NanoQuest SDK

MEMS-Based FT-IR Sensor

Installation and Operation Manual

For Product: NANOQ-2.5

Locations

Americas

Ocean Insight, Inc.
8060 Bryan Dairy Rd., Largo, FL 33777, USA

Manufacturing & Logistics
4301 Metric Dr., Winter Park, FL 32792, USA

Sales: info@oceaninsight.com
Orders: orders@oceaninsight.com
Support: techsupport@oceaninsight.com

Phone: +1 727.733.2447
Fax: +1 727.733.3962

Europe, Middle East & Africa

Sales & Support
Geograaf 24, 6921 EW Duiven, The Netherlands

Manufacturing & Logistics
Maybaachstrasse 11, 73760 Ostfildern, Germany

Email: info@oceaninsight.eu

Netherlands: +31 26-319-0500
Netherlands Fax: +31 26-319-0505
Germany: +49 711-341696-0
UK: +44 1865-819922
France: +33 442-386-588

Asia

Ocean Insight Asia
666 Gubei Rd., Kirin Tower Suite 601B
Changning District, Shanghai, PRC, 200336

Email: asiasales@oceaninsight.com

China: +86 21-6295-6600
China Fax: +86 21-6295-6708
Japan & Korea: +82 10-8514-3797

www.oceaninsight.com

Table of Contents

About This Manual

Cautions

Caution: Do not let contaminants get into the bench. Keep the protective cap on the slit aperture when not connected to an accessory, probe or fiber.

Caution: Do not immerse the device in any fluid, place fluids on top of or attempt to clean with liquid detergents or cleaning agents. This may cause an electrical hazard. Do not use if accidental wetting occurs.

Caution: Consult local codes and ordinances for proper disposal of equipment and other consumable goods.

Caution: Do not use if device is dropped and/or damaged. Have an authorized service representative check the device before using again.

Caution: Be sure to install any software BEFORE connecting the spectrometer to your PC or host system. The software installs the drivers required for spectrometer installation. If you do not install the software first, the system may not properly recognize the spectrometer.

Caution: The user of this spectrometer shall have the sole responsibility for any malfunction which results from improper use, faulty maintenance, improper repair, damage or alteration by anyone other than Ocean Insight or their authorized service personnel.

Caution: Do not apply excessive vibration or shock to the device. Although vibration rejection mode is supported, excessive vibrations may damage the unit.

Caution: Do not use organic solvents in cleaning. Wipe with a dry and clean tissue.

Caution: When attempting to connect the fiber, do not apply excessive force to the optical connector. Excessive force may damage the connector and will affect measurement results.

Warranty

For the most current warranty information, please visit OceanInsight.com.

Certifications and Compliance

Warning

The authority to operate this equipment is conditioned by the requirement that no modifications will be made to the equipment unless the changes or modifications are expressly approved by the manufacturer.

WEEE Compliance

The WEEE symbol on the product indicates that the product must not be disposed of with normal household waste. Instead, such marked waste equipment must be disposed of by arranging to return to a designated collection point for the recycling of waste electrical and electronic equipment. Separating and recycling this waste equipment at the time of disposal will help to conserve natural resources and ensure that the equipment is recycled in a manner that protects human health and the environment

This device has been tested and complies with the following standards:

EMC Directive 2014/30/EU

EN 61326-1:2013

ISO Certification

Ocean Insight, the applied spectral knowledge company, has been certified for ISO 9001:2015 certification applicable to the design and manufacture of electro-optical equipment.

Overview

NanoQuest is a spectral sensing module whose working principle is based on the standard Fourier Transform Infrared (FT-IR) spectroscopy technique commonly used in conventional spectrometers. The core engine of FT-IR spectrometers is a Michelson interferometer. In NanoQuest, the whole Michelson interferometer is integrated on a single silicon chip.

