter FilterType2D
<i>FilterNormalDirection</i>	The normal of the direction the 2D-filter will be applied to the data, e.g. Direction_3 for filtering each single B-scan

```
6.25.3.19 predefinedFilter3D() void predefinedFilter3D (
    DataHandle Data,
    FilterType3D FilterType )
```

Applies the predefined 3D-Filter to the Data.

Parameters

<i>Data</i>	The DataHandle the filter will be applied to
<i>FilterType</i>	Selection of a predefined filter FilterType3D

6.26 Polarization

Polarization Sensitive OCT Processing Routines.

Typedefs

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

Enumerations

- `enum PolarizationPropertyInt {
PolarizationProcessing_DOPU_Z = 0,
PolarizationProcessing_DOPU_X = 1,
PolarizationProcessing_DOPU_Y = 2,
PolarizationProcessing_DOPU_FilterType = 3,
PolarizationProcessing_BScanAveraging = 4,
PolarizationProcessing_BScanAveraged = 9,
PolarizationProcessing_AveragingZ = 5,
PolarizationProcessing_AveragingX = 6,
PolarizationProcessing_AveragingY = 7,
PolarizationProcessing_AScanAveraging = 8 }`

Values that determine the behaviour of the Polarization processing routines.

- `enum PolarizationPropertyFloat {
PolarizationProcessing_IntensityThreshold_dB = 0,
PolarizationProcessing_PMDCorrectionAngle_rad = 1,
PolarizationProcessing_CentralWavelength_nm = 2,
PolarizationProcessing_OpticalAxisOffset_rad = 3 }`

Values that determine the behaviour of the Polarization processing routines.

- `enum PolarizationFlag {
PolarizationProcessing_ApplyThresholding = 0,
PolarizationProcessing_YAxisIsFrameAxis = 1 }`

Flags that determine the behaviour of the Polarization processing routines.

- `enum PolarizationDOPUFilterType {
PolarizationProcessing_DOPU_Median,
PolarizationProcessing_DOPU_Average,
PolarizationProcessing_DOPU_Gaussian,
PolarizationProcessing_DOPU_GaussianWithFFT }`

Values that determine the behaviour of temporal filter, if enabled.

- `enum PolarizationRetarder {
Retarder_Quarter_Wave = 0,
Retarder_Half_Wave = 1 }`

List of available polarization retarders in a polarization control unit.

Functions

- `SPECTRALRADAR_API void setPolarizationFlag (PolarizationProcessingHandle Polarization, PolarizationFlag Flag, BOOL OnOff)`
Sets the polarization processing flags.
- `SPECTRALRADAR_API BOOL getPolarizationFlag (PolarizationProcessingHandle Polarization, PolarizationFlag Flag)`
Gets the desired polarization processing flag.

- **SPECTRALRADAR_API** `PolarizationProcessingHandle createPolarizationProcessing (void)`
Returns a Polarization processing handle to the Processing routines for polarization analysis.
- **SPECTRALRADAR_API** `void clearPolarizationProcessing (PolarizationProcessingHandle Polarization)`
Clears the polarization processing routines and frees the memory that has been allocated for these to work properly.
- **SPECTRALRADAR_API** `int getPolarizationPropertyInt (PolarizationProcessingHandle Polarization, PolarizationPropertyInt Property)`
Gets the desired polarization processing property.
- **SPECTRALRADAR_API** `void setPolarizationPropertyInt (PolarizationProcessingHandle Polarization, PolarizationPropertyInt Property, int Value)`
Sets polarization processing properties.
- **SPECTRALRADAR_API** `double getPolarizationPropertyFloat (PolarizationProcessingHandle Polarization, PolarizationPropertyFloat Property)`
Gets the desired polarization processing floating-point property.
- **SPECTRALRADAR_API** `void setPolarizationPropertyFloat (PolarizationProcessingHandle Polarization, PolarizationPropertyFloat Property, double Value)`
Sets the desired polarization processing floating-point property.
- **SPECTRALRADAR_API** `void setPolarizationOutputI (PolarizationProcessingHandle Polarization, DataHandle Intensity)`
Sets the location of the resulting polarization intensity output (Stokes parameter I).
- **SPECTRALRADAR_API** `void setPolarizationOutputQ (PolarizationProcessingHandle Polarization, DataHandle StokesQ)`
Sets the location of the resulting Stokes parameter Q.
- **SPECTRALRADAR_API** `void setPolarizationOutputU (PolarizationProcessingHandle Polarization, DataHandle StokesU)`
Sets the location of the resulting Stokes parameter U.
- **SPECTRALRADAR_API** `void setPolarizationOutputV (PolarizationProcessingHandle Polarization, DataHandle StokesV)`
Sets the location of the resulting Stokes parameter V.
- **SPECTRALRADAR_API** `void setPolarizationOutputDOPU (PolarizationProcessingHandle Polarization, DataHandle DOPU)`
Sets the location of the resulting DOPU (Degree Of Polarization Uniformity).
- **SPECTRALRADAR_API** `void setPolarizationOutputRetardation (PolarizationProcessingHandle Polarization, DataHandle Retardation)`
Sets the location of the resulting retardation.
- **SPECTRALRADAR_API** `void setPolarizationOutputOpticAxis (PolarizationProcessingHandle Polarization, DataHandle OpticAxis)`
Sets the location of the resulting optic axis.
- **SPECTRALRADAR_API** `void executePolarizationProcessing (PolarizationProcessingHandle Polarization, ComplexDataHandle Data_Camera1, ComplexDataHandle PData_Camera0)`
Executes the polarization processing of the input data and returns, if previously setup, intensity, retardation, and phase differences.
- **SPECTRALRADAR_API** `void saveFileMetadataPolarization (OCTFileHandle FileHandle, PolarizationProcessingHandle PolProc)`
Saves metadata to the specified file. These metadata specify the operational arguments needed by the polarization processing routines to redo the polarization-analysis starting from two `ComplexDataHandle` delivered by `Proc_0` and `Proc_1`.
- **SPECTRALRADAR_API** `PolarizationProcessingHandle createPolarizationProcessingForFile (OCTFileHandle FileHandle)`
Loads metadata to the specified file. These metadata specify the operational arguments needed by the polarization processing routines to redo the polarization-analysis starting from two `ComplexDataHandle` delivered by `Proc_0` and `Proc_1`, exactly as they were done before the file was written.

6.26.1 Detailed Description

Polarization Sensitive OCT Processing Routines.

This section deals with polarization sensitive OCT (PS-OCT).

6.26.2 Typedef Documentation

6.26.2.1 **PolarizationProcessingHandle** `PolarizationProcessingHandle`

Handle used for Polarization processing.

Definition at line 121 of file [SpectralRadar_Handles.h](#).

6.26.3 Enumeration Type Documentation

6.26.3.1 **PolarizationDOPUFilterType** `enum PolarizationDOPUFilterType`

Values that determine the behaviour of temporal filter, if enabled.

Enumerator

PolarizationProcessing_DOPU_Median	Median.
PolarizationProcessing_DOPU_Average	Average.
PolarizationProcessing_DOPU_Gaussian	Convolution with a Gaussian kernel.
PolarizationProcessing_DOPU_GaussianWithFFT	FFT convolution with a Gaussian kernel (presumably more efficient for very large kernels).

Definition at line 1025 of file [SpectralRadar_Types.h](#).

6.26.3.2 **PolarizationFlag** `enum PolarizationFlag`

Flags that determine the behaviour of the Polarization processing routines.

Enumerator

PolarizationProcessing_ApplyThresholding	TRUE if thresholding should be applied. In that case, the total intensity will be compared to the threshold. If lower, then the computed Stokes parameters will be set to zero.
PolarizationProcessing_YAxisIsFrameAxis	When post-processing a multiframe B-scan, this flag signals that the many frames are encoded along the Y-axis.

Definition at line 564 of file [SpectralRadar_Properties.h](#).

6.26.3.3 **PolarizationPropertyFloat** `enum PolarizationPropertyFloat`

Values that determine the behaviour of the Polarization processing routines.

Enumerator

PolarizationProcessing_IntensityThreshold_dB	Threshold value to enable/disable features of PS computation based on the total intensity value.
PolarizationProcessing_PMDCorrectionAngle_rad	Correction angle (in radians) to get circularly polarized light at the upper surface of the sample. This angle is a compensation fo the polarization-mode-dispersion (PMD). More in detail, this angle is used to compute a phasor ($\exp(i\alpha)$), that will be applied to the complex reflectivities vector associated with camera 0.
PolarizationProcessing_CentralWavelength_nm	Assuming a gaussian light source, the value of the wavenumber with maximal intensity (in nm).
PolarizationProcessing_OpticalAxisOffset_rad	Refer to a particular orientation on the sample holder. The angle should be expressed in radians.

Definition at line 546 of file [SpectralRadar_Properties.h](#).

6.26.3.4 **PolarizationPropertyInt** `enum PolarizationPropertyInt`

Values that determine the behaviour of the Polarization processing routines.

Enumerator

PolarizationProcessing_DOPU_Z	Number of pixels for DOPU averaging in the z-direction.
PolarizationProcessing_DOPU_X	Number of pixels for DOPU averaging in the z-direction.
PolarizationProcessing_DOPU_Y	Number of pixels for DOPU averaging in the y-direction.
PolarizationProcessing_DOPU_FilterType	DOPU filter specification. See PolarizationDOPUFilterType .
PolarizationProcessing_BScanAveraging	Number of frames for averaging.
PolarizationProcessing_BScanAveraged	Number of frames actually averaged (transiently different from PolarizationProcessing_BScanAveraging until enough frames are available).
PolarizationProcessing_AveragingZ	Number of pixels for averaging along the x axis.
PolarizationProcessing_AveragingX	Number of pixels for averaging along the y axis.
PolarizationProcessing_AveragingY	Number of pixels for averaging along the z axis.
PolarizationProcessing_AScanAveraging	A-Scan averaging. This parameter influences the way data get acquired, it cannot be changed for offline processing.

Definition at line 520 of file [SpectralRadar_Properties.h](#).

6.26.3.5 **PolarizationRetarder** enum `PolarizationRetarder`

List of available polarization retarders in a polarization control unit.

Enumerator

<code>Retarder_Quality_Wave</code>	Build-in quarter-wave plate ($\lambda/4$) to control polarization state.
<code>Retarder_Half_Wave</code>	Build-in half-wave plate ($\lambda/2$) to control polarization state.

Definition at line 1050 of file [SpectralRadar_Types.h](#).

6.26.4 Function Documentation

6.26.4.1 **clearPolarizationProcessing()** void clearPolarizationProcessing (`PolarizationProcessingHandle Polarization`)

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

Parameters

<code>Polarization</code>	The <code>PolarizationProcessingHandle</code> that was initially provided by createPolarizationProcessing() or createPolarizationProcessingForFile() .
---------------------------	--

6.26.4.2 **createPolarizationProcessing()** `PolarizationProcessingHandle createPolarizationProcessing (void)`

Returns a Polarization processing handle to the Processing routines for polarization analysis.

Returns

A handle to the polarization processing routines.

6.26.4.3 **createPolarizationProcessingForFile()** `PolarizationProcessingHandle createPolarizationProcessingForFile (OCTFileHandle FileHandle)`

Loads metadata to the specified file. These metadata specify the operational arguments needed by the polarization processing routines to redo the polarization-analysis starting from two `ComplexDataHandle` delivered by `Proc_0` and `Proc_1`, exactly as they were done before the file was written.

Parameters

in	<i>FileHandle</i>	A valid (non null) handle of OCTFile (OCTFileHandle), previously obtained with the function createOCTFile .
----	-------------------	---

Returns

A valid (non null) polarization-processing handle to the processing routines for polarization analysis.

```
6.26.4.4 executePolarizationProcessing() void executePolarizationProcessing (
    PolarizationProcessingHandle Polarization,
    ComplexDataHandle Data_Camera1,
    ComplexDataHandle Data_Camera0 )
```

Executes the polarization processing of the input data and returns, if previously setup, intensity, retardation, and phase differences.

Parameters

<i>Data_Camera1</i>	Complex data obtained with executeProcessing starting from spectral data of the sensor "1".
<i>Data_Camera0</i>	Complex data obtained with executeProcessing starting from spectral data of the sensor "0".

If the sample does not alter the polarization state of the incident light (e.g. it is a mirror), the interference pattern will be captured by the first sensor (Camera-0), except for residual signals due to the non ideality of the optical components. If the setup had no QWPs, the opposite would happen: the interference pattern would've been captured by the second sensor (Camera-1). The setup has the QWPs at precise angles for the system to be equally sensitive to both components.

Parameters

<i>Polarization</i>	The PolarizationProcessingHandle that was initially provided by createPolarizationProcessing() or createPolarizationProcessingForFile() .
---------------------	---

```
6.26.4.5 getPolarizationFlag() BOOL getPolarizationFlag (
    PolarizationProcessingHandle Polarization,
    PolarizationFlag Flag )
```

Gets the desired polarization processing flag.

Parameters

<i>Polarization</i>	The PolarizationProcessingHandle that was initially provided by createPolarizationProcessing() or createPolarizationProcessingForFile() .
<i>Flag</i>	One of the options supported by PolarizationFlag .

Returns

True if the option has been activated.

6.26.4.6 `getPolarizationPropertyFloat()` `double getPolarizationPropertyFloat (`
`PolarizationProcessingHandle Polarization,`
`PolarizationPropertyFloat Property)`

Gets the desired polarization processing floating-point property.

Parameters

<i>Polarization</i>	The <code>PolarizationProcessingHandle</code> that was initially provided by <code>createPolarizationProcessing()</code> or <code>createPolarizationProcessingForFile()</code> .
<i>Property</i>	The floating point property (s. <code>PolarizationPropertyFloat</code>) whose value is inquired.

Returns

The value of the floating point property.

6.26.4.7 `getPolarizationPropertyInt()` `int getPolarizationPropertyInt (`
`PolarizationProcessingHandle Polarization,`
`PolarizationPropertyInt Property)`

Gets the desired polarization processing property.

Parameters

<i>Polarization</i>	The <code>PolarizationProcessingHandle</code> that was initially provided by <code>createPolarizationProcessing()</code> or <code>createPolarizationProcessingForFile()</code> .
<i>Property</i>	The integral property (s. <code>PolarizationPropertyInt</code>) whose value is inquired.

Returns

The value of the integral property.

6.26.4.8 `saveFileMetadataPolarization()` `void saveFileMetadataPolarization (`
`OCTFileHandle FileHandle,`
`PolarizationProcessingHandle PolProc)`

Saves metadata to the specified file. These metadata specify the operational arguments needed by the polarization processing routines to redo the polarization-analysis starting from two `ComplexDataHandle` delivered by `Proc_0` and `Proc_1`.

Parameters

in	<i>FileHandle</i>	A valid (non null) handle of OCTFile (OCTFileHandle), previously obtained with the function createOCTFile .
in	<i>PolProc</i>	A valid (non null) polarization-processing handle to the processing routines for polarization analysis.

```
6.26.4.9 setPolarizationFlag() void setPolarizationFlag (
    PolarizationProcessingHandle Polarization,
    PolarizationFlag Flag,
    BOOL OnOff )
```

Sets the polarization processing flags.

Parameters

<i>Polarization</i>	The PolarizationProcessingHandle that was initially provided by createPolarizationProcessing() or createPolarizationProcessingForFile() .
<i>Flag</i>	One of the options supported by PolarizationFlag .
<i>OnOff</i>	True if the option should be activated.

```
6.26.4.10 setPolarizationOutputDOPU() void setPolarizationOutputDOPU (
    PolarizationProcessingHandle Polarization,
    DataHandle DOPU )
```

Sets the location of the resulting DOPU (Degree Of Polarization Uniformity).

Parameters

<i>Polarization</i>	The PolarizationProcessingHandle that was initially provided by createPolarizationProcessing() or createPolarizationProcessingForFile() .
<i>DOPU</i>	A handle (DataHandle) previously obtained with createData (). It will hold the computed DOPU data upon return.

The range of the DOPU is [0,1]. It takes the value when light appears to be completely unpolarized, and 1 when the opposite is the case.

```
6.26.4.11 setPolarizationOutputI() void setPolarizationOutputI (
    PolarizationProcessingHandle Polarization,
    DataHandle Intensity )
```

Sets the location of the resulting polarization intensity output (Stokes parameter I).

Parameters

<i>Polarization</i>	The PolarizationProcessingHandle that was initially provided by createPolarizationProcessing() .
<i>Intensity</i>	A handle (DataHandle) previously obtained with createData (). It will hold the computed intensity data upon return.
<small>Generated by Doxygen</small>	

```
6.26.4.12 setPolarizationOutputOpticAxis() void setPolarizationOutputOpticAxis (
    PolarizationProcessingHandle Polarization,
    DataHandle OpticAxis )
```

Sets the location of the resulting optic axis.

Parameters

Polarization	The PolarizationProcessingHandle that was initially provided by createPolarizationProcessing() or createPolarizationProcessingForFile() .
OpticAxis	A handle (DataHandle) previously obtained with createData() . It will hold the computed optic-axis data upon return.

The range of the optic axil is [-pi/2,pi/2].

```
6.26.4.13 setPolarizationOutputQ() void setPolarizationOutputQ (
    PolarizationProcessingHandle Polarization,
    DataHandle StokesQ )
```

Sets the location of the resulting Stokes parameter Q.

Parameters

Polarization	The PolarizationProcessingHandle that was initially provided by createPolarizationProcessing() or createPolarizationProcessingForFile() .
StokesQ	A handle (DataHandle) previously obtained with createData() . It will hold the computed Stokes parm. Q data upon return.

The range of Q is [-1,1]. It takes the value -1 when the polarization is 100% parallel (zero degrees), and the value 1 when the polarization is 100% perpendicular (ninety degrees).

```
6.26.4.14 setPolarizationOutputRetardation() void setPolarizationOutputRetardation (
    PolarizationProcessingHandle Polarization,
    DataHandle Retardation )
```

Sets the location of the resulting retardation.

Parameters

Polarization	The PolarizationProcessingHandle that was initially provided by createPolarizationProcessing() .
Retardation	A handle (DataHandle) previously obtained with createData() . It will hold the computed retardation data upon return.

The range of the retardation is [0,pi/2].

```
6.26.4.15 setPolarizationOutputU() void setPolarizationOutputU (
```

```
PolarizationProcessingHandle Polarization,
DataHandle StokesU )
```

Sets the location of the resulting Stokes parameter U.

Parameters

<i>Polarization</i>	The PolarizationProcessingHandle that was initially provided by createPolarizationProcessing() or createPolarizationProcessingForFile() .
<i>StokesU</i>	A handle (DataHandle) previously obtained with createData() . It will hold the computed Stokes parm. U data upon return.

The range of U is [-1,1]. It takes the value -1 when the polarization is 100% at -45 degrees, and the value 1 when the polarization is 100% at 45 degrees.

6.26.4.16 setPolarizationOutputV() void setPolarizationOutputV (

```
PolarizationProcessingHandle Polarization,
DataHandle StokesV )
```

Sets the location of the resulting Stokes parameter U.

Parameters

<i>Polarization</i>	The PolarizationProcessingHandle that was initially provided by createPolarizationProcessing() or createPolarizationProcessingForFile() .
<i>StokesV</i>	A handle (DataHandle) previously obtained with createData() . It will hold the computed Stokes parm. V data upon return.

The range of V is [-1,1]. It takes the value -1 when the reflected light is 100% left-hand circularly polarized, and the value 1 when the reflected light is 100% right-hand circularly polarized.

6.26.4.17 setPolarizationPropertyFloat() void setPolarizationPropertyFloat (

```
PolarizationProcessingHandle Polarization,
PolarizationPropertyFloat Property,
double Value )
```

Sets the desired polarization processing floating-point property.

Parameters

<i>Polarization</i>	The PolarizationProcessingHandle that was initially provided by createPolarizationProcessing() or createPolarizationProcessingForFile() .
<i>Property</i>	The floating point property (s. PolarizationPropertyFloat) whose value should be set.
<i>Value</i>	The new value for the property.

6.26.4.18 setPolarizationPropertyInt() void setPolarizationPropertyInt (

```
PolarizationProcessingHandle Polarization,
```

```
PolarizationPropertyInt Property,  
int Value )
```

Sets polarization processing properties.

Parameters

<i>Polarization</i>	The PolarizationProcessingHandle that was initially provided by createPolarizationProcessing() or createPolarizationProcessingForFile() .
<i>Property</i>	The integral property (s. PolarizationPropertyInt) whose value should be set.
<i>Value</i>	The new value for the property.

6.27 Polarization Adjustment

Typedefs

- `typedef void(__stdcall * cbRetardationChanged) (PolarizationRetarder, double)`

Defines the function prototype for the polarization adjustment retardation callback (see also [setPolarizationAdjustmentRetardationChanged\(\)](#)). The argument contains the current (unitless) position (see also [setPolarizationAdjustmentRetardation\(\)](#)) of the specified [PolarizationRetarder](#).

Functions

- `SPECTRALRADAR_API BOOL isPolarizationAdjustmentAvailable (OCTDeviceHandle Dev)`
Returns whether or not a motorized polarization adjustment stage is available for the specified device.
- `SPECTRALRADAR_API void setPolarizationAdjustmentRetardationChangedCallback (OCTDeviceHandle Dev, cbRetardationChanged Callback)`
Registers the callback to get notified when the polarization adjustment retardation has changed.
- `SPECTRALRADAR_API void setPolarizationAdjustmentRetardation (OCTDeviceHandle Dev, PolarizationRetarder Retarder, double Retardation, WaitForCompletion Wait)`
Sets the retardation of the specified retarder in the polarization adjustment. The retardation is a unitless value between 0 and 1, which represents the full adjustment range of the retarder. The retarder may take some time to physically reach the new Retardation. Use the Wait parameter to choose if the function should block until the new position is reached.
- `SPECTRALRADAR_API double getPolarizationAdjustmentRetardation (OCTDeviceHandle Dev, PolarizationRetarder Retarder)`
Gets the current retardation of the specified retarder in the polarization adjustment. If [setPolarizationAdjustmentRetardation\(\)](#) was used in a non-blocking fashion, the function returns the current position of the retarder, not the final target position.

6.27.1 Detailed Description

6.27.2 TypeDef Documentation

6.27.2.1 `cbRetardationChanged` `typedef void(__stdcall* cbRetardationChanged) (PolarizationRetarder, double)`

Defines the function prototype for the polarization adjustment retardation callback (see also [setPolarizationAdjustmentRetardationChanged\(\)](#)). The argument contains the current (unitless) position (see also [setPolarizationAdjustmentRetardation\(\)](#)) of the specified [PolarizationRetarder](#).

Definition at line 1089 of file [SpectralRadar_Types.h](#).

6.27.3 Function Documentation

6.27.3.1 `getPolarizationAdjustmentRetardation()` `double getPolarizationAdjustmentRetardation (OCTDeviceHandle Dev, PolarizationRetarder Retarder)`

Gets the current retardation of the specified retarder in the polarization adjustment. If [setPolarizationAdjustmentRetardation\(\)](#) was used in a non-blocking fashion, the function returns the current position of the retarder, not the final target position.

Parameters

<i>Dev</i>	the OCTDeviceHandle that was initially provided by initDevice.
<i>Retarder</i>	the PolarizationRetarder which shall be queried

Returns

The current unitless Retardation of the selected Retarder ($0 \leq \text{Retardation} \leq 1$)

6.27.3.2 isPolarizationAdjustmentAvailable() [SPECTRALRADAR_API](#) [BOOL](#) isPolarizationAdjustmentAvailable ([OCTDeviceHandle](#) Dev)

Returns whether or not a motorized polarization adjustment stage is available for the specified device.

Parameters

<i>Dev</i>	the OCTDeviceHandle that was initially provided by initDevice.
------------	--

Returns

true if a polarization adjustment is available

6.27.3.3 setPolarizationAdjustmentRetardation() double setPolarizationAdjustmentRetardation ([OCTDeviceHandle](#) Dev, [PolarizationRetarder](#) Retarder, double Retardation, [WaitForCompletion](#) Wait)

Sets the retardation of the specified retarder in the polarization adjustment. The retardation is a unitless value between 0 and 1, which represents the full adjustment range of the retarder. The retarder may take some time to physically reach the new Retardation. Use the Wait parameter to choose if the function should block until the new position is reached.

Parameters

<i>Dev</i>	the OCTDeviceHandle that was initially provided by initDevice.
<i>Retarder</i>	the PolarizationRetarder which shall be adjusted
<i>Retardation</i>	the new retardation value ($0 \leq \text{retardation} \leq 1$)
<i>Wait</i>	specify WaitForCompletion Wait to block until the new Retardation value has been reached

6.27.3.4 setPolarizationAdjustmentRetardationChangedCallback() void setPolarizationAdjustmentRetardationChangedCallback (

```
OCTDeviceHandle Dev,  
cbRetardationChanged Callback )
```

Registers the callback to get notified when the polarization adjustment retardation has changed.

Parameters

<i>Dev</i>	the OCTDeviceHandle that was initially provided by initDevice.
<i>Callback</i>	the Callback to register.

6.28 Reference Intensity Control

Typedefs

- `typedef void(__stdcall * cbReferenceIntensityControlValueChanged) (double)`

*Defines the function prototype for the reference intensity control status callback (see also [setReferenceIntensityControlCallback\(\)](#)).
The argument contains the current (unitless) intensity between 0 and 1 (see also [setReferenceIntensityControlValue\(\)](#)).*

Functions

- **SPECTRALRADAR_API** `BOOL isReferenceIntensityControlAvailable (OCTDeviceHandle Dev)`
Returns whether or not an automated reference intensity control is available for the specified device.
- **SPECTRALRADAR_API** `void setReferenceIntensityControlCallback (OCTDeviceHandle Dev, cbReferenceIntensityControlValue Callback)`
Registers the callback to get notified when the reference intensity has changed.
- **SPECTRALRADAR_API** `void setReferenceIntensityControlValue (OCTDeviceHandle Dev, double ReferenceIntensity, WaitForCompletion Wait)`
Sets the reference intensity of the specified device. The intensity is a unitless value between 0 and 1, which represents the full adjustment range of the reference intensity control, but may or may not be linear. The control may take some time to physically reach the new intensity. Use the Wait parameter to choose if the function should block until the new intensity is reached.
- **SPECTRALRADAR_API** `double getReferenceIntensityControlValue (OCTDeviceHandle Dev)`
Gets the current reference intensity of the specified device. If `setReferenceIntensityControlValue` was used in a non-blocking fashion, the function returns the current value of the control, not the final target value.

6.28.1 Detailed Description

6.28.2 Typedef Documentation

6.28.2.1 `cbReferenceIntensityControlValueChanged` `typedef void(__stdcall* cbReferenceIntensityControlValueChanged) (double)`

Defines the function prototype for the reference intensity control status callback (see also [setReferenceIntensityControlCallback\(\)](#)).
The argument contains the current (unitless) intensity between 0 and 1 (see also [setReferenceIntensityControlValue\(\)](#)).

Definition at line 1093 of file [SpectralRadar_Types.h](#).

6.28.3 Function Documentation

6.28.3.1 `getReferenceIntensityControlValue()` `double getReferenceIntensityControlValue (OCTDeviceHandle Dev)`

Gets the current reference intensity of the specified device. If `setReferenceIntensityControlValue` was used in a non-blocking fashion, the function returns the current value of the control, not the final target value.

Parameters

<i>Dev</i>	the OCTDeviceHandle that was initially provided by initDevice.
------------	--

Returns

The current unitless reference intensity of the selected device ($0 \leq \text{ReferenceIntensity} \leq 1$)

6.28.3.2 isReferenceIntensityControlAvailable() [SPECTRALRADAR_API](#) [BOOL](#) [isReferenceIntensityControlAvailable](#)

```
OCTDeviceHandle Dev )
```

Returns whether or not an automated reference intensity control is available for the specified device.

Parameters

<i>Dev</i>	the OCTDeviceHandle that was initially provided by initDevice.
------------	--

Returns

true if a reference intensity control is available

6.28.3.3 setReferenceIntensityControlCallback() [void](#) [setReferenceIntensityControlCallback](#)

```
( OCTDeviceHandle Dev,
    cbReferenceIntensityControlValueChanged Callback )
```

Registers the callback to get notified when the reference intensity has changed.

Parameters

<i>Dev</i>	the OCTDeviceHandle that was initially provided by initDevice.
<i>Callback</i>	the Callback to register.

6.28.3.4 setReferenceIntensityControlValue() [void](#) [setReferenceIntensityControlValue](#)

```
( OCTDeviceHandle Dev,
    double ReferenceIntensity,
    WaitForCompletion Wait )
```

Sets the reference intensity of the specified device. The intensity is a unitless value between 0 and 1, which represents the full adjustment range of the reference intensity control, but may or may not be linear. The control may take some time to physically reach the new intensity. Use the Wait parameter to choose if the function should block until the new intensity is reached.

Parameters

<i>Dev</i>	the OCTDeviceHandle that was initially provided by initDevice.
<i>ReferenceIntensity</i>	the new reference intensity value ($0 \leq \text{ReferenceIntensity} \leq 1$)
<i>Wait</i>	specify WaitForCompletion Wait to block until the new intensity value has been reached

6.29 Amplification Control

Functions

- **SPECTRALRADAR_API** `BOOL isAmplificationControlAvailable (OCTDeviceHandle Dev)`
Returns whether or not the sampling amplification of specified device can be adjusted.
- **SPECTRALRADAR_API** `int getAmplificationControlNumberOfSteps (OCTDeviceHandle Dev)`
Gets the number of discrete amplification control steps available on the specified device. Please note that the largest amplification step is `getAmplificationControlNumberOfSteps()` - 1.
- **SPECTRALRADAR_API** `void setAmplificationControlStep (OCTDeviceHandle Dev, int Step)`
Sets the sampling amplification on the the specified device. The lowest amplification is always 0. In general, the amplification should be set as high as possible without going into saturation.
- **SPECTRALRADAR_API** `int getAmplificationControlStep (OCTDeviceHandle Dev)`
Gets the current sampling amplification of the specified device.

6.29.1 Detailed Description

6.29.2 Function Documentation

6.29.2.1 `getAmplificationControlNumberOfSteps()` `int getAmplificationControlNumberOfSteps (OCTDeviceHandle Dev)`

Gets the number of discrete amplification control steps available on the specified device. Please note that the largest amplification step is `getAmplificationControlNumberOfSteps()` - 1.

Parameters

<code>Dev</code>	the <code>OCTDeviceHandle</code> that was initially provided by <code>initDevice</code> .
------------------	---

Returns

The number of amplification steps.

6.29.2.2 `getAmplificationControlStep()` `int getAmplificationControlStep (OCTDeviceHandle Dev)`

Gets the current sampling amplification of the specified device.

Parameters

<code>Dev</code>	the <code>OCTDeviceHandle</code> that was initially provided by <code>initDevice</code> .
------------------	---

Returns

The current amplification step of the selected device ($0 \leq \text{Step} \leq \text{getAmplificationControlNumberOfSteps}()$)

6.29.2.3 isAmplificationControlAvailable() `SPECTRALRADAR_API BOOL isAmplificationControlAvailable(OCTDeviceHandle Dev)`

Returns whether or not the sampling amplification of specified device can be adjusted.

Parameters

<i>Dev</i>	the OCTDeviceHandle that was initially provided by initDevice.
------------	--

Returns

true if an amplification control is available

6.29.2.4 setAmplificationControlStep() `void setAmplificationControlStep(OCTDeviceHandle Dev, int Step)`

Sets the sampling amplification on the the specified device. The lowest amplification is always 0. In general, the amplification should be set as high as possible without going into saturation.

Parameters

<i>Dev</i>	the OCTDeviceHandle that was initially provided by initDevice.
<i>Step</i>	Which amplification step to use. $0 \leq \text{AmplificationStep} < \text{getAmplificationControlNumberOfSteps}()$

6.30 General Information

Functions

- **SPECTRALRADAR_API** void `getSoftwareVersion` (char **Version*, int *Length*)

Returns the current software version.

6.30.1 Detailed Description

6.30.2 Function Documentation

6.30.2.1 `getSoftwareVersion()` void `getSoftwareVersion` (
 char * *Version*,
 int *Length*)

Returns the current software version.

Parameters

out	<i>Version</i>	The current software version returned from the function.
in	<i>Length</i>	Length of the user-provided buffer.

7 Data Structure Documentation

7.1 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.

7.1.1 Detailed Description

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

This data structure is an ANSI C equivalent to the C++ data type
`std::complex<float>`

. Notice that arrays of complex data are always in interleaved format (real and imaginary parts of each element in contiguous memory addresses) and not in split format, where real and imaginary parts are stored in separate arrays.

Definition at line 37 of file [SpectralRadar_Types.h](#).

7.1.2 Field Documentation

7.1.2.1 `data float data[2]`

`data[0]` is the real part and `data[1]` is the imaginary part.

Definition at line 39 of file [SpectralRadar_Types.h](#).

8 File Documentation

8.1 SpectralRadar.h File Reference

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

Macros

- `#define SPECTRALRADAR_API`
Export/Import of define of DLL members.

Functions

- **SPECTRALRADAR_API void setErrorsPerThreadFlag (bool getErrorsPerThread)**
Sets the whether or not errors are returned per thread in `isError` and `getError`. By default, errors are collected globally and a call to `getError` will return the last error from any thread. If this flag is set to true, only errors from the current thread will be returned.
- **SPECTRALRADAR_API ErrorCode isError (void)**
Returns error code. The error flag will not be cleared; a following call to `getError` thus provides detailed error information.
- **SPECTRALRADAR_API ErrorCode getError (char *Message, int StringSize)**
Returns an error code and a message if an error occurred. The error flag will be cleared.
- **SPECTRALRADAR_API void setLog (LogOutputType Type, const char *Filename)**
Specifies where to write text output by the SDK. The respective text output might help to debug applications or identify errors and faults.
- **SPECTRALRADAR_API int getDataPropertyInt (DataHandle Data, DataPropertyInt Selection)**
Returns the selected integer property of the specified data.
- **SPECTRALRADAR_API double getDataPropertyFloat (DataHandle Data, DataPropertyFloat Selection)**
Returns the selected floating point property of the specified data.
- **SPECTRALRADAR_API void copyData (DataHandle DataSource, DataHandle DataDestination)**
Copies the content of the specified source to the specified destination.
- **SPECTRALRADAR_API void copyDialogContent (DataHandle DataSource, float *Destination)**
Copies the data in the specified data object (`DataHandle`) into the specified pointer.
- **SPECTRALRADAR_API float * getDataPtr (DataHandle Data)**
The returned pointer points to memory owned by `SpectralRadar.dll`. The user should not attempt to free it.
- **SPECTRALRADAR_API void reserveData (DataHandle Data, int Size1, int Size2, int Size3)**
Reserves the amount of data specified. This might improve performance if appending data to the `DataHandle` as no additional memory needs to be reserved then.
- **SPECTRALRADAR_API void resizeData (DataHandle Data, int Size1, int Size2, int Size3)**
Resizes the respective data object. In general the data will be 1-dimensional if `Size2` and `Size3` are equal to 1, 2-dimensional if `Size3` is equal to 1 or 3-dimensional if all, `Size1`, `Size2`, `Size3`, are unequal to 1.
- **SPECTRALRADAR_API void setDataRange (DataHandle Data, double range1, double range2, double range3)**
Sets the range in mm in the 3 axes represented in the `RealData` buffer.
- **SPECTRALRADAR_API void setDataContent (DataHandle Data, float *NewContent)**
*Sets the data content of the data object. The data chunk pointed to by `NewContent` needs to be of the size expected by the data object, i. e. `Size1*Size2*Size3*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 orientation of the data object to the given orientation.
- **SPECTRALRADAR_API int getComplexDataPropertyInt (ComplexDataHandle Data, DataPropertyInt Selection)**
Returns the selected integer property of the specified data.
- **SPECTRALRADAR_API double getComplexDataPropertyFloat (ComplexDataHandle Data, DataPropertyFloat Selection)**
Returns the selected floating-point property of the specified data.
- **SPECTRALRADAR_API void copyComplexDialogContent (ComplexDataHandle DataSource, ComplexFloat *Destination)**
Copies the content of the complex data to the pointer specified as destination.
- **SPECTRALRADAR_API void copyComplexData (ComplexDataHandle DataSource, ComplexDataHandle DataDestination)**
Copies the contents of the specified `ComplexDataHandle` to the specified destination `ComplexDataHandle`.