The NanoQuest software application has two editions:

- NanoQuest Software – Basic Edition This is a Graphical User Interface (GUI) software that enables plotting, saving, and loading NIR spectra measured by NanoQuest. This edition is used for demonstration and evaluation purposes.
- NanoQuest Software - SDK Edition: This is a Software Development Kit (SDK) that enables the direct interface with the NanoQuest via a set of APIs. This edition is used to control the NanoQuest and to build end-usage application software.

The SDK and Basic editions of NanoQuest Software share the same libraries.

This document depicts the requirements to operate the SDK, and explains the different APIs, and communication protocols.

Operating Systems

NanoQuest SDK can operate on the following platforms:

- Microsoft Windows XP (both x86 and x64)
- Microsoft Windows Vista (both x86 and x64)
- Microsoft Windows 7, 8, and 10 (both x86 and x64)
- Ubuntu 12.04 (both x86 and x64)
- Debian 32 (ARM Architecture)

SDK Package

The SDK package consists of the following folders:

- SDK<version number>
    - bin: Output folder for the user application
    - bin_debian_arm_x86: contains spectrometer libraries, jar and configuration files to be used for Debian 32bit platform on ARM architecture.
    - bin_ubuntu_x86: contains spectrometer libraries, jar and configuration files to be used for Ubuntu 32bit platform
    - bin_ubuntu_x64: contains spectrometer libraries, jar and configuration files to be used for Ubuntu 64bit platform
    - bin_win_x86: contains spectrometer libraries, jar and configuration files to be used for Windows 32bit platform
    - bin_win_x64: contains spectrometer libraries, jar and configuration files to be used for Windows 64bit platform
    - nanoquest: contains the source code of NanoQuest Software for demonstration purpose.

Installation

NanoQuest Software – Basic Edition should be installed before proceeding with the SDK installation steps.

After downloading the SDK package the following steps should be performed in an IDE, such as Eclipse (https://www.eclipse.org/ide/):

- Open a new project: Click File->New->Java Project
- Uncheck “Use default location”
- In the “Location” field, browse to the location of the SDK package (e.g. D:\SDK v4.1)
- Press the “Next” button
- Under the source tab, the SDK package hierarchy should be displayed. Select the folder corresponding to your operating system. Right-click and select “Use as source folder”
- Note: Ensure that only 2 folders are marked as source folders (nanoquest/src and the folder corresponding to your operating system)
- Ensure that the “Default output folder” field contains the path to the bin folder (e.g. D:\SDK v4.1\bin)

- Click on the libraries tab, remove any paths that don't belong to your operating system
- Press the "Finish" button
- From the menu select "Run->Run Configurations"
- Write click on "Java Application" and click "New"
- Under the "Main" tab, in the "Main class" field, click on "Search"
- In the "Select Main Type" window, type "UserInterface" and select it from the list. Press "OK"
- Click on the "Arguments" tab. In the "VM arguments" field, type the following commands:

  -Djava.library.path="<path to the SDK libraries corresponding to your platform>" -Dswing.defaultlaf=com.sun.java.swing.plaf.nimbus.NimbusLookAndFeel

  For example:

  -Djava.library.path="D:/SDK v4.1/bin_winx64" -Dswing.defaultlaf=com.sun.java.swing.plaf.nimbus.NimbusLookAndFeel

- Click the "Run" button

Software Architecture

NanoQuest SDK has the components described below. The interfaces among these components are as shown in **Figure** 1:

- Application software
  - nanoquest.jar: The source code of NanoQuest Software - Basic Edition is delivered as for reference. This component should be replaced by the end-use application software.
  - 3rd party modules used by nanoquest.jar:
    - jcommon-1.0.21.jar
    - jfreechart-1.0.17.jar
    - log4j-1.2.17.jar

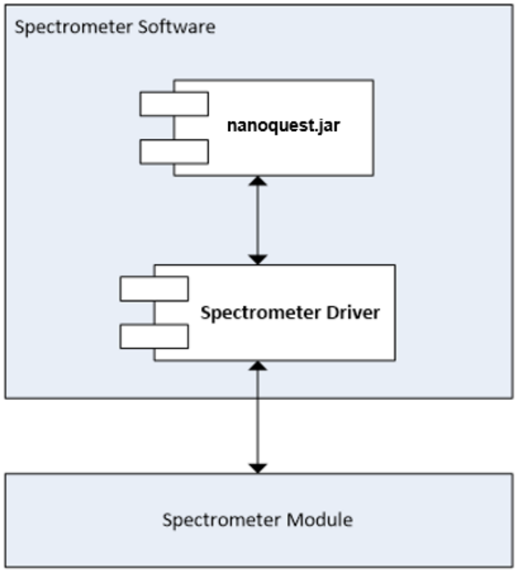

Figure 1: SDK Components

        ▪ miglayout15-swing.jar

- Spectrometer driver:
    - p2AppManager.jar (which is the only component from which nanoquest.jar calls the different APIs)
    - TAIFDriver.jar
    - cyDriver.dll
    - spectrometerDSP.dll
    - 3rd party modules:
        - libusb-1.0.dll
        - log4j-1.2.17.jar
        - pthreadGC2.dll

Operation Flowchart

The application software should follow the steps shown in **Figure** 2 to operate successfully. The steps can be summarized as follows:

1. Perform the initialization sequence described in **Figure** 3.
2. Wait in idle state for a run command.
3. When receiving a run command:
    a. Switch the device on
    b. Set the correct calibration folder to use in run procedure
    c. Wait for run to be finished
    d. Request output data
    e. Switch the device off
4. Return to idle state waiting for new run command

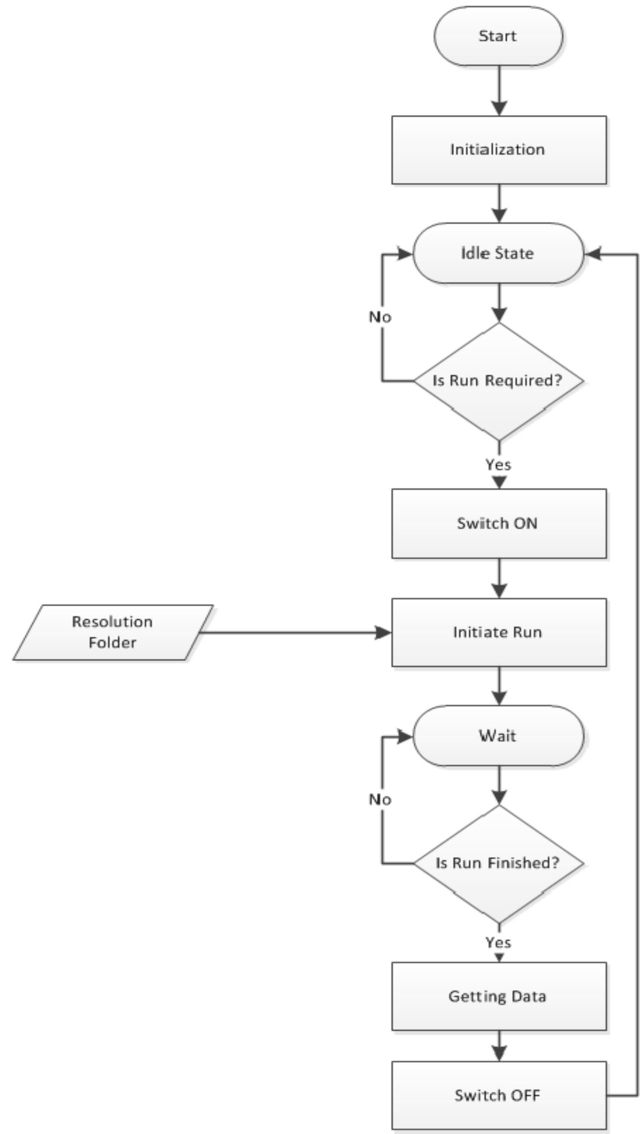

Figure 2: Operation flowchart

Details on the sequences of each step are described in upcoming sections

Notes:

1. The spectrometer driver can serve only one run command at a given time.
2. Checking device connectivity is valid only in idle state.
3. Resolution folder is a set of files that are stored on the spectrometer module's memory (EEPROM). All the files are read at the initialization step and one of them is selected by the application software to be used at the run step. The hierarchy of the resolution folder is as follows:
    a. Conf_Files:
        i. Temperature
            1. Resolution1 (8nm)
            2. Resolution2 (16 nm)
        ii. savedOpticalSettings

APIs

p2AppManager.jar

This component is the interface of the spectrometer driver and it is responsible for the following:

- Communication among the different application components.
- Simple processing on input and output parameters/data.

p2AppManager APIs

The p2AppManager component has the following APIs:

Interface: P2AppManagerImpl()

Description: Component Constructor

Inputs	Outputs	Return	Type
- **String dir (optional): Set the working directory of the SDK.**	-	-	Sync

Interface: addObserver()

Description: Add the caller as an observer in the p2AppManager

Inputs	Outputs	Return	Type
Reference to the caller instance	-	-	Sync

Notes:

- Guidelines to get the status of the software:
    - Your class should implement “Observer” interface.
    - The class should add itself as an observer to “p2AppManager” class through `addObserver()` method.
    - `Update()` method will be invoked from p2AppManager once an action has been finished. This method should be overridden also in your class.

Interface: getDeviceId()

Description: Gets the ID of the connected spectrometer module.

Inputs	Outputs	Return	Type
-	String deviceID	Spectrometer ID	Sync

Interface: initializeCore()

Description: Begin initializing the connected board

Inputs	Outputs	Return	Type
-	-	p2AppManagerStatus: see Table 3 .	Async

Interface: setSettings()

Description: Set the relative path of the resolution folder to be used during the upcoming runs

Inputs	Outputs	Return	Type
- **String resolutionFolder: resolution folder to be used** - **String reloadRegister (optional): flag set to true if you want to load a new register file to the module, false if you are using the same file**	-	-	Sync

Interface: setOpticalSettings()

Description: Set the optical gain to the selected one.

Inputs	Outputs	Return	Type
- **String opticalGainSetting: name of the optical gain** - **String opticalGainPrefix: See Table 1**	-	p2AppManager Status: see Table 3	Sync

Interface: runSpec()

Description: Generate Spectrum (relative to background measurement)

Inputs	Outputs	Return	Type
- **String runTime: Scan time in milliseconds** - **String isSample: flag set by *false* if background measurement and *true* if sample measurement** - **String apodization (optional)** - **String zeroPadding (optional)** **See Table 1**	-	p2AppManager Status: see Table 3	Async

Interface: getSpecData()

Description: Get data corresponding to runSpec function

Inputs	Outputs	Return	Type
-	See Table 2	double[][]	Sync

Interface: runInterSpec()

Description: Generate Interferogram and Power Spectral Density

Inputs	Outputs	Return	Type
- **String runTime: Scan time in milliseconds** - **String apodization (optional)** - **String zeroPadding (optional)** **See Table 1**	-	p2AppManagerStatus: see Table 3	Async

Interface: getInterSpecData()

Description: Get data corresponding to runInterSpec command

Inputs	Outputs	Return	Type
-	See Table 2	double[][]	Sync

Interface: checkDeviceStatus()

Description: Check the current status of the connected device

Inputs	Outputs	Return	Type
-	-	p2AppManagerStatus: see Table 3	Sync

Interface: switchDevice()

Description: Switch the device on and off

Inputs	Outputs	Return	Type
- **String on: true if you want to switch device on, false otherwise.** - **String openLoop: False for P2 modules, default to true for prior modules**	-	p2AppManagerStatus: see Table 3	Async