- **SPECTRALRADAR_API** `ComplexFloat * getComplexDataPtr (ComplexDataHandle Data)`
The returned pointer points to memory owned by SpectralRadar.dll. The user should not attempt to free it.
- **SPECTRALRADAR_API** `void setComplexDataContent (ComplexDataHandle Data, ComplexFloat *NewContent)`
Sets the data content of the ComplexDataHandle to the content specified by the pointer.
- **SPECTRALRADAR_API** `void reserveComplexData (ComplexDataHandle Data, int Size1, int Size2, int Size3)`
Reserves the amount of data specified. This might improve performance if appending data to the ComplexDataHandle as no additional memory needs to be reserved then.
- **SPECTRALRADAR_API** `void resizeComplexData (ComplexDataHandle Data, int Size1, int Size2, int Size3)`
Resizes the respective data object. In general the data will be 1-dimensional if Size2 and Size3 are equal to 1, 2-dimensional if Size3 is equal to 1 dn 3-dimensional if all, Size1, Size2, Size3, are unequal to 1.
- **SPECTRALRADAR_API** `void setComplexDataRange (ComplexDataHandle Data, double range1, double range2, double range3)`
Sets the range in mm in the 3 axes represented in the RealData buffer.
- **SPECTRALRADAR_API** `int getColoredDataPropertyInt (ColoredDataHandle ColData, DataPropertyInt Selection)`
Returns the selected integer property of the specified colored data.
- **SPECTRALRADAR_API** `double getColoredDataPropertyFloat (ColoredDataHandle ColData, DataPropertyFloat Selection)`
Returns the selected integer property of the specified colored data.
- **SPECTRALRADAR_API** `void copyColoredData (ColoredDataHandle ImageSource, ColoredDataHandle ImageDestination)`
Copies the contents of the specified ColoredDataHandle to the specified destination ColoredDataHandle.
- **SPECTRALRADAR_API** `void copyColoredDataContent (ColoredDataHandle Source, unsigned long *Destination)`
Copies the data in the specified colored data object (ColoredDataHandle) into the specified pointer.
- **SPECTRALRADAR_API** `void copyColoredDataContentAligned (ColoredDataHandle ImageSource, unsigned long *Destination, int Stride)`
Copies the data in the specified colored data object (ColoredDataHandle) into the specified pointer.
- **SPECTRALRADAR_API** `unsigned long * getColoredDataPtr (ColoredDataHandle ColData)`
The returned pointer points to memory owned by SpectralRadar.dll. The user should not attempt to free it.
- **SPECTRALRADAR_API** `void resizeColoredData (ColoredDataHandle ColData, int Size1, int Size2, int Size3)`
Resizes the respective colored data object. In general the data will be 1-dimensional if Size2 and Size3 are equal to 1, 2-dimensional if Size3 is equal to 1 dn 3-dimensional if all, Size1, Size2, Size3, are unequal to 1.
- **SPECTRALRADAR_API** `void reserveColoredData (ColoredDataHandle ColData, int Size1, int Size2, int Size3)`
Reserves the amount of colored data specified. This might improve performance if appending data to the ColoredDataHandle as no additional memory needs to be reserved then.
- **SPECTRALRADAR_API** `void setColoredDataContent (ColoredDataHandle ColData, unsigned long *NewContent)`
Sets the data content of the colored data object. The data chunk pointed to by NewContent needs to be of the size expected by the data object, i. e. $Size1 \times Size2 \times Size3 \times \text{sizeof}(unsigned\ long)$.
- **SPECTRALRADAR_API** `void setColoredDataRange (ColoredDataHandle Data, double range1, double range2, double range3)`
Sets the range in mm in the 3 axes represented in the data object buffer.
- **SPECTRALRADAR_API** `DataOrientation getColoredDataOrientation (ColoredDataHandle Data)`
Returns the data orientation of the colored data object.
- **SPECTRALRADAR_API** `void setColoredDataOrientation (ColoredDataHandle Data, DataOrientation Orientation)`
Sets the data orientation of the colored data object to the given orientation.
- **SPECTRALRADAR_API** `void copyRawDataContent (RawDataHandle RawDataSource, void *DataContent)`
Copies the content of the raw data into the specified buffer.

- **SPECTRALRADAR_API** void `copyRawData (RawDataHandle RawDataSource, RawDataHandle RawData ← Target)`
Copies raw data content and metadata into the specified target handle.
- **SPECTRALRADAR_API** void * `getRawDataPtr (RawDataHandle RawDataSource)`
Notice that raw data refers to the spectra as acquired, without processing of any kind.
- **SPECTRALRADAR_API** int `getRawDataPropertyInt (RawDataHandle RawData, RawDataPropertyInt Property)`
Notice that raw data refers to the spectra as acquired, without processing of any kind.
- **SPECTRALRADAR_API** double `getRawDataPropertyFloat (RawDataHandle Data, RawDataPropertyFloat 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 `reserveRawData (RawDataHandle Raw, int Size1, int Size2, int Size3)`
Reserves the amount of data specified. This might improve performance if appending data to the `RawDataHandle` as no additional memory needs to be reserved then.
- **SPECTRALRADAR_API** void `resizeRawData (RawDataHandle Raw, int Size1, int Size2, int Size3)`
Resizes the specified raw data buffer accordingly.
- **SPECTRALRADAR_API** void `setRawDataContent (RawDataHandle RawData, void *NewContent)`
Sets the content of the raw data buffer. The size of the `RawDataHandle` needs to be adjusted first, as otherwise not all data might be copied.
- **SPECTRALRADAR_API** void `setScanSpectra (RawDataHandle RawData, int NumberOfScanRegions, int *ScanRegions)`
Notice that raw data refers to the spectra as acquired, without processing of any kind.
- **SPECTRALRADAR_API** void `setApodizationSpectra (RawDataHandle RawData, int NumberOfApoRegions, int *ApodizationRegions)`
Notice that raw data refers to the spectra as acquired, without processing of any kind.
- **SPECTRALRADAR_API** int `getNumberOfScanRegions (RawDataHandle Raw)`
Returns the number of regions that have been acquired that contain scan data, i. e. spectra that are used to compute A-scans.
- **SPECTRALRADAR_API** int `getNumberOfApodizationRegions (RawDataHandle Raw)`
Returns the number of regions in the raw data containing spectra that are supposed to be used for apodization.
- **SPECTRALRADAR_API** void `getScanSpectra (RawDataHandle Raw, int *SpectralIndex)`
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 *SpectralIndex)`
Returns the indices of spectra that contain apodization data, i. e. spectra that are supposed to be used as input for apodization.
- **SPECTRALRADAR_API** RawDataHandle `createRawData (void)`
Notice that raw data refers to the spectra as acquired, without processing of any kind.
- **SPECTRALRADAR_API** void `clearRawData (RawDataHandle Raw)`
Notice that raw data refers to the spectra as acquired, without processing of any kind.
- **SPECTRALRADAR_API** DataHandle `createData (void)`
Creates a 1-dimensional data object, containing floating point data.
- **SPECTRALRADAR_API** DataHandle `createGradientData (int Size)`
Creates a 1-dimensional data object, containing floating point data with equidistant arranged values between [0, size-1] with distance 1/(size-1).
- **SPECTRALRADAR_API** void `clearData (DataHandle Data)`
Clears the specified `DataHandle` object.
- **SPECTRALRADAR_API** ColoredDataHandle `createColoredData (void)`
Creates a colored data object (`ColoredDataHandle`).
- **SPECTRALRADAR_API** void `clearColoredData (ColoredDataHandle Volume)`

- Clears a colored volume object.
• **SPECTRALRADAR_API** ComplexDataHandle createComplexData (void)
Creates a data object holding complex data (*ComplexDataHandle*).
• **SPECTRALRADAR_API** void clearComplexData (ComplexDataHandle Data)
Clears a data object holding complex data (*ComplexDataHandle*).
• **SPECTRALRADAR_API** OCTDeviceHandle initDevice (void)
Initializes the installed device.
• **SPECTRALRADAR_API** int getDevicePropertyInt (OCTDeviceHandle Dev, DevicePropertyInt Selection)
Returns properties of the device belonging to the specified *OCTDeviceHandle*.
• **SPECTRALRADAR_API** const char * getDevicePropertyString (OCTDeviceHandle Dev, DevicePropertyString Selection)
Returns properties of the device belonging to the specified *OCTDeviceHandle*.
• **SPECTRALRADAR_API** double getDevicePropertyFloat (OCTDeviceHandle Dev, DevicePropertyFloat Selection)
Returns properties of the device belonging to the specified *OCTDeviceHandle*.
• **SPECTRALRADAR_API** BOOL getDeviceFlag (OCTDeviceHandle Dev, DeviceFlag Selection)
Returns properties of the device belonging to the specified *OCTDeviceHandle*.
• **SPECTRALRADAR_API** void setDeviceFlag (OCTDeviceHandle Dev, DeviceFlag Selection, BOOL Value)
Sets the selected flag of the device belonging to the specified *OCTDeviceHandle*.
• **SPECTRALRADAR_API** BOOL getStaticDeviceFlag (StaticDeviceFlag Selection)
Returns properties of the device available before device initialization.
• **SPECTRALRADAR_API** void setStaticDeviceFlag (StaticDeviceFlag Selection, BOOL Value)
Sets the selected flag of the device available before device initialization.
• **SPECTRALRADAR_API** void closeDevice (OCTDeviceHandle Dev)
Closes the device opened previously with *initDevice*.
• **SPECTRALRADAR_API** void moveScanner (OCTDeviceHandle Dev, ProbeHandle Probe, ScanAxis Axis, double Position_mm)
Manually moves the scanner to a given position.
• **SPECTRALRADAR_API** void moveScannerToApoPosition (OCTDeviceHandle Dev, ProbeHandle Probe)
Moves the scanner to the apodization position.
• **SPECTRALRADAR_API** int getNumberOfDevicePresetCategories (OCTDeviceHandle Dev)
If the hardware supports multiple presets, the function returns the number of categories in which presets can be set.
• **SPECTRALRADAR_API** const char * getDevicePresetCategoryName (OCTDeviceHandle Dev, int Category)
Gets a descriptor/name for the respective preset category.
• **SPECTRALRADAR_API** int getDevicePresetCategoryIndex (OCTDeviceHandle Dev, const char *Name)
Gets the index of a preset category from the name of the category.
• **SPECTRALRADAR_API** void setDevicePreset (OCTDeviceHandle Dev, int Category, ProbeHandle Probe, ProcessingHandle Proc, int Preset)
Sets the preset of the device. Using presets the sensitivity and acquisition speed of the device can be influenced.
• **SPECTRALRADAR_API** int getDevicePreset (OCTDeviceHandle Dev, int Category)
Gets the currently used device preset.
• **SPECTRALRADAR_API** const char * getDevicePresetDescription (OCTDeviceHandle Dev, int Category, int Preset)
Returns a description of the selected device preset. Using the description more information about sensitivity and acquisition speed of the respective set can be found.
• **SPECTRALRADAR_API** int getNumberOfDevicePresets (OCTDeviceHandle Dev, int Category)
Returns the number of available device presets.
• **SPECTRALRADAR_API** void setRequiredSLDOnTime_s (int Time_s)
Sets the time the SLD needs to be switched on before any measurement can be started. Default is 3 seconds.
• **SPECTRALRADAR_API** void resetCamera (void)
Resets the spectrometer camera.
• **SPECTRALRADAR_API** BOOL isDeviceAvailable (void)

- **SPECTRALRADAR_API** DeviceState `getDeviceState` (void)
Returns the state of supported base-unit.
- **SPECTRALRADAR_API** void `getDeviceError` (char *ErrorMsg, int StringSize)
Returns the error message given by the hardware.
- **SPECTRALRADAR_API** int `getNumberOfInternalDeviceValues` (OCTDeviceHandle Dev)
Returns the number of Analog-to-Digital Converter present in the device.
- **SPECTRALRADAR_API** void `getInternalDeviceValueName` (OCTDeviceHandle Dev, int Index, char *Name, int NameStringSize, char *Unit, int UnitStringSize)
Returns names and unit for the specified Analog-to-Digital Converter.
- **SPECTRALRADAR_API** double `getInternalDeviceValueByName` (OCTDeviceHandle Dev, const char *Name)
Returns the value of the specified Analog-to-Digital Converter (ADC):
- **SPECTRALRADAR_API** double `getInternalDeviceValueByIndex` (OCTDeviceHandle Dev, int Index)
Returns the value of the selected ADC.
- **SPECTRALRADAR_API** int `getNumberOfOutputDeviceValues` (OCTDeviceHandle Dev)
Returns the number of output values.
- **SPECTRALRADAR_API** void `getOutputDeviceValueName` (OCTDeviceHandle Dev, int Index, char *Name, int NameStringSize, char *Unit, int UnitStringSize)
Returns names and units of the requested output values.
- **SPECTRALRADAR_API** BOOL `doesOutputDeviceValueExist` (OCTDeviceHandle Dev, const char *Name)
Returns whether the requested output device values exists or not.
- **SPECTRALRADAR_API** void `setOutputDeviceValueByName` (OCTDeviceHandle Dev, const char *Name, double value)
Sets the specified output value.
- **SPECTRALRADAR_API** void `setInternalDeviceValueByIndex` (OCTDeviceHandle Dev, int Index, double Value)
Sets the value of the selected ADC.
- **SPECTRALRADAR_API** void `setOutputDeviceValueByIndex` (OCTDeviceHandle Dev, int Index, double Value)
Sets the value of the selected ADC.
- **SPECTRALRADAR_API** void `getOutputDeviceValueRangeByName` (OCTDeviceHandle Dev, const char *Name, double *Min, double *Max)
Gives the range of the specified output value.
- **SPECTRALRADAR_API** void `getOutputDeviceValueRangeByIndex` (OCTDeviceHandle Dev, int Index, double *Min, double *Max)
Gives the range of the specified output value.
- **SPECTRALRADAR_API** ProbeHandle `initCurrentProbe` (OCTDeviceHandle Dev)
Initializes the probe that is currently selected in the GUI. If no probe is configured, an error is raised.
- **SPECTRALRADAR_API** ProbeHandle `initProbe` (OCTDeviceHandle Dev, const char *ProbeFile)
Initializes a probe specified by ProbeFile.
- **SPECTRALRADAR_API** ProbeHandle `initDefaultProbe` (OCTDeviceHandle Dev, const char *Type, const char *Objective)
Creates a standard probe using standard parameters for the specified probe type.
- **SPECTRALRADAR_API** ProbeHandle `initProbeFromOCTFile` (OCTDeviceHandle Dev, OCTFileHandle File)
Creates a probe using the parameters from the specified OCT file.
- **SPECTRALRADAR_API** void `saveProbe` (ProbeHandle Probe, const char *ProbeFile)
Saves the current properties of the `ProbeHandle` to a specified INI file to be reloaded using the `initProbe()` function.
- **SPECTRALRADAR_API** void `setProbeParameterInt` (ProbeHandle Probe, ProbeParameterInt Selection, int Value)
Sets integer parameter of the specified probe.

- **SPECTRALRADAR_API** void `setProbeParameterFloat` (`ProbeHandle` Probe, `ProbeParameterFloat` Selection, double Value)
Sets floating point parameters of the specified probe.
- **SPECTRALRADAR_API** int `getProbeParameterInt` (`ProbeHandle` Probe, `ProbeParameterInt` Selection)
Gets integer parameters of the specified probe.
- **SPECTRALRADAR_API** double `getProbeParameterFloat` (`ProbeHandle` Probe, `ProbeParameterFloat` Selection)
Gets floating point parameters of the specified probe.
- **SPECTRALRADAR_API** BOOL `getProbeFlag` (`ProbeHandle` Probe, `ProbeFlag` Selection)
Returns the selected boolean value of the specified probe.
- **SPECTRALRADAR_API** void `setProbeParameterString` (`ProbeHandle` Probe, `ProbeParameterString` Selection, const char *Value)
Sets a string property of the specified probe.
- **SPECTRALRADAR_API** const char * `getProbeParameterString` (`ProbeHandle` Probe, `ProbeParameterString` Selection)
Gets the desired string property of the specified probe.
- **SPECTRALRADAR_API** void `positionToProbeVoltage` (`OCTDeviceHandle` Handle, `ProbeHandle` Probe, double Position_X_mm, double Position_Y_mm, double *Volt_X, double *Volt_Y)
Convert scan position in mm to probe output voltage.
- **SPECTRALRADAR_API** const char * `getProbeType` (`ProbeHandle` Probe)
Gets the type of the specified probe.
- **SPECTRALRADAR_API** void `setProbeType` (`ProbeHandle` Probe, const char *Type)
Sets the type of the specified probe.
- **SPECTRALRADAR_API** void `closeProbe` (`ProbeHandle` Probe)
Closes the probe and frees all memory associated with it.
- **SPECTRALRADAR_API** void `CameraPixelToPosition` (`ProbeHandle` Probe, `ColoredDataHandle` Image, int PixelX, int PixelY, double *PosX, double *PosY)
Computes the physical position of a camera pixel of the video camera in the probe. It assumes a properly calibrated device.
- **SPECTRALRADAR_API** void `PositionToCameraPixel` (`ProbeHandle` Probe, `ColoredDataHandle` Image, double PosX, double PosY, int *PixelX, int *PixelY)
Computes the pixel of the video camera corresponding to a physical position. It needs to be assured that the device is properly calibrated.
- **SPECTRALRADAR_API** void `visualizeScanPatternOnDevice` (`OCTDeviceHandle` Dev, `ProbeHandle` Probe, `ScanPatternHandle` Pattern, BOOL ShowRawPattern)
Visualizes the scan pattern on top of the camera image; if appropriate hardware is used for visualization.
- **SPECTRALRADAR_API** void `visualizeScanPatternOnImage` (`ProbeHandle` Probe, `ScanPatternHandle` ScanPattern, `ColoredDataHandle` VideoImage)
Visualizes the scan pattern on top of the camera image; scan pattern data is written into the image.
- **SPECTRALRADAR_API** `ScanPatternHandle` `createNoScanPattern` (`ProbeHandle` Probe, int AScans, int NumberOfScans)
Creates a simple scan pattern that does not move the galvo. Use this pattern for point scans and/or non-scanning probes. The pattern will however use a specified amount of trigger signals. For continuous acquisition use NumberOfScans set to 1.
- **SPECTRALRADAR_API** `ScanPatternHandle` `createAScanPattern` (`ProbeHandle` Probe, int AScans, double PosX_mm, double PosY_mm)
Creates a scan pattern used to acquire a specific amount of Ascans at a specific position.
- **SPECTRALRADAR_API** `ScanPatternHandle` `createBScanPattern` (`ProbeHandle` Probe, double Range_mm, int AScans)
Creates a horizontal rectilinear-segment B-scan pattern that moves the galvo over a specified range.
- **SPECTRALRADAR_API** `ScanPatternHandle` `createBScanPatternManual` (`ProbeHandle` Probe, double StartX_mm, double StartY_mm, double StopX_mm, double StopY_mm, int AScans)
Creates a B-scan pattern specified by start and end points.

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

Creates an ideal B-scan pattern assuming scanners with infinite speed. No correction factors are taken into account. This is only used for internal purposes and not as a scan pattern designed to be output to the galvo drivers.
- **SPECTRALRADAR_API ScanPatternHandle createCirclePattern (ProbeHandle Probe, double Radius_mm, int AScans)**

Creates a circle scan pattern.
- **SPECTRALRADAR_API ScanPatternHandle createVolumePattern (ProbeHandle Probe, double RangeX_mm, int SizeX, double RangeY_mm, int SizeY, ScanPatternApodizationType ApoType, ScanPatternAcquisitionOrder AcqOrder)**

Creates a simple volume pattern.
- **SPECTRALRADAR_API ScanPatternHandle createVolumePatternEx (ProbeHandle Probe, double RangeX_mm, int SizeX, double RangeY_mm, int SizeY, double CenterX_mm, double CenterY_mm, double Angle_rad, ScanPatternApodizationType ApoType, ScanPatternAcquisitionOrder AcqOrder)**

Creates a simple volume pattern.
- **SPECTRALRADAR_API void updateScanPattern (ScanPatternHandle Pattern)**

Updates the specified pattern (ScanPatternHandle) and computes the full look-up-table.
- **SPECTRALRADAR_API void rotateScanPattern (ScanPatternHandle Pattern, double Angle_rad)**

Rotates the specified pattern (ScanPatternHandle), counter-clockwise. The rotation is relative to current angle, not to the horizontal. That is, after multiple invocations of this function the final rotation is the addition of all rotations.
- **SPECTRALRADAR_API void rotateScanPatternEx (ScanPatternHandle Pattern, double Angle_rad, int Index)**

Counter-clockwise rotates the scan Index (0-based, i.e. zero for the first, one for the second, and so on) of the specified volume scan pattern (ScanPatternHandle). The rotation is relative to current angle, not to the horizontal. That is, after multiple invocations of this function the final rotation is the addition of all rotations.
- **SPECTRALRADAR_API void shiftScanPattern (ScanPatternHandle Pattern, double ShiftX_mm, double ShiftY_mm)**

Shifts the specified pattern (ScanPatternHandle). The shift is relative to current position, not to (0,0). That is, after multiple invocations of this function the final shift is the addition of all shifts.
- **SPECTRALRADAR_API void shiftScanPatternEx (ScanPatternHandle Pattern, double ShiftX_mm, double ShiftY_mm, BOOL ShiftApo, int Index)**

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

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

Returns the number of points in the specified scan pattern (ScanPatternHandle), including apodization and flyback.
- **SPECTRALRADAR_API void getScanPatternLUT (ScanPatternHandle Pattern, double *VoltX, double *VoltY)**

Returns the voltages that will be applied to reach the positions to be scanned, in the specified scan pattern (ScanPatternHandle).
- **SPECTRALRADAR_API int getScanPointsSize (ScanPatternHandle Pattern)**

Returns the number of points in the specified scan pattern (ScanPatternHandle), including apodization and flyback.
- **SPECTRALRADAR_API void getScanPoints (ScanPatternHandle Pattern, double *PosX_mm, double *PosY_mm)**

Returns the position coordinates (in mm) of the points that in the specified scan pattern (ScanPatternHandle).
- **SPECTRALRADAR_API void clearScanPattern (ScanPatternHandle Pattern)**

Clears the specified scan pattern (ScanPatternHandle).
- **SPECTRALRADAR_API ScanPatternHandle createFreeformScanPattern2D (ProbeHandle Probe, double *PosX_mm, double *PosY_mm, int Size, int AScans, InterpolationMethod InterpolationMethod, BOOL CloseScanPattern)**

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

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

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

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

Creates a volume scan pattern of arbitrary form with equidistant sampled scan points.
- **SPECTRALRADAR_API void interpolatePoints2D** (double *OrigPosX, double *OrigPosY, int Size, double *InterpPosX, double *InterpPosY, int NewSize, **InterpolationMethod** InterpolationMet, **BoundaryCondition** BoundaryCond)

Interpolates the imaginary curve defined by the given sequence of points with the specified `InterpolationMethod`. The coordinates are abstract and this funcion has no sideeffects that could affect any physical property. The original and the interpolated coordinates have a meaning for the user, but no consequence for SpectralRadar.
- **SPECTRALRADAR_API void inflatePoints** (double *PosX, double *PosY, int Size, double *InflatedPosX, double *InflatedPosY, int NumberOfInflationLines, double RangeOfInflation, **InflationMethod** Method)

Inflates the provided curve in space with the specified `InflationMethod`. It can be used to create scan patterns of arbitrary forms with `createFreeformScanPattern3DFromLUT` if the used positions correspond to coordinates of the valid scan field in mm.
- **SPECTRALRADAR_API void saveScanPointsToFile** (double *ScanPosX_mm, double *ScanPosY_mm, int *ScanIndices, int Size, const char *Filename, **ScanPointsDataFormat** DataFormat)

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

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

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

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

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

Creates a `DataHandle` from the specified scan points and corresponding indices.
- **SPECTRALRADAR_API size_t projectMemoryRequirement** (**OCTDeviceHandle** Handle, **ScanPatternHandle** Pattern, **AcquisitionType** type)

Returns the size of the required memory, e.g. for a raw data object, in bytes to acquire the scan pattern once.
- **SPECTRALRADAR_API void startMeasurement** (**OCTDeviceHandle** Dev, **ScanPatternHandle** Pattern, **AcquisitionType** Type)

starts a continuous measurement BScans.
- **SPECTRALRADAR_API void getRawData** (**OCTDeviceHandle** Dev, **RawDataHandle** RawData)

Acquires data and stores the data unprocessed.

- **SPECTRALRADAR_API** void `getRawDataEx` (OCTDeviceHandle Dev, RawDataHandle RawData, int Cameraldx)
Acquires data with the specific camera given with camera index and stores the data unprocessed.
- **SPECTRALRADAR_API** void `getAnalogInputData` (OCTDeviceHandle Dev, DataHandle Output)
Acquires analog data and stores the result floating point voltages. Data from apodisation and flyback regions will be discarded.
- **SPECTRALRADAR_API** void `getAnalogInputDataEx` (OCTDeviceHandle Dev, DataHandle Data, DataHandle ApoData)
Acquires analog data and stores the result floating point voltages. Data from flyback regions will be discarded, data from apodisation regions will be stored in ApoData.
- **SPECTRALRADAR_API** void `setAnalogInputChannelEnabled` (OCTDeviceHandle Dev, int ChannelIndex, bool Enable)
Enables or disables an analog input channel. Use `getDevicePropertyInt` with flag `Device_NumOfAnalogInputChannels` to determine the number of available channels.
- **SPECTRALRADAR_API** bool `getAnalogInputChannelEnabled` (OCTDeviceHandle Dev, int ChannelIndex)
Returns the current status (enabled/disabled) of an analog input channel. Use `getDevicePropertyInt` with flag `Device_NumOfAnalogInputChannels` to determine the number of available channels.
- **SPECTRALRADAR_API** const char * `getAnalogInputChannelName` (OCTDeviceHandle Dev, int ChannelIndex)
Returns the name of an analog input channel. Use `getDevicePropertyInt` with flag `Device_NumOfAnalogInputChannels` to determine the number of available channels.
- **SPECTRALRADAR_API** void `stopMeasurement` (OCTDeviceHandle Dev)
stops the current measurement.
- **SPECTRALRADAR_API** void `measureSpectra` (OCTDeviceHandle Dev, int NumberOfSpectra, RawDataHandle Raw)
Acquires the desired number of spectra (raw data without processing) without moving galvo scanners.
- **SPECTRALRADAR_API** void `measureSpectraEx` (OCTDeviceHandle Dev, int NumberOfSpectra, RawDataHandle Raw, int CameralIndex)
Acquires the desired number of spectra (raw data without processing) without moving galvo scanners, for the desired camera.
- **SPECTRALRADAR_API** void `measureSpectraContinuousEx` (OCTDeviceHandle Dev, int NumberOfSpectra, RawDataHandle Raw, int CameralIndex)
Acquires the desired number of spectra (raw data without processing) without moving galvo scanners, for the desired camera. Starts a continuous acquisition in the background and uses ongoing acquisition if possible, to avoid latency from start/stop measurement.
- **SPECTRALRADAR_API** ProcessingHandle `createProcessing` (int SpectrumSize, int BytesPerRawPixel, BOOL Signed, float ScalingFactor, float MinElectrons, Processing_FFTType Type, float FFTOversampling)
Creates processing routines with the desired properties.
- **SPECTRALRADAR_API** ProcessingHandle `createProcessingForDevice` (OCTDeviceHandle Dev)
Creates processing routines for the specified device (`OCTDeviceHandle`).
- **SPECTRALRADAR_API** ProcessingHandle `createProcessingForDeviceEx` (OCTDeviceHandle Dev, int CameralIndex)
Creates processing routines for the specified device (`OCTDeviceHandle`) with camera index.
- **SPECTRALRADAR_API** ProcessingHandle `createProcessingForOCTFile` (OCTFileHandle File)
Creates processing routines for the specified OCT file (`OCTFileHandle`), such that the processing conditions are exactly the same as those when the file had been saved.
- **SPECTRALRADAR_API** ProcessingHandle `createProcessingForOCTFileEx` (OCTFileHandle File, const int CameralIndex)
Creates processing routines for the specified OCT file (`OCTFileHandle`), such that the processing conditions are exactly the same as those when the file had been saved.
- **SPECTRALRADAR_API** int `getInputSize` (ProcessingHandle Proc)
Returns the expected input size (pixels per spectrum) of the processing algorithms.
- **SPECTRALRADAR_API** int `getAScanSize` (ProcessingHandle Proc)
Returns the number of pixels in an A-Scan that can be obtained (computed) with the given processing routines.

- **SPECTRALRADAR_API** void `setApodizationWindow (ProcessingHandle Proc, ApodizationWindow Window)`
Sets the windowing function that will be used for apodization (this apodization has nothing to do with the reference spectra measured without a sample!). The selected windowing function will be used in all subsequent processings right before the fast Fourier transformation.
- **SPECTRALRADAR_API** ApodizationWindow `getApodizationWindow (ProcessingHandle Proc)`
Returns the current windowing function that is being used for apodization, `ApodizationWindow` (this apodization is not the reference spectrum measured without a sample!).
- **SPECTRALRADAR_API** void `setApodizationWindowParameter (ProcessingHandle Proc, ApodizationWindowParameter Selection, double Value)`
Sets the apodization window parameter, such as window width or ratio between constant and cosine part. Notice that this apodization is unrelated to the reference spectrum measured without a sample!.
- **SPECTRALRADAR_API** double `getApodizationWindowParameter (ProcessingHandle Proc, ApodizationWindowParameter Selection)`
Gets the apodization window parameter, such as window width or ratio between constant and cosine part. Notice that this apodization is unrelated to the reference spectrum measured without a sample!.
- **SPECTRALRADAR_API** void `getCurrentApodizationEdgeChannels (ProcessingHandle Proc, int *LeftPix, int *RightPix)`
Returns the pixel positions of the left/right edge channels of the current apodization. Here apodization refers to the reference spectra measured without sample.
- **SPECTRALRADAR_API** void `setProcessingDechirpAlgorithm (ProcessingHandle Proc, Processing_FFTType Type, float Oversampling)`
Sets the algorithm to be used for dechirping the input spectra.
- **SPECTRALRADAR_API** void `setProcessingParameterInt (ProcessingHandle Proc, ProcessingParameterInt Selection, int Value)`
Sets the specified integer value processing parameter.
- **SPECTRALRADAR_API** int `getProcessingParameterInt (ProcessingHandle Proc, ProcessingParameterInt Selection)`
Returns the specified integer value processing parameter.
- **SPECTRALRADAR_API** void `setProcessingParameterFloat (ProcessingHandle Proc, ProcessingParameterFloat Selection, double Value)`
Sets the specified floating point processing parameter.
- **SPECTRALRADAR_API** double `getProcessingParameterFloat (ProcessingHandle Proc, ProcessingParameterFloat Selection)`
Gets the specified floating point processing parameter.
- **SPECTRALRADAR_API** void `setProcessingFlag (ProcessingHandle Proc, ProcessingFlag Flag, BOOL Value)`
Sets the specified processing flag.
- **SPECTRALRADAR_API** BOOL `getProcessingFlag (ProcessingHandle Proc, ProcessingFlag Flag)`
Returns TRUE if the specified processing flag is set, FALSE otherwise.
- **SPECTRALRADAR_API** void `setProcessingAveragingAlgorithm (ProcessingHandle Proc, ProcessingAveragingAlgorithm Algorithm)`
Sets the algorithm that will be used for averaging during the processing.
- **SPECTRALRADAR_API** void `setCalibration (ProcessingHandle Proc, CalibrationData Selection, DataHandle Data)`
Sets the calibration data.
- **SPECTRALRADAR_API** void `getCalibration (ProcessingHandle Proc, CalibrationData Selection, DataHandle Data)`
Retrieves the desired calibration vector.
- **SPECTRALRADAR_API** void `measureCalibration (OCTDeviceHandle Dev, ProcessingHandle Proc, CalibrationData Selection)`
Measures the specified calibration parameters and uses them in subsequent processing.
- **SPECTRALRADAR_API** void `measureCalibrationEx (OCTDeviceHandle Dev, ProcessingHandle Proc, CalibrationData Selection, int CameralIndex)`
Measures the specified calibration parameters and uses them in subsequent processing with specified camera index.