Interface: wavelengthCalibrationBG()

Description: Perform first step of the wavelength calibration using background reading

Inputs	Outputs	Return	Type
- **String runTime: Scan time in milliseconds** - **String apodization** - **String zeroPadding** **See Table 1**	-	p2AppManagerStatus: see Table 3	Async

Interface: wavelengthCalibration()

Description: Perform second step of the wavelength calibration using a known calibrator (sample)

Inputs	Outputs	Return	Type
- **String runTime: Scan time in milliseconds** - **String calibratorType[1]: name of the sample to be used** - **String apodization** - **String zeroPadding** **See Table 1**	-	p2AppManagerStatus: see Table 3	Async

Interface: runCalibCorr()

Description: Perform wavelength self-correction using two burst correction technique

Inputs	Outputs	Return	Type
- **String runTime: run time in milliseconds** - **String apodization** - **String zeroPadding** **See Table 1**	-	p2AppManagerStatus: see Table 3	Async

Interface: updateFFT_SettingsInterSpec()

Description: Update Interferogram based on selected FFT settings

Inputs	Outputs	Return	Type
- **String apodization** - **String zeroPadding** **See Table 1**	-	p2AppManagerStatus: see Table 3	Sync

[1] calibratorType: Name of the calibrator file under mems/standard_calibrators

Interface: updateFFT_SettingsSpec()

Description: Update Spectrum based on selected FFT settings

Inputs	Outputs	Return	Type
- **String apodization** - **String zeroPadding** **See Table 1**	-	p2AppManagerStatus: see Table 3	Sync

Interface: runInterSpecGainAdj()

Description: Add a new gain settings to get an Interferogram

Inputs	Outputs	Return	Type
- **String runTime: time needed to adjust the gain in milliseconds**	-	p2AppManagerStatus: see Table 3	Async

Interface: getGainAdjustInterSpecData()

Description: Get the gain settings corresponding to runInterSpecGainAdj()

Inputs	Outputs	Return	Type
-	-	double[][]	Sync

Interface: saveInterSpecGainSettings()

Description: Save the gain settings returned from getGainAdjustInterSpecData() in the calibration folder

Inputs	Outputs	Return	Type
- **String optionName: name to be used to save the settings** - **double[][] result: gain settings returning from getGainAdjustInterSpecData()**	-	p2AppManagerStatus: see Table 3	Sync

Interface: runSpecGainAdjBG()

Description: Add a new gain for the spectrum using background

Inputs	Outputs	Return	Type
- **String runTime: time needed to adjust the gain in milliseconds**	-	p2AppManagerStatus: see Table 3	Async

Interface: getGainAdjustSpecData()

Description: Get gain settings corresponding to runSpecGainAdjBG()

Inputs	Outputs	Return	Type
-	-	double[][]	Sync

Interface: burnSettings()

Description: Burn the gain settings and wavenumber correction values on the module

Inputs	Outputs	Return	Type
-	-	p2AppManagerStatus: see Table 3	Async

Interface: burnSpecificSettings()

Description: Burn specific gain settings and enable/disable the saving of the wavenumber correction values on the module

Inputs	Outputs	Return	Type
- **String [] settingsToBurn: List containing the name of the gain settings to burn** - **String updateCorrection: flag if set to true it saves the correction values to the module.**	-	p2AppManagerStatus: see Table 3	Async

Interface: saveSpecGainSettings()

Description: Save the gain settings returned from getGainAdjustSpecData() in the calibration folder

Inputs	Outputs	Return	Type
- **String optionName: name to be used to save the settings** - **double[][] result: gain settings returning from getGainAdjustSpecData()**	-	p2AppManagerStatus: see Table 3	Sync

Interface: restoreDefaultSettings()

Description: Restore the default gain settings and wavenumber correction settings from the module