- **SPECTRALRADAR_API** void `measureApodizationSpectra` (`OCTDeviceHandle` Dev, `ProbeHandle` Probe, `ProcessingHandle` Proc)
Measures the apodization spectra in the defined apodization position and size and uses them in subsequent processing.
- **SPECTRALRADAR_API** void `saveCalibrationDefault` (`ProcessingHandle` Proc, `CalibrationData` Selection)
Saves the selected calibration in its default path. This same default path will be used by SpectralRadar in subsequent executions to retrieve the calibration data.
- **SPECTRALRADAR_API** void `saveCalibrationDefaultEx` (`ProcessingHandle` Proc, `CalibrationData` Selection, int CameraIndex)
Saves the selected calibration in its default path, for the selected camera. This same default path will be used by SpectralRadar in.
- **SPECTRALRADAR_API** void `saveCalibration` (`ProcessingHandle` Proc, `CalibrationData` Selection, const char *Path)
Saves the selected calibration in the specified path.
- **SPECTRALRADAR_API** void `loadCalibration` (`ProcessingHandle` Proc, `CalibrationData` Selection, const char *Path)
Will load a specified calibration file and its content will be used for subsequent processing.
- **SPECTRALRADAR_API** void `setSpectrumOutput` (`ProcessingHandle` Proc, `DataHandle` Spectrum)
Sets the location for the resulting spectral data.
- **SPECTRALRADAR_API** void `setOffsetCorrectedSpectrumOutput` (`ProcessingHandle` Proc, `DataHandle` OffsetCorrectedSpectrum)
Sets the location for the resulting offset corrected spectral data.
- **SPECTRALRADAR_API** void `setDCCorrectedSpectrumOutput` (`ProcessingHandle` Proc, `DataHandle` DC↔CorrectedSpectrum)
Sets the location for the resulting DC removed spectral data.
- **SPECTRALRADAR_API** void `setApodizedSpectrumOutput` (`ProcessingHandle` Proc, `DataHandle` ApodizedSpectrum)
Sets the location for the resulting apodized spectral data.
- **SPECTRALRADAR_API** void `setComplexDataOutput` (`ProcessingHandle` Proc, `ComplexDataHandle` ComplexScan)
Sets the pointer to the resulting complex scans that will be written after subsequent processing executions.
- **SPECTRALRADAR_API** void `setProcessedDataOutput` (`ProcessingHandle` Proc, `DataHandle` Scan)
Sets the pointer to the resulting scans that will be written after subsequent processing executions.
- **SPECTRALRADAR_API** void `setColoredDataOutput` (`ProcessingHandle` Proc, `ColoredDataHandle` Scan, `ColoringHandle` Color)
Sets the pointer to the resulting colored scans that will be written after subsequent processing executions.
- **SPECTRALRADAR_API** void `setTransposedColoredDataOutput` (`ProcessingHandle` Proc, `ColoredDataHandle` Scan, `ColoringHandle` Color)
Sets the pointer to the resulting colored scans that will be written after subsequent processing executions. The orientation of the colored data will be transposed in such a way that the first axis (normally z-axis) will be the x-axis (the depth of each individual A-scan) and the second axis (normally x-axis) will be the z-axis.
- **SPECTRALRADAR_API** void `executeProcessing` (`ProcessingHandle` Proc, `RawDataHandle` RawData)
Executes the processing. The results will be stored as requested through the functions `setProcessedDataOutput()`, `setComplexDataOutput()`, `setColoredDataOutput()` (including coloring properties) and similar ones. In all cases, sizes and ranges will be adjusted automatically to the right values.
- **SPECTRALRADAR_API** void `finishProcessing` (`ProcessingHandle` Proc, `ComplexDataHandle` ComplexData)
Completes the processing. The results will be stored as requested through the functions `setProcessedDataOutput()`, `setComplexDataOutput()`, `setColoredDataOutput()` (including coloring properties) and similar ones. In all cases, sizes and ranges will be adjusted automatically to the right values.
- **SPECTRALRADAR_API** void `clearProcessing` (`ProcessingHandle` Proc)
Clears the processing instance and frees all temporary memory that was associated with it. Processing threads will be stopped.
- **SPECTRALRADAR_API** void `computeDispersion` (`DataHandle` Spectrum1, `DataHandle` Spectrum2, `DataHandle` Chirp, `DataHandle` Disp)

- Computes the dispersion and chirp of the two provided spectra, where both spectra need to have been subjected to same dispersion mismatch. Both spectra need to have been acquired for different path length differences.*
- **SPECTRALRADAR_API** void `computeDispersionByCoeff` (double Quadratic, **DataHandle** Chirp, **DataHandle** Disp)
Computes dispersion by a quadratic approximation specified by the quadratic factor.
 - **SPECTRALRADAR_API** void `computeDispersionByImage` (**DataHandle** LinearKSpectra, **DataHandle** Chirp, **DataHandle** Disp)
Guesses the dispersion based on the spectral data specified. The spectral data needs to be linearized in wavenumber before using this function.
 - **SPECTRALRADAR_API** int `getNumberOfDispersionPresets` (**ProcessingHandle** Proc)
Gets the number of dispersion presets.
 - **SPECTRALRADAR_API** const char * `getDispersionPresetName` (**ProcessingHandle** Proc, int Index)
Gets the name of the dispersion preset specified with index.
 - **SPECTRALRADAR_API** void `setDispersionPresetByName` (**ProcessingHandle** Proc, const char *Name)
Sets the dispersion preset specified with name.
 - **SPECTRALRADAR_API** void `setDispersionPresetByIndex` (**ProcessingHandle** Proc, int Index)
Sets the dispersion preset specified with index.
 - **SPECTRALRADAR_API** void `setDispersionPresets` (**ProcessingHandle** Proc, **ProbeHandle** Probe)
Sets the dispersion presets for the probe.
 - **SPECTRALRADAR_API** Processing_FFTType `getProcessing_FFTType` (**ProcessingHandle** Proc)
Retrieve the active FFT Type.
 - **SPECTRALRADAR_API** void `setDispersionCorrectionType` (**ProcessingHandle** Proc, **DispersionCorrectionType** Type)
Sets the active dispersion correction type.
 - **SPECTRALRADAR_API** DispersionCorrectionType `getDispersionCorrectionType` (**ProcessingHandle** Proc)
Sets the active dispersion correction type.
 - **SPECTRALRADAR_API** void `setDispersionQuadraticCoeff` (**ProcessingHandle** Proc, double Coeff)
Sets the coefficient for the quadratic correction of the dispersion.
 - **SPECTRALRADAR_API** double `getDispersionQuadraticCoeff` (**ProcessingHandle** Proc)
Sets the coefficient for the quadratic correction of the dispersion.
 - **SPECTRALRADAR_API** const char * `getCurrentDispersionPresetName` (**ProcessingHandle** Proc)
Gets the name of the active dispersion preset.
 - **SPECTRALRADAR_API** void `exportData` (**DataHandle** Data, **DataExportFormat** Format, const char *File←Name)
*Exports data (**DataHandle**) to a file. The number of dimensions is handled automatically upon analysis of the first argument.*
 - **SPECTRALRADAR_API** void `exportDataAsImage` (**DataHandle** Data, **ColoringHandle** Color, **ColoredDataExportFormat** Format, **Direction** SliceNormalDirection, const char *FileName, int ExportOptionMask)
*Exports 2-dimensional and 3-dimensional data (**DataHandle**) as image data (such as BMP, PNG, JPEG, ...).*
 - **SPECTRALRADAR_API** void `exportComplexData` (**ComplexDataHandle** Data, **ComplexDataExportFormat** Format, const char *FileName)
*Exports 1-, 2- and 3-dimensional complex data (**ComplexDataHandle**).*
 - **SPECTRALRADAR_API** void `exportColoredData` (**ColoredDataHandle** Data, **ColoredDataExportFormat** Format, **Direction** SliceNormalDirection, const char *FileName, int ExportOptionMask)
*Exports colored data (**ColoredDataHandle**).*
 - **SPECTRALRADAR_API** void `importColoredData` (**ColoredDataHandle** ColoredData, **DataImportFormat** Format, const char *FileName)
*Imports colored data (**ColoredDataHandle**) with the specified format and copied it into a data object (**ColoredDataHandle**)*
 - **SPECTRALRADAR_API** void `importData` (**DataHandle** Data, **DataImportFormat** Format, const char *File←Name)
*Imports data with the specified format and copies it into a data object (**DataHandle**).*

- **SPECTRALRADAR_API** void `exportRawData (RawDataHandle Raw, RawDataExportFormat Format, const char *FileName)`
Exports the specified data to disk.
- **SPECTRALRADAR_API** void `importRawData (RawDataHandle Raw, RawDataImportFormat Format, const char *FileName)`
Imports the specified data from disk.
- **SPECTRALRADAR_API** void `appendRawData (RawDataHandle Raw, RawDataHandle DataToAppend, Direction Dir)`
Appends the new raw data to the old raw data perpendicular to the specified direction.
- **SPECTRALRADAR_API** void `getRawDataSliceAtIndex (RawDataHandle Raw, RawDataHandle Slice, Direction SliceNormalDirection, int Index)`
Returns a slice of raw data perpendicular to the specified direction at the specified index.
- **SPECTRALRADAR_API** double `analyzeData (DataHandle Data, DataAnalyzation Selection)`
Analyzes the given data, extracts the selected feature, and returns the computed value.
- **SPECTRALRADAR_API** double `analyzeAScan (DataHandle Data, AScanAnalyzation Selection)`
Analyzes the given A-scan data, extracts the selected feature, and returns the computed value.
- **SPECTRALRADAR_API** void `analyzePeaksInAScan (DataHandle Data, AScanAnalyzation Selection, int NumberOfPeaksToAnalyze, int MinDistBetweenPeaks, double *Result)`
Analyzes the given A-scan data, extracts the selected feature, and returns the computed value. It returns the result of multiple peaks compared to the function `analyzeAScan`.
- **SPECTRALRADAR_API** void `transposeData (DataHandle DataIn, DataHandle DataOut)`
Transposes the given data and writes the result to `DataOut`. First and second axes will be swaped.
- **SPECTRALRADAR_API** void `transposeDataInplace (DataHandle Data)`
Transposes the given `Data`. First and second axes will be swaped.
- **SPECTRALRADAR_API** void `transposeAndScaleData (DataHandle DataIn, DataHandle DataOut, float Min, float Max)`
Transposes the given data and writes the result to `DataOut`. First and second axes will be swaped, and the range of the entries will be scaled in such a way, that the range [Min,Max] will be mapped onto the range [0,1].
- **SPECTRALRADAR_API** void `normalizeData (DataHandle Data, float Min, float Max)`
Scales the given data in such a way, that the range [Min, Max] is mapped onto the range [0,1].
- **SPECTRALRADAR_API** void `getDataSliceAtPos (DataHandle Data, DataHandle Slice, Direction Slice← NormalDirection, double Pos_mm)`
Returns a slice of data perpendicular to the specified direction at the specified position.
- **SPECTRALRADAR_API** void `getComplexDataSlicePos (ComplexDataHandle Data, ComplexDataHandle Slice, Direction SliceNormalDirection, double Pos_mm)`
Returns a slice of complex data perpendicular to the specified direction at the specified position.
- **SPECTRALRADAR_API** void `getColoredDataSlicePos (ColoredDataHandle Data, ColoredDataHandle Slice, Direction SliceNormalDirection, double Pos_mm)`
Returns a slice of colored data perpendicular to the specified direction at the specified position.
- **SPECTRALRADAR_API** void `getDataSliceAtIndex (DataHandle Data, DataHandle Slice, Direction Slice← NormalDirection, int Index)`
Returns a slice of data perpendicular to the specified direction at the specified index.
- **SPECTRALRADAR_API** void `getComplexDataSliceIndex (ComplexDataHandle Data, ComplexDataHandle Slice, Direction SliceNormalDirection, int Index)`
Returns a slice of complex data perpendicular to the specified direction at the specified index.
- **SPECTRALRADAR_API** void `getColoredDataSliceIndex (ColoredDataHandle Data, ColoredDataHandle Slice, Direction SliceNormalDirection, int Index)`
Returns a slice of colored data perpendicular to the specified direction at the specified index.
- **SPECTRALRADAR_API** void `computeDataProjection (DataHandle Data, DataHandle Slice, Direction ProjectionDirection, DataAnalyzation Selection)`
Returns a single slice of data, in which each pixel value is the feature extracted through an analysis along the specified direction.
- **SPECTRALRADAR_API** void `appendData (DataHandle Data, DataHandle DataToAppend, Direction Dir)`

- Appends the new data to the provided data, perpendicular to the specified direction.
- **SPECTRALRADAR_API** void `appendComplexData` (`ComplexDataHandle` Data, `ComplexDataHandle` DataToAppend, `Direction` Dir)
 - Appends the new data to the provided data, perpendicular to the specified direction.
- **SPECTRALRADAR_API** void `appendColoredData` (`ColoredDataHandle` Data, `ColoredDataHandle` DataToAppend, `Direction` Dir)
 - Appends the new data to the provided data, perpendicular to the specified direction.
- **SPECTRALRADAR_API** void `cropData` (`DataHandle` Data, `Direction` Dir, int IndexMax, int IndexMin)
 - Crops the data along the desired direction at the given indices. Upon return the data will only contain those slices whose indices were in the interval [IndexMin, IndexMax], counted along the cropping direction.
- **SPECTRALRADAR_API** void `cropComplexData` (`ComplexDataHandle` Data, `Direction` Dir, int IndexMax, int IndexMin)
 - Crops the complex data along the desired direction at the given indices. Upon return the data will only contain those slices whose indices were in the interval [IndexMin, IndexMax], counted along the cropping direction.
- **SPECTRALRADAR_API** void `cropColoredData` (`ColoredDataHandle` Data, `Direction` Dir, int IndexMax, int IndexMin)
 - Crops the colored data along the desired direction at the given indices. Upon return the data will only contain those slices whose indices were in the interval [IndexMin, IndexMax], counted along the cropping direction.
- **SPECTRALRADAR_API** void `separateData` (`DataHandle` Data1, `DataHandle` Data2, int SeparationIndex, `Direction` Dir)
 - Separates the data at the given index at specific separation direction. The first part of the separated data will remain in Data1, the second separated in Data2.
- **SPECTRALRADAR_API** void `separateComplexData` (`ComplexDataHandle` Data1, `ComplexDataHandle` Data2, int SeparationIndex, `Direction` Dir)
 - Separates the data at the given index at specific separation direction. The first part of the separated data will remain in Data1, the second separated in Data2.
- **SPECTRALRADAR_API** void `separateColoredData` (`ColoredDataHandle` Data1, `ColoredDataHandle` Data2, int SeparationIndex, `Direction` Dir)
 - Separates the data at the given index at specific separation direction. The first part of the separated data will remain in Data1, the second separated in Data2.
- **SPECTRALRADAR_API** void `flipData` (`DataHandle` Data, `Direction` FlippingDir)
 - Mirrors the data across a plane perpendicular to the given direction.
- **SPECTRALRADAR_API** void `flipComplexData` (`ComplexDataHandle` Data, `Direction` FlippingDir)
 - Mirrors the data across a plane perpendicular to the given direction.
- **SPECTRALRADAR_API** void `flipColoredData` (`ColoredDataHandle` Data, `Direction` FlippingDir)
 - Mirrors the data across a plane perpendicular to the given direction.
- **SPECTRALRADAR_API** `ImageFieldHandle` `createImageField` (`void`)
 - Creates an object holding image field data.
- **SPECTRALRADAR_API** `ImageFieldHandle` `createImageFieldFromProbe` (`ProbeHandle` Probe)
 - Creates an object holding image field data from the specified Probe Handle.
- **SPECTRALRADAR_API** void `clearImageField` (`ImageFieldHandle` ImageField)
 - Frees an object holding image field data.
- **SPECTRALRADAR_API** void `saveImageField` (`ImageFieldHandle` ImageField, const char *Path)
 - Saves data containing image field data.
- **SPECTRALRADAR_API** void `loadImageField` (`ImageFieldHandle` ImageField, const char *Path)
 - Loads data containing image field data.
- **SPECTRALRADAR_API** void `determineImageField` (`ImageFieldHandle` ImageField, `ScanPatternHandle` Pattern, `DataHandle` Surface)
 - Determines the image field correction for the given surface data, previously measured with the given scan pattern.
- **SPECTRALRADAR_API** void `determineImageFieldWithMask` (`ImageFieldHandle` ImageField, `ScanPatternHandle` Pattern, `DataHandle` Surface, `DataHandle` Mask)
 - Determines the image field correction for the given surface data, previously measured with the given scan pattern. The positive entries of the mask determine the points that actually enter in the computation.

- **SPECTRALRADAR_API** void `correctImageField` (`ImageFieldHandle` `ImageField`, `ScanPatternHandle` `Pattern`, `DataHandle` `Data`)
Applies the image field correction to the given B-Scan or volume data.
- **SPECTRALRADAR_API** void `correctImageFieldComplex` (`ImageFieldHandle` `ImageField`, `ScanPatternHandle` `Pattern`, `ComplexDataHandle` `Data`)
Applies the image field correction to the complex B-Scan or volume complex data.
- **SPECTRALRADAR_API** void `correctSurface` (`ImageFieldHandle` `ImageField`, `ScanPatternHandle` `Pattern`, `DataHandle` `Surface`)
Applies the image field correction to the given Surface. Surface must contain depth values as a function of x/y coordinates.
- **SPECTRALRADAR_API** void `setImageFieldInProbe` (`ImageFieldHandle` `ImageField`, `ProbeHandle` `Probe`)
Sets the specified image field to the specified Probe handle. Notice that no probe file will be automatically saved.
- **SPECTRALRADAR_API** `VisualCalibrationHandle` `createVisualCalibration` (`OCTDeviceHandle` `Device`, double `TargetCornerLength_mm`, `BOOL` `CheckAngle`, `BOOL` `SaveData`)
Creates handle used for visual calibration.
- **SPECTRALRADAR_API** void `clearVisualCalibration` (`VisualCalibrationHandle` `Handle`)
Clear handle and frees all related memory.
- **SPECTRALRADAR_API** `BOOL` `visualCalibrate_1st_CameraScaling` (`VisualCalibrationHandle` `Handle`, `ProbeHandle` `Probe`, `ColoredDataHandle` `Image`)
This is the first step in visual calibration. For this, the calibration target needs to be placed under the objective. Returns TRUE if the first step succeeds.
- **SPECTRALRADAR_API** `BOOL` `visualCalibrate_2nd_Galvo` (`VisualCalibrationHandle` `Handle`, `ProbeHandle` `Probe`, `ColoredDataHandle` `Image`)
This is the second step in visual calibration. For this, the calibration target or and infrared viewing card needs to be placed under the objective. Returns TRUE if the second step succeeds.
- **SPECTRALRADAR_API** `BOOL` `visualCalibrate_previewImage` (`VisualCalibrationHandle` `Handle`, `ColoredDataHandle` `Image`)
Provides a preview image for the current calibration.
- **SPECTRALRADAR_API** void `visualCalibration_getHoles` (`VisualCalibrationHandle` `Handle`, int *`x0`, int *`y0`, int *`x1`, int *`y1`, int *`x2`, int *`y2`)
Provides currently located hole positions of the three-hole target.
- **SPECTRALRADAR_API** const char * `visualCalibrate_Status` (`VisualCalibrationHandle` `Handle`)
Gives a status message of the currently executed visual calibration.
- **SPECTRALRADAR_API** `DopplerProcessingHandle` `createDopplerProcessing` (`void`)
Returns a handle for the use of Doppler-computation routines.
- **SPECTRALRADAR_API** `DopplerProcessingHandle` `createDopplerProcessingForFile` (`OCTFileHandle` `File`)
Returns a handle for the use of Doppler-computation routines. The handle is created based on a saved OCT file.
- **SPECTRALRADAR_API** int `getDopplerPropertyInt` (`DopplerProcessingHandle` `Handle`, `DopplerPropertyInt` `Property`)
Gets the value of the given Doppler processing property.
- **SPECTRALRADAR_API** void `setDopplerPropertyInt` (`DopplerProcessingHandle` `Handle`, `DopplerPropertyInt` `Property`, int `Value`)
Sets the value of the given Doppler processing property.
- **SPECTRALRADAR_API** double `getDopplerPropertyFloat` (`DopplerProcessingHandle` `Doppler`, `DopplerPropertyFloat` `Property`)
Gets the value of the given Doppler processing property.
- **SPECTRALRADAR_API** void `setDopplerPropertyFloat` (`DopplerProcessingHandle` `Handle`, `DopplerPropertyFloat` `Property`, float `Value`)
Sets the value of the given Doppler processing property.
- **SPECTRALRADAR_API** `BOOL` `getDopplerFlag` (`DopplerProcessingHandle` `Handle`, `DopplerFlag` `Flag`)
Gets the given Doppler processing flag.
- **SPECTRALRADAR_API** void `setDopplerFlag` (`DopplerProcessingHandle` `Handle`, `DopplerFlag` `Flag`, `BOOL` `OnOff`)

- Sets the given Doppler processing flag.*
- **SPECTRALRADAR_API** void `setDopplerAmplitudeOutput` (`DopplerProcessingHandle` Handle, `DataHandle` AmpOut)

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

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

Executes the Doppler processing of the input data and returns phases and amplitudes.
 - **SPECTRALRADAR_API** void `dopplerPhaseToVelocity` (`DopplerProcessingHandle` Doppler, `DataHandle` In↔Out)

Scales phases computed by Doppler OCT to actual flow velocities in scan direction.
 - **SPECTRALRADAR_API** void `dopplerVelocityToPhase` (`DopplerProcessingHandle` Doppler, `DataHandle` In↔Out)

Scales flow velocities computed by Doppler OCT back to original phase differences.
 - **SPECTRALRADAR_API** void `clearDopplerProcessing` (`DopplerProcessingHandle` Handle)

Closes the Doppler processing routines and frees the memory that has been allocated for these to work properly.
 - **SPECTRALRADAR_API** void `getDopplerOutputSize` (`DopplerProcessingHandle` Handle, int `Size1In`, int `Size2In`, int *`Size1Out`, int *`Size2Out`)

Returns the final size of the Doppler output if executeDopplerProcessing is executed using data of the specified input size.
 - **SPECTRALRADAR_API** void `calcContrast` (`DataHandle` ApodizedSpectrum, `DataHandle` Contrast)

Computes the contrast for the specified (apodized) spectrum.
 - **SPECTRALRADAR_API** `SettingsHandle` `initSettingsFile` (const char *Path)

*Loads a settings file (usually *.ini); and prepares its properties to be read.*
 - **SPECTRALRADAR_API** int `getSettingsEntryInt` (`SettingsHandle` SettingsFile, const char *Node, int Default↔Value)

Gets an integer number from the specified ini file (see `SettingsHandle` and `initSettingsFile`);
 - **SPECTRALRADAR_API** double `getSettingsEntryFloat` (`SettingsHandle` SettingsFile, const char *Node, double DefaultValue)

Gets an floating point number from the specified ini file (see `SettingsHandle` and `initSettingsFile`);
 - **SPECTRALRADAR_API** void `getSettingsEntryFloatArray` (`SettingsHandle` SettingsFile, const char *Node, const double *DefaultValues, double *Values, int *Size)

Gets an array of floating point numbers from the specified ini file (see `SettingsHandle` and `initSettingsFile`);
 - **SPECTRALRADAR_API** const char * `getSettingsEntryString` (`SettingsHandle` SettingsFile, const char *Node, const char *Value, const char *Default)

Gets a string from the specified ini file (see `SettingsHandle` and `initSettingsFile`); The resulting const char ptr will be valid until the settings file is closed by `closeSettingsFile`).*
 - **SPECTRALRADAR_API** void `setSettingsEntryInt` (`SettingsHandle` SettingsFile, const char *Node, int Value)

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

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

Sets a string in the specified ini file (see `SettingsHandle` and `initSettingsFile`);
 - **SPECTRALRADAR_API** void `saveSettings` (`SettingsHandle` SettingsFile)

Saves the changes to the specified Settings file.
 - **SPECTRALRADAR_API** void `closeSettingsFile` (`SettingsHandle` Handle)

Closes the specified ini file and stores the set entries .

- **SPECTRALRADAR_API** `ColoringHandle createColoring32Bit (ColorScheme Color, ColoringByteOrder ByteOrder)`

Creates processing that can be used to color given floating point B-scans to 32 bit colored images.
- **SPECTRALRADAR_API** `ColoringHandle createCustomColoring32Bit (int LUTSize, unsigned long *LUT)`

Create custom coloring using the specified color look-up-table.
- **SPECTRALRADAR_API** `void setColoringBoundaries (ColoringHandle Colorng, float Min_dB, float Max_dB)`

Sets the boundaries in dB which are used by the coloring algorithm to map colors to floating point values in dB.
- **SPECTRALRADAR_API** `void setColoringEnhancement (ColoringHandle Coloring, ColorEnhancement Enhancement)`

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

Colors a given data object (`DataHandle`) into a given colored object (`ColoredDataHandle`).
- **SPECTRALRADAR_API** `void colorizeDopplerData (ColoringHandle AmpColoring, ColoringHandle Phase←Coloring, DataHandle AmpData, DataHandle PhaseData, ColoredDataHandle Output, double MinSignal_dB, BOOL Transpose)`

Colors a two given data object (`DataHandle`) using overlay and intensity to represent phase and amplitude data. Used for Doppler imaging.
- **SPECTRALRADAR_API** `void colorizeDopplerDataEx (ColoringHandle AmpColoring, ColoringHandle PhaseColoring[2], DataHandle AmpData, DataHandle PhaseData, ColoredDataHandle Output, double MinSignal_dB, BOOL Transpose)`

Colors a two given data object (`DataHandle`) using overlay and intensity to represent phase and amplitude data. Used for Doppler imaging. In the extended version, two `ColoringHandles` can be specified, two provide different coloring for increasing and decreasing phase, for example.
- **SPECTRALRADAR_API** `void clearColoring (ColoringHandle Handle)`

Clears the coloring previously created by `createColoring32Bit`.
- **SPECTRALRADAR_API** `void getMaxCameralImageSize (OCTDeviceHandle Dev, int *SizeX, int *SizeY)`

Returns the maximum possible camera image size for the current device.
- **SPECTRALRADAR_API** `void getCameralImage (OCTDeviceHandle Dev, ColoredDataHandle Image)`

Gets a camera image.
- **SPECTRALRADAR_API** `unsigned long InterpretReferenceIntensity (float intensity)`

interprets the reference intensity and gives a color code that reflects its state.
- **SPECTRALRADAR_API** `unsigned long InterpretReferenceIntensitySingleValue (float DesiredIntensity, float Tolerance, float CurrentIntensity)`

interprets the reference intensity and gives green color code when the reference intensity is the DesiredIntensity plus/minus the Tolerance, otherwise red is returned.
- **SPECTRALRADAR_API** `void getConfigPath (char *Path, int StrSize)`

Returns the path that hold the config files.
- **SPECTRALRADAR_API** `void getPluginPath (char *Path, int StrSize)`

Returns the path that hold the plugins.
- **SPECTRALRADAR_API** `void getInstallationPath (char *Path, int StrSize)`

Returns the installation path.
- **SPECTRALRADAR_API** `double getReferenceIntensity (ProcessingHandle Proc)`

Returns an absolute value that indicates the reference intensity that was present when the currently used apodization was determined.
- **SPECTRALRADAR_API** `double getRelativeReferenceIntensity (OCTDeviceHandle Dev, ProcessingHandle Proc)`

Returns a value larger than 0.0 and smaller than 1.0 that indicates the reference intensity (relative to saturation) that was present when the currently used apodization was determined.
- **SPECTRALRADAR_API** `double getRelativeSaturation (ProcessingHandle Proc)`

Returns a value larger than 0.0 and smaller than 1.0 that indicates the saturation of the sensor that was present during the last processing cycle.

- **SPECTRALRADAR_API** BufferHandle `createMemoryBuffer (void)`
Creates a buffer holding data and colored data.
- **SPECTRALRADAR_API** void `appendToBuffer (BufferHandle, DataHandle, ColoredDataHandle)`
Appends specified data and colored data to the requested buffer.
- **SPECTRALRADAR_API** void `purgeBuffer (BufferHandle)`
Discards all data.
- **SPECTRALRADAR_API** int `getBufferSize (BufferHandle)`
Returns the currently available data sets in the buffer.
- **SPECTRALRADAR_API** int `getBufferFirstIndex (BufferHandle)`
Returns the index of the first data sets available in the buffer.
- **SPECTRALRADAR_API** int `getBufferLastIndex (BufferHandle)`
Returns the index of one past the last data sets available in the buffer.
- **SPECTRALRADAR_API** DataHandle `getBufferData (BufferHandle, int Index)`
Returns the data in the buffer.
- **SPECTRALRADAR_API** ColoredDataHandle `getColoredBufferData (BufferHandle, int Index)`
Returns the colored data in the buffer.
- **SPECTRALRADAR_API** void `clearBuffer (BufferHandle BufferHandle)`
Clears the buffer and frees all data and colored data objects in it.
- **SPECTRALRADAR_API** 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 SpectralIn, DataHandle SpectraOut, DataHandle Chirp)`
Linearizes the spectral data using the given chirp vector.
- **SPECTRALRADAR_API** const char * `DataObjectName_SpectralData (int index)`
Returns the filename of the spectral-data object with the specified index.
- **SPECTRALRADAR_API** OCTFileHandle `createOCTFile (OCTFormat format)`
Creates a handle to an OCT file of the given format.
- **SPECTRALRADAR_API** void `clearOCTFile (OCTFileHandle Handle)`
Clears the given OCT file handle and frees its resources.
- **SPECTRALRADAR_API** int `getFileDataObjectCount (OCTFileHandle Handle)`
Returns the number of data objects in the OCT file. This number will vary depending on the file's format and contents (Files with the .oct extension may contain multiple OCT data objects depending on their internal structure).
- **SPECTRALRADAR_API** void `loadFile (OCTFileHandle Handle, const char *Filename)`
Loads the actual OCT data file from a file system. The file must have the format given in `createOCTFile()`.
- **SPECTRALRADAR_API** void `saveFile (OCTFileHandle Handle, const char *Filename)`
Saves the OCT data file in the given fully qualified path name.
- **SPECTRALRADAR_API** void `saveChangesToFile (OCTFileHandle Handle)`
Saves the OCT data file in the file previously opened with `loadFile()`. Only changes will be saved.
- **SPECTRALRADAR_API** void `copyFileMetadata (OCTFileHandle SrcHandle, OCTFileHandle DstHandle)`
Copies metadata from one OCT file to another.
- **SPECTRALRADAR_API** bool `containsFileMetadataFloat (OCTFileHandle Handle, FileMetadataFloat Floatfield)`
Returns true if the given metadata field is present in the file.
- **SPECTRALRADAR_API** double `getFileMetadataFloat (OCTFileHandle Handle, FileMetadataFloat Floatfield)`
Returns the value of the given file metadata field as a floating point number if found.
- **SPECTRALRADAR_API** void `setFileMetadataFloat (OCTFileHandle Handle, FileMetadataFloat Floatfield, double Value)`
Sets the value of the given file metadata field as a floating point number.
- **SPECTRALRADAR_API** bool `containsFileMetadataInt (OCTFileHandle Handle, FileMetadataInt Intfield)`
Returns true if the given metadata field is present in the file.

- **SPECTRALRADAR_API** int getFileMetadataInt (OCTFileHandle Handle, FileMetadataInt Intfield)
Returns the value of the given file metadata field as an integer if found.
- **SPECTRALRADAR_API** void setFileMetadataInt (OCTFileHandle Handle, FileMetadataInt Intfield, int Value)
Sets the value of the given file metadata field as an integer.
- **SPECTRALRADAR_API** bool containsFileMetadataString (OCTFileHandle Handle, FileMetadataString Stringfield)
Returns true if the given metadata field is present in the file.
- **SPECTRALRADAR_API** const char * getFileMetadataString (OCTFileHandle Handle, FileMetadataString Stringfield)
Returns the value of the given file metadata field as a string if found.
- **SPECTRALRADAR_API** void setFileMetadataString (OCTFileHandle Handle, FileMetadataString Stringfield, const char *Content)
Sets the value of the given file metadata field as a string.
- **SPECTRALRADAR_API** bool containsFileMetadataFlag (OCTFileHandle Handle, FileMetadataFlag Boolfield)
Returns true if the given metadata field is present in the file.
- **SPECTRALRADAR_API** BOOL getFileMetadataFlag (OCTFileHandle Handle, FileMetadataFlag Boolfield)
Gets the boolean value of the given file metadata field.
- **SPECTRALRADAR_API** void setFileMetadataFlag (OCTFileHandle Handle, FileMetadataFlag Boolfield, BOOL Value)
Sets the boolean value of the given file metadata field.
- **SPECTRALRADAR_API** void saveFileMetadata (OCTFileHandle Handle, OCTDeviceHandle Dev, ProcessingHandle Proc, ProbeHandle Probe, ScanPatternHandle Pattern)
Saves meta information from the given device, processing, probe and scan pattern instances in the metadata block of the given file handle. This information will be available in files of type FileFormat_OCITY; mileage on other formats may vary according to their description.
- **SPECTRALRADAR_API** void setFileMetadataTimestamp (OCTFileHandle File, time_t Timestamp)
Saves provided timestamp to meta information to the given file handle. This information will be available in files of type FileFormat_OCITY; mileage on other formats may vary according to their description.
- **SPECTRALRADAR_API** time_t getFileMetadataTimestamp (OCTFileHandle File)
Returns the specified timestamp from the meta information of the given file handle. This information will be available in files of type FileFormat_OCITY; mileage on other formats may vary according to their description.
- **SPECTRALRADAR_API** void saveFileMetadataDoppler (OCTFileHandle Handle, DopplerProcessingHandle DopplerProc)
Saves meta information from the given DopplerProcessingHandle. A corresponding DopplerProcessingHandle can then be recreated using createDopplerProcessingForFile.
- **SPECTRALRADAR_API** void saveFileMetadataSpeckle (OCTFileHandle Handle, SpeckleVarianceHandle SpeckleVarianceProc)
Saves meta information from the given SpeckleVarianceHandle. A corresponding SpeckleVarianceHandle can then be recreated using initSpeckleVarianceForFile.
- **SPECTRALRADAR_API** void loadCalibrationFromFile (OCTFileHandle Handle, ProcessingHandle Proc)
Loads Chirp, Offset, and Apodization vectors from the given OCT file into the given processing object.
- **SPECTRALRADAR_API** void loadCalibrationFromFileEx (OCTFileHandle Handle, ProcessingHandle Proc, const int CameralIndex)
Loads Chirp, Offset, and Apodization vectors from the given OCT file into the given processing object.
- **SPECTRALRADAR_API** void saveCalibrationToFile (OCTFileHandle Handle, ProcessingHandle Proc)
Saves Chirp, Offset, and Apodization vectors from the given processing object into the given OCT file.
- **SPECTRALRADAR_API** void saveCalibrationToFileEx (OCTFileHandle Handle, ProcessingHandle Proc, int CameralIndex)
Saves Chirp, Offset, and Apodization vectors from the given processing object into the given OCT file.
- **SPECTRALRADAR_API** void getFileRealData (OCTFileHandle Handle, DataHandle Data, int Index)
Retrieves a RealData object from the OCT file at the given index with $0 \leq index < \text{getFileDataObjectCount}(\text{OCTFileHandle handle})$. Users must ensure that the data handle is properly prepared and destroyed.

- **SPECTRALRADAR_API** void `getFileColoredData (OCTFileHandle Handle, ColoredDataHandle Data, size_t Index)`

*Retrieves a ColoredData object from the OCT file at the given index with $0 \leq index < \text{getFileSize}(\text{OCTFileHandle handle})$.
Users must ensure that the data handle is properly prepared and destroyed.*
- **SPECTRALRADAR_API** void `getFileComplexData (OCTFileHandle Handle, ComplexDataHandle Data, size_t Index)`

*Retrieves a ComplexData object from the OCT file at the given index with $0 \leq index < \text{getFileSize}(\text{OCTFileHandle handle})$.
Users must ensure that the data handle is properly prepared and destroyed.*
- **SPECTRALRADAR_API** void `getFileRawData (OCTFileHandle Handle, RawDataHandle Data, size_t Index)`

*Retrieves a RawData object from the OCT file at the given index with $0 \leq index < \text{getFileSize}(\text{OCTFileHandle handle})$.
Users must ensure that the data handle is properly prepared and destroyed.*
- **SPECTRALRADAR_API** void `getFile (OCTFileHandle Handle, size_t Index, const char *FilenameOnDisk)`

Retrieves a data object of arbitrary type from the OCT file at the given index with $0 \leq index < \text{getFileSize}(\text{OCTFileHandle handle})$ and stores it at the given fully qualified path.
- **SPECTRALRADAR_API** int `findFileDataObject (OCTFileHandle Handle, const char *Search)`

Searches for a data object the name of which contains the given string and returns its index, -1 if not found.
- **SPECTRALRADAR_API** BOOL `containsFileDataObject (OCTFileHandle Handle, const char *Search)`

Searches for a data object the name of which contains the given string and returns TRUE if at least one data object name matches.
- **SPECTRALRADAR_API** BOOL `containsFileRawData (OCTFileHandle Handle)`

Returns TRUE if the file contains raw data objects.
- **SPECTRALRADAR_API** void `addFileRealData (OCTFileHandle Handle, DataHandle Data, const char *DataObjectName)`

Adds a RealData object to the OCT file; DataObjectName will be its name inside the OCT file if applicable. The object that the DataHandle refers to must live until after `saveFile()` has been called.
- **SPECTRALRADAR_API** void `addFileColoredData (OCTFileHandle Handle, ColoredDataHandle Data, const char *DataObjectName)`

Adds a ColoredData object to the OCT file; DataObjectName will be its name inside the OCT file if applicable. The object that the ColoredDataHandle refers to must live until after `saveFile()` has been called.
- **SPECTRALRADAR_API** void `addFileComplexData (OCTFileHandle Handle, ComplexDataHandle Data, const char *DataObjectName)`

Adds a ComplexData object to the OCT file; DataObjectName will be its name inside the OCT file if applicable. The object that the ComplexDataHandle refers to must live until after `saveFile()` has been called.
- **SPECTRALRADAR_API** void `addFileRawData (OCTFileHandle Handle, RawDataHandle Data, const char *DataObjectName)`

Adds raw Data object to the OCT file; DataObjectName will be its name inside the OCT file if applicable. The object that the RawDataHandle refers to must live until after `saveFile` has been called.
- **SPECTRALRADAR_API** void `addFileText (OCTFileHandle Handle, const char *FilenameOnDisk, const char *DataObjectName)`

Adds a text object read from FilenameOnDisk to the OCT file; DataObjectName will be its name inside the OCT file if applicable. The file identified by filenameOnDisk must exist until after `saveFile()` has been called.
- **SPECTRALRADAR_API** DataObjectType `getFileDataObjectType (OCTFileHandle Handle, int Index)`

Returns the type of the data object at the given Index in the OCT file.
- **SPECTRALRADAR_API** void `getFileDataObjectName (OCTFileHandle Handle, int Index, char *Filename, int Length)`

Returns the name of the data object at the given Index in the OCT file.
- **SPECTRALRADAR_API** int `getFileDataSizeX (OCTFileHandle Handle, size_t Index)`

Returns the pixel count in X of the data object at the given Index in the OCT file.
- **SPECTRALRADAR_API** int `getFileDataSizeY (OCTFileHandle Handle, size_t Index)`

Returns the pixel count in Y of the data object at the given Index in the OCT file.
- **SPECTRALRADAR_API** int `getFileDataSizeZ (OCTFileHandle Handle, size_t Index)`

Returns the pixel count in Z of the data object at the given Index in the OCT file.
- **SPECTRALRADAR_API** float `getFileDataRangeX (OCTFileHandle Handle, size_t Index)`

Returns the range (usually in mm) in X of the data object at the given Index in the OCT file.

- **SPECTRALRADAR_API** float `getFileDataRangeY` (`OCTFileHandle` Handle, `size_t` Index)
Returns the range (usually in mm) in Y of the data object at the given Index in the OCT file.
- **SPECTRALRADAR_API** float `getFileDataRangeZ` (`OCTFileHandle` Handle, `size_t` Index)
Returns the range (usually in mm) in Z of the data object at the given Index in the OCT file.
- **SPECTRALRADAR_API** void `clearMarkerList` (`OCTFileHandle` Handle)
Clears the marker list of a given OCT file. This removes all line and point markers from the file.
- **SPECTRALRADAR_API** void `copyMarkerListFromRealData` (`OCTFileHandle` Handle, `DataHandle` Data)
coordinates, so re-use is possible.
- **SPECTRALRADAR_API** void `copyMarkerListToRealData` (`OCTFileHandle` Handle, `DataHandle` Data)
coordinates, so re-use is possible.
- **SPECTRALRADAR_API** void `addFileMetadataPreset` (`OCTFileHandle` Handle, const char *Category, const char *PresetDescription)
Adds one of the presets set during acquisition for the `OCTFileHandle`.
- **SPECTRALRADAR_API** int `getFileMetadataNumberOfPresets` (`OCTFileHandle` Handle)
Gets the number of presets that were set during the acquisition.
- **SPECTRALRADAR_API** const char * `getFileMetadataPresetCategory` (`OCTFileHandle` Handle, int Index)
Gets the preset category belonging to the preset with given Index.
- **SPECTRALRADAR_API** const char * `getFileMetadataPresetDescription` (`OCTFileHandle` Handle, int Index)
Gets the preset description belonging to the preset with given Index.
- **SPECTRALRADAR_API** SpeckleVarianceHandle `initSpeckleVariance` (void)
Initializes the speckle variance contrast processing instance.
- **SPECTRALRADAR_API** SpeckleVarianceHandle `initSpeckleVarianceForFile` (`OCTFileHandle` File)
Initializes the speckle variance contrast processing instance, based on the parameters stored in an OCT file.
- **SPECTRALRADAR_API** void `closeSpeckleVariance` (SpeckleVarianceHandle Handle)
Closes the speckle variance contrast processing instance and frees all used resources.
- **SPECTRALRADAR_API** void `setSpeckleVariancePropertyInt` (SpeckleVarianceHandle Handle, `SpeckleVariancePropertyInt` Property, int value)
Sets the given integer property to the given value.
- **SPECTRALRADAR_API** int `getSpeckleVariancePropertyInt` (SpeckleVarianceHandle Handle, `SpeckleVariancePropertyInt` Property)
Sets the given floating point property to the given value.
- **SPECTRALRADAR_API** void `setSpeckleVariancePropertyFloat` (SpeckleVarianceHandle Handle, `SpeckleVariancePropertyFloat` Property, double value)
Returns the value of the given integer property.
- **SPECTRALRADAR_API** double `getSpeckleVariancePropertyFloat` (SpeckleVarianceHandle Handle, `SpeckleVariancePropertyFloat` Property)
Returns the value of the given floating point property.
- **SPECTRALRADAR_API** void `setSpeckleVarianceType` (SpeckleVarianceHandle SpeckleVar, `SpeckleVarianceType` Type)
Sets the speckle variance type to the given value.
- **SPECTRALRADAR_API** SpeckleVarianceType `getSpeckleVarianceType` (SpeckleVarianceHandle SpeckleVar)
Returns the speckle variance type the instance is using.
- **SPECTRALRADAR_API** void `computeSpeckleVariance` (SpeckleVarianceHandle SpeckleVar, `ComplexDataHandle` CompDataIn, `DataHandle` DataOutMean, `DataHandle` DataOutVar)
Computes the speckle variance contrast and returns the mean and variance values in DataOutMean and DataOutVar.
- **SPECTRALRADAR_API** void `setTriggerMode` (`OCTDeviceHandle` Dev, `DeviceTriggerType` TriggerMode)
Sets the trigger mode for the OCT device used for acquisition. Additional hardware may be needed.
- **SPECTRALRADAR_API** `DeviceTriggerType` `getTriggerMode` (`OCTDeviceHandle` Dev)
Returns the trigger mode used for acquisition.
- **SPECTRALRADAR_API** `BOOL` `isTriggerModeAvailable` (`OCTDeviceHandle` Dev, `DeviceTriggerType` TriggerMode)

- **SPECTRALRADAR_API** `BOOL isTriggerIOModeAvailable (OCTDeviceHandle Dev, DeviceTriggerIOType TriggerMode)`

Returns whether the specified trigger mode is possible or not for the used device.
- **SPECTRALRADAR_API** `void setTriggerIOMode (OCTDeviceHandle Dev, DeviceTriggerIOType TriggerMode)`

Returns whether the specified trigger IO mode is possible or not for the used device.
- **SPECTRALRADAR_API** `void setTriggerIOMode (OCTDeviceHandle Dev, DeviceTriggerIOType TriggerMode)`

Sets the trigger mode for the trigger IO channel of the OCT device.
- **SPECTRALRADAR_API** `DeviceTriggerIOType getTriggerIOMode (OCTDeviceHandle Dev)`

Returns the trigger IO mode used for acquisition.
- **SPECTRALRADAR_API** `void setTriggerIOConfiguration (OCTDeviceHandle Dev, size_t Offset, size_t Divider)`

Configures the parameters for trigger mode for the trigger IO channel of the OCT device. If the trigger IO channel is set to output, it will start generating a pulses after n=Offset A-Scans. It will then generate a pulse every after d=Divider A-Scans. If the trigger IO channel is set to input, Offset is ignored. Whenever a trigger pulse is received on the trigger IO channel, the system will perform d=Divider A-Scans and wait for the next trigger.
- **SPECTRALRADAR_API** `void setTriggerTimeout_s (OCTDeviceHandle Dev, int Timeout_s)`

Sets the timeout of the camera in seconds (useful in external trigger mode).
- **SPECTRALRADAR_API** `int getTriggerTimeout_s (OCTDeviceHandle Dev)`

Returns the timeout of the camera in seconds (not used in trigger mode Trigger_FreeRunning).
- **SPECTRALRADAR_API** `int getScanPatternPropertyInt (ScanPatternHandle ScanPattern, ScanPatternPropertyInt Property)`

Returns the specified property of the scan pattern.
- **SPECTRALRADAR_API** `double getScanPatternPropertyFloat (ScanPatternHandle Pattern, ScanPatternPropertyFloat Selection)`

Returns the specified property of the scan pattern.
- **SPECTRALRADAR_API** `double expectedAcquisitionTime_s (ScanPatternHandle ScanPattern, OCTDeviceHandle Dev)`

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

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

Returns whether the acquisition type is available for the scan pattern.
- **SPECTRALRADAR_API** `BOOL checkAvailableMemoryForRawData (OCTDeviceHandle Dev, ScanPatternHandle Pattern, ptrdiff_t AdditionalMemory)`

Checks whether sufficient memory is available for raw data acquired with the specified scan pattern.
- **SPECTRALRADAR_API** `double QuantumEfficiency (OCTDeviceHandle Dev, double CenterWavelength_nm, double PowerIntoSpectrometer_W, DataHandle Spectrum_e)`

Calculates the quantum efficiency from the processed input spectrum in the Data instance.
- **SPECTRALRADAR_API** `void determineSurface (DataHandle Volume, DataHandle Surface)`

Performs a minimal segmentation of the data, by finding a surface that is compromised of the highest signals from each A-scan. From the 3D input data, the output data will 2D data, where each data pixel contains the depth of the respective surface as a function of the x- and y-pixel position.
- **SPECTRALRADAR_API** `unsigned long long getFreeMemory ()`

Returns the amount of free system memory. Function is available for convenience.
- **SPECTRALRADAR_API** `void absComplexData (const ComplexDataHandle ComplexData, DataHandle Abs)`

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

- Converts the complex values from the `ComplexDataHandle` to its dB values and writes them to `DataHandle`.
 - **SPECTRALRADAR_API** void `argComplexData` (`ComplexDataHandle` `ComplexData`, `DataHandle` `Arg`)
Converts the complex values from the `ComplexDataHandle` to its phase angle values and writes them to `DataHandle`.
 - **SPECTRALRADAR_API** void `realComplexData` (`ComplexDataHandle` `ComplexData`, `DataHandle` `Real`)
Writes the real part of the complex values from the `ComplexDataHandle` to `DataHandle`.
 - **SPECTRALRADAR_API** void `imagComplexData` (`ComplexDataHandle` `ComplexData`, `DataHandle` `Imag`)
Writes the imaginary part of the complex values from the `ComplexDataHandle` to `DataHandle`.
- **SPECTRALRADAR_API** void `determineDynamicRange_dB` (`DataHandle` `Data`, double *`MinRange_dB`, double *`MaxRange_dB`)
Gives a rough estimation of the dynamic range of the specified data object.
- **SPECTRALRADAR_API** void `determineDynamicRangeWithMinRange_dB` (`DataHandle` `Data`, double *`MinRange_dB`, double *`MaxRange_dB`, double `MinDynamicRange_dB`)
Gives a rough estimation of the dynamic range of the specified data object.
- **SPECTRALRADAR_API** void `medianFilter1D` (`DataHandle` `Data`, int `Rank`, `Direction` `FilterDirection`)
Computes a 1D-median filter on the specified data.
- **SPECTRALRADAR_API** void `medianFilter2D` (`DataHandle` `Data`, int `Rank`, `Direction` `FilterNormalDirection`)
Computes a 2D-median filter on the specified data.
- **SPECTRALRADAR_API** void `pepperFilter2D` (`DataHandle` `Data`, `PepperFilterType` `Type`, float `Threshold`, `Direction` `FilterNormalDirection`)
Removes pepper-noise (very low values, i. e. dark spots in the data). This enhances the visual (colored) representation of the data.
- **SPECTRALRADAR_API** void `convolutionFilter1D` (`DataHandle` `Data`, int `FilterSize`, float *`FilterKernel`, `Direction` `FilterDirection`)
Calculates a mathematical convolution of the Data and the 1D-FilterKernel.
- **SPECTRALRADAR_API** void `convolutionFilter2D` (`DataHandle` `Data`, int `FilterSize1`, int `FilterSize2`, float *`FilterKernel`, `Direction` `FilterNormalDirection`)
Calculates a mathematical convolution of the Data and the 2D-FilterKernel.
- **SPECTRALRADAR_API** void `convolutionFilter3D` (`DataHandle` `Data`, int `FilterSize1`, int `FilterSize2`, int `FilterSize3`, float *`FilterKernel`)
Calculates a mathematical convolution of the Data and the 3D-FilterKernel.
- **SPECTRALRADAR_API** void `predefinedFilter1D` (`DataHandle` `Data`, `FilterType1D` `Filter`, `Direction` `FilterDirection`)
Applies the predefined 1D-Filter to the Data.
- **SPECTRALRADAR_API** void `predefinedFilter2D` (`DataHandle` `Data`, `FilterType2D` `Filter`, `Direction` `FilterNormalDirection`)
Applies the predefined 2D-Filter to the Data.
- **SPECTRALRADAR_API** void `predefinedFilter3D` (`DataHandle` `Data`, `FilterType3D` `FilterType`)
Applies the predefined 3D-Filter to the Data.
- **SPECTRALRADAR_API** void `predefinedComplexFilter2D` (`ComplexDataHandle` `ComplexData`, `ComplexFilterType2D` `Type`, `Direction` `FilterNormalDirection`)
Applies the predefined 2D-Filter to the ComplexData.
- **SPECTRALRADAR_API** void `darkFieldComplexFilter2D` (`ComplexDataHandle` `ComplexData`, double `Radius`, `Direction` `FilterNormalDirection`)
Filters the image such that the image contrast comes from light scattered by the sample.
- **SPECTRALRADAR_API** void `brightFieldComplexFilter2D` (`ComplexDataHandle` `ComplexData`, double `Radius`, `Direction` `FilterNormalDirection`)
Filters the image such that the image contrast comes from absorbance of light in the sample.
- **SPECTRALRADAR_API** void `polynomialFitAndEval1D` (int `Size`, const float *`OrigPosX`, const float *`OrigY`, int `DegreePolynom`, int `EvalSize`, const float *`EvalPosX`, float *`EvalY`)
Computes the polynomial fit of the given 1D data.
- **SPECTRALRADAR_API** float `calcParabolaMaximum` (float `x0`, float `y0`, float `yLeft`, float `yRight`, float *`peakHeight`)

- Computes the x-position of the highest peak of the parabola given by the point $x0$, $y0$, $yLeft$, $yRight$. $y0$ needs to be the point with the highest value.
- **SPECTRALRADAR_API** void `crossCorrelatedProjection` (`DataHandle DataIn`, `DataHandle DataOut`)

Upon return `DataOut` contains an average of all B-Scans in `DataIn`. Right before averaging, the datasets are cross-correlated to eliminate registration errors.
- **SPECTRALRADAR_API** void `averagingResizeFilter` (`DataHandle DataIn`, `DataHandle DataOut`, int `Decimation1`, int `Decimation2`, int `Decimation3`)

Resizes a data object by averaging. The number of pixels that are averaged on each axis can be specified by the Decimation parameters. E.g. if `Decimation2` is set to 3, `Dimension2` of the resulting `DataOut` object will be 1/3 of `Dimension2` of `DataIn`.
- **SPECTRALRADAR_API** void `thresholdDopplerData` (`DataHandle Phase`, `DataHandle Intensity`, float `intensityThreshold`, float `phaseTargetValue`)

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

Returns two statistical interpretations of the current light intensity on the sensor.
- **SPECTRALRADAR_API** int `getNumberOfProbeConfigs` ()

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

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

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

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

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

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

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

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

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

Sets the `Shape` of the valid scan range for the `ProbeHandle`. All possible scan-range shapes are defined in `ProbeScanRangeShape`.
- **SPECTRALRADAR_API** void `setQuadraticProbeFactors` (`ProbeHandle Probe`, double *`QuadFactorsX`, double *`QuadFactorsY`, int `NumberOfFactors`)

Sets the probe calibration factors.
- **SPECTRALRADAR_API** int `getObjectivePropertyInt` (const char *`Objective`, `ObjectivePropertyInt Selection`)

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

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

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

- **SPECTRALRADAR_API** void `addProbeButtonCallback (OCTDeviceHandle Dev, cbProbeMessageReceived Callback)`
Registers a callback function to notify when a button on the probe has been pressed. The int parameter passed to the callback function will contain the pressed button's ID. Caution: Since the callbacks will not be called in separate threads but in the order of addition, make sure that the callback function returns as soon as possible.
- **SPECTRALRADAR_API** void `removeProbeButtonCallback (OCTDeviceHandle Dev, cbProbeMessageReceived Callback)`
Removes a previously registered probe button callback function.
- **SPECTRALRADAR_API** BOOL `isRefstageAvailable (OCTDeviceHandle Dev)`
Returns whether a motorized reference stage is available or not for the specified device. Please note that a motorized reference stage is not included in all systems.
- **SPECTRALRADAR_API** RefstageStatus `getRefstageStatus (OCTDeviceHandle Dev)`
Returns the current status of the reference stage, e.g. if it is moving.
- **SPECTRALRADAR_API** double `getRefstageLength_mm (OCTDeviceHandle Dev, ProbeHandle Probe)`
Returns the total length in mm of the reference stage.
- **SPECTRALRADAR_API** double `getRefstagePosition_mm (OCTDeviceHandle Dev, ProbeHandle Probe)`
Returns the current position in mm of the reference stage.
- **SPECTRALRADAR_API** void `homeRefstage (OCTDeviceHandle Dev, RefstageWaitForMovement WaitForMoving)`
Homes the reference stage to calibrate the zero position.
- **SPECTRALRADAR_API** void `moveRefstageToPosition_mm (OCTDeviceHandle Dev, ProbeHandle Probe, double Pos_mm, RefstageSpeed Speed, RefstageWaitForMovement WaitForMoving)`
Moves the reference stage to the specified position in mm.
- **SPECTRALRADAR_API** void `moveRefstage_mm (OCTDeviceHandle Dev, ProbeHandle Probe, double Length_mm, RefstageMovementDirection Direction, RefstageSpeed Speed, RefstageWaitForMovement WaitForMoving)`
Moves the reference stage with the specified length in mm.
- **SPECTRALRADAR_API** void `startRefstageMovement (OCTDeviceHandle Dev, RefstageMovementDirection Direction, RefstageSpeed Speed)`
Starts the movement of the reference stage with the chosen speed. Please note that the movement does not stop until `stopRefstageMovement` is called.
- **SPECTRALRADAR_API** void `stopRefstageMovement (OCTDeviceHandle Dev)`
Stops the movement of the reference stage.
- **SPECTRALRADAR_API** void `setRefstageSpeed (OCTDeviceHandle Dev, RefstageSpeed Speed)`
Sets the velocity of the movement of the reference stage.
- **SPECTRALRADAR_API** void `setRefstageStatusCallback (OCTDeviceHandle Dev, cbRefstageStatusChanged Callback)`
Registers the callback to get notified if the reference stage status changed.
- **SPECTRALRADAR_API** void `setRefstagePosChangedCallback (OCTDeviceHandle Dev, cbRefstagePositionChanged Callback)`
Registers the callback to get notified if the reference stage position changed.
- **SPECTRALRADAR_API** double `getRefstageMinPosition_mm (OCTDeviceHandle Dev, ProbeHandle Probe)`
Returns the minimal position in mm the reference stage can move to.
- **SPECTRALRADAR_API** double `getRefstageMaxPosition_mm (OCTDeviceHandle Dev, ProbeHandle Probe)`
Returns the maximal position in mm the reference stage can move to.
- **SPECTRALRADAR_API** void `setLightSourceTimeoutCallback (OCTDeviceHandle Dev, lightSourceStateCallback Callback)`
Sets a callback function that will be invoked by the SDK whenever the state of the lightsource of the device changes.
- **SPECTRALRADAR_API** void `setLightSourceTimeout_s (OCTDeviceHandle Dev, double Timeout)`
Sets a the timeout in seconds, after which the OCT lightsource will be turned off if no scanning is performed.
- **SPECTRALRADAR_API** double `getLightSourceTimeout_s (OCTDeviceHandle Dev)`
Gets a the timeout in seconds, after which the OCT lightsource will be turned off if no scanning is performed.

- **SPECTRALRADAR_API** void `setPolarizationFlag` (`PolarizationProcessingHandle` `Polarization`, `PolarizationFlag`, `BOOL` `OnOff`)
Sets the polarization processing flags.
- **SPECTRALRADAR_API** `BOOL` `getPolarizationFlag` (`PolarizationProcessingHandle` `Polarization`, `PolarizationFlag`)
Gets the desired polarization processing flag.
- **SPECTRALRADAR_API** `PolarizationProcessingHandle` `createPolarizationProcessing` (`void`)
Returns a Polarization processing handle to the Processing routines for polarization analysis.
- **SPECTRALRADAR_API** void `clearPolarizationProcessing` (`PolarizationProcessingHandle` `Polarization`)
Clears the polarization processing routines and frees the memory that has been allocated for these to work properly.
- **SPECTRALRADAR_API** int `getPolarizationPropertyInt` (`PolarizationProcessingHandle` `Polarization`, `PolarizationPropertyInt` `Property`)
Gets the desired polarization processing property.
- **SPECTRALRADAR_API** void `setPolarizationPropertyInt` (`PolarizationProcessingHandle` `Polarization`, `PolarizationPropertyInt` `Property`, int `Value`)
Sets polarization processing properties.
- **SPECTRALRADAR_API** double `getPolarizationPropertyFloat` (`PolarizationProcessingHandle` `Polarization`, `PolarizationPropertyFloat` `Property`)
Gets the desired polarization processing floating-point property.
- **SPECTRALRADAR_API** void `setPolarizationPropertyFloat` (`PolarizationProcessingHandle` `Polarization`, `PolarizationPropertyFloat` `Property`, double `Value`)
Sets the desired polarization processing floating-point property.
- **SPECTRALRADAR_API** void `setPolarizationOutputI` (`PolarizationProcessingHandle` `Polarization`, `DataHandle` `Intensity`)
Sets the location of the resulting polarization intensity output (Stokes parameter I).
- **SPECTRALRADAR_API** void `setPolarizationOutputQ` (`PolarizationProcessingHandle` `Polarization`, `DataHandle` `StokesQ`)
Sets the location of the resulting Stokes parameter Q.
- **SPECTRALRADAR_API** void `setPolarizationOutputU` (`PolarizationProcessingHandle` `Polarization`, `DataHandle` `StokesU`)
Sets the location of the resulting Stokes parameter U.
- **SPECTRALRADAR_API** void `setPolarizationOutputV` (`PolarizationProcessingHandle` `Polarization`, `DataHandle` `StokesV`)
Sets the location of the resulting Stokes parameter U.
- **SPECTRALRADAR_API** void `setPolarizationOutputDOPU` (`PolarizationProcessingHandle` `Polarization`, `DataHandle` `DOPU`)
Sets the location of the resulting DOPU (Degree Of Polarization Uniformity).
- **SPECTRALRADAR_API** void `setPolarizationOutputRetardation` (`PolarizationProcessingHandle` `Polarization`, `DataHandle` `Retardation`)
Sets the location of the resulting retardation.
- **SPECTRALRADAR_API** void `setPolarizationOutputOpticAxis` (`PolarizationProcessingHandle` `Polarization`, `DataHandle` `OpticAxis`)
Sets the location of the resulting optic axis.
- **SPECTRALRADAR_API** void `executePolarizationProcessing` (`PolarizationProcessingHandle` `Polarization`, `ComplexDataHandle` `Data_Camera1`, `ComplexDataHandle` `PData_Camera0`)
Executes the polarization processing of the input data and returns, if previously setup, intensity, retardation, and phase differences.
- **SPECTRALRADAR_API** void `saveFileMetadataPolarization` (`OCTFileHandle` `FileHandle`, `PolarizationProcessingHandle` `PolProc`)
Saves metadata to the specified file. These metadata specify the operational arguments needed by the polarization processing routines to redo the polarization-analysis starting from two `ComplexDataHandle` delivered by `Proc_0` and `Proc_1`.
- **SPECTRALRADAR_API** `PolarizationProcessingHandle` `createPolarizationProcessingForFile` (`OCTFileHandle` `FileHandle`)

Loads metadata to the specified file. These metadata specify the operational arguments needed by the polarization processing routines to redo the polarization-analysis starting from two `ComplexDataHandle` delivered by `Proc_0` and `Proc_1`, exactly as they were done before the file was written.

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

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

- **SPECTRALRADAR_API double analyzeComplexAScan (ComplexDataHandle AScanIn, AScanAnalysis Selection)**

Analyzes the given complex A-scan data, extracts the selected feature, and returns the computed value.

- **SPECTRALRADAR_API BOOL isPolarizationAdjustmentAvailable (OCTDeviceHandle Dev)**

Returns whether or not a motorized polarization adjustment stage is available for the specified device.

- **SPECTRALRADAR_API void setPolarizationAdjustmentRetardationChangedCallback (OCTDeviceHandle Dev, cbRetardationChanged Callback)**

Registers the callback to get notified when the polarization adjustment retardation has changed.

- **SPECTRALRADAR_API void setPolarizationAdjustmentRetardation (OCTDeviceHandle Dev, PolarizationRetarder Retarder, double Retardation, WaitForCompletion Wait)**

Sets the retardation of the specified retarder in the polarization adjustment. The retardation is a unitless value between 0 and 1, which represents the full adjustment range of the retarder. The retarder may take some time to physically reach the new Retardation. Use the Wait parameter to choose if the function should block until the new position is reached.

- **SPECTRALRADAR_API double getPolarizationAdjustmentRetardation (OCTDeviceHandle Dev, PolarizationRetarder Retarder)**

Gets the current retardation of the specified retarder in the polarization adjustment. If `setPolarizationAdjustmentRetardation` was used in a non-blocking fashion, the function returns the current position of the retarder, not the final target position.

- **SPECTRALRADAR_API BOOL isReferenceIntensityControlAvailable (OCTDeviceHandle Dev)**

Returns whether or not an automated reference intensity control is available for the specified device.

- **SPECTRALRADAR_API void setReferenceIntensityControlCallback (OCTDeviceHandle Dev, cbReferenceIntensityControlValue Callback)**

Registers the callback to get notified when the reference intensity has changed.

- **SPECTRALRADAR_API void setReferenceIntensityControlValue (OCTDeviceHandle Dev, double ReferenceIntensity, WaitForCompletion Wait)**

Sets the reference intensity of the specified device. The intensity is a unitless value between 0 and 1, which represents the full adjustment range of the reference intensity control, but may or may not be linear. The control may take some time to physically reach the new intensity. Use the Wait parameter to choose if the function should block until the new intensity is reached.

- **SPECTRALRADAR_API double getReferenceIntensityControlValue (OCTDeviceHandle Dev)**

Gets the current reference intensity of the specified device. If `setReferenceIntensityControlValue` was used in a non-blocking fashion, the function returns the current value of the control, not the final target value.

- **SPECTRALRADAR_API BOOL isAmplificationControlAvailable (OCTDeviceHandle Dev)**

Returns whether or not the sampling amplification of specified device can be adjusted.

- **SPECTRALRADAR_API int getAmplificationControlNumberOfSteps (OCTDeviceHandle Dev)**

Gets the number of discrete amplification control steps available on the specified device. Please note that the largest amplification step is `getAmplificationControlNumberOfSteps()` - 1.

- **SPECTRALRADAR_API void setAmplificationControlStep (OCTDeviceHandle Dev, int Step)**

Sets the sampling amplification on the the specified device. The lowest amplification is always 0. In general, the amplification should be set as high as possible without going into saturation.

- **SPECTRALRADAR_API int getAmplificationControlStep (OCTDeviceHandle Dev)**

Gets the current sampling amplification of the specified device.

- **SPECTRALRADAR_API void getSoftwareVersion (char *Version, int Length)**

Returns the current software version.

- **SPECTRALRADAR_API void useProbeCalibration (bool OnOff)**

Enable or disable use of probe calibration. Needs to be called before `initProbe`.

- **SPECTRALRADAR_API void extractLine (DataHandle Data, DataHandle Res, double P1_1_mm, double P1_2_mm, double P2_1_mm, double P2_2_mm)**

- Extract 1D data from 2D along a specified line.*

 - **SPECTRALRADAR_API** int `extractLocalMaxima` (**DataHandle** Data1D, int N, double *dataPos_mm, double *dataHeight)
Extract multiple local maxima from 1D data.
 - **SPECTRALRADAR_API** int `extractLocalMaximaEx` (**DataHandle** Data1D, int N, double minDist, double *dataPos_mm, double *dataHeight)
Extract multiple local maxima from 1D data while mainting a minimum distance between peaks.
 - **SPECTRALRADAR_API** void `readData` (**DataHandle** Data, const char *filename, int SizeZ, int SizeX, int SizeY)
Read data object from raw data stream in file.
 - **SPECTRALRADAR_API** double `getExpectedHannFWHM` (**OCTDeviceHandle** Dev, **ProcessingHandle** Proc)
Returns the theoretical width of a hann point-spread function (PSF) given the current spectral width and edge channels.
 - **SPECTRALRADAR_API** void `calcContrastEx` (**ProcessingHandle** Proc, **DataHandle** ApodizedSpectrum, **DataHandle** Contrast)
Calculates the spectrometer's contrast given a modulated spectrum, a properly calibrated processing (containing an apodization spectrum) as a function of the camera pixel.
 - **SPECTRALRADAR_API** double `calcSpectrometerImagingQualityIndex` (**DataHandle** Spectrum)
Given a maximally modulated spectrum, this function calculates a quality indicator that describes the image quality of the specturm onto the line camera.
 - **SPECTRALRADAR_API** void `analyzeMaxPeak` (**DataHandle** Data, float *PeakHeight_dB, float *PeakFWHM_Pixel)
This function analyzes the highest peak in the given A-scan data and yields its height (in dB) and full-width at half maximum (FWHM) in pixels.
 - **SPECTRALRADAR_API** void `analyzeMaxPeakEx` (**DataHandle** Data, float *PeakHeight_dB, float *PeakFWHM_Pixel, float *Points, float *Spline, int *SplineSize)
This function analyzes the highest peak in the given A-scan data and yields its height (in dB) and full-width at half maximum (FWHM) in pixels. In addition to `analyzeMaxPeak` this function gives additional data that was used for the determination, such as the relevant points for maximum, and FWHM computations, as well as the interpolated spline.
 - **SPECTRALRADAR_API** void `extractAScan` (**DataHandle** Data, **DataHandle** Res, int x0, int y0)
Extracts an Ascan at a given x and y coordinates and stores it in a different data handle.

Variables

ExportOptions

Specifies additional export options to be used with functions such as `exportDataAsImage()`. Multiple options can be combined by bit-wise or ("|"). Different options can be used for different export format. If an option is not supported by an export format, it is ignored.

- const int **ExportOption_None** = 0x00000000
- const int **ExportOption_DrawScaleBar** = 0x00000001
Draw scale bar on exported image.
- const int **ExportOption_DrawMarkers** = 0x00000002
Draw markers on exported image.
- const int **ExportOption_UsePhysicalAspectRatio** = 0x00000004
Honor physical aspect ratio when exporting data (width and height of each pixel will have the same physical dimensions).
- const int **ExportOption_Flip_X_Axis** = 0x00000008
Flip X-axis.
- const int **ExportOption_Flip_Y_Axis** = 0x00000010
Flip Y-axis.
- const int **ExportOption_Flip_Z_Axis** = 0x00000020
Flip Z-axis.

8.1.1 Detailed Description

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

Definition in file [SpectralRadar.h](#).

8.1.2 Macro Definition Documentation

8.1.2.1 SPECTRALRADAR_API `#define SPECTRALRADAR_API`

Export/Import of define of DLL members.

Definition at line 223 of file [SpectralRadar.h](#).

8.1.3 Function Documentation

8.1.3.1 checkAvailableMemoryForRawData() `BOOL checkAvailableMemoryForRawData (OCTDeviceHandle Dev, ScanPatternHandle Pattern, ptrdiff_t AdditionalMemory)`

Checks whether sufficient memory is available for raw data acquired with the specified scan pattern.

AdditionalMemory The parameter specifies additional memory that will be required during the measurement (from [startMeasurement\(\)](#) to [stopMeasruement\(\)](#)) unknown to the SDK and/or memory that will be freed/available prior to the call of [startMeasurement\(\)](#).

8.1.3.2 clearVisualCalibration() `void clearVisualCalibration (VisualCalibrationHandle Handle)`

Clear handle and frees all related memory.

Parameters

in	<code>Handle</code>	A handle of a visual calibration (VisualCalibrationHandle). If the handle is a nullptr, this function does nothing. In most cases this handle will have been previously created with the function createVisualCalibration .
----	---------------------	---

Warning

ThorImageOCT uses this and other functions to calibrate the galvo, assuming a very specific sequence of actions and conditions, as explained in the ThorImageOCT. For this function to properly work, the user need to re-create the same sequence of actions and conditions. Please use the ThorImageOCT software to perform probe calibrations, if at all necessary.

8.1.3.3 closeSpeckleVariance() `void closeSpeckleVariance (`
`SpeckleVarianceHandle Handle)`

Closes the speckle variance contrast processing instance and frees all used resources.

Parameters

in	<i>Handle</i>	A handle of speckle variance routines (SpeckleVarianceHandle). If the handle is a nullptr, this function does nothing.
----	---------------	--

8.1.3.4 computeSpeckleVariance() `void computeSpeckleVariance (`
`SpeckleVarianceHandle SpeckleVar,`
`ComplexDataHandle CompDataIn,`
`DataHandle DataOutMean,`
`DataHandle DataOutVar)`

Computes the speckle variance contrast and returns the mean and variance values in DataOutMean and DataOutVar.

8.1.3.5 createAScanPattern() `ScanPatternHandle createAScanPattern (`
`ProbeHandle Probe,`
`int AScans,`
`double PosX_mm,`
`double PosY_mm)`

Creates a scan pattern used to acquire a specific amount of Ascans at a specific position.

Parameters

in	<i>Probe</i>	A valid (non null) handle of a probe.
in	<i>AScans</i>	The number of A-Scans that will be measured.
in	<i>PosX_mm</i>	The position of the light spot, in millimeter.
in	<i>PosY_mm</i>	The position of the light spot, in millimeter.

Returns

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

```
8.1.3.6 createVisualCalibration() VisualCalibrationHandle createVisualCalibration (
    OCTDeviceHandle Device,
    double TargetCornerLength_mm,
    BOOL CheckAngle,
    BOOL SaveData )
```

Creates handle used for visual calibration.

Parameters

in	<i>Device</i>	A valid (non null) OCT device handle (<code>OCTDeviceHandle</code>), previously generated with the function <code>initDevice</code> .
in	<i>TargetCornerLength_mm</i>	The length of the edge
in	<i>CheckAngle</i>	A flag stating if the the sample's position is in a right angle with respect to the camera image (TRUE) or not (FALSE).
in	<i>SaveData</i>	If TRUE, debug information will be dumped. Kindly say FALSE.

Returns

A valid handle of a visual calibration (`VisualCalibrationHandle`).

Warning

ThorImageOCT uses this and other functions to calibrate the galvo, assuming a very specific sequence of actions and conditions, as explained in the ThorImageOCT. For this function to properly work, the user need to re-create the same sequence of actions and conditions. Please use the ThorImageOCT software to perform probe calibrations, if at all necessary.

```
8.1.3.7 dopplerPhaseToVelocity() void dopplerPhaseToVelocity (
    DopplerProcessingHandle Handle,
    DataHandle InOut )
```

Scales phases computed by Doppler OCT to actual flow velocities in scan direction.

Parameters

in	<i>Handle</i>	A valid (non null) handle of Doppler processing routines (<code>DopplerProcessingHandle</code>), obtained with the function <code>createDopplerProcessing</code> .
in, out	<i>InOut</i>	A handle of data representing first phase data that will then be modified to contain velocity data.

This requires the Doppler scan rate, Doppler angle and center velocity of the Doppler object to be set correctly.

```
8.1.3.8 getFreeMemory() unsigned long long getFreeMemory ( )
```

Returns the amount of free system memory. Function is available for convenience.

```
8.1.3.9 getSpeckleVariancePropertyFloat() double getSpeckleVariancePropertyFloat (
    SpeckleVarianceHandle Handle,
    SpeckleVariancePropertyFloat Property )
```

Returns the value of the given floating point property.

```
8.1.3.10 getSpeckleVariancePropertyInt() int getSpeckleVariancePropertyInt (
    SpeckleVarianceHandle Handle,
    SpeckleVariancePropertyInt Property )
```

Sets the given floating point property to the given value.

```
8.1.3.11 getSpeckleVarianceType() void SpeckleVarianceType getSpeckleVarianceType (
    SpeckleVarianceHandle SpeckleVar )
```

Returns the speckle variance type the instance is using.

```
8.1.3.12 initSpeckleVariance() SpeckleVarianceHandle initSpeckleVariance (
    void )
```

Initializes the speckle variance contrast processing instance.

```
8.1.3.13 initSpeckleVarianceForFile() SPECTRALRADAR_API SpeckleVarianceHandle initSpeckleVariance←
ForFile (
    OCTFileHandle File )
```

Initializes the speckle variance contrast processing instance, based on the parameters stored in an OCT file.

Parameters

in	File	A handle to the OCT-File used to create the speckle variance processing routines from.
----	------	--

```
8.1.3.14 setLog() setLog (
    LogOutputType Type,
    const char * Filename )
```

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

Parameters

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

8.1.3.15 setSpeckleVariancePropertyFloat() void setSpeckleVariancePropertyFloat (
SpeckleVarianceHandle Handle,
SpeckleVariancePropertyFloat Property,
double value)

Returns the value of the given integer property.

8.1.3.16 setSpeckleVariancePropertyInt() void setSpeckleVariancePropertyInt (
SpeckleVarianceHandle Handle,
SpeckleVariancePropertyInt Property,
int value)

Sets the given integer property to the given value.

8.1.3.17 setSpeckleVarianceType() void setSpeckleVarianceType (
SpeckleVarianceHandle SpeckleVar,
SpeckleVarianceType Type)

Sets the speckle variance type to the given value.

8.1.3.18 visualCalibrate_1st_CameraScaling() BOOL visualCalibrate_1st_CameraScaling (
VisualCalibrationHandle Handle,
ProbeHandle Probe,
ColoredDataHandle Image)

This is the first step in visual calibration. For this, the calibration the target needs to be placed under the objective.
Returns TRUE if the first step succeeds.

Parameters

in	<i>Handle</i>	A handle of a valid (non null) visual calibration (VisualCalibrationHandle), previously created with the function createVisualCalibration .
in	<i>Probe</i>	A valid (non null) probe handle (ProbeHandle), previously generated with the function initProbe .
in	<i>Image</i>	Video snapshot to use for calibration

Warning

ThorImageOCT uses this and other functions to calibrate the galvo, assuming a very specific sequence of actions and conditions, as explained in the ThorImageOCT. For this function to properly work, the user need to re-create the same sequence of actions and conditions. Please use the ThorImageOCT software to perform probe calibrations, if at all necessary.

8.1.3.19 visualCalibrate_2nd_Galvo() `BOOL visualCalibrate_2nd_Galvo (`
`VisualCalibrationHandle Handle,`
`ProbeHandle Probe,`
`ColoredDataHandle Image)`

This is the second step in visual calibration. For this, the calibration target or and infrared vieweing card needs to be placed under the objective. Returns TRUE if the second step succeeds.

Parameters

in	<i>Handle</i>	A handle of a valid (non null) visual calibration (VisualCalibrationHandle), previously created with the function createVisualCalibration .
in	<i>Probe</i>	A valid (non null) probe handle (ProbeHandle), previously generated with the function initProbe .
in	<i>Image</i>	Video snapshot to use for calibration

It is assumed that the function [visualCalibrate_1st_CameraScaling](#) has been previously successfully invoked.