Inputs	Outputs	Return	Type
-	-	p2AppManagerStatus: see Table 3	Async

Interface: setWorkingDirectory()

Description: Sets the working directory of the application

Inputs	Outputs	Return	Type
- **String dir: Path to the working directory**	-		Async

Interface: getWorkingDirectory()

Description: return the current working directory of the application

Inputs	Outputs	Return	Type
-	-	- String : Path to the working directory	Async

Input Data Format

Parameter	Description	Value	Description
Apodization	Shape of the window to be used to multiply the Interferogram before FFT	0	Rectangular
		1	Tukey .25
ZeroPadding	Number of points to be added to the Interferogram before FFT	0	No points to add
		1	1*VALUE= number of points to add [2]
		3	3*VALUE= number of points to add
		7	7*VALUE= number of points to add
OpticalGainPrefix	Identifier between Interferogram gain settings and Spectrum gain settings	_InterSpec_	To retrieve the gain in case of background or interferogram
		Spec	To retrieve the gain in case of Sample

Table 1: Input data format

[2] VALUE: Parameter in Conf_Files/param.conf file

Output Data Format

Two-dimensional array holds the spectrum/interferogram data, which consists of the following arrays:

API Name	Array Index	Description	Data set	Axis	Units
getInterSpecData()	0	Optical path difference values	Interferogram	X	µm
	1	Photodetector's current intensity values (Interference pattern)	Interferogram	Y	nA
	2	Wavenumber values	Spectrum	X	cm-1
	3	Power spectral density (PSD) values	Spectrum	Y	a.u.
getSpecData()	2	Wavenumber values	Spectrum	X	cm-1
	3	Absorbance values (relative to background measurement)	Spectrum	Y	Abs.

Table 2: Output data format

p2AppManagerStatus

Status Code	Enum	Message
0	*NO_ERROR*	No error
1	*DEVICE_BUSY_ERROR*	Device is busy.
2	*BOARD_DISTCONNECTED_ERROR*	NanoQuest Software does not detect any connected NanoQuest module
3	*BOARD_NOT_INITIALIZED_ERROR*	NanoQuest module is not initialized
4	*UNKNOWN_ERROR*	Unknown error. Contact Ocean Insight.
7	*CONFIG_FILES_LOADING_ERROR*	Error in loading resolution folder
8	*CONFIG_PARAM_LENGTH_ERROR*	Error in resolution folder format
11	*INVALID_RUN_TIME_ERROR*	Invalid scan time
23	*INAVLID_REG_FILE_FORMAT_ERROR*	Error in resolution folder format
24	*NO_OF_SCANS_DSP_ERROR*	DSP error
25	*DSP_INTERFEROGRAM_POST_PROCESSINF_ERROR*	DSP error
26	*DSP_INTERFEROGRAM_POST_EMPTY_DATA_ERROR*	DSP error
27	*DSP_INTERFEROGRAM_POST_BAD_DATA_ERROR*	DSP error
28	*UPDATE_CORR_FILE_ERROR*	Error updating resolution folder
29	*WHITE_LIGHT_PROCESSING_ERROR*	Error in saving background data
30	*DSP_INTERFEROGRAM_FFT_POST_PROCESSINF_ERROR*	DSP error
31	*INVALID_RUN_PARAMETERS_ERROR*	Invalid run parameters
32	*INVALID_RUN_TIME_NOT_EQUAL_BG_RUN_TIME_ERROR*	Background measurement scan time is not equal to sample measurement scan time
33	*NO_VALID_BG_DATA_ERROR*	No valid background measurement found
34	*INTERFERO_FILE_CREATION_ERROR*	Error occurred during saving interferogram data
35	*PSD_FILE_CREATION_ERROR*	Error occurred during saving PSD data