Warning

ThorImageOCT uses this and other functions to calibrate the galvo, assuming a very specific sequence of actions and conditions, as explained in the ThorImageOCT. For this function to properly work, the user need to re-create the same sequence of actions and conditions. Please use the ThorImageOCT software to perform probe calibrations, if at all necessary.

8.1.3.20 visualCalibrate_previewImage() `BOOL visualCalibrate_previewImage (`
`VisualCalibrationHandle Handle,`
`ColoredDataHandle Image)`

Provides a preview image for the current calibration.

Parameters

in	<i>Handle</i>	A handle of a valid (non null) visual calibration (VisualCalibrationHandle), previously created with the function createVisualCalibration .
out	<i>Image</i>	A valid (non null) handle of colored data (ColoredDataHandle). The preview image will be written here.

Warning

ThorImageOCT uses this and other functions to calibrate the galvo, assuming a very specific sequence of actions and conditions, as explained in the ThorImageOCT. For this function to properly work, the user need to re-create the same sequence of actions and conditions. Please use the ThorImageOCT software to perform probe calibrations, if at all necessary.

8.1.3.21 visualCalibrate_Status() `const char * visualCalibrate_Status (VisualCalibrationHandle Handle)`

Gives a status message of the currently executed visual calibration.

Parameters

in	<i>Handle</i>	A handle of a valid (non null) visual calibration (VisualCalibrationHandle), previously created with the function createVisualCalibration .
----	---------------	---

Returns

The status message of the currently executed visual calibration.

Warning

ThorImageOCT uses this and other functions to calibrate the galvo, assuming a very specific sequence of actions and conditions, as explained in the ThorImageOCT. For this function to properly work, the user need to re-create the same sequence of actions and conditions. Please use the ThorImageOCT software to perform probe calibrations, if at all necessary.

8.1.3.22 visualCalibration_getHoles() `void visualCalibration_getHoles (VisualCalibrationHandle Handle,`

```
int * x0,
int * y0,
int * x1,
int * y1,
int * x2,
int * y2 )
```

provides currently located hole positions of the three-hole target.

Parameters

in	<i>Handle</i>	A handle of a valid (non null) visual calibration (VisualCalibrationHandle), previously created with the function createVisualCalibration .
out	<i>x0</i>	The x-coordinate of the first hole.
out	<i>y0</i>	The y-coordinate of the first hole.
out	<i>x1</i>	The x-coordinate of the second hole.
out	<i>y1</i>	The y-coordinate of the second hole.
out	<i>x2</i>	The x-coordinate of the third hole.
out	<i>y2</i>	The y-coordinate of the third hole.

Warning

ThorImageOCT uses this and other functions to calibrate the galvo, assuming a very specific sequence of actions and conditions, as explained in the ThorImageOCT. For this function to properly work, the user need to re-create the same sequence of actions and conditions. Please use the ThorImageOCT software to perform probe calibrations, if at all necessary.

8.2 SpectralRadar.h

```

00001 #ifndef _SPECTRALRADAR_H
00002 #define _SPECTRALRADAR_H
00003
00004 #include <stddef.h>
00005 #include <time.h>
00006
00007 /* The global SpectralRadar.h contains all function definition, as well as doxygen docs for
   introduction, groups, and the respective functions
00008 * SpectralRadar_Types.h includes all enum type definitions and callback definitions & their doxygen
   documentation (except for properties required for getter/setter functions
00009 * SpectralRadar_Properties.h includes all property enums for getter/setters & their documentation
00010 * SepctralRadar_Handles.h contains all handles and their documentation */
00011
00012 #include "SpectralRadar_Types.h"
00013 #include "SpectralRadar_Properties.h"
00014 #include "SpectralRadar_Handles.h"
00015
00016 #ifdef __cplusplus
00017     #ifdef SPECTRALRADAR_EXPORTS
00018         #define SPECTRALRADAR_API __declspec(dllexport)
00019     #else
00020         #define SPECTRALRADAR_API __declspec(dllimport)
00021     #endif
00022 #else
00023     #define SPECTRALRADAR_API
00024 #endif
00025
00026 #ifdef __cplusplus
00027 extern "C" {
00028 #endif
00029     SPECTRALRADAR_API void setErrorsPerThreadFlag(bool getErrorsPerThread);
00030
00031
00032     SPECTRALRADAR_API ErrorCode isError(void);
00033
00034     SPECTRALRADAR_API ErrorCode getError(char* Message, int StringSize);
00035
00036
00037     SPECTRALRADAR_API void setLog(LogOutputType Type, const char* Filename);
00038
00039
00040     SPECTRALRADAR_API int getDataPropertyInt(DataHandle Data, DataPropertyInt Selection);
00041
00042     SPECTRALRADAR_API double getDataPropertyFloat(DataHandle Data, DataPropertyFloat Selection);
00043
00044     SPECTRALRADAR_API void copyData(DataHandle DataSource, DataHandle DataDestination);
00045
00046     SPECTRALRADAR_API void copyDialogContent(DataHandle DataSource, float* Destination);
00047
00048     SPECTRALRADAR_API float* getDataPtr(DataHandle Data);
00049
00050     SPECTRALRADAR_API void reserveData(DataHandle Data, int Size1, int Size2, int Size3);
00051
00052     SPECTRALRADAR_API void resizeData(DataHandle Data, int Size1, int Size2, int Size3);
00053
00054     SPECTRALRADAR_API void setDataRange(DataHandle Data, double range1, double range2, double
   range3);
00055
00056     SPECTRALRADAR_API void setDataContent(DataHandle Data, float* NewContent);
00057
00058     SPECTRALRADAR_API DataOrientation getDataOrientation(DataHandle Data);
00059
00060     SPECTRALRADAR_API void setDataOrientation(DataHandle Data, DataOrientation Orientation);
00061
00062     SPECTRALRADAR_API int getComplexDataPropertyInt(ComplexDataHandle Data, DataPropertyInt
   Selection);
00063
00064

```

```
00380     SPECTRALRADAR_API double getComplexDataPropertyFloat(ComplexDataHandle Data, DataPropertyFloat
Selection);
00381
00390     SPECTRALRADAR_API void copyComplexDataContent(ComplexDataHandle DataSource, ComplexFloat*
Destination);
00391
00397     SPECTRALRADAR_API void copyComplexData(ComplexDataHandle DataSource, ComplexDataHandle
DataDestination);
00398
00406     SPECTRALRADAR_API ComplexFloat* getComplexDataPtr(ComplexDataHandle Data);
00407
00416     SPECTRALRADAR_API void setComplexDataContent(ComplexDataHandle Data, ComplexFloat*
NewContent);
00417
00426     SPECTRALRADAR_API void reserveComplexData(ComplexDataHandle Data, int Size1, int Size2, int
Size3);
00427
00436     SPECTRALRADAR_API void resizeComplexData(ComplexDataHandle Data, int Size1, int Size2, int
Size3);
00437
00445     SPECTRALRADAR_API void setComplexDataRange(ComplexDataHandle Data, double range1, double
range2, double range3);
00446
00453     SPECTRALRADAR_API int getColoredDataPropertyInt(ColoredDataHandle ColData, DataPropertyInt
Selection);
00454
00461     SPECTRALRADAR_API double getColoredDataPropertyFloat(ColoredDataHandle ColData,
DataPropertyFloat Selection);
00462
00468     SPECTRALRADAR_API void copyColoredData(ColoredDataHandle ImageSource, ColoredDataHandle
ImageDestination);
00469
00478     SPECTRALRADAR_API void copyColoredDataContent(ColoredDataHandle Source, unsigned long*
Destination);
00479
00489     SPECTRALRADAR_API void copyColoredDataContentAligned(ColoredDataHandle ImageSource, unsigned
long* Destination, int Stride);
00490
00498     SPECTRALRADAR_API unsigned long* getColoredDataPtr(ColoredDataHandle ColData);
00499
00508     SPECTRALRADAR_API void resizeColoredData(ColoredDataHandle ColData, int Size1, int Size2, int
Size3);
00509
00518     SPECTRALRADAR_API void reserveColoredData(ColoredDataHandle ColData, int Size1, int Size2, int
Size3);
00519
00529     SPECTRALRADAR_API void setColoredDataContent(ColoredDataHandle ColData, unsigned long*
NewContent);
00530
00538     SPECTRALRADAR_API void setColoredDataRange(ColoredDataHandle Data, double range1, double
range2, double range3);
00539
00545     SPECTRALRADAR_API DataOrientation getColoredDataOrientation(ColoredDataHandle Data);
00546
00552     SPECTRALRADAR_API void setColoredDataOrientation(ColoredDataHandle Data, DataOrientation
Orientation);
00553
00565     SPECTRALRADAR_API void copyRawDataContent(RawDataHandle RawDataSource, void* DataContent);
00566
00576     SPECTRALRADAR_API void copyRawData(RawDataHandle RawDataSource, RawDataHandle RawDataTarget);
00577
00586     SPECTRALRADAR_API void* getRawDataPtr(RawDataHandle RawDataSource);
00587
00596     SPECTRALRADAR_API int getRawDataPropertyInt(RawDataHandle RawData, RawDataPropertyInt
Property);
00597
00606     SPECTRALRADAR_API double getRawDataPropertyFloat(RawDataHandle Data, RawDataPropertyFloat
Property);
00607
00616     SPECTRALRADAR_API void setRawDataBytesPerPixel(RawDataHandle Raw, int BytesPerPixel);
00617
00627     SPECTRALRADAR_API void reserveRawData(RawDataHandle Raw, int Size1, int Size2, int Size3);
00628
00638     SPECTRALRADAR_API void resizeRawData(RawDataHandle Raw, int Size1, int Size2, int Size3);
00639
00651     SPECTRALRADAR_API void setRawDataContent(RawDataHandle RawData, void* NewContent);
00652
00670     SPECTRALRADAR_API void setScanSpectra(RawDataHandle RawData, int NumberOfScanRegions, int*
ScanRegions);
00671
00689     SPECTRALRADAR_API void setApodizationSpectra(RawDataHandle RawData, int NumberOfApoRegions,
int* ApodizationRegions);
00690
00702     SPECTRALRADAR_API int getNumberOfScanRegions(RawDataHandle Raw);
00703
00715     SPECTRALRADAR_API int getNumberOfApodizationRegions(RawDataHandle Raw);
00716
```

```

00729     SPECTRALRADAR_API void getScanSpectra(RawDataHandle Raw, int* SpectraIndex);
00730
00743     SPECTRALRADAR_API void getApodizationSpectra(RawDataHandle Raw, int* SpectraIndex);
00744
00754     SPECTRALRADAR_API RawDataHandle createRawData(void);
00755
00762     SPECTRALRADAR_API void clearRawData(RawDataHandle Raw);
00763
00768     SPECTRALRADAR_API DataHandle createData(void);
00769
00775     SPECTRALRADAR_API DataHandle createGradientData(int Size);
00776
00781     SPECTRALRADAR_API void clearData(DataHandle Data);
00782
00787     SPECTRALRADAR_API ColoredDataHandle createColoredData(void);
00788
00793     SPECTRALRADAR_API void clearColoredData(ColoredDataHandle Volume);
00794
00799     SPECTRALRADAR_API ComplexDataHandle createComplexData(void);
00800
00805     SPECTRALRADAR_API void clearComplexData(ComplexDataHandle Data);
00806
00807
00817     SPECTRALRADAR_API OCTDeviceHandle initDevice(void);
00818
00825     SPECTRALRADAR_API int getDevicePropertyInt(OCTDeviceHandle Dev, DevicePropertyInt Selection);
00826
00834     SPECTRALRADAR_API const char* getDevicePropertyString(OCTDeviceHandle Dev,
DevicePropertyString Selection);
00835
00842     SPECTRALRADAR_API double getDevicePropertyFloat(OCTDeviceHandle Dev, DevicePropertyFloat Selection);
00843
00845     SPECTRALRADAR_API BOOL getDeviceFlag(OCTDeviceHandle Dev, DeviceFlag Selection);
00851
00858     SPECTRALRADAR_API void setDeviceFlag(OCTDeviceHandle Dev, DeviceFlag Selection, BOOL Value);
00859
00865     SPECTRALRADAR_API BOOL getStaticDeviceFlag(StaticDeviceFlag Selection);
00866
00872     SPECTRALRADAR_API void setStaticDeviceFlag(StaticDeviceFlag Selection, BOOL Value);
00873
00879     SPECTRALRADAR_API void closeDevice(OCTDeviceHandle Dev);
00880
00888     SPECTRALRADAR_API void moveScanner(OCTDeviceHandle Dev, ProbeHandle Probe, ScanAxis Axis,
double Position_mm);
00889
00895     SPECTRALRADAR_API void moveScannerToApoPosition(OCTDeviceHandle Dev, ProbeHandle Probe);
00896
00907     SPECTRALRADAR_API int getNumberOfDevicePresetCategories(OCTDeviceHandle Dev);
00908
00922     SPECTRALRADAR_API const char* getDevicePresetCategoryName(OCTDeviceHandle Dev, int Category);
00923
00937     SPECTRALRADAR_API int getDevicePresetCategoryIndex(OCTDeviceHandle Dev, const char* Name);
00938
00954     SPECTRALRADAR_API void setDevicePreset(OCTDeviceHandle Dev, int Category, ProbeHandle Probe,
ProcessingHandle Proc, int Preset);
00955
00972     SPECTRALRADAR_API int getDevicePreset(OCTDeviceHandle Dev, int Category);
00973
00991     SPECTRALRADAR_API const char* getDevicePresetDescription(OCTDeviceHandle Dev, int Category,
int Preset);
00992
01008     SPECTRALRADAR_API int getNumberOfDevicePresets(OCTDeviceHandle Dev, int Category);
01009
01014     SPECTRALRADAR_API void setRequiredSLDOnTime_s(int Time_s);
01015
01019     SPECTRALRADAR_API void resetCamera(void);
01020
01028     SPECTRALRADAR_API BOOL isDeviceAvailable(void);
01029
01037     SPECTRALRADAR_API DeviceState getDeviceState(void);
01038
01048     SPECTRALRADAR_API void getDeviceError(char* ErrorMsg, int StringSize);
01049
01058     SPECTRALRADAR_API int getNumberOfInternalDeviceValues(OCTDeviceHandle Dev);
01059
01073     SPECTRALRADAR_API void getInternalDeviceValueName(OCTDeviceHandle Dev, int Index, char* Name,
int NameStringSize, char* Unit, int UnitStringSize);
01074
01083     SPECTRALRADAR_API double getInternalDeviceValueByName(OCTDeviceHandle Dev, const char* Name);
01084
01093     SPECTRALRADAR_API double getInternalDeviceValueByIndex(OCTDeviceHandle Dev, int Index);
01094
01097
01101     SPECTRALRADAR_API int getNumberOfOutputDeviceValues(OCTDeviceHandle Dev);
01102
01106     SPECTRALRADAR_API void getOutputDeviceValueName(OCTDeviceHandle Dev, int Index, char* Name,

```

```
    int NameStringSize, char* Unit, int UnitStringSize);
01107    SPECTRALRADAR_API BOOL doesOutputDeviceValueExist(OCTDeviceHandle Dev, const char* Name);
01112    SPECTRALRADAR_API void setOutputDeviceValueByName(OCTDeviceHandle Dev, const char* Name,
01113        double value);
01117    SPECTRALRADAR_API void setInternalDeviceValueByIndex(OCTDeviceHandle Dev, int Index, double
01118        Value);
01123    SPECTRALRADAR_API void setOutputDeviceValueByIndex(OCTDeviceHandle Dev, int Index, double
01124        Value);
01129    SPECTRALRADAR_API void getOutputDeviceValueRangeByName(OCTDeviceHandle Dev, const char* Name,
01130        double* Min, double* Max);
01135    SPECTRALRADAR_API void getOutputDeviceValueRangeByIndex(OCTDeviceHandle Dev, int Index,
01136        double* Min, double* Max);
01141
01152
01153
01159    SPECTRALRADAR_API ProbeHandle initCurrentProbe(OCTDeviceHandle Dev);
01160
01174    SPECTRALRADAR_API ProbeHandle initProbe(OCTDeviceHandle Dev, const char* ProbeFile);
01175
01183    SPECTRALRADAR_API ProbeHandle initDefaultProbe(OCTDeviceHandle Dev, const char* Type, const
01184        char* Objective);
01191    SPECTRALRADAR_API ProbeHandle initProbeFromOCTFile(OCTDeviceHandle Dev, OCTFileHandle File);
01192
01202    SPECTRALRADAR_API void saveProbe(ProbeHandle Probe, const char* ProbeFile);
01203
01211    SPECTRALRADAR_API void setProbeParameterInt(ProbeHandle Probe, ProbeParameterInt Selection,
01212        int Value);
01220    SPECTRALRADAR_API void setProbeParameterFloat(ProbeHandle Probe, ProbeParameterFloat
01221        Selection, double Value);
01229    SPECTRALRADAR_API int getProbeParameterInt(ProbeHandle Probe, ProbeParameterInt Selection);
01230
01238    SPECTRALRADAR_API double getProbeParameterFloat(ProbeHandle Probe, ProbeParameterFloat
01239        Selection);
01247    SPECTRALRADAR_API BOOL getProbeFlag(ProbeHandle Probe, ProbeFlag Selection);
01248
01256    SPECTRALRADAR_API void setProbeParameterString(ProbeHandle Probe, ProbeParameterString
01257        Selection, const char* Value);
01266    SPECTRALRADAR_API const char* getProbeParameterString(ProbeHandle Probe, ProbeParameterString
01267        Selection);
01268
01278    SPECTRALRADAR_API void positionToProbeVoltage(OCTDeviceHandle Handle, ProbeHandle Probe,
01279        double Position_X_mm, double Position_Y_mm, double * Volt_X, double * Volt_Y);
01286    SPECTRALRADAR_API const char* getProbeType(ProbeHandle Probe);
01287
01294    SPECTRALRADAR_API void setProbeType(ProbeHandle Probe, const char* Type);
01295
01302    SPECTRALRADAR_API void closeProbe(ProbeHandle Probe);
01303
01304
01305
01317    SPECTRALRADAR_API void CameraPixelToPosition(ProbeHandle Probe, ColoredDataHandle Image, int
01318        PixelX, int PixelY, double* PosX, double* PosY);
01329    SPECTRALRADAR_API void PositionToCameraPixel(ProbeHandle Probe, ColoredDataHandle Image,
01330        double PosX, double PosY, int* PixelX, int* PixelY);
01339    SPECTRALRADAR_API void visualizeScanPatternOnDevice(OCTDeviceHandle Dev, ProbeHandle Probe,
01340        ScanPatternHandle Pattern, BOOL ShowRawPattern);
01348    SPECTRALRADAR_API void visualizeScanPatternOnImage(ProbeHandle Probe, ScanPatternHandle
01349        ScanPattern, ColoredDataHandle VideoImage);
01362    SPECTRALRADAR_API ScanPatternHandle createNoScanPattern(ProbeHandle Probe, int AScans, int
01363        NumberOfScans);
01372    SPECTRALRADAR_API ScanPatternHandle createAScanPattern(ProbeHandle Probe, int AScans, double
01373        PosX_mm, double PosY_mm);
01388    SPECTRALRADAR_API ScanPatternHandle createBScanPattern(ProbeHandle Probe, double Range_mm, int
01389        AScans);
01401    SPECTRALRADAR_API ScanPatternHandle createBScanPatternManual(ProbeHandle Probe, double
01402        StartX_mm, double StartY_mm, double StopX_mm, double StopY_mm, int AScans);
```

```

01412     SPECTRALRADAR_API ScanPatternHandle createIdealBScanPattern(ProbeHandle Probe, double
01413         Range_mm, int AScans);
01423     SPECTRALRADAR_API ScanPatternHandle createCirclePattern(ProbeHandle Probe, double Radius_mm,
01424         int AScans);
01458     SPECTRALRADAR_API ScanPatternHandle createVolumePattern(ProbeHandle Probe, double RangeX_mm,
01459         int SizeX, double RangeY_mm, int SizeY, ScanPatternApodizationType ApoType,
01460         ScanPatternAcquisitionOrder AcqOrder);
01497     SPECTRALRADAR_API ScanPatternHandle createVolumePatternEx(ProbeHandle Probe, double RangeX_mm,
01498         int SizeX, double RangeY_mm, int SizeY, double CenterX_mm, double CenterY_mm, double Angle_rad,
01499         ScanPatternApodizationType ApoType, ScanPatternAcquisitionOrder AcqOrder);
01503     SPECTRALRADAR_API void updateScanPattern(ScanPatternHandle Pattern);
01512     SPECTRALRADAR_API void rotateScanPattern(ScanPatternHandle Pattern, double Angle_rad);
01513
01527     SPECTRALRADAR_API void rotateScanPatternEx(ScanPatternHandle Pattern, double Angle_rad, int
01528         Index);
01537     SPECTRALRADAR_API void shiftScanPattern(ScanPatternHandle Pattern, double ShiftX_mm, double
01538         ShiftY_mm);
01554     SPECTRALRADAR_API void shiftScanPatternEx(ScanPatternHandle Pattern, double ShiftX_mm, double
01555         ShiftY_mm, BOOL ShiftApo, int Index);
01562     SPECTRALRADAR_API void zoomScanPattern(ScanPatternHandle Pattern, double Factor);
01563
01571     SPECTRALRADAR_API int getScanPatternLUTSize(ScanPatternHandle Pattern);
01572
01583     SPECTRALRADAR_API void getScanPatternLUT(ScanPatternHandle Pattern, double* VoltX, double*
01584         VoltY);
01590     SPECTRALRADAR_API int getScanPointsSize(ScanPatternHandle Pattern);
01591
01600     SPECTRALRADAR_API void getScanPoints(ScanPatternHandle Pattern, double* PosX_mm, double*
01601         PosY_mm);
01606     SPECTRALRADAR_API void clearScanPattern(ScanPatternHandle Pattern);
01607
01610
01611
01612
01626     SPECTRALRADAR_API ScanPatternHandle createFreeformScanPattern2D(ProbeHandle Probe, double*
01627         PosX_mm, double* PosY_mm, int Size, int AScans,
01628             InterpolationMethod InterpolationMethod, BOOL CloseScanPattern);
01644     SPECTRALRADAR_API ScanPatternHandle createFreeformScanPattern2DFromLUT(ProbeHandle Probe,
01645         double* PosX_mm, double* PosY_mm, int Size, BOOL ClosedScanPattern);
01664     SPECTRALRADAR_API ScanPatternHandle createFreeformScanPattern3DFromLUT(ProbeHandle Probe,
01665         double* PosX_mm, double* PosY_mm, int AScansPerBScan, int NumberOfBScans,
01666             BOOL ClosedScanPattern, ScanPatternApodizationType ApoType,
01667             ScanPatternAcquisitionOrder AcqOrder);
01690     SPECTRALRADAR_API ScanPatternHandle createFreeformScanPattern3D(ProbeHandle Probe, double*
01691         PosX_mm, double* PosY_mm, int* ScanIndices, int Size, int NumberOfAScansPerBScan,
01692             InterpolationMethod InterpolationMethod, BOOL CloseScanPattern,
01693             ScanPatternApodizationType ApoType, ScanPatternAcquisitionOrder AcqOrder);
01706     SPECTRALRADAR_API void interpolatePoints2D(double* OrigPosX, double* OrigPosY, int Size,
01707         double* InterpPosX, double* InterpPosY, int NewSize,
01708             InterpolationMethod InterpolationMet, BoundaryCondition BoundaryCond);
01721     SPECTRALRADAR_API void inflatePoints(double* PosX, double* PosY, int Size, double*
01722         InflatedPosX, double* InflatedPosY, int NumberOfInflationLines,
01723             double RangeOfInflation, InflationMethod Method);
01734     SPECTRALRADAR_API void saveScanPointsToFile(double* ScanPosX_mm, double* ScanPosY_mm, int*
01735         ScanIndices, int Size, const char* Filename, ScanPointsDataFormat DataFormat);
01742     SPECTRALRADAR_API int getSizeOfScanPointsFromFile(const char* Filename, ScanPointsDataFormat
01743         DataFormat);
01754     SPECTRALRADAR_API void loadScanPointsFromFile(double* ScanPosX_mm, double* ScanPosY_mm, int*
01755         ScanIndices, int Size, const char* Filename, ScanPointsDataFormat DataFormat);
01763     SPECTRALRADAR_API int getSizeOfScanPointsFromDataHandle(DataHandle ScanPoints);
01764
01774     SPECTRALRADAR_API void getScanPointsFromDataHandle(DataHandle ScanPoints, double* PosX_mm,
01775         double* PosY_mm, int* ScanIndices, int Length);
01785     SPECTRALRADAR_API DataHandle createDataHandleFromScanPoints(double* PosX_mm, double* PosY_mm,
01786         int* ScanIndices, int Length);
01786

```

```
01789
01790
01794     SPECTRALRADAR_API size_t projectMemoryRequirement(OCTDeviceHandle Handle, ScanPatternHandle
Pattern, AcquisitionType type);
01795
01810     SPECTRALRADAR_API void startMeasurement(OCTDeviceHandle Dev, ScanPatternHandle Pattern,
AcquisitionType Type);
01811
01824     SPECTRALRADAR_API void getRawData(OCTDeviceHandle Dev, RawDataHandle RawData);
01825
01844     SPECTRALRADAR_API void getRawDataEx(OCTDeviceHandle Dev, RawDataHandle RawData, int
CameraIdx);
01845
01854     SPECTRALRADAR_API void getAnalogInputData(OCTDeviceHandle Dev, DataHandle Output);
01855
01865     SPECTRALRADAR_API void getAnalogInputDataEx(OCTDeviceHandle Dev, DataHandle Data, DataHandle
ApoData);
01866
01874     SPECTRALRADAR_API void setAnalogInputChannelEnabled(OCTDeviceHandle Dev, int ChannelIndex,
bool Enable);
01875
01882     SPECTRALRADAR_API bool getAnalogInputChannelEnabled(OCTDeviceHandle Dev, int ChannelIndex);
01883
01890     SPECTRALRADAR_API const char* getAnalogInputChannelName(OCTDeviceHandle Dev, int
ChannelIndex);
01891
01896     SPECTRALRADAR_API void stopMeasurement(OCTDeviceHandle Dev);
01897
01913     SPECTRALRADAR_API void measureSpectra(OCTDeviceHandle Dev, int NumberOfSpectra, RawDataHandle
Raw);
01914
01937     SPECTRALRADAR_API void measureSpectraEx(OCTDeviceHandle Dev, int NumberOfSpectra,
RawDataHandle Raw, int CameraIndex);
01938
01957     SPECTRALRADAR_API void measureSpectraContinuousEx(OCTDeviceHandle Dev, int NumberOfSpectra,
RawDataHandle Raw, int CameraIndex);
01958
01961
01962
01978     SPECTRALRADAR_API ProcessingHandle createProcessing(int SpectrumSize, int BytesPerRawPixel,
BOOL Signed, float ScalingFactor, float MinElectrons, Processing_FFTType Type, float
FFTOversampling);
01979
01990     SPECTRALRADAR_API ProcessingHandle createProcessingForDevice(OCTDeviceHandle Dev);
01991
02003     SPECTRALRADAR_API ProcessingHandle createProcessingForDeviceEx(OCTDeviceHandle Dev, int
CameraIndex);
02004
02011     SPECTRALRADAR_API ProcessingHandle createProcessingForOCTFile(OCTFileHandle File);
02012
02022     SPECTRALRADAR_API ProcessingHandle createProcessingForOCTFileEx(OCTFileHandle File, const int
CameraIndex);
02023
02033     SPECTRALRADAR_API int getInputSize(ProcessingHandle Proc);
02034
02047     SPECTRALRADAR_API int getAScanSize(ProcessingHandle Proc);
02048
02064     SPECTRALRADAR_API void setApodizationWindow(ProcessingHandle Proc, ApodizationWindow Window);
02065
02074     SPECTRALRADAR_API ApodizationWindow getApodizationWindow(ProcessingHandle Proc);
02075
02085     SPECTRALRADAR_API void setApodizationWindowParameter(ProcessingHandle Proc,
ApodizationWindowParameter Selection, double Value);
02086
02096     SPECTRALRADAR_API double getApodizationWindowParameter(ProcessingHandle Proc,
ApodizationWindowParameter Selection);
02097
02118     SPECTRALRADAR_API void getCurrentApodizationEdgeChannels(ProcessingHandle Proc, int* LeftPix,
int* RightPix);
02119
02128     SPECTRALRADAR_API void setProcessingDechirpAlgorithm(ProcessingHandle Proc, Processing_FFTType
Type, float Oversampling);
02129
02138     SPECTRALRADAR_API void setProcessingParameterInt(ProcessingHandle Proc, ProcessingParameterInt
Selection, int Value);
02139
02148     SPECTRALRADAR_API int getProcessingParameterInt(ProcessingHandle Proc, ProcessingParameterInt
Selection);
02149
02158     SPECTRALRADAR_API void setProcessingParameterFloat(ProcessingHandle Proc,
ProcessingParameterFloat Selection, double Value);
02159
02168     SPECTRALRADAR_API double getProcessingParameterFloat(ProcessingHandle Proc,
ProcessingParameterFloat Selection);
02169
02178     SPECTRALRADAR_API void setProcessingFlag(ProcessingHandle Proc, ProcessingFlag Flag, BOOL
Value);
```

```

02179     SPECTRALRADAR_API BOOL getProcessingFlag(ProcessingHandle Proc, ProcessingFlag Flag);
02180
02181     SPECTRALRADAR_API void setProcessingAveragingAlgorithm(ProcessingHandle Proc,
02182     ProcessingAveragingAlgorithm Algorithm);
02183
02184     SPECTRALRADAR_API void setCalibration(ProcessingHandle Proc, CalibrationData Selection,
02185     DataHandle Data);
02186
02187     SPECTRALRADAR_API void getCalibration(ProcessingHandle Proc, CalibrationData Selection,
02188     DataHandle Data);
02189
02190     SPECTRALRADAR_API void measureCalibration(OCTDeviceHandle Dev, ProcessingHandle Proc,
02191     CalibrationData Selection);
02192
02193     SPECTRALRADAR_API void measureCalibrationEx(OCTDeviceHandle Dev, ProcessingHandle Proc,
02194     CalibrationData Selection, int CameraIndex);
02195
02196     SPECTRALRADAR_API void measureApodizationSpectra(OCTDeviceHandle Dev, ProbeHandle Probe,
02197     ProcessingHandle Proc);
02198
02199     SPECTRALRADAR_API void saveCalibrationDefault(ProcessingHandle Proc, CalibrationData
02200     Selection);
02201
02202     SPECTRALRADAR_API void saveCalibrationDefaultEx(ProcessingHandle Proc, CalibrationData
02203     Selection, int CameraIndex);
02204
02205     SPECTRALRADAR_API void saveCalibration(ProcessingHandle Proc, CalibrationData Selection, const
02206     char* Path);
02207
02208     SPECTRALRADAR_API void loadCalibration(ProcessingHandle Proc, CalibrationData Selection, const
02209     char* Path);
02210
02211     SPECTRALRADAR_API void setSpectrumOutput(ProcessingHandle Proc, DataHandle Spectrum);
02212
02213     SPECTRALRADAR_API void setOffsetCorrectedSpectrumOutput(ProcessingHandle Proc, DataHandle
02214     OffsetCorrectedSpectrum);
02215
02216     SPECTRALRADAR_API void setDCCorrectedSpectrumOutput(ProcessingHandle Proc, DataHandle
02217     DCCorrectedSpectrum);
02218
02219     SPECTRALRADAR_API void setApodizedSpectrumOutput(ProcessingHandle Proc, DataHandle
02220     ApodizedSpectrum);
02221
02222     SPECTRALRADAR_API void setComplexDataOutput(ProcessingHandle Proc, ComplexDataHandle
02223     ComplexScan);
02224
02225     SPECTRALRADAR_API void setProcessedDataOutput(ProcessingHandle Proc, DataHandle Scan);
02226
02227     SPECTRALRADAR_API void setColoredDataOutput(ProcessingHandle Proc, ColoredDataHandle Scan,
02228     ColoringHandle Color);
02229
02230     SPECTRALRADAR_API void setTransposedColoredDataOutput(ProcessingHandle Proc, ColoredDataHandle
02231     Scan, ColoringHandle Color);
02232
02233     SPECTRALRADAR_API void executeProcessing(ProcessingHandle Proc, RawDataHandle RawData);
02234
02235     SPECTRALRADAR_API void finishProcessing(ProcessingHandle Proc, ComplexDataHandle ComplexData);
02236
02237     SPECTRALRADAR_API void clearProcessing(ProcessingHandle Proc);
02238
02239     SPECTRALRADAR_API void computeDispersion(DataHandle Spectrum1, DataHandle Spectrum2,
02240     DataHandle Chirp, DataHandle Disp);
02241
02242     SPECTRALRADAR_API void computeDispersionByCoeff(double Quadratic, DataHandle Chirp, DataHandle
02243     Disp);
02244
02245     SPECTRALRADAR_API void computeDispersionByImage(DataHandle LinearKSpectra, DataHandle Chirp,
02246     DataHandle Disp);
02247
02248     SPECTRALRADAR_API int getNumberOfDispersionPresets(ProcessingHandle Proc);
02249
02250     SPECTRALRADAR_API const char* getDispersionPresetName(ProcessingHandle Proc, int Index);
02251
02252     SPECTRALRADAR_API void setDispersionPresetByName(ProcessingHandle Proc, const char* Name);
02253
02254     SPECTRALRADAR_API void setDispersionPresetByIndex(ProcessingHandle Proc, int Index);
02255
02256     SPECTRALRADAR_API void setDispersionPresets(ProcessingHandle Proc, ProbeHandle Probe);
02257
02258     SPECTRALRADAR_API Processing_FFTType getProcessing_FFTType(ProcessingHandle Proc);
02259
02260     SPECTRALRADAR_API void setDispersionCorrectionType(ProcessingHandle Proc,
02261     DispersionCorrectionType Type);
02262
02263     SPECTRALRADAR_API DispersionCorrectionType getDispersionCorrectionType(ProcessingHandle Proc);
02264

```

```
02608     SPECTRALRADAR_API void setDispersionQuadraticCoeff(ProcessingHandle Proc, double Coeff);
02609
02617     SPECTRALRADAR_API double getDispersionQuadraticCoeff(ProcessingHandle Proc);
02618
02626     SPECTRALRADAR_API const char* getCurrentDispersionPresetName(ProcessingHandle Proc);
02627
02630
02631
02639     SPECTRALRADAR_API void exportData(DataHandle Data, DataExportFormat Format, const char*
02640     FileName);
02653         SPECTRALRADAR_API void exportDataAsImage(DataHandle Data, ColoringHandle Color,
02653         ColoredDataExportFormat Format, Direction SliceNormalDirection, const char* FileName, int
02654         ExportOptionMask);
02662         SPECTRALRADAR_API void exportComplexData(ComplexDataHandle Data, ComplexDataExportFormat
02663         Format, const char* FileName);
02674         SPECTRALRADAR_API void exportColoredData(ColoredDataHandle Data, ColoredDataExportFormat
02675         Format, Direction SliceNormalDirection, const char* FileName, int ExportOptionMask);
02675
02683         SPECTRALRADAR_API void importColoredData(ColoredDataHandle ColoredData, DataImportFormat
02684         Format, const char* FileName);
02692         SPECTRALRADAR_API void importData(DataHandle Data, DataImportFormat Format, const char*
02693         FileName);
02703         SPECTRALRADAR_API void exportRawData(RawDataHandle Raw, RawDataExportFormat Format, const
02704         char* FileName);
02714         SPECTRALRADAR_API void importRawData(RawDataHandle Raw, RawDataImportFormat Format, const
02715         char* FileName);
02722             const int ExportOption_None = 0x00000000;
02727             const int ExportOption_DrawScaleBar = 0x00000001;
02729             const int ExportOption_DrawMarkers = 0x00000002;
02731             const int ExportOption_UsePhysicalAspectRatio = 0x00000004;
02733             const int ExportOption_Flip_X_Axis = 0x00000008;
02735             const int ExportOption_Flip_Y_Axis = 0x00000010;
02737             const int ExportOption_Flip_Z_Axis = 0x00000020;
02739
02742
02743
02755     SPECTRALRADAR_API void appendRawData(RawDataHandle Raw, RawDataHandle DataToAppend, Direction
02756     Dir);
02770         SPECTRALRADAR_API void getRawDataSliceAtIndex(RawDataHandle Raw, RawDataHandle Slice,
02771     Direction SliceNormalDirection, int Index);
02778         SPECTRALRADAR_API double analyzeData(DataHandle Data, DataAnalyzation Selection);
02779
02788         SPECTRALRADAR_API double analyzeAScan(DataHandle Data, AScanAnalyzation Selection);
02789
02800         SPECTRALRADAR_API void analyzePeaksInAScan(DataHandle Data, AScanAnalyzation Selection, int
02801     NumberOfPeaksToAnalyze, int MinDistBetweenPeaks, double* Result);
02809         SPECTRALRADAR_API void transposeData(DataHandle DataIn, DataHandle DataOut);
02810
02816         SPECTRALRADAR_API void transposeDataInplace(DataHandle Data);
02817
02829         SPECTRALRADAR_API void transposeAndScaleData(DataHandle DataIn, DataHandle DataOut, float Min,
02830     float Max);
02830
02837         SPECTRALRADAR_API void normalizeData(DataHandle Data, float Min, float Max);
02838
02855         SPECTRALRADAR_API void getDataSliceAtPos(DataHandle Data, DataHandle Slice, Direction
02856     SliceNormalDirection, double Pos_mm);
02873         SPECTRALRADAR_API void getComplexDataSlicePos(ComplexDataHandle Data, ComplexDataHandle Slice,
02874     Direction SliceNormalDirection, double Pos_mm);
02891         SPECTRALRADAR_API void getColoredDataSlicePos(ColoredDataHandle Data, ColoredDataHandle Slice,
02892     Direction SliceNormalDirection, double Pos_mm);
02907         SPECTRALRADAR_API void getDataSliceAtIndex(DataHandle Data, DataHandle Slice, Direction
02908     SliceNormalDirection, int Index);
02923         SPECTRALRADAR_API void getComplexDataSliceIndex(ComplexDataHandle Data, ComplexDataHandle
02924     Slice, Direction SliceNormalDirection, int Index);
02939         SPECTRALRADAR_API void getColoredDataSliceIndex(ColoredDataHandle Data, ColoredDataHandle
02940     Slice, Direction SliceNormalDirection, int Index);
02951         SPECTRALRADAR_API void computeDataProjection(DataHandle Data, DataHandle Slice, Direction
02952     ProjectionDirection, DataAnalyzation Selection);
02963         SPECTRALRADAR_API void appendData(DataHandle Data, DataHandle DataToAppend, Direction Dir);
```

```

02964
02975     SPECTRALRADAR_API void appendComplexData(ComplexDataHandle Data, ComplexDataHandle
02976         DataToAppend, Direction Dir);
02977
02987     SPECTRALRADAR_API void appendColoredData(ColoredDataHandle Data, ColoredDataHandle
02988         DataToAppend, Direction Dir);
02989
02997     SPECTRALRADAR_API void cropData(DataHandle Data, Direction Dir, int IndexMax, int IndexMin);
02998
03007     SPECTRALRADAR_API void cropComplexData(ComplexDataHandle Data, Direction Dir, int IndexMax,
03008         int IndexMin);
03009
03017     SPECTRALRADAR_API void cropColoredData(ColoredDataHandle Data, Direction Dir, int IndexMax,
03018         int IndexMin);
03019
03028     SPECTRALRADAR_API void separateData(DataHandle Data1, DataHandle Data2, int SeparationIndex,
03029         Direction Dir);
03030
03039     SPECTRALRADAR_API void separateComplexData(ComplexDataHandle Data1, ComplexDataHandle Data2,
03040         int SeparationIndex, Direction Dir);
03041
03050     SPECTRALRADAR_API void separateColoredData(ColoredDataHandle Data1, ColoredDataHandle Data2,
03051         int SeparationIndex, Direction Dir);
03052
03057     SPECTRALRADAR_API void flipData(DataHandle Data, Direction FlippingDir);
03058
03064     SPECTRALRADAR_API void flipComplexData(ComplexDataHandle Data, Direction FlippingDir);
03065
03071     SPECTRALRADAR_API void flipColoredData(ColoredDataHandle Data, Direction FlippingDir);
03072
03077     SPECTRALRADAR_API ImageFieldHandle createImageField(void);
03078
03084     SPECTRALRADAR_API ImageFieldHandle createImageFieldFromProbe(ProbeHandle Probe);
03085
03090     SPECTRALRADAR_API void clearImageField(ImageFieldHandle ImageField);
03091
03098     SPECTRALRADAR_API void saveImageField(ImageFieldHandle ImageField, const char* Path);
03099
03106     SPECTRALRADAR_API void loadImageField(ImageFieldHandle ImageField, const char* Path);
03107
03127     SPECTRALRADAR_API void determineImageField(ImageFieldHandle ImageField, ScanPatternHandle
03128         Pattern, DataHandle Surface);
03129
03154     SPECTRALRADAR_API void determineImageFieldWithMask(ImageFieldHandle ImageField,
03155         ScanPatternHandle Pattern, DataHandle Surface, DataHandle Mask);
03156
03168     SPECTRALRADAR_API void correctImageField(ImageFieldHandle ImageField, ScanPatternHandle
03169         Pattern, DataHandle Data);
03170
03182     SPECTRALRADAR_API void correctImageFieldComplex(ImageFieldHandle ImageField, ScanPatternHandle
03183         Pattern, ComplexDataHandle Data);
03184
03196     SPECTRALRADAR_API void correctSurface(ImageFieldHandle ImageField, ScanPatternHandle Pattern,
03197         DataHandle Surface);
03198
03205     SPECTRALRADAR_API void setImageFieldInProbe(ImageFieldHandle ImageField, ProbeHandle Probe);
03206
03213
03225     SPECTRALRADAR_API VisualCalibrationHandle createVisualCalibration(OCTDeviceHandle Device,
03226         double TargetCornerLength_mm, BOOL CheckAngle, BOOL SaveData);
03227
03235     SPECTRALRADAR_API void clearVisualCalibration(VisualCalibrationHandle Handle);
03236
03247     SPECTRALRADAR_API BOOL visualCalibrate_1st_CameraScaling(VisualCalibrationHandle Handle,
03248         ProbeHandle Probe, ColoredDataHandle Image);
03249
03261     SPECTRALRADAR_API BOOL visualCalibrate_2nd_Galvo(VisualCalibrationHandle Handle, ProbeHandle
03262         Probe, ColoredDataHandle Image);
03263
03272     SPECTRALRADAR_API BOOL visualCalibrate_previewImage(VisualCalibrationHandle Handle,
03273         ColoredDataHandle Image);
03274
03288     SPECTRALRADAR_API void visualCalibration_getHoles(VisualCalibrationHandle Handle, int* x0,
03289         int* y0, int* x1, int* y1, int* x2, int* y2);
03290
03299     SPECTRALRADAR_API const char* visualCalibrate_Status(VisualCalibrationHandle Handle);
03300
03303
03304
03309     SPECTRALRADAR_API DopplerProcessingHandle createDopplerProcessing(void);
03310
03311
03316     SPECTRALRADAR_API DopplerProcessingHandle createDopplerProcessingForFile(OCTFileHandle File);
03317
03318
03326     SPECTRALRADAR_API int getDopplerPropertyInt(DopplerProcessingHandle Handle, DopplerPropertyInt
03327         Property);

```

```
03327
03328
03329
03330
03331     SPECTRALRADAR_API void setDopplerPropertyInt(DopplerProcessingHandle Handle,
03332     DopplerPropertyInt Property, int Value);
03333
03334     SPECTRALRADAR_API double getDopplerPropertyFloat(DopplerProcessingHandle Doppler,
03335     DopplerPropertyFloat Property);
03336
03337     SPECTRALRADAR_API void setDopplerPropertyFloat(DopplerProcessingHandle Handle,
03338     DopplerPropertyFloat Property, float Value);
03339
03340     SPECTRALRADAR_API BOOL getDopplerFlag(DopplerProcessingHandle Handle, DopplerFlag Flag);
03341
03342     SPECTRALRADAR_API void setDopplerFlag(DopplerProcessingHandle Handle, DopplerFlag Flag, BOOL
03343     OnOff);
03344
03345     SPECTRALRADAR_API void setDopplerAmplitudeOutput(DopplerProcessingHandle Handle, DataHandle
03346     AmpOut);
03347
03348     SPECTRALRADAR_API void setDopplerPhaseOutput(DopplerProcessingHandle Handle, DataHandle
03349     PhasesOut);
03350
03351     SPECTRALRADAR_API void executeDopplerProcessing(DopplerProcessingHandle Handle,
03352     ComplexDataHandle Input);
03353
03354     SPECTRALRADAR_API void dopplerPhaseToVelocity(DopplerProcessingHandle Doppler, DataHandle
03355     InOut);
03356
03357     SPECTRALRADAR_API void dopplerVelocityToPhase(DopplerProcessingHandle Doppler, DataHandle
03358     InOut);
03359
03360     SPECTRALRADAR_API void clearDopplerProcessing(DopplerProcessingHandle Handle);
03361
03362     SPECTRALRADAR_API void getDopplerOutputSize(DopplerProcessingHandle Handle, int Size1In, int
03363     Size2In, int* Size1Out, int* Size2Out);
03364
03365     SPECTRALRADAR_API void calcContrast(DataHandle ApodizedSpectrum, DataHandle Contrast);
03366
03367
03368     SPECTRALRADAR_API SettingsHandle initSettingsFile(const char* Path);
03369
03370     SPECTRALRADAR_API int getSettingsEntryInt(SettingsHandle SettingsFile, const char* Node, int
03371     DefaultValue);
03372
03373     SPECTRALRADAR_API double getSettingsEntryFloat(SettingsHandle SettingsFile, const char* Node,
03374     double DefaultValue);
03375
03376     SPECTRALRADAR_API void getSettingsEntryFloatArray(SettingsHandle SettingsFile, const char*
03377     Node, const double* DefaultValues, double* Values, int* Size);
03378
03379     SPECTRALRADAR_API const char* getSettingsEntryString(SettingsHandle SettingsFile, const char*
03380     Node, const char* Default);
03381
03382     SPECTRALRADAR_API void setSettingsEntryInt(SettingsHandle SettingsFile, const char* Node, int
03383     Value);
03384
03385     SPECTRALRADAR_API void setSettingsEntryFloat(SettingsHandle SettingsFile, const char* Node,
03386     double Value);
03387
03388     SPECTRALRADAR_API void setSettingsEntryString(SettingsHandle SettingsFile, const char* Node,
03389     const char* Value);
03390
03391     SPECTRALRADAR_API void saveSettings(SettingsHandle SettingsFile);
03392
03393     SPECTRALRADAR_API void closeSettingsFile(SettingsHandle Handle);
03394
03395
03396     SPECTRALRADAR_API ColoringHandle createColoring32Bit(ColorScheme Color, ColoringByteOrder
03397     ByteOrder);
03398
03399     SPECTRALRADAR_API ColoringHandle createCustomColoring32Bit(int LUTSize, unsigned long* LUT);
03400
03401     SPECTRALRADAR_API void setColoringBoundaries(ColoringHandle Colordg, float Min_dB, float
03402     Max_dB);
03403
03404     SPECTRALRADAR_API void setColoringEnhancement(ColoringHandle Coloring, ColorEnhancement
03405     Enhancement);
03406
03407     SPECTRALRADAR_API void colorizeData(ColoringHandle Coloring, DataHandle Data,
03408     ColoredDataHandle ColoredData, BOOL Transpose);
03409
03410     SPECTRALRADAR_API void colorizeDopplerData(ColoringHandle AmpColoring, ColoringHandle
03411     PhaseColoring, DataHandle AmpData, DataHandle PhaseData, ColoredDataHandle Output, double
03412     MinSignal_dB, BOOL Transpose);
03413
03414     SPECTRALRADAR_API void colorizeDopplerDataEx(ColoringHandle AmpColoring, ColoringHandle
```

```

PhaseColoring[2], DataHandle AmpData, DataHandle PhaseData, ColoredDataHandle Output, double
03563    MinSignal_dB, BOOL Transpose);
03569     SPECTRALRADAR_API void clearColoring(ColorHandle Handle);
03570
03574
03578     SPECTRALRADAR_API void getMaxCameraImageSize(OCTDeviceHandle Dev, int* SizeX, int* SizeY);
03579
03583     SPECTRALRADAR_API void getCameraImage(OCTDeviceHandle Dev, ColoredDataHandle Image);
03584
03585
03586 // HELPER
03587
03590
03601     SPECTRALRADAR_API unsigned long InterpretReferenceIntensity(float intensity);
03602
03614     SPECTRALRADAR_API unsigned long InterpretReferenceIntensitySingleValue(float DesiredIntensity,
float Tolerance, float CurrentIntensity);
03615
03619     SPECTRALRADAR_API void getConfigPath(char* Path, int StrSize);
03620
03624     SPECTRALRADAR_API void getPluginPath(char* Path, int StrSize);
03625
03629     SPECTRALRADAR_API void getInstallationPath(char* Path, int StrSize);
03630
03637     SPECTRALRADAR_API double getReferenceIntensity(ProcessingHandle Proc);
03638
03644     SPECTRALRADAR_API double getRelativeReferenceIntensity(OCTDeviceHandle Dev, ProcessingHandle
Proc);
03645
03649     SPECTRALRADAR_API double getRelativeSaturation(ProcessingHandle Proc);
03650
03657     SPECTRALRADAR_API BufferHandle createMemoryBuffer(void);
03658
03664     SPECTRALRADAR_API void appendToBuffer(BufferHandle, DataHandle, ColoredDataHandle);
03665
03669     SPECTRALRADAR_API void purgeBuffer(BufferHandle);
03670
03674     SPECTRALRADAR_API int getBufferSize(BufferHandle);
03675
03679     SPECTRALRADAR_API int getBufferFirstIndex(BufferHandle);
03680
03684     SPECTRALRADAR_API int getBufferLastIndex(BufferHandle);
03685
03689     SPECTRALRADAR_API DataHandle getData(BufferHandle, int Index);
03690
03694     SPECTRALRADAR_API ColoredDataHandle getData(BufferHandle, int Index);
03695
03700     SPECTRALRADAR_API void clearBuffer(BufferHandle BufferHandle);
03701
03702
03703
03704
03705
03709     SPECTRALRADAR_API void computeLinearKRawData(ComplexDataHandle ComplexDataAfterFFT, DataHandle
LinearKData);
03710
03714     SPECTRALRADAR_API void linearizeSpectralData(DataHandle SpectraIn, DataHandle SpectraOut,
DataHandle Chirp);
03715
03717
03718
03719
03722     static const char* DataObjectName_VideoImage = "data\\VideoImage.data";
03725     static const char* DataObjectName_PreviewImage = "data\\PreviewImage.data";
03728     static const char* DataObjectName_OCTData = "data\\Intensity.data";
03729     static const char* DataObjectName_VarianceData = "data\\Variance.data";
03730     static const char* DataObjectName_PhaseData = "data\\Phase.data";
03731     static const char* DataObjectName_Spectral1DData = "data\\SpectralFloat.data";
03732     static const char* DataObjectName_AnalogInputData = "data\\AnalogInput.data";
03735     static const char* DataObjectName_ComplexOCTData = "data\\Complex.data";
03736     static const char* DataObjectName_FreeformScanPoints = "data\\CustomScanPoints.data";
03737     static const char* DataObjectName_FreeformScanPointsInterpolated =
03738         "data\\CustomScanPointsInterpolated.data";
03744     SPECTRALRADAR_API const char* DataObjectName_SpectralData(int index);
03745
03749     static const char* AcquisitionMode_1D = "Mode1D";
03753     static const char* AcquisitionMode_2D = "Mode2D";
03757     static const char* AcquisitionMode_3D = "Mode3D";
03758     static const char* AcquisitionMode_Doppler = "ModeDoppler";
03759     static const char* AcquisitionMode_Speckle = "ModeSpeckle";
03760     static const char* AcquisitionMode_PolarizationSensitive = "ModePolarization";
03761     static const char* AcquisitionMode_PolarizationSensitive_3D = "ModePolarization3D";
03762
03766     SPECTRALRADAR_API OCTFileHandle createOCTFile(OCTFormat format);
03767

```

```
03772     SPECTRALRADAR_API void clearOCTFile(OCTFileHandle Handle);
03773
03778     SPECTRALRADAR_API int getFileDataObjectCount(OCTFileHandle Handle);
03779
03785     SPECTRALRADAR_API void loadFile(OCTFileHandle Handle, const char* Filename);
03786
03792     SPECTRALRADAR_API void saveFile(OCTFileHandle Handle, const char* Filename);
03793
03798     SPECTRALRADAR_API void saveChangesToFile(OCTFileHandle Handle);
03799
03807     SPECTRALRADAR_API void copyFileMetadata(OCTFileHandle SrcHandle, OCTFileHandle DstHandle);
03808
03814     SPECTRALRADAR_API bool containsFileMetadataFloat(OCTFileHandle Handle, FileMetadataFloat
03815     Floatfield);
03821     SPECTRALRADAR_API double getFileMetadataFloat(OCTFileHandle Handle, FileMetadataFloat
03822     Floatfield);
03829     SPECTRALRADAR_API void setFileMetadataFloat(OCTFileHandle Handle, FileMetadataFloat
03830     Floatfield, double Value);
03836     SPECTRALRADAR_API bool containsFileMetadataInt(OCTFileHandle Handle, FileMetadataInt
03837     Intfield);
03843     SPECTRALRADAR_API int getFileMetadataInt(OCTFileHandle Handle, FileMetadataInt Intfield);
03844
03852     SPECTRALRADAR_API void setFileMetadataInt(OCTFileHandle Handle, FileMetadataInt Intfield, int
03853     Value);
03859     SPECTRALRADAR_API bool containsFileMetadataString(OCTFileHandle Handle, FileMetadataString
03860     Stringfield);
03866     SPECTRALRADAR_API const char* getFileMetadataString(OCTFileHandle Handle, FileMetadataString
03867     Stringfield);
03874     SPECTRALRADAR_API void setFileMetadataString(OCTFileHandle Handle, FileMetadataString
03875     Stringfield, const char* Content);
03881     SPECTRALRADAR_API bool containsFileMetadataFlag(OCTFileHandle Handle, FileMetadataFlag
03882     Boolfield);
03888     SPECTRALRADAR_API BOOL getFileMetadataFlag(OCTFileHandle Handle, FileMetadataFlag Boolfield);
03896     SPECTRALRADAR_API void setFileMetadataFlag(OCTFileHandle Handle, FileMetadataFlag Boolfield,
03897     BOOL Value);
03908     SPECTRALRADAR_API void saveFileMetadata(OCTFileHandle Handle, OCTDeviceHandle Dev,
03909     ProcessingHandle Proc, ProbeHandle Probe, ScanPatternHandle Pattern);
03915     SPECTRALRADAR_API void setFileMetadataTimestamp(OCTFileHandle File, time_t Timestamp);
03916
03921     SPECTRALRADAR_API time_t getFileMetadataTimestamp(OCTFileHandle File);
03922
03928     SPECTRALRADAR_API void saveFileMetadataDoppler(OCTFileHandle Handle, DopplerProcessingHandle
03929     DopplerProc);
03935     SPECTRALRADAR_API void saveFileMetadataSpeckle(OCTFileHandle Handle, SpeckleVarianceHandle
03936     SpeckleVarianceProc);
03937
03945     SPECTRALRADAR_API void loadCalibrationFromFile(OCTFileHandle Handle, ProcessingHandle Proc);
03946
03955     SPECTRALRADAR_API void loadCalibrationFromFileEx(OCTFileHandle Handle, ProcessingHandle Proc,
03956     const int CameraIndex);
03964     SPECTRALRADAR_API void saveCalibrationToFile(OCTFileHandle Handle, ProcessingHandle Proc);
03965
03974     SPECTRALRADAR_API void saveCalibrationToFileEx(OCTFileHandle Handle, ProcessingHandle Proc,
03975     int CameraIndex);
03983     SPECTRALRADAR_API void getFileRealData(OCTFileHandle Handle, DataHandle Data, int Index);
03984
03992     SPECTRALRADAR_API void getFileColoredData(OCTFileHandle Handle, ColoredDataHandle Data, size_t
03993     Index);
04001     SPECTRALRADAR_API void getFileComplexData(OCTFileHandle Handle, ComplexDataHandle Data, size_t
04002     Index);
04011     SPECTRALRADAR_API void getFileRawData(OCTFileHandle Handle, RawDataHandle Data, size_t Index);
04012
04019     SPECTRALRADAR_API void getFile(OCTFileHandle Handle, size_t Index, const char*
04020     FilenameOnDisk);
04026     SPECTRALRADAR_API int findFileDataObject(OCTFileHandle Handle, const char* Search);
04027
04033     SPECTRALRADAR_API BOOL containsFileDataObject(OCTFileHandle Handle, const char* Search);
```

```

04040     SPECTRALRADAR_API BOOL containsFileRawData(OCTFileHandle Handle);
04041
04048     SPECTRALRADAR_API void addFileRealData(OCTFileHandle Handle, DataHandle Data, const char*
DataObjectName);
04049
04056     SPECTRALRADAR_API void addFileColoredData(OCTFileHandle Handle, ColoredDataHandle Data, const
char* DataObjectName);
04057
04064     SPECTRALRADAR_API void addFileComplexData(OCTFileHandle Handle, ComplexDataHandle Data, const
char* DataObjectName);
04065
04075     SPECTRALRADAR_API void addFileRawData(OCTFileHandle Handle, RawDataHandle Data, const char*
DataObjectName);
04076
04084     SPECTRALRADAR_API void addFileText(OCTFileHandle Handle, const char* FilenameOnDisk, const
char* DataObjectName);
04085
04092     SPECTRALRADAR_API DataObjectType getFileDataObjectType(OCTFileHandle Handle, int Index);
04093
04101     SPECTRALRADAR_API void getFileDataObjectName(OCTFileHandle Handle, int Index, char* Filename,
int Length);
04102
04109     SPECTRALRADAR_API int getFileDataSizeX(OCTFileHandle Handle, size_t Index);
04110
04117     SPECTRALRADAR_API int getFileDataSizeY(OCTFileHandle Handle, size_t Index);
04118
04125     SPECTRALRADAR_API int getFileDataSizeZ(OCTFileHandle Handle, size_t Index);
04126
04133     SPECTRALRADAR_API float getFileDataRangeX(OCTFileHandle Handle, size_t Index);
04134
04141     SPECTRALRADAR_API float getFileDataRangeY(OCTFileHandle Handle, size_t Index);
04142
04149     SPECTRALRADAR_API float getFileDataRangeZ(OCTFileHandle Handle, size_t Index);
04150
04155     SPECTRALRADAR_API void clearMarkerList(OCTFileHandle Handle);
04156
04166     SPECTRALRADAR_API void copyMarkerListFromRealData(OCTFileHandle Handle, DataHandle Data);
04167
04176     SPECTRALRADAR_API void copyMarkerListToRealData(OCTFileHandle Handle, DataHandle Data);
04177
04184     SPECTRALRADAR_API void addFileMetadataPreset(OCTFileHandle Handle, const char* Category, const
char* PresetDescription);
04185
04190     SPECTRALRADAR_API int getFileMetadataNumberOfPresets(OCTFileHandle Handle);
04191
04197     SPECTRALRADAR_API const char* getFileMetadataPresetCategory(OCTFileHandle Handle, int Index);
04198
04204     SPECTRALRADAR_API const char* getFileMetadataPresetDescription(OCTFileHandle Handle, int
Index);
04205
04206
04207
04211     SPECTRALRADAR_API SpeckleVarianceHandle initSpeckleVariance(void);
04212
04217     SPECTRALRADAR_API SpeckleVarianceHandle initSpeckleVarianceForFile(OCTFileHandle File);
04218
04223     SPECTRALRADAR_API void closeSpeckleVariance(SpeckleVarianceHandle Handle);
04224     // OBSOLETE
04225     //SPECTRALRADAR_API void setAveraging(SpeckleVarianceHandle SpeckleVar, int Av1, int Av2, int
Av3);
04226
04230     SPECTRALRADAR_API void setSpeckleVariancePropertyInt(SpeckleVarianceHandle Handle,
SpeckleVariancePropertyInt Property, int value);
04231
04235     SPECTRALRADAR_API int getSpeckleVariancePropertyInt(SpeckleVarianceHandle Handle,
SpeckleVariancePropertyInt Property);
04236
04240     SPECTRALRADAR_API void setSpeckleVariancePropertyFloat(SpeckleVarianceHandle Handle,
SpeckleVariancePropertyFloat Property, double value);
04241
04245     SPECTRALRADAR_API double getSpeckleVariancePropertyFloat(SpeckleVarianceHandle Handle,
SpeckleVariancePropertyFloat Property);
04246
04250     SPECTRALRADAR_API void setSpeckleVarianceType(SpeckleVarianceHandle SpeckleVar,
SpeckleVarianceType Type);
04251
04255     SPECTRALRADAR_API SpeckleVarianceType getSpeckleVarianceType(SpeckleVarianceHandle
SpeckleVar);
04256
04260     SPECTRALRADAR_API void computeSpeckleVariance(SpeckleVarianceHandle SpeckleVar,
ComplexDataHandle CompDataIn, DataHandle DataOutMean, DataHandle DataOutVar);
04261
04262
04263
04266
04272     SPECTRALRADAR_API void setTriggerMode(OCTDeviceHandle Dev, DeviceTriggerType TriggerMode);
04273

```

```
04279     SPECTRALRADAR_API DeviceTriggerType getTriggerMode(OCTDeviceHandle Dev);
04280
04286     SPECTRALRADAR_API BOOL isTriggerModeAvailable(OCTDeviceHandle Dev, DeviceTriggerType
TriggerMode);
04287
04293     SPECTRALRADAR_API BOOL isTriggerIOModeAvailable(OCTDeviceHandle Dev, DeviceTriggerIOType
TriggerMode);
04294
04301     SPECTRALRADAR_API void setTriggerIOMode(OCTDeviceHandle Dev, DeviceTriggerIOType TriggerMode);
04302
04308     SPECTRALRADAR_API DeviceTriggerIOType getTriggerIOMode(OCTDeviceHandle Dev);
04309
04319     SPECTRALRADAR_API void setTriggerIOConfiguration(OCTDeviceHandle Dev, size_t Offset, size_t
Divider);
04320
04326     SPECTRALRADAR_API void setTriggerTimeout_s(OCTDeviceHandle Dev, int Timeout_s);
04327
04333     SPECTRALRADAR_API int getTriggerTimeout_s(OCTDeviceHandle Dev);
04334
04338     SPECTRALRADAR_API int getScanPatternPropertyInt(ScanPatternHandle ScanPattern,
ScanPatternPropertyInt Property);
04339
04343     SPECTRALRADAR_API double getScanPatternPropertyFloat(ScanPatternHandle Pattern,
ScanPatternPropertyFloat Selection);
04344
04348     SPECTRALRADAR_API double expectedAcquisitionTime_s(ScanPatternHandle ScanPattern,
OCTDeviceHandle Dev);
04349
04353     SPECTRALRADAR_API ScanPatternAcquisitionOrder getScanPatternAcqOrder(ScanPatternHandle
ScanPattern);
04354
04358     SPECTRALRADAR_API BOOL isAcqTypeForScanPatternAvailable(ScanPatternHandle ScanPattern,
AcquisitionType AcqType);
04359
04365     SPECTRALRADAR_API BOOL checkAvailableMemoryForRawData(OCTDeviceHandle Dev, ScanPatternHandle
Pattern, ptrdiff_t AdditionalMemory);
04366
04370     SPECTRALRADAR_API double QuantumEfficiency(OCTDeviceHandle Dev, double CenterWavelength_nm,
double PowerIntoSpectrometer_W, DataHandle Spectrum_e);
04371
04376     SPECTRALRADAR_API void determineSurface(DataHandle Volume, DataHandle Surface);
04377
04380     SPECTRALRADAR_API unsigned long long getFreeMemory();
04381
04385     SPECTRALRADAR_API void absComplexData(const ComplexDataHandle ComplexData, DataHandle Abs);
04386
04390     SPECTRALRADAR_API void logAbsComplexData(const ComplexDataHandle ComplexData, DataHandle dB);
04391
04395     SPECTRALRADAR_API void argComplexData(ComplexDataHandle ComplexData, DataHandle Arg);
04396
04400     SPECTRALRADAR_API void realComplexData(ComplexDataHandle ComplexData, DataHandle Real);
04401
04405     SPECTRALRADAR_API void imagComplexData(ComplexDataHandle ComplexData, DataHandle Imag);
04406
04410
04419     SPECTRALRADAR_API void determineDynamicRange_dB(DataHandle Data, double* MinRange_dB, double*
MaxRange_dB);
04420
04428     SPECTRALRADAR_API void determineDynamicRangeWithMinRange_dB(DataHandle Data, double*
MinRange_dB, double* MaxRange_dB, double MinDynamicRange_dB);
04429
04436     SPECTRALRADAR_API void medianFilter1D(DataHandle Data, int Rank, Direction FilterDirection);
04437
04444     SPECTRALRADAR_API void medianFilter2D(DataHandle Data, int Rank, Direction
FilterNormalDirection);
04445
04456     SPECTRALRADAR_API void pepperFilter2D(DataHandle Data, PepperFilterType Type, float Threshold,
Direction FilterNormalDirection);
04457
04465     SPECTRALRADAR_API void convolutionFilter1D(DataHandle Data, int FilterSize, float*
FilterKernel, Direction FilterDirection);
04466
04475     SPECTRALRADAR_API void convolutionFilter2D(DataHandle Data, int FilterSize1, int FilterSize2,
float* FilterKernel, Direction FilterNormalDirection);
04476
04485     SPECTRALRADAR_API void convolutionFilter3D(DataHandle Data, int FilterSize1, int FilterSize2,
int FilterSize3, float* FilterKernel);
04486
04493     SPECTRALRADAR_API void predefinedFilter1D(DataHandle Data, FilterType1D Filter, Direction
FilterDirection);
04494
04501     SPECTRALRADAR_API void predefinedFilter2D(DataHandle Data, FilterType2D Filter, Direction
FilterNormalDirection);
04502
04508     SPECTRALRADAR_API void predefinedFilter3D(DataHandle Data, FilterType3D FilterType);
04509
04516     SPECTRALRADAR_API void predefinedComplexFilter2D(ComplexDataHandle ComplexData,
```

```

        ComplexFilterType2D Type, Direction FilterNormalDirection);
04517
04524     SPECTRALRADAR_API void darkFieldComplexFilter2D(ComplexDataHandle ComplexData, double Radius,
04525     Direction FilterNormalDirection);
04525
04532     SPECTRALRADAR_API void brightFieldComplexFilter2D(ComplexDataHandle ComplexData, double
Radius, Direction FilterNormalDirection);
04533
04544     SPECTRALRADAR_API void polynomialFitAndEval1D(int Size, const float* OrigPosX, const float*
OrigY, int DegreePolynom, int EvalSize, const float* EvalPosX, float* EvalY);
04545
04554     SPECTRALRADAR_API float calcParabolaMaximum(float x0, float y0, float yLeft, float yRight,
float* peakHeight);
04555
04560     SPECTRALRADAR_API void crossCorrelatedProjection(DataHandle DataIn, DataHandle DataOut);
04561
04572     SPECTRALRADAR_API void averagingResizeFilter(DataHandle DataIn, DataHandle DataOut, int
Decimation1, int Decimation2, int Decimation3);
04573
04577     SPECTRALRADAR_API void thresholdDopplerData(DataHandle Phase, DataHandle Intensity, float
intensityThreshold, float phaseTargetValue);
04578
04583     SPECTRALRADAR_API void getCurrentIntensityStatistics(OCTDeviceHandle Dev, ProcessingHandle
Proc, float* relToRefIntensity, float* relToProjAbsIntensity);
04584
04589     SPECTRALRADAR_API int getNumberOfProbeConfigs();
04590
04597     SPECTRALRADAR_API void getProbeConfigName(int Index, char* ProbeName, int StringSize);
04598
04599
04600
04605     SPECTRALRADAR_API int getNumberOfAvailableProbes(void);
04606
04615     SPECTRALRADAR_API void getAvailableProbe(int Index, char* ProbeName, int StringSize);
04616
04625     SPECTRALRADAR_API void getProbeDisplayName(const char* ProbeName, char* DisplayName, int
StringSize);
04626
04635     SPECTRALRADAR_API void getObjectiveDisplayName(const char* ObjectiveName, char* DisplayName,
int StringSize);
04636
04642     SPECTRALRADAR_API int getNumberOfCompatibleObjectives(const char* ProbeName);
04643
04653     SPECTRALRADAR_API void getCompatibleObjective(int Index, const char* ProbeName, char*
Objective, int StringSize);
04654
04659     SPECTRALRADAR_API ProbeScanRangeShape getProbeMaxScanRangeShape(ProbeHandle Probe);
04660
04666     SPECTRALRADAR_API void setProbeMaxScanRangeShape(ProbeHandle Probe, ProbeScanRangeShape
Shape);
04667
04675     SPECTRALRADAR_API void setQuadraticProbeFactors(ProbeHandle Probe, double* QuadFactorsX,
double* QuadFactorsY, int NumberOfFactors);
04676
04682     SPECTRALRADAR_API int getObjectivePropertyInt(const char* Objective, ObjectivePropertyInt
Selection);
04683
04689     SPECTRALRADAR_API double getObjectivePropertyFloat(const char* Objective,
ObjectivePropertyFloat Selection);
04690
04694     SPECTRALRADAR_API const char* getObjectivePropertyString(const char* Objective,
ObjectivePropertyString Selection);
04695
04696
04697
04701     SPECTRALRADAR_API void addProbeButtonCallback(OCTDeviceHandle Dev, cbProbeMessageReceived
Callback);
04702
04706     SPECTRALRADAR_API void removeProbeButtonCallback(OCTDeviceHandle Dev, cbProbeMessageReceived
Callback);
04707
04708
04709
04714     SPECTRALRADAR_API BOOL isRefstageAvailable(OCTDeviceHandle Dev);
04715
04720     SPECTRALRADAR_API RefstageStatus getRefstageStatus(OCTDeviceHandle Dev);
04721
04727     SPECTRALRADAR_API double getRefstageLength_mm(OCTDeviceHandle Dev, ProbeHandle Probe);
04728
04734     SPECTRALRADAR_API double getRefstagePosition_mm(OCTDeviceHandle Dev, ProbeHandle Probe);
04735
04741     SPECTRALRADAR_API void homeRefstage(OCTDeviceHandle Dev, RefstageWaitForMovement
WaitForMoving);
04742
04751     SPECTRALRADAR_API void moveRefstageToPosition_mm(OCTDeviceHandle Dev, ProbeHandle Probe,
double Pos_mm, RefstageSpeed Speed, RefstageWaitForMovement WaitForMoving);
04752

```

```
04762     SPECTRALRADAR_API void moveRefstage_mm(OCTDeviceHandle Dev, ProbeHandle Probe, double
Length_mm, RefstageMovementDirection Direction, RefstageSpeed Speed, RefstageWaitForMovement
WaitForMoving);
04763
04770     SPECTRALRADAR_API void startRefstageMovement(OCTDeviceHandle Dev, RefstageMovementDirection
Direction, RefstageSpeed Speed);
04771
04776     SPECTRALRADAR_API void stopRefstageMovement(OCTDeviceHandle Dev);
04777
04783     SPECTRALRADAR_API void setRefstageSpeed(OCTDeviceHandle Dev, RefstageSpeed Speed);
04784
04790     SPECTRALRADAR_API void setRefstageStatusCallback(OCTDeviceHandle Dev, cbRefstageStatusChanged
Callback);
04791
04797     SPECTRALRADAR_API void setRefstagePosChangedCallback(OCTDeviceHandle Dev,
cbRefstagePositionChanged Callback);
04798
04804     SPECTRALRADAR_API double getRefstageMinPosition_mm(OCTDeviceHandle Dev, ProbeHandle Probe);
04805
04811     SPECTRALRADAR_API double getRefstageMaxPosition_mm(OCTDeviceHandle Dev, ProbeHandle Probe);
04812
04813     // After leaving the SLD on for the specified amount of time, it will be switched off,
automatically
04814     // Controlled by following functions/enums/callbacks.
04815
04816
04817
04823     SPECTRALRADAR_API void setLightSourceTimeoutCallback(OCTDeviceHandle Dev,
lightSourceStateCallback Callback);
04824
04830     SPECTRALRADAR_API void setLightSourceTimeout_s(OCTDeviceHandle Dev, double Timeout);
04831
04837     SPECTRALRADAR_API double getLightSourceTimeout_s(OCTDeviceHandle Dev);
04838
04839     // Polarization Processing
04840
04841
04846
04853     SPECTRALRADAR_API void setPolarizationFlag(PolarizationProcessingHandle Polarization,
PolarizationFlag Flag, BOOL OnOff);
04854
04861     SPECTRALRADAR_API BOOL getPolarizationFlag(PolarizationProcessingHandle Polarization,
PolarizationFlag Flag);
04862
04867     SPECTRALRADAR_API PolarizationProcessingHandle createPolarizationProcessing(void);
04868
04873     SPECTRALRADAR_API void clearPolarizationProcessing(PolarizationProcessingHandle Polarization);
04874
04881     SPECTRALRADAR_API int getPolarizationPropertyInt(PolarizationProcessingHandle Polarization,
PolarizationPropertyInt Property);
04882
04889     SPECTRALRADAR_API void setPolarizationPropertyInt(PolarizationProcessingHandle Polarization,
PolarizationPropertyInt Property, int Value);
04890
04897     SPECTRALRADAR_API double getPolarizationPropertyFloat(PolarizationProcessingHandle Polarization,
PolarizationPropertyFloat Property);
04898
04905     SPECTRALRADAR_API void setPolarizationPropertyFloat(PolarizationProcessingHandle Polarization,
PolarizationPropertyFloat Property, double Value);
04906
04912     SPECTRALRADAR_API void setPolarizationOutputI(PolarizationProcessingHandle Polarization,
DataHandle Intensity);
04913
04922     SPECTRALRADAR_API void setPolarizationOutputQ(PolarizationProcessingHandle Polarization,
DataHandle StokesQ);
04923
04932     SPECTRALRADAR_API void setPolarizationOutputU(PolarizationProcessingHandle Polarization,
DataHandle StokesU);
04933
04942     SPECTRALRADAR_API void setPolarizationOutputV(PolarizationProcessingHandle Polarization,
DataHandle StokesV);
04943
04951     SPECTRALRADAR_API void setPolarizationOutputDOPU(PolarizationProcessingHandle Polarization,
DataHandle DOPU);
04952
04960     SPECTRALRADAR_API void setPolarizationOutputRetardation(PolarizationProcessingHandle
Polarization, DataHandle Retardation);
04961
04969     SPECTRALRADAR_API void setPolarizationOutputOpticAxis(PolarizationProcessingHandle
Polarization, DataHandle OpticAxis);
04970
04982     SPECTRALRADAR_API void executePolarizationProcessing(PolarizationProcessingHandle
Polarization, ComplexDataHandle Data_Camer0, ComplexDataHandle PData_Camera0);
04983
04990     SPECTRALRADAR_API void saveFileMetadataPolarization(OCTFileHandle FileHandle,
PolarizationProcessingHandle PolProc);
04991
```

```

04999     SPECTRALRADAR_API PolarizationProcessingHandle
05000         createPolarizationProcessingForFile(OCTFileHandle FileHandle);
05008     SPECTRALRADAR_API void updateAfterPresetChange(OCTDeviceHandle Dev, ProbeHandle Probe,
05009         ProcessingHandle Proc, int CameraIndex);
05010
05019     SPECTRALRADAR_API double analyzeComplexAScan(ComplexDataHandle AScanIn, AScanAnalyzation
05020         Selection);
05021     // Polarization Adjustment
05023
05029     SPECTRALRADAR_API BOOL isPolarizationAdjustmentAvailable(OCTDeviceHandle Dev);
05030
05036     SPECTRALRADAR_API void setPolarizationAdjustmentRetardationChangedCallback(OCTDeviceHandle
05037         Dev, cbRetardationChanged Callback);
05037
05045     SPECTRALRADAR_API void setPolarizationAdjustmentRetardation(OCTDeviceHandle Dev,
05046         PolarizationRetarder Retarder, double Retardation, WaitForCompletion Wait);
05046
05053     SPECTRALRADAR_API double getPolarizationAdjustmentRetardation(OCTDeviceHandle Dev,
05054         PolarizationRetarder Retarder);
05055     // Reference Intensity Control
05057
05063     SPECTRALRADAR_API BOOL isReferenceIntensityControlAvailable(OCTDeviceHandle Dev);
05064
05070     SPECTRALRADAR_API void setReferenceIntensityControlCallback(OCTDeviceHandle Dev,
05071         cbReferenceIntensityControlValueChanged Callback);
05071
05078     SPECTRALRADAR_API void setReferenceIntensityControlValue(OCTDeviceHandle Dev, double
05079         ReferenceIntensity, WaitForCompletion Wait);
05085     SPECTRALRADAR_API double getReferenceIntensityControlValue(OCTDeviceHandle Dev);
05086
05087     // Amplification Control
05089
05095     SPECTRALRADAR_API BOOL isAmplificationControlAvailable(OCTDeviceHandle Dev);
05096
05102     SPECTRALRADAR_API int getAmplificationControlNumberOfSteps(OCTDeviceHandle Dev);
05103
05109     SPECTRALRADAR_API void setAmplificationControlStep(OCTDeviceHandle Dev, int Step);
05110
05116     SPECTRALRADAR_API int getAmplificationControlStep(OCTDeviceHandle Dev);
05117
05118     // General information
05120
05126     SPECTRALRADAR_API void getSoftwareVersion(char* Version, int Length);
05127
05133     SPECTRALRADAR_API void useProbeCalibration(bool OnOff);
05134
05145     SPECTRALRADAR_API void extractLine(DataHandle Data, DataHandle Res, double P1_1_mm, double
05146         P1_2_mm, double P2_1_mm, double P2_2_mm);
05156
05156     SPECTRALRADAR_API int extractLocalMaxima(DataHandle DataID, int N, double* dataPos_mm, double*
05157         dataHeight);
05157
05168     SPECTRALRADAR_API int extractLocalMaximaEx(DataHandle DataID, int N, double minDist, double*
05169         dataPos_mm, double* dataHeight);
05169
05179     SPECTRALRADAR_API void readData(DataHandle Data, const char* filename, int SizeZ, int SizeX,
05180         int SizeY);
05180
05188     SPECTRALRADAR_API double getExpectedHannFWHM(OCTDeviceHandle Dev, ProcessingHandle Proc);
05189
05197     SPECTRALRADAR_API void calcContrastEx(ProcessingHandle Proc, DataHandle ApodizedSpectrum,
05198         DataHandle Contrast);
05198
05205     SPECTRALRADAR_API double calcSpectrometerImagingQualityIndex(DataHandle Spectrum);
05206
05214     SPECTRALRADAR_API void analyzeMaxPeak(DataHandle Data, float* PeakHeight_dB, float*
05214         PeakFWHM_Pixel);
05215
05226     SPECTRALRADAR_API void analyzeMaxPeakEx(DataHandle Data, float* PeakHeight_dB, float*
05226         PeakFWHM_Pixel, float* Points, float* Spline, int* SplineSize);
05227
05236     SPECTRALRADAR_API void extractAScan(DataHandle Data, DataHandle Res, int x0, int y0);
05237
05238
05239
05240 #ifdef __cplusplus
05241 }
05242 #endif
05243
05244 #endif // _SPECTRALRADAR_H

```

8.3 SpectralRadar_Handles.h File Reference

Header containing all handles of the SDK. When including [SpectralRadar.h](#) this file will automatically be included.

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_Buffer * BufferHandle`
The BufferHandle identifies a data buffer.
- `typedef struct C_OCTDevice * OCTDeviceHandle`
The OCTDeviceHandle type is used as Handle for using the SpectralRadar.
- `typedef struct C_Probe * ProbeHandle`
Handle for controlling the galvo scanner.
- `typedef struct C_ScanPattern * ScanPatternHandle`
Handle for creating, manipulating, and discarding a scan pattern.
- `typedef struct C_Processing * ProcessingHandle`
Handle for a processing routine.
- `typedef struct C_DopplerProcessing * DopplerProcessingHandle`
Handle used for Doppler processing.
- `typedef struct C_SpeckleVariance * SpeckleVarianceHandle`
Handle used for SpeckleVariance processing.
- `typedef struct C_PolarizationProcessing * PolarizationProcessingHandle`
Handle used for Polarization processing.
- `typedef struct C_Coloring32Bit * ColoringHandle`
Handle for routines that color available 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.

8.3.1 Detailed Description

Header containing all handles of the SDK. When including [SpectralRadar.h](#) this file will automatically be included.

Definition in file [SpectralRadar_Handles.h](#).

8.4 SpectralRadar_Handles.h

```
00001 #ifndef SPECTRALRADAR_HANDLES_H
00002 #define SPECTRALRADAR_HANDLES_H
00003
00004 // includes all enums required for public getter and setter functions
00005 // getter and setter enums will, likely, only be used in the wrapper functions.
00006
00007
00008
00009
00010
00011 #ifdef __cplusplus
00012 extern "C" {
00013 #endif
00014
00015 struct C_RawData;
00016 typedef struct C_RawData* RawDataHandle;
00017
00018 struct C_Data;
00019 typedef struct C_Data* DataHandle;
00020
00021 struct C_ColoredData;
00022 typedef struct C_ColoredData* ColoredDataHandle;
00023
00024 struct C_ComplexData;
00025 typedef struct C_ComplexData* ComplexDataHandle;
00026
00027 struct C_Buffer;
00028 typedef struct C_Buffer* BufferHandle;
00029
00030 struct C_OCTDevice;
00031 typedef struct C_OCTDevice* OCTDeviceHandle;
00032
00033
00034 struct C_Probe;
00035 typedef struct C_Probe* ProbeHandle;
00036
00037 struct C_ScanPattern;
00038 typedef struct C_ScanPattern* ScanPatternHandle;
00039
00040 struct C_Processing;
00041 typedef struct C_Processing* ProcessingHandle;
00042
00043 struct C_DopplerProcessing;
00044 typedef struct C_DopplerProcessing* DopplerProcessingHandle;
00045
00046 struct C_SpeckleVariance;
00047 typedef struct C_SpeckleVariance* SpeckleVarianceHandle;
00048
00049 struct C_PolarizationProcessing;
00050 typedef struct C_PolarizationProcessing* PolarizationProcessingHandle;
00051
00052 struct C_Coloring32Bit;
00053 typedef struct C_Coloring32Bit* ColoringHandle;
00054
00055 struct C_ImageFieldCorrection;
00056 typedef struct C_ImageFieldCorrection* ImageFieldHandle;
00057
00058 struct C_VisualCalibration;
00059 typedef struct C_VisualCalibration* VisualCalibrationHandle;
00060
00061 struct C_MarkerList;
00062 typedef struct C_MarkerList* MarkerListHandle;
00063
00064 struct C_OCTFileHandle;
00065 typedef struct C_FileHandling* OCTFileHandle;
00066
00067 struct C_Settings;
00068 typedef struct C_Settings* SettingsHandle;
00069
00070 #ifdef __cplusplus
00071 }
00072 #endif
00073
00074 #endif // SPECTRALRADAR_HANDLES_H
```

8.5 SpectralRadar_Properties.h File Reference

Header containing all property enums of the SDK. When including [SpectralRadar.h](#) this file will automatically be included.

Enumerations

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

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

- enum `RawDataPropertyFloat` {
 `RawData_Range1`,
 `RawData_Range2`,
 `RawData_Range3` }

Floating point properties of raw data (`RawDataHandle`) that can be retrieved with the function `getRawDataPropertyFloat`. Some of these parameters will be directly copied from the `ScanPatternHandle` that was used to acquire the data.

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

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

- enum `DataPropertyFloat` {
 `Data_Spacing1`,
 `Data_Spacing2`,
 `Data_Spacing3`,
 `Data_Range1`,
 `Data_Range2`,
 `Data_Range3` }

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

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

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

- enum `DevicePropertyInt` {
 `Device_SpectrumElements`,
 `Device_BytesPerElement`,
 `Device_MaxLiveVolumeRenderingScans`,
 `Device_BitDepth`,
 `Device_NumOfCameras`,
 `Device_RevisionNumber`,

```
Device_NumOfAnalogInputChannels,  
Device_MinimumSpectraPerBuffer }
```

Integer properties of the device that can be retrieved with the function [getDevicePropertyInt](#).

- enum [DevicePropertyString](#) {
 Device_Type,
 Device_Series,
 Device_SerialNumber,
 Device_HardwareConfig }

String-properties of the device that can be retrieved with the function [getDevicePropertyString](#).

- enum [DeviceFlag](#) {
 Device_On = 0,
 Device_CameraAvailable = 1,
 Device_SLDAvailable = 2,
 Device_SLDStatus = 3,
 Device_LaserDiodeStatus = 4,
 Device_CameraShowScanPattern = 5,
 Device_ProbeControllerAvailable = 6,
 Device_DatalsSigned = 7,
 Device_IsSweptSource = 8,
 Device_AnalogInputAvailable = 9,
 Device_HasMasterBoard = 10,
 Device_HasControlBoard = 11,
 Device_SLDStatusQuick = 12 }

Boolean properties of the device that can be retrieved with the function [getDeviceFlag](#).

- enum [StaticDeviceFlag](#) {
 Device_PowerControlAvailable = 0,
 Device_PowerOn = 1 }

Boolean properties of the device that can be queried before the device is initialized. Retrieved with the function [getStaticDeviceFlag](#).

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

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

- enum [ProbeParameterString](#) {
 Probe_Name,

```
Probe_SerialNumber,  
Probe_Description,  
Probe_Objective }
```

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

- enum `ProbeParameterInt` {
 `Probe_ApodizationCycles`,
 `Probe_Oversampling`,
 `Probe_Oversampling_SlowAxis`,
 `Probe_SpeckleReduction` }

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

- enum `ProbeFlag` {
 `Probe_Camerainverted_X`,
 `Probe_Camerainverted_Y`,
 `Probe_HasMEMSScanner`,
 `Probe_ApoOnlyX` }

Boolean parameters describing the behaviour of the Probe.

- enum `ProcessingParameterInt` {
 `Processing_SpectrumAveraging`,
 `Processing_AScanAveraging`,
 `Processing_BScanAveraging`,
 `Processing_ZeroPadding`,
 `Processing_NumberOfThreads`,
 `Processing_FourierAveraging` }

Parameters that set the behaviour of the processing algorithms.

- enum `ProcessingParameterFloat` {
 `Processing_ApodizationDamping`,
 `Processing_MinElectrons`,
 `Processing_FFTOversampling`,
 `Processing_MaxSensorValue` }

Parameters that set the behaviour of the processing algorithms.

- enum `CalibrationData` {
 `Calibration_OffsetErrors`,
 `Calibration_ApodizationSpectrum`,
 `Calibration_ApodizationVector`,
 `Calibration_Dispersion`,
 `Calibration_Chirp`,
 `Calibration_ExtendedAdjust`,
 `Calibration_FixedPattern` }

Data describing the calibration of the processing routines.

- enum `ProcessingFlag` {
 `Processing_UseOffsetErrors`,
 `Processing_RemoveDCSpectrum`,
 `Processing_RemoveAdvancedDCSpectrum`,
 `Processing_UseApodization`,
 `Processing_UseScanForApodization`,
 `Processing_UseUndersamplingFilter`,
 `Processing_UseDispersionCompensation`,
 `Processing_UseDechirp`,
 `Processing_UseExtendedAdjust`,
 `Processing_FullRangeOutput`,
 `Processing_FilterDC`,
 `Processing_UseAutocorrCompensation`,
 `Processing_UseDEFR`,
 `Processing_OnlyWindowing`,
 `Processing_RemoveFixedPattern`,
 `Processing_CalculateSaturation` }

Flags that set the behaviour of the processing algorithms.

- enum `DopplerPropertyInt` {
 `Doppler_Averaging_1`,
 `Doppler_Averaging_2`,
 `Doppler_Stride_1`,
 `Doppler_Stride_2` }

Values that determine the behaviour of the Doppler processing routines.

- enum `DopplerPropertyFloat` {
 `Doppler_RefractiveIndex`,
 `Doppler_ScanRate_Hz`,
 `Doppler_CenterWavelength_nm`,
 `Doppler_DopplerAngle_Deg` }

Values that determine the behaviour of the Doppler processing routines.

- enum `DopplerFlag` { `Doppler_VelocityScaling` }

Flags that determine the behaviour of the Doppler processing routines.

- enum `SpeckleVariancePropertyInt` {
 `SpeckleVariance_Averaging_1`,
 `SpeckleVariance_Averaging_2`,
 `SpeckleVariance_Averaging_3` }

Enum identifying different properties of typ int for speckle variance processing.

- enum `SpeckleVariancePropertyFloat` { `SpeckleVariance_Threshold` }
Enum identifying different properties of typ float for speckle variance processing.
- enum `ScanPatternPropertyInt` {
 `ScanPattern_SizeTotal`,
 `ScanPattern_Cycles`,
 `ScanPattern_SizeCycle`,
 `ScanPattern_SizePreparationCycle`,
 `ScanPattern_SizeImagingCycle`,
 `ScanPattern_SizePreparationScan` }

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

- enum `ScanPatternPropertyFloat` {
 `ScanPattern_RangeX`,
 `ScanPattern_RangeY`,
 `ScanPattern_CenterX`,
 `ScanPattern_CenterY`,
 `ScanPattern_Angle`,
 `ScanPattern_MeanLength_mm` }

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

- enum `ObjectivePropertyString` {
 `Objective_DisplayName`,
 `Objective_Mount` }

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

- enum `ObjectivePropertyInt` {
 `Objective_RangeMaxX_mm`,
 `Objective_RangeMaxY_mm` }

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

- enum `PolarizationPropertyInt` {
 `PolarizationProcessing_DOPU_Z` = 0,
 `PolarizationProcessing_DOPU_X` = 1,
 `PolarizationProcessing_DOPU_Y` = 2,
 `PolarizationProcessing_DOPU_FilterType` = 3,
 `PolarizationProcessing_BScanAveraging` = 4,
 `PolarizationProcessing_BScanAveraged` = 9,
 `PolarizationProcessing_AveragingZ` = 5,
 `PolarizationProcessing_AveragingX` = 6,

```
PolarizationProcessing_AveragingY = 7,  
PolarizationProcessing_AScanAveraging = 8 }
```

Values that determine the behaviour of the Polarization processing routines.

- enum `PolarizationPropertyFloat` {
 `PolarizationProcessing_IntensityThreshold_dB` = 0,
 `PolarizationProcessing_PMDCorrectionAngle_rad` = 1,
 `PolarizationProcessing_CentralWavelength_nm` = 2,
 `PolarizationProcessing_OpticalAxisOffset_rad` = 3 }

Values that determine the behaviour of the Polarization processing routines.

- enum `PolarizationFlag` {
 `PolarizationProcessing_ApplyThresholding` = 0,
 `PolarizationProcessing_YAxisIsFrameAxis` = 1 }

Flags that determine the behaviour of the Polarization processing routines.

8.5.1 Detailed Description

Header containing all property enums of the SDK. When including `SpectralRadar.h` this file will automatically be included.

Definition in file `SpectralRadar_Properties.h`.

8.5.2 Enumeration Type Documentation

8.5.2.1 `SpeckleVariancePropertyFloat` enum `SpeckleVariancePropertyFloat`

Enum identifying different properties of typ float for speckle variance processing.

Definition at line 454 of file `SpectralRadar_Properties.h`.

8.5.2.2 `SpeckleVariancePropertyInt` enum `SpeckleVariancePropertyInt`

Enum identifying different properties of typ int for speckle variance processing.

Definition at line 442 of file `SpectralRadar_Properties.h`.

8.6 SpectralRadar_Properties.h

```

00001 #ifndef SPECTRALRADAR_PROPERTIES_H
00002 #define SPECTRALRADAR_PROPERTIES_H
00003
00004 // includes all enums required for public getter and setter functions
00005 // getter and setter enums will, likely, only be used in the wrapper functions.
00006
00009
00010 #ifdef __cplusplus
00011 extern "C" {
00012 #endif
00013
00017 typedef enum {
00019     RawData_Size1,
00021     RawData_Size2,
00023     RawData_Size3,
00025     RawData_NumberOfElements,
00027     RawData_SizeInBytes,
00029     RawData_BytesPerElement,
00038     RawData_LostFrames
00039 } RawDataPropertyInt;
00040
00041
00046 typedef enum {
00048     RawData_Range1,
00050     RawData_Range2,
00052     RawData_Range3,
00053 } RawDataPropertyFloat;
00054
00058 typedef enum {
00060     Data_Dimensions,
00062     Data_Size1,
00064     Data_Size2,
00066     Data_Size3,
00068     Data_NumberOfElements,
00070     Data_SizeInBytes,
00072     Data_BytesPerElement
00073 } DataPropertyInt;
00074
00078 typedef enum {
00080     Data_Spacing1,
00082     Data_Spacing2,
00084     Data_Spacing3,
00086     Data_Range1,
00088     Data_Range2,
00090     Data_Range3
00091 } DataPropertyFloat;
00092
00096 typedef enum {
00098     Device_FullWellCapacity,
00100     Device_zSpacing,
00102     Device_zRange,
00104     Device_SignalAmplitudeMin_dB,
00106     Device_SignalAmplitudeLow_dB,
00108     Device_SignalAmplitudeHigh_dB,
00110     Device_SignalAmplitudeMax_dB,
00112     Device_BinToElectronScaling,
00114     Device_Temperature,
00116     Device_SLD_OnTime_sec,
00118     Device_CenterWavelength_nm,
00120     Device_SpectralWidth_nm, // FIXME Spectrometer bandwidth
00122     Device_MaxTriggerFrequency_Hz,
00123     // Expected line rate depending on the chosen camera preset.
00124     Device_LineRate_Hz
00125 } DevicePropertyFloat;
00126
00127
00131 typedef enum {
00133     Device_SpectrumElements,
00135     Device_BytesPerElement,
00137     Device_MaxLiveVolumeRenderingScans,
00139     Device_BitDepth,
00141     Device_NumOfCameras,
00143     Device_RevisionNumber,
00145     Device_NumOfAnalogInputChannels,
00147     Device_MinimumSpectraPerBuffer,
00148 } DevicePropertyInt;
00149
00153 typedef enum {
00155     Device_Type,
00157     Device_Series,
00159     Device_SerialNumber,
00161     Device_HardwareConfig,
00162 } DevicePropertyString;
00163
00164

```

```
00168 typedef enum {
00170     Device_On = 0,
00172     Device_CameraAvailable = 1,
00174     Device_SLDAvailable = 2,
00176     Device_SLDStatus = 3,
00179     Device_LaserDiodeStatus = 4,
00181     Device_CameraShowScanPattern = 5,
00183     Device_ProbeControllerAvailable = 6,
00185     Device_DataIsSigned = 7,
00187     Device_IsSweptSource = 8,
00189     Device_AnalogInputAvailable = 9,
00191     Device_HasMasterBoard = 10,
00193     Device_HasControlBoard = 11,
00195         Device_SLDStatusQuick = 12,
00196 } DeviceFlag;
00197
00201 typedef enum {
00203     Device_PowerControlAvailable = 0,
00205     Device_PowerOn = 1,
00206 } StaticDeviceFlag;
00207
00214 typedef enum {
00216     Probe_FactorX,
00218     Probe_OffsetX,
00220     Probe_FactorY,
00222     Probe_OffsetY,
00224     Probe_FlybackTime_Sec,
00226     Probe_ExpansionTime_Sec,
00228     Probe_RotationTime_Sec,
00231     Probe_ExpectedScanRate_Hz,
00233     Probe_CameraScalingX,
00235     Probe_CameraOffsetX,
00237     Probe_CameraScalingY,
00239     Probe_CameraOffsetY,
00241     Probe_CameraAngle,
00243     Probe_RangeMaxX,
00245     Probe_RangeMaxY,
00247     Probe_MaximumSlope_XY,
00249     Probe_SpeckleSize,
00251     Probe_ApoVoltageX,
00253     Probe_ApoVoltageY,
00255     Probe_ReferenceStageOffset,
00257     Probe_FiberOpticalPathLength_mm,
00259     Probe_ProbeOpticalPathLength_mm,
00261     Probe_ObjectiveOpticalPathLength_mm,
00263     Probe_ObjectiveFocalLength_mm,
00264 } ProbeParameterFloat;
00265
00270 typedef enum {
00272     Probe_Name,
00274     Probe_SerialNumber,
00277     Probe_Description,
00280     Probe_Objective
00281 } ProbeParameterString;
00282
00283
00287 typedef enum {
00289     Probe_ApodizationCycles,
00291     Probe_Oversampling,
00293     Probe_Oversampling_SlowAxis,
00295     Probe_SpeckleReduction
00296 } ProbeParameterInt;
00297
00301 typedef enum {
00303     Probe_CameraInverted_X,
00305     Probe_CameraInverted_Y,
00307     Probe_HasMEMSScanner,
00309     Probe_ApoOnlyX
00310 } ProbeFlag;
00311
00315 typedef enum {
00317     Processing_SpectrumAveraging,
00319     Processing_AScanAveraging,
00321     Processing_BScanAveraging,
00323     Processing_ZeroPadding,
00325     Processing_NumberOfThreads,
00327     Processing_FourierAveraging
00328 } ProcessingParameterInt;
00329
00333 typedef enum {
00335     Processing_ApodizationDamping,
00337     Processing_MinElectrons,
00339     Processing_FFTOversampling,
00341     Processing_MaxSensorValue
00342 } ProcessingParameterFloat;
00343
00347 typedef enum {
```

```

00349     Calibration_OffsetErrors,
00351     Calibration_ApodizationSpectrum,
00353     Calibration_ApodizationVector,
00355     Calibration_Dispersion,
00357     Calibration_Chirp,
00359     Calibration_ExtendedAdjust,
00361     Calibration_FixedPattern
00362 } CalibrationData;
00363
00367 typedef enum {
00369     Processing_UseOffsetErrors, // 0
00371     Processing_RemoveDCSpectrum, // 1
00373     Processing_RemoveAdvancedDCSpectrum, // 2
00375     Processing_UseApodization, // 3
00377     Processing_UseScanForApodization, // 4
00379     Processing_UseUndersamplingFilter, // 5
00381     Processing_UseDispersionCompensation, // 6
00383     Processing_UseDechirp, // 7
00385     Processing_UseExtendedAdjust, // 8
00387     Processing_FullRangeOutput, // 9
00389     Processing_FilterDC, // 10
00391     Processing_UseAutocorrCompensation, // 11
00393     Processing_UseDEFR, // 12
00395     Processing_OnlyWindowing, // 13
00397     Processing_RemoveFixedPattern, // 14
00399     Processing_CalculateSaturation //15
00400 } ProcessingFlag;
00401
00406 typedef enum {
00408     Doppler_Averaging_1,
00410     Doppler_Averaging_2,
00412     Doppler_Stride_1,
00414     Doppler_Stride_2
00415 } DopplerPropertyInt;
00416
00420 typedef enum {
00422     Doppler_RefractiveIndex,
00424     Doppler_ScanRate_Hz,
00426     Doppler_CenterWavelength_nm,
00428     Doppler_DopplerAngle_Deg
00429 } DopplerPropertyFloat;
00430
00434 typedef enum {
00436     Doppler_VelocityScaling,
00437 } DopplerFlag;
00438
00442 typedef enum {
00443     // Averaging on Z-Axis
00444     SpeckleVariance_Averaging_1,
00445     // Averaging on X-Axis
00446     SpeckleVariance_Averaging_2,
00447     // Averaging on Y-Axis
00448     SpeckleVariance_Averaging_3
00449 } SpeckleVariancePropertyInt;
00450
00454 typedef enum {
00455     // Threshold used for processed data
00456     SpeckleVariance_Threshold
00457 } SpeckleVariancePropertyFloat;
00458
00462 typedef enum {
00464     ScanPattern_SizeTotal,
00466     ScanPattern_Cycles,
00468     ScanPattern_SizeCycle,
00471     ScanPattern_SizePreparationCycle,
00473     ScanPattern_SizeImagingCycle,
00476     ScanPattern_SizePreparationScan,
00477 } ScanPatternPropertyInt;
00478
00482 typedef enum {
00484     ScanPattern_RangeX,
00486     ScanPattern_RangeY,
00488     ScanPattern_CenterX,
00490     ScanPattern_CenterY,
00492     ScanPattern_Angle,
00494     ScanPattern_MeanLength_mm
00495 } ScanPatternPropertyFloat;
00496
00500 typedef enum {
00502     Objective_DisplayName,
00504     Objective_Mount
00505 } ObjectivePropertyString;
00506
00510 typedef enum {
00512     Objective_RangeMaxX_mm,
00514     Objective_RangeMaxY_mm
00515 } ObjectivePropertyInt;

```

```

00516
00520 typedef enum {
00522     PolarizationProcessing_DOPU_Z = 0,
00524     PolarizationProcessing_DOPU_X = 1,
00526     PolarizationProcessing_DOPU_Y = 2,
00528     PolarizationProcessing_DOPU_FilterType = 3,
00530     PolarizationProcessing_BScanAveraging = 4,
00532     PolarizationProcessing_BScanAveraged = 9,
00534     PolarizationProcessing_AveragingZ = 5,
00536     PolarizationProcessing_AveragingX = 6,
00538     PolarizationProcessing_AveragingY = 7,
00540     PolarizationProcessing_AScanAveraging = 8,
00541 } PolarizationPropertyInt;
00542
00546 typedef enum
00547 {
00549     PolarizationProcessing_IntensityThreshold_dB = 0,
00553     PolarizationProcessing_PMDCorrectionAngle_rad = 1,
00555     PolarizationProcessing_CentralWavelength_nm = 2,
00558     PolarizationProcessing_OpticalAxisOffset_rad = 3,
00559 } PolarizationPropertyFloat;
00560
00564 typedef enum {
00567     PolarizationProcessing_ApplyThresholding = 0,
00569     PolarizationProcessing_YAxisIsFrameAxis = 1,
00570 } PolarizationFlag;
00571
00572 #ifdef __cplusplus
00573 }
00574 #endif
00575
00576 #endif // SPECTRALRADAR_PROPERTIES_H

```

8.7 SpectralRadar_Types.h File Reference

Header containing all types of the SDK. When including [SpectralRadar.h](#) this file will automatically be included.

Data Structures

- struct [ComplexFloat](#)
A standard complex data type that is used to access complex data.

Macros

- **#define __stdcall**
- **#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 void(__stdcall * cbRefstageStatusChanged) (RefstageStatus)**
Defines the function prototype for the reference stage status callback (see also [setRefstageStatusCallback\(\)](#)). The argument contains the current status of the reference stage when called.
- **typedef void(__stdcall * cbRefstagePositionChanged) (double)**
Defines the function prototype for the reference stage position change callback (see also [setRefstagePosChangedCallback\(\)](#)). The argument contains the reference stage position in mm when called.
- **typedef void(__stdcall * cbProbeMessageReceived) (int)**

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

- `typedef void(__stdcall * cbRetardationChanged) (PolarizationRetarder, double)`

Defines the function prototype for the polarization adjustment retardation callback (see also [setPolarizationAdjustmentRetardationChanged\(\)](#)).
The argument contains the current (unitless) position (see also [setPolarizationAdjustmentRetardation\(\)](#)) of the specified [PolarizationRetarder](#).

- `typedef void(__stdcall * cbReferenceIntensityControlValueChanged) (double)`

Defines the function prototype for the reference intensity control status callback (see also [setReferenceIntensityControlCallback\(\)](#)).
The argument contains the current (unitless) intensity between 0 and 1 (see also [setReferenceIntensityControlValue\(\)](#)).

- `typedef void(__stdcall * genericProgressCallback) (double progress, const char *msg)`

Definition for a generic callback function indicating progress. The argument will be passed a value from 0.0 to 1.0 indicating progress in a respective process.

- `typedef void(__stdcall * lightSourceStateCallback) (LightSourceState)`

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

Enumerations

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

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

- `enum LogOutputType {
 Standard,
 File,
 None }`

Specifies where to write text output by the SDK.

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

Analysis types accepted by the functions [analyzeData](#) and [computeDataProjection](#).

- `enum AScanAnalyzation {
 Data_Noise_dB,
 Data_Noise_electrons,
 Data_PeakPos_Pixel,
 Data_PeakPos_PhysUnits,
 Data_PeakHeight_dB,
 Data_PeakWidth_6dB,
 Data_PeakWidth_20dB,
 Data_PeakWidth_40dB,
 Data_PeakPhase,
 Data_PeakRealPart,
 Data_PeakImagPart }`

Analysis types accepted by the functions [analyzeAScan](#) and [analyzeComplexAScan](#).

- `enum DataOrientation {
 DataOrientation_ZXY,
 DataOrientation_ZYX,
 DataOrientation_XZY,
 DataOrientation_XYZ,
 DataOrientation_YXZ,
 DataOrientation_YZX,
 DataOrientation_ZTX,
 DataOrientation_ZXT }`

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

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

Axis selection for the function `moveScanner`.

- enum `DeviceState` {
 `DeviceState_Unavailable` = 0,
 `DeviceState_Standby` = 1,
 `DeviceState_Enabled` = 2,
 `DeviceState_NoConfiguration` = 3,
 `DeviceState_Error` = 4,
 `DeviceState_Startup` = 5 }

Results for function `getDeviceState`.

- enum `ScanPatternAcquisitionOrder` {
 `ScanPattern_AcqOrderFrameByFrame`,
 `ScanPattern_AcqOrderAll` }

Parameters describing the behaviour of the scan pattern.

- enum `ScanPatternApodizationType` {
 `ScanPattern_ApoOneForAll`,
 `ScanPattern_ApoEachBScan` }

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

- enum `InflationMethod` { `Inflation_NormalDirection` }

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

- enum `InterpolationMethod` {
 `Interpolation_Linear`,
 `Interpolation_Spline` }

Selects the interpolation method.

- enum `BoundaryCondition` {
 `BoundaryCondition_Standard`,
 `BoundaryCondition_Natural`,
 `BoundaryCondition_Periodic` }

Selects the boundary conditions for the interpolation.

- enum `ScanPointsDataFormat` {
 `ScanPoints_DataFormat_TXT`,
 `ScanPoints_DataFormat_RAWandSRM` }

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

- enum `AcquisitionType` {
 `Acquisition_AsyncContinuous`,
 `Acquisition_AsyncFinite`,
 `Acquisition_Sync` }

Determines the kind of acquisition process. The type of acquisition process affects e.g. whether consecutive B-scans are acquired or if it is possible to lose some data.

- enum `Processing_FFTType` {
 `Processing_StandardFFT`,
 `Processing_StandardNDFT`,
 `Processing_iFFT`,
 `Processing_NFFT1`,
 `Processing_NFFT2`,
 `Processing_NFFT3`,
 `Processing_NFFT4` }

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

- enum `DispersionCorrectionType` {
 `Dispersion_None`,
 `Dispersion_QuadraticCoeff`,
 `Dispersion_Preset`,
 `Dispersion_Manual` }

To select the dispersion correction algorithm.

- enum `ApodizationWindow` {
 `Apodization_Hann` = 0,
 `Apodization_Hamming` = 1,
 `Apodization_Gauss` = 2,
 `Apodization_TaperedCosine` = 3,
 `Apodization_Blackman` = 4,
 `Apodization_BlackmanHarris` = 5,
 `Apodization_LightSourceBased` = 6,
 `Apodization_Unknown` = 999 }

To select the apodization window function.

- enum `ProcessingAveragingAlgorithm` {
 `Processing_Averaging_Min`,
 `Processing_Averaging_Mean`,
 `Processing_Averaging_Median`,
 `Processing_Averaging_Norm2`,
 `Processing_Averaging_Max`,
 `Processing_Averaging_Fourier_Min`,
 `Processing_Averaging_Fourier_Norm4`,
 `Processing_Averaging_Fourier_Max`,
 `Processing_Averaging_StandardDeviationAbs`,
 `Processing_Averaging_PhaseMatched` }

This sets the averaging algorithm to be used for processing.

- enum `ApodizationWindowParameter` {
 `ApodizationWindowParameter_Sigma`,
 `ApodizationWindowParameter_Ratio`,
 `ApodizationWindowParameter_Frequency` }

Sets certain parameters that are used by the window functions to be applied during apodization.

- enum `DataExportFormat` {
 `DataExport_SRML`,
 `DataExport_RAWL`,
 `DataExport_CVSL`,
 `DataExport_TXTL`,
 `DataExport_TableTXTL`,
 `DataExport_FitsL`,
 `DataExport_VFFL`,
 `DataExport_VTKL`,
 `DataExport_TIFFL` }

Export format for any data represented by a `DataHandle`.

- enum `ComplexDataExportFormat` { `ComplexDataExport_RAW` }

Export format for complex data.

- enum `ColoredDataExportFormat` {
 `ColoredDataExport_SRML`,
 `ColoredDataExport_RAWL`,
 `ColoredDataExport_BMPL`,
 `ColoredDataExport_PNGL`,
 `ColoredDataExport_JPGL`,
 `ColoredDataExport_PDFL`,
 `ColoredDataExport_TIFFL` }

Export format for images (`ColoredDataHandle`).

- enum `Direction` {
 `Direction_1`,
 `Direction_2`,
 `Direction_3` }

Specifies a direction. In the default orientation, the first orientation is the Z-axis (parallel to the illumination-ray during the measurement), the second is the X-axis, and the third is the Y-axis.

- enum `DataImportFormat` { `DataImport_SRM` }

Supported import format to load data from disk.

- enum `RawDataExportFormat` {
 `RawDataExport_RAW`,
 `RawDataExport_SRR` }

Supported raw data export formats to store data to disk.

- enum `RawDataImportFormat` { `RawDataImport_SRR` }

Supported raw data import formats to load data from disk.

- enum `Plane2D` {
 `Plane2D_12`,
 `Plane2D_23`,
 `Plane2D_13` }

Planes for slices of the volume data.

- enum `ColorScheme` {
 `ColorScheme_BlackAndWhite` = 0,
 `ColorScheme_Inverted` = 1,
 `ColorScheme_Color` = 2,
 `ColorScheme_BlackAndOrange` = 3,
 `ColorScheme_BlackAndRed` = 4,
 `ColorScheme_BlackRedAndYellow` = 5,
 `ColorScheme_DopplerPhase` = 6,
 `ColorScheme_BlueAndBlack` = 7,
 `ColorScheme_PolarizationRetardation` = 8,
 `ColorScheme_GreenBlueAndBlack` = 9,
 `ColorScheme_BlackAndRedYellow` = 10,
 `ColorScheme_TransparentAndWhite` = 11,
 `ColorScheme_GreenBlueWhiteRedYellow` = 12,
 `ColorScheme_BlueGreenBlackYellowRed` = 13,
 `ColorScheme_RedGreenBlue` = 14,
 `ColorScheme_GreenBlueRed` = 15,
 `ColorScheme_BlueRedGreen` = 16,
 `ColorScheme_GreenBlueRedGreen` = 17,
 `ColorScheme_BlueRedGreenBlue` = 18,
 `ColorScheme_Inverse_RedGreenBlue` = 19,
 `ColorScheme_Inverse_GreenBlueRed` = 20,
 `ColorScheme_Inverse_BlueRedGreen` = 21,
 `ColorScheme_Inverse_GreenBlueRedGreen` = 22,
 `ColorScheme_Inverse_BlueRedGreenBlue` = 23,
 `ColorScheme_RedYellowGreenBlueRed` = 24,
 `ColorScheme_RedGreenBlueRed` = 25,
 `ColorScheme_Inverse_RedGreenBlueRed` = 26,
 `ColorScheme_RedYellowBlue` = 27,
 `ColorScheme_Inverse_RedYellowBlue` = 28,
 `ColorScheme_DEM_Normal` = 29,
 `ColorScheme_Inverse_DEM_Normal` = 30,
 `ColorScheme_DEM_Blind` = 31,
 `ColorScheme_Inverse_DEM_Blind` = 32,
 `ColorScheme_WhiteBlackWhite` = 33,
 `ColorScheme_BlackWhiteBlack` = 34 } }

selects the ColorScheme of the data to transform real data to colored data.

- enum `ColoringByteOrder` {
 `Coloring_RGBA` = 0,
 `Coloring_BGRA` = 1,
 `Coloring_ARGB` = 2 }
 Selects the byte order of the coloring to be applied.
- enum `ColorEnhancement` {
 `ColorEnhancement_None` = 0,
 `ColorEnhancement_Sine` = 1,
 `ColorEnhancement_Parable` = 2,
 `ColorEnhancement_Cubic` = 3,
 `ColorEnhancement_Sqrt` = 4 }
 Selects the byte order of the coloring to be applied.
- enum `OCTFileFormat` {
 `FileFormat_OCITY`,
 `FileFormat_IMG`,
 `FileFormat_SDR`,
 `FileFormat_SRM`,
 `FileFormat_TIFF32`}
 Enum identifying possible file formats.
- enum `DataObjectType` {
 `DataObjectType_Real`,
 `DataObjectType_Colored`,
 `DataObjectType_Complex`,
 `DataObjectType_Raw`,
 `DataObjectType_Binary`,
 `DataObjectType_Text`,
 `DataObjectType_Unknown` = 999 }
 Enum identifying.
- enum `FileMetadataFloat` {
 `FileMetadata_RefractiveIndex`,
 `FileMetadata_RangeX`,
 `FileMetadata_RangeY`,
 `FileMetadata_RangeZ`,
 `FileMetadata_CenterX`,
 `FileMetadata_CenterY`,
 `FileMetadata_Angle`,
 `FileMetadata_BinToElectronScaling`,
 `FileMetadata_CentralWavelength_nm`,
 `FileMetadata_SourceBandwidth_nm`,
 `FileMetadata_MinElectrons`,
 `FileMetadata_QuadraticDispersionCorrectionFactor`,
 `FileMetadata_SpeckleVarianceThreshold`,
 `FileMetadata_ScanTime_Sec`,
 `FileMetadata_ReferenceIntensity`,
 `FileMetadata_ScanPause_Sec`,
 `FileMetadata_Zoom`,
 `FileMetadata_MinPointDistance`,
 `FileMetadata_MaxPointDistance`,
 `FileMetadata_FFTOversampling`,
 `FileMetadata_FullWellCapacity`,
 `FileMetadata_Saturation`,
 `FileMetadata_CameraLineRate_Hz`,
 `FileMetadata_PMDCorrectionAngle_rad`,
 `FileMetadata_OpticalAxisOffset_rad`,
 `FileMetadata_ReferenceLength_mm`,
 `FileMetadata_ReferenceIntensityControl_Value`,
 `FileMetadata_PolarizationAdjustment_QuarterWave`,
 `FileMetadata_PolarizationAdjustment_HalfWave` }
 Enum identifying.

Enum identifying file metadata fields of floating point type.

- enum `FileMetadataInt` {
 `FileMetadata_ProcessState`,
 `FileMetadata_SizeX`,
 `FileMetadata_SizeY`,
 `FileMetadata_SizeZ`,
 `FileMetadata_Oversampling`,
 `FileMetadata_IntensityAveragedSpectra`,
 `FileMetadata_IntensityAveragedAScans`,
 `FileMetadata_IntensityAveragedBScans`,
 `FileMetadata_DopplerAverageX`,
 `FileMetadata_DopplerAverageZ`,
 `FileMetadata_ApoWindow`,
 `FileMetadata_DeviceBitDepth`,
 `FileMetadata_SpectrometerElements`,
 `FileMetadata_ExperimentNumber`,
 `FileMetadata_DeviceBytesPerPixel`,
 `FileMetadata_SpeckleAveragingFastAxis`,
 `FileMetadata_SpeckleAveragingSlowAxis`,
 `FileMetadata_Processing_FFTType`,
 `FileMetadata_NumOfCameras`,
 `FileMetadata_SelectedCamera`,
 `FileMetadata_ApodizationType`,
 `FileMetadata_AcquisitionOrder`,
 `FileMetadata_DOPUFilter`,
 `FileMetadata_DOPUAverageZ`,
 `FileMetadata_DOPUAverageX`,
 `FileMetadata_DOPUAverageY`,
 `FileMetadata_PolarizationAverageZ`,
 `FileMetadata_PolarizationAverageX`,
 `FileMetadata_PolarizationAverageY`,
 `FileMetadata_SamplingAmplification`,
 `FileMetadata_SamplingAmplificationSteps`,
 `FileMetadata_AnalogInputNumOfChannels` }

Enum identifying file metadata fields of integral type.

- enum `FileMetadataString` {
 `FileMetadata_DeviceSeries`,
 `FileMetadata_DeviceName`,
 `FileMetadata_Serial`,
 `FileMetadata_Comment`,
 `FileMetadata_CustomInfo`,
 `FileMetadata_AcquisitionMode`,
 `FileMetadata_Study`,
 `FileMetadata_DispersionPreset`,
 `FileMetadata_ProbeName`,
 `FileMetadata_FreeformScanPatternInterpolation`,
 `FileMetadata_HardwareConfig`,
 `FileMetadata_OrigVersion`,
 `FileMetadata_LastModVersion`,
 `FileMetadata_AnalogInputActiveChannels`,
 `FileMetadata_AnalogInputChannelNames` }

Enum identifying file metadata fields of character string type.

- enum `FileMetadataFlag` {
 `FileMetadata_OffsetApplied`,
 `FileMetadata_DCSubtracted`,
 `FileMetadata_ApoApplied`,
 `FileMetadata_DechirpApplied`,
 `FileMetadata_UndersamplingFilterApplied`,

```
FileMetadata_DispersionCompensationApplied,
FileMetadata_QuadraticDispersionCorrectionUsed,
FileMetadata_ImageFieldCorrectionApplied,
FileMetadata_ScanLineShown,
FileMetadata_AutoCorrCompensationUsed,
FileMetadata_BScanCrossCorrelation,
FileMetadata_DCSubtractedAdvanced,
FileMetadata_OnlyWindowing,
FileMetadata_RawDataIsSigned,
FileMetadata_FreeformScanPatternIsActive,
FileMetadata_FreeformScanPatternCloseLoop,
FileMetadata_IsSweptSource,
FileMetadata_DopplerOversampling }
```

Enum identifying file metadata fields of bool type.

- enum **FileMetadata_ProcessingState** {
 RawSpectra,
 ProcessedIntensity,
 RawSpectraAndProcessedIntensity,
 ProcessedIntensityAndPhase,
 RawSpectraAndProcessedIntensityAndPhase,
 psSpeckleVariance,
 psRawSpectraAndSpeckleVariance,
 psColored,
 psUnknown = 999 }

Enum to specify the processing state of the stored data.

- enum **SpeckleVarianceType** {
 SpeckleVariance_LogscaleVariance_Linear,
 SpeckleVariance_LogscaleVariance_Logscale,
 SpeckleVariance_LinearVariance_Linear,
 SpeckleVariance_LinearVariance_Logscale,
 SpeckleVariance_ComplexVariance_Linear,
 SpeckleVariance_ComplexVariance_Logscale }

Enum identifying different speckle variance processing types.

- enum **DeviceTriggerType** {
 Trigger_FreeRunning,
 Trigger_TrigBoard_ExternalStart,
 Trigger_External_AScan }

Enum identifying trigger types for the OCT system.

- enum **DeviceTriggerIOType** {
 TriggerIO_Disabled,
 TriggerIO_Output,
 TriggerIO_Input }

Enum identifying trigger types for the secondary trigger IO channel.

- enum **PepperFilterType** {
 PepperFilter_Horizontal,
 PepperFilter_Vertical,
 PepperFilter_Star,
 PepperFilter_Block }

Specifies the type of pepper filter to be applied.

- enum **ComplexFilterType2D** { **FilterComplex2D_PhaseContrast** }

Specifies the type of filter to be applied to complex data.

- enum **FilterType1D** { **Filter1D_Gaussian_5** }

Specifies the type of 1D-filter to be applied. All filters are normalized.

- enum **FilterType2D** {
 Filter2D_Gaussian_3x3,
 Filter2D_Gaussian_5x5,

```
Filter2D_Prewitt_Horizontal_3x3,
Filter2D_Prewitt_Vertical_3x3,
Filter2D_NonlinearPrewitt_3x3,
Filter2D_Sobel_Horizontal_3x3,
Filter2D_Sobel_Vertical_3x3,
Filter2D_NonlinearSobel_3x3,
Filter2D_Laplacian_NoDiagonal_3x3,
Filter2D_Laplacian_3x3 }
```

Specifies the type of 2D-filter to be applied. All filters are normalized.

- enum `FilterType3D { Filter3D_Gaussian_3x3x3 }`

Specifies the type of 3D-filter to be applied. All filters are normalized.

- enum `ObjectivePropertyFloat { Objective_FocalLength_mm,`
`Objective_OpticalPathLength }`

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

- enum `ProbeScanRangeShape { Probe_ScanRange_Rectangular,`
`Probe_ScanRange_Round }`

The shape of the maximal valid scan range.

- enum `RefstageStatus { RefStage_Status_Idle = 0,`
`RefStage_Status_Homing = 1,`
`RefStage_Status_Moving = 2,`
`RefStage_Status_MovingTo = 3,`
`RefStage_Status_Stopping = 4,`
`RefStage_Status_NotAvailable = 5,`
`RefStage_Status_Undefined = -1 }`

Defines the status of the motorized reference stage.

- enum `RefstageSpeed { RefStage_Speed_Slow = 0,`
`RefStage_Speed_Fast = 1,`
`RefStage_Speed_VerySlow = 2,`
`RefStage_Speed_VeryFast = 3 }`

Defines the velocity of movement for the motorized reference stage.

- enum `RefstageWaitForMovement { RefStage_Movement_Wait = 0,`
`RefStage_Movement_Continue = 1 }`

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

- enum `RefstageMovementDirection { RefStage_MoveShorter = 0,`
`RefStage_MoveLonger = 1 }`

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

- enum `PolarizationDOPUFilterType { PolarizationProcessing_DOPU_Median,`
`PolarizationProcessing_DOPU_Average,`
`PolarizationProcessing_DOPU_Gaussian,`
`PolarizationProcessing_DOPU_GaussianWithFFT }`

Values that determine the behaviour of temporal filter, if enabled.

- enum `WaitForCompletion { Wait = 0,`
`Continue = 1 }`

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

- enum `PolarizationRetarder` {
 `Retarder_Quarter_Wave` = 0,
 `Retarder_Half_Wave` = 1 }
- List of available polarization retarders in a polarization control unit.*
- enum `SpectrumDirectionType` {
 `LongToShortWavelengths`,
 `ShortToLongWavelengths`,
 `UnknownSpectrumDirection` }
- Describes the orientation of the spectrometer, i.e., if the first pixels correspond to longer or shorter wavelengths.*
- enum `LightSourceState` {
 `Activating`,
 `On`,
 `Off` }
- Values that define the state of the light source.*

8.7.1 Detailed Description

Header containing all types of the SDK. When including `SpectralRadar.h` this file will automatically be included.

Definition in file `SpectralRadar_Types.h`.

8.7.2 Macro Definition Documentation

8.7.2.1 FALSE `#define FALSE 0`

FALSE for use with data type `BOOL`.

Definition at line 26 of file `SpectralRadar_Types.h`.

8.7.2.2 TRUE `#define TRUE 1`

TRUE for use with data type `BOOL`.

Definition at line 22 of file `SpectralRadar_Types.h`.

8.7.3 Typedef Documentation

8.7.3.1 BOOL `BOOL`

A standard boolean data type used in the API.

Definition at line 18 of file `SpectralRadar_Types.h`.

8.7.4 Enumeration Type Documentation

8.7.4.1 SpeckleVarianceType enum SpeckleVarianceType

Enum identifying different speckle variance processing types.

Definition at line 833 of file [SpectralRadar_Types.h](#).

8.8 SpectralRadar_Types.h

```
00001 #ifndef SPECTRALRADAR_TYPES_H
00002 #define SPECTRALRADAR_TYPES_H
00003
00004 #ifndef _WIN32
00005     #define __stdcall
00006 #endif
00007
00008 // includes all types (enums, handles, other typedefs) required for public API, except getter and
// setter enums
00009
00010
00011 #ifdef __cplusplus
00012     extern "C" {
00013 #endif
00014     typedef int BOOL;
00015
00016 #define TRUE 1
00017
00018 #define FALSE 0
00019
00020
00021 // needed for compatibility reasons if compiled with x byte aligned struct data.
00022
00023     typedef struct {
00024         float data[2];
00025     } ComplexFloat;
00026
00027
00028
00029     typedef enum {
00030         NoError = 0x0000,
00031         Error = 0xE000
00032     } ErrorCode;
00033
00034
00035     typedef enum {
00036         Standard,
00037         File,
00038         None
00039     } LogOutputType;
00040
00041
00042
00043     typedef enum {
00044         Data_Min,
00045         Data_Mean,
00046         Data_Max,
00047         Data_MaxDepth,
00048         Data_Median,
00049     } DataAnalyzation;
00050
00051
00052     typedef enum {
00053         Data_Noise_dB,
00054         Data_Noise_electrons,
00055         Data_PeakPos_Pixel,
00056         Data_PeakPos_PhysUnits,
00057         Data_PeakHeight_dB,
00058         Data_PeakWidth_6dB,
00059         Data_PeakWidth_20dB,
00060         Data_PeakWidth_40dB,
00061         Data_PeakPhase,
00062         Data_PeakRealPart,
00063         Data_PeakImagPart
00064     } AScanAnalyzation;
00065
00066
00067     typedef enum {
00068         DataOrientation_ZXY,
00069         DataOrientation_ZYX,
00070         DataOrientation_XZY,
```

```

00131     DataOrientation_XYZ,
00132     DataOrientation_YXZ,
00133     DataOrientation_YZX,
00134     DataOrientation_ZTX,
00135     DataOrientation_ZXT,
00136 } DataOrientation;
00137
00144 typedef enum {
00146     ScanAxis_X = 0,
00148     ScanAxis_Y = 1
00149 } ScanAxis;
00150
00154 typedef enum {
00156     DeviceState_Unavailable = 0,
00158     DeviceState_Standby = 1,
00160     DeviceState_Enabled = 2,
00162     DeviceState_NoConfiguration = 3,
00164         DeviceState_Error = 4,
00166     DeviceState_Startup = 5,
00167 } DeviceState;
00168
00169
00170
00174 typedef enum ScanPatternAcquisitionOrder_ {
00177     ScanPattern_AcqOrderFrameByFrame,
00179     ScanPattern_AcqOrderAll
00180 } ScanPatternAcquisitionOrder;
00181
00187 typedef enum ScanPatternApodizationType_ {
00189     ScanPattern_ApoOneForAll,
00192     ScanPattern_ApoEachBScan
00193 } ScanPatternApodizationType;
00194
00198 typedef enum InflationMethod_{
00200     Inflation_NormalDirection
00201 } InflationMethod;
00202
00206 typedef enum InterpolationMethod_{
00208     Interpolation_Linear,
00210     Interpolation_Spline,
00211 } InterpolationMethod;
00212
00216 typedef enum BoundaryCondition_{
00218     BoundaryCondition_Standard,
00220     BoundaryCondition_Natural,
00223     BoundaryCondition_Periodic,
00224 } BoundaryCondition;
00225
00229 typedef enum ScanPointsDataFormat_{
00231     ScanPoints_DataFormat_TXT,
00233     ScanPoints_DataFormat_RAWandSRM,
00234 } ScanPointsDataFormat;
00235
00240 typedef enum AcquisitionType_ {
00248     Acquisition_AsyncContinuous,
00251     Acquisition_AsyncFinite,
00255     Acquisition_Sync
00256 } AcquisitionType;
00257
00262 typedef enum {
00264     Processing_StandardFFT,
00266     Processing_StandardNDFT,
00268     Processing_iFFT,
00270     Processing_NFFT1,
00272     Processing_NFFT2,
00274     Processing_NFFT3,
00276     Processing_NFFT4,
00277 } Processing_FFTType;
00278
00282 typedef enum {
00284     Dispersion_None,
00286     Dispersion_QuadraticCoeff,
00288     Dispersion_Preset,
00290     Dispersion_Manual
00291 } DispersionCorrectionType;
00292
00296 typedef enum ApodizationWindow_ {
00298     Apodization_Hann = 0,
00300     Apodization_Hamming = 1,
00302     Apodization_Gauss = 2,
00304     Apodization_TaperedCosine = 3,
00306     Apodization_Blackman = 4,
00308     Apodization_BlackmanHarris = 5,
00310     Apodization_LightSourceBased = 6,
00312     Apodization_Unknown = 999
00313 } ApodizationWindow;
00314

```

```
00315
00319     typedef enum {
00320         Processing_Averaging_Min,
00321         Processing_Averaging_Mean,
00322         Processing_Averaging_Median,
00323         Processing_Averaging_Norm2,
00324         Processing_Averaging_Max,
00325         Processing_Averaging_Fourier_Min,
00326         Processing_Averaging_Fourier_Norm4,
00327         Processing_Averaging_Fourier_Max,
00328         Processing_Averaging_StandardDeviationAbs,
00329         Processing_Averaging_PhaseMatched,
00330     } ProcessingAveragingAlgorithm;
00332
00336     typedef enum {
00338         ApodizationWindowParameter_Sigma,
00339         ApodizationWindowParameter_Ratio,
00340         ApodizationWindowParameter_Frequency
00341     } ApodizationWindowParameter;
00345
00350     typedef enum DataExportFormat_ {
00352         DataExport_SRM,
00354         DataExport_RAW,
00356         DataExport_CSV,
00358         DataExport_TXT,
00360         DataExport_TableTXT,
00362         DataExport_Fits,
00364         DataExport_VPF,
00366         DataExport_VTK,
00368         DataExport_TIFF
00369     } DataExportFormat;
00370
00374     typedef enum {
00376         ComplexDataExport_RAW
00377     } ComplexDataExportFormat;
00378
00382     typedef enum {
00384         ColoredDataExport_SRM,
00385         ColoredDataExport_RAW,
00386         ColoredDataExport_BMP,
00387         ColoredDataExport_PNG,
00388         ColoredDataExport_JPG,
00389         ColoredDataExport_PDF,
00390         ColoredDataExport_TIFF
00391     } ColoredDataExportFormat;
00399
00404     typedef enum {
00406         Direction_1,
00407         Direction_2,
00408         Direction_3
00409     } Direction;
00412
00416     typedef enum {
00419         DataImport_SRM
00420     } DataImportFormat;
00421
00425     typedef enum {
00427         RawDataExport_RAW,
00428         RawDataExport_SRR
00429     } RawDataExportFormat;
00431
00435     typedef enum {
00437         RawDataImport_SRR
00438     } RawDataImportFormat;
00439
00444     typedef enum {
00446         Plane2D_12,
00447         Plane2D_23,
00448         Plane2D_13
00449     } Plane2D;
00452
00453
00458     typedef enum {
00460         ColorScheme_BlackAndWhite = 0,
00461         ColorScheme_Inverted = 1,
00462         ColorScheme_Color = 2,
00463         ColorScheme_BlackAndOrange = 3,
00464         ColorScheme_BlackAndRed = 4,
00465         ColorScheme_BlackRedAndYellow = 5,
00466         ColorScheme_DopplerPhase = 6,
00467         ColorScheme_BlueAndBlack = 7,
00468         ColorScheme_PolarizationRetardation = 8,
00469         ColorScheme_GreenBlueAndBlack = 9,
00470         ColorScheme_BlackAndRedYellow = 10,
00471         ColorScheme_TransparentAndWhite = 11,
00472         ColorScheme_GreenBlueWhiteRedYellow = 12,
00473         ColorScheme_BlueGreenBlackYellowRed = 13,
00474 }
```

```

00489     ColorScheme_RedGreenBlue = 14,
00491     ColorScheme_GreenBlueRed = 15,
00493     ColorScheme_BlueRedGreen = 16,
00495     ColorScheme_GreenBlueRedGreen = 17,
00497     ColorScheme_BlueRedGreenBlue = 18,
00499     ColorScheme_Inverse_RedGreenBlue = 19,
00501     ColorScheme_Inverse_GreenBlueRed = 20,
00503     ColorScheme_Inverse_BlueRedGreen = 21,
00505     ColorScheme_Inverse_GreenBlueRedGreen = 22,
00507     ColorScheme_Inverse_BlueRedGreenBlue = 23,
00509     ColorScheme_RedYellowGreenBlueRed = 24,
00511     ColorScheme_RedGreenBlueRed = 25,
00513     ColorScheme_Inverse_RedGreenBlueRed = 26,
00515     ColorScheme_RedYellowBlue = 27,
00517     ColorScheme_Inverse_RedYellowBlue = 28,
00519     ColorScheme_DEM_Normal = 29,
00520     ColorScheme_Inverse_DEM_Normal = 30,
00522     ColorScheme_DEM_Blind = 31,
00523     ColorScheme_Inverse_DEM_Blind = 32,
00524     ColorScheme_WhiteBlackWhite = 33,
00525     ColorScheme_BlackWhiteBlack = 34
00526 } ColorScheme;
00527
00531     typedef enum {
00533         Coloring_RGBA = 0,
00535         Coloring_BGRA = 1,
00537         Coloring_ARGB = 2
00538     } ColoringByteOrder;
00539
00543     typedef enum {
00545         ColorEnhancement_None = 0,
00547         ColorEnhancement_Sine = 1,
00549         ColorEnhancement_Parable = 2,
00551         ColorEnhancement_Cubic = 3,
00553         ColorEnhancement_Sqrt = 4
00554     } ColorEnhancement;
00555
00559     typedef enum _OCTFileFormat {
00560         FileFormat_OCITY,
00561         FileFormat_IMG,
00562         FileFormat_SDR,
00563         FileFormat_SRM,
00564         FileFormat_TIFF32
00565     } OCTFileFormat;
00566
00567
00571     typedef enum _DataObjectType {
00572         DataObjectType_Real,
00573         DataObjectType_Colored,
00574         DataObjectType_Complex,
00575         DataObjectType_Raw,
00576         DataObjectType_Binary,
00577         DataObjectType_Text,
00578         DataObjectType_Unknown = 999
00579     } DataObjectType;
00580
00581
00585     typedef enum {
00587         FileMetadata_RefractiveIndex, // = 0
00589         FileMetadata_RangeX,
00591         FileMetadata_RangeY,
00593         FileMetadata_RangeZ,
00595         FileMetadata_CenterX,
00597         FileMetadata_CenterY,
00599         FileMetadata_Angle,
00601         FileMetadata_BinToElectronScaling,
00603         FileMetadata_CentralWavelength_nm,
00605         FileMetadata_SourceBandwidth_nm,
00607         FileMetadata_MinElectrons,
00609         FileMetadata_QuadraticDispersionCorrectionFactor,
00611         FileMetadata_SpeckleVarianceThreshold,
00613         FileMetadata_ScanTime_Sec,
00615         FileMetadata_ReferenceIntensity,
00617         FileMetadata_ScanPause_Sec,
00619         FileMetadata_Zoom,
00621         FileMetadata_MinPointDistance,
00623         FileMetadata_MaxPointDistance,
00625         FileMetadata_FFTOverSampling,
00627         FileMetadata_FullWellCapacity,
00630         FileMetadata_Saturation,
00632         FileMetadata_CameraLineRate_Hz,
00634         FileMetadata_PMDCorrectionAngle_rad,
00636         FileMetadata_OpticalAxisOffset_rad,
00638         FileMetadata_ReferenceLength_mm,
00640         FileMetadata_ReferenceIntensityControl_Value,
00642         FileMetadata_PolarizationAdjustment_QuarterWave,
00644         FileMetadata_PolarizationAdjustment_HalfWave, // 28
00646
00647

```

```
00648     } FileMetadataFloat;
00649
00650     typedef enum {
00651         FileMetadata_ProcessState, // = 0
00652         FileMetadata_SizeX,
00653         FileMetadata_SizeY,
00654         FileMetadata_SizeZ,
00655         FileMetadata_Oversampling,
00656         FileMetadata_IntensityAveragedSpectra,
00657         FileMetadata_IntensityAveragedAScans,
00658         FileMetadata_IntensityAveragedBScans,
00659         FileMetadata_DopplerAverageX,
00660         FileMetadata_DopplerAverageZ,
00661         FileMetadata_ApoWindow,
00662         FileMetadata_DeviceBitDepth,
00663         FileMetadata_SpectrometerElements,
00664         FileMetadata_ExperimentNumber,
00665         FileMetadata_DeviceBytesPerPixel,
00666         FileMetadata_SpeckleAveragingFastAxis,
00667         FileMetadata_SpeckleAveragingSlowAxis,
00668         FileMetadata_Processing_FFTType,
00669         FileMetadata_NumOfCameras, // 18
00670         FileMetadata_SelectedCamera,
00671         FileMetadata_ApodizationType,
00672         FileMetadata_AcquisitionOrder,
00673         FileMetadata_DOPUFilter,
00674         FileMetadata_DOPUAverageZ,
00675         FileMetadata_DOPUAverageX,
00676         FileMetadata_DOPUAverageY,
00677         FileMetadata_PolarizationAverageZ,
00678         FileMetadata_PolarizationAverageX,
00679         FileMetadata_PolarizationAverageY,
00680         FileMetadata_SamplingAmplification,
00681         FileMetadata_SamplingAmplificationSteps,
00682         FileMetadata_AnalogInputNumOfChannels // 31
00683     } FileMetadataInt;
00684
00685     typedef enum {
00686         FileMetadata_DeviceSeries, // 0
00687         FileMetadata_DeviceName,
00688         FileMetadata_Serial,
00689         FileMetadata_Comment,
00690         FileMetadata_CustomInfo,
00691         FileMetadata_AcquisitionMode,
00692         FileMetadata_Study,
00693         FileMetadata_DispersionPreset,
00694         FileMetadata_ProbeName,
00695         FileMetadata_FreeformScanPatternInterpolation,
00696         FileMetadata_HardwareConfig,
00697         FileMetadata_OrigVersion,
00698         FileMetadata_LastModVersion,
00699         FileMetadata_AnalogInputActiveChannels,
00700         FileMetadata_AnalogInputChannelNames, // 14
00701
00702     } FileMetadataString;
00703
00704     typedef enum {
00705         FileMetadata_OffsetApplied, // = 0
00706         FileMetadata_DCSubtracted,
00707         FileMetadata_ApoApplied,
00708         FileMetadata_DeChirpApplied,
00709         FileMetadata_UndersamplingFilterApplied,
00710         FileMetadata_DispersionCompensationApplied,
00711         FileMetadata_QuadraticDispersionCorrectionUsed,
00712         FileMetadata_ImageFieldCorrectionApplied,
00713         FileMetadata_ScanLineShown,
00714         FileMetadata_AutoCorrCompensationUsed,
00715         FileMetadata_BScanCrossCorrelation,
00716         FileMetadata_DCSubtractedAdvanced,
00717         FileMetadata_OnlyWindowing,
00718         FileMetadata_RawDataIsSigned,
00719         FileMetadata_FreeformScanPatternIsActive,
00720         FileMetadata_FreeformScanPatternCloseLoop,
00721         FileMetadata_IsSweptSource, // 16
00722         FileMetadata_DopplerOversampling // 17
00723     } FileMetadataFlag;
00724
00725     typedef enum {
00726         RawSpectra,
00727         ProcessedIntensity,
00728         RawSpectraAndProcessedIntensity,
00729         ProcessedIntensityAndPhase,
00730         RawSpectraAndProcessedIntensityAndPhase,
00731         psSpeckleVariance,
00732         psRawSpectraAndSpeckleVariance,
00733         psColored,
00734         psUnknown = 999
00735 }
```

```

00828     }FileMetadata_ProcessingState;
00829
00830     typedef enum {
00831         // Logscale speckle variance with linear output
00832         SpeckleVariance_LogscaleVariance_Linear,
00833         // Logscale speckle variance with logscale output
00834         SpeckleVariance_LogscaleVariance_Logscale,
00835         // Linear speckle variance with linear output
00836         SpeckleVariance_LinearVariance_Linear,
00837         // Linear speckle variance with logscale output
00838         SpeckleVariance_LinearVariance_Logscale,
00839         // Complex speckle variance with linear output
00840         SpeckleVariance_ComplexVariance_Linear,
00841         // Complex speckle variance with logscale output
00842         SpeckleVariance_ComplexVariance_Logscale
00843     } SpeckleVarianceType;
00844
00845
00846
00847
00848
00849
00850
00851     typedef enum DeviceTriggerType_ {
00852         Trigger_FreeRunning,
00853         Trigger_TrigBoard_ExternalStart,
00854         Trigger_External_AScan
00855     } DeviceTriggerType;
00856
00857
00858
00859     typedef enum DeviceTriggerIOType_ {
00860         TriggerIO_Disabled,
00861         TriggerIO_Output,
00862         TriggerIO_Input,
00863     } DeviceTriggerIOType;
00864
00865
00866
00867
00868
00869
00870     typedef enum {
00871         PepperFilter_Horizontal,
00872         PepperFilter_Vertical,
00873         PepperFilter_Star,
00874         PepperFilter_Block
00875     } PepperFilterType;
00876
00877
00878
00879
00880     typedef enum {
00881         FilterComplex2D_PhaseContrast
00882     } ComplexFilterType2D;
00883
00884
00885
00886
00887
00888
00889
00890     typedef enum {
00891         Filter1D_Gaussian_5
00892     } FilterTypeID;
00893
00894
00895
00896
00897
00898
00899
00900
00901     typedef enum {
00902         Filter2D_Gaussian_3x3,
00903         Filter2D_Gaussian_5x5,
00904         Filter2D_Prewitt_Horizontal_3x3,
00905         Filter2D_Prewitt_Vertical_3x3,
00906         Filter2D_NonlinearPrewitt_3x3,
00907         Filter2D_Sobel_Horizontal_3x3,
00908         Filter2D_Sobel_Vertical_3x3,
00909         Filter2D_NonlinearSobel_3x3,
00910         Filter2D_Laplacian_NoDiagonal_3x3,
00911         Filter2D_Laplacian_3x3
00912     } FilterType2D;
00913
00914
00915
00916
00917
00918
00919
00920
00921
00922
00923
00924
00925
00926
00927
00928
00929
00930
00931
00932
00933
00934
00935
00936
00937     typedef enum {
00938         Filter3D_Gaussian_3x3x3
00939     } FilterType3D;
00940
00941
00942
00943
00944
00945
00946
00947     typedef enum {
00948         Objective_FocalLength_mm,
00949         Objective_OpticalPathLength
00950     } ObjectivePropertyFloat;
00951
00952
00953
00954
00955
00956
00957     typedef enum {
00958         Probe_ScanRange_Rectangular,
00959         Probe_ScanRange_Round
00960     } ProbeScanRangeShape;
00961
00962
00963
00964
00965
00966     typedef enum ReferenceStatus_ {
00967
00968         RefStage_Status_Idle = 0,
00969         RefStage_Status_Homing = 1,
00970         RefStage_Status_Moving = 2,
00971         RefStage_Status_MovingTo = 3,
00972         RefStage_Status_Stopping = 4,
00973         RefStage_Status_NotAvailable = 5,
00974         RefStage_Status_Undefined = -1
00975     } RefstageStatus;
00976
00977
00978
00979
00980
00981
00982
00983

```

```
00984
00988     typedef enum
00989     {
00991         RefStage_Speed_Slow = 0,
00993         RefStage_Speed_Fast = 1,
00995         RefStage_Speed_VerySlow = 2,
00997         RefStage_Speed_VeryFast = 3
00998     } RefstageSpeed;
00999
01003     typedef enum RefstageWaitForMovement_
01004     {
01006         RefStage_Movement_Wait = 0,
01008         RefStage_Movement_Continue = 1
01009     } RefstageWaitForMovement;
01010
01014     typedef enum
01015     {
01017         RefStage_MoveShorter = 0,
01019         RefStage_MoveLonger = 1
01020     } RefstageMovementDirection;
01021
01025     typedef enum PolarizationDOPUFilterType_ {
01027         PolarizationProcessing_DOPU_Median,
01029         PolarizationProcessing_DOPU_Average,
01031         PolarizationProcessing_DOPU_Gaussian,
01033         PolarizationProcessing_DOPU_GaussianWithFFT,
01034     } PolarizationDOPUFilterType;
01035
01036
01040     typedef enum WaitForCompletion_{
01042         Wait = 0,
01044         Continue = 1
01045     } WaitForCompletion;
01046
01050     typedef enum PolarizationRetarder_
01051     {
01053         Retarder_Quality_Wave = 0,
01055         Retarder_Half_Wave = 1,
01056     } PolarizationRetarder;
01057
01062     typedef enum SpectrumDirectionType_ {
01064         LongToShortWavelengths,
01066         ShortToLongWavelengths,
01068         UnknownSpectrumDirection
01069     } SpectrumDirectionType;
01070
01071
01075     typedef void(__stdcall* cbRefstageStatusChanged)(RefstageStatus);
01076
01080     typedef void(__stdcall* cbRefstagePositionChanged)(double);
01081
01085     typedef void(__stdcall* cbProbeMessageReceived)(int);
01086
01089     typedef void(__stdcall* cbRetardationChanged)(PolarizationRetarder, double);
01090
01093     typedef void(__stdcall* cbReferenceIntensityControlValueChanged)(double);
01094
01099     typedef void(__stdcall* genericProgressCallback)(double progress, const char* msg);
01100
01104     typedef enum LightSourceState_
01105     {
01107         Activating,
01109         On,
01111         Off
01112     } LightSourceState;
01113
01117     typedef void(__stdcall* lightSourceStateCallback)(LightSourceState);
01118
01119 #ifdef __cplusplus
01120 }
01121 #endif
01122
01123 #endif // SPECTRALRADAR_TYPES_H
```