Status Code	Enum	Message
36	*SPECTRUM_FILE_CREATION_ERROR*	Error occurred during saving spectrum data
37	*GRAPHS_FOLDER_CREATION_ERROR*	Error occurred during creating data folder
42	*INITIATE_MIPDRIVER_ERROR*	Error occurred during NanoQuest module initialization
43	*INVALID_BOARD_CONFIGURATION_ERROR*	Error occurred during NanoQuest module initialization
50	*DATA_STREAMING_TAIF_ERROR*	Error occurred during streaming from NanoQuest module
51	*DATA_STREAMING_ERROR*	Error occurred during streaming from NanoQuest module
52	*INVALID_NOTIFICATION_ERROR*	Error occurred during result return
53	*INVALID_ACTION_ERROR*	Invalid action performed
54	*INVALID_DEVICE_ERROR*	Invalid device is attached
55	*THREADING_ERROR*	Threading error occurred
56	*BOARD_ALREADY_INITIALIZED*	NanoQuest module is already initialized successfully
57	*INITIALIZATION_IN_PROGRESS*	Initialization sequence is in progress
58	*SW_DOESNOT_SUPPORT_THIS_FEATURE*	Requested command is not supported
60	*ACTUATION_SETTING_ERROR*	Error occurred during the setup of actuation settings
61	*DEVICE_IS_TURNED_OFF_ERROR*	NanoQuest module is switched off
62	*ASIC_REGISTER_WRITING_ERROR*	Error occurred during writing to chip registers

Table 3: p2AppManagerStatus values

Sequence Diagrams

Initialization

The initialization scenario should be run at least once for the connected NanoQuest module. The scenario consists of the following steps:

1. Construct the p2AppManager.jar through calling `p2AppManagerImpl()`
2. Add your class as an observer to be notified by the p2AppManager when asking for an asynchronous action
3. Board initialization through calling `InitializeCore()`
4. Waiting for finishing initialization
5. Your class will be notified when module initialization is finished

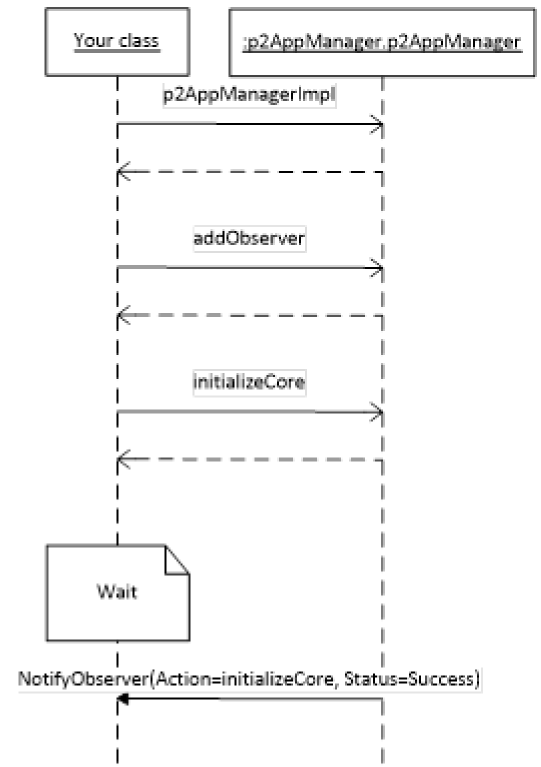

Figure 3: Initialization Sequence

Interferogram & PSD Run

The Interferogram & PSD scenario consists of the following steps:

1. Switch on the module through calling `switchDevice(on=true)`
2. Set the resolution folder through calling `setSettings(resolutionFolder=`<selected calibration folder>`)`
3. Set the optical settings through calling setOpticalSettings(opticalGainSettings,"_InterSpec_")
4. Start the run procedure through calling `runInterSpec(RunTime)`
5. Wait for finishing run
6. Your class will be notified when the run is finished
7. Get the data through calling `getInterSpecData()`
8. Switch off the module through calling `switchDevice(off=false)`

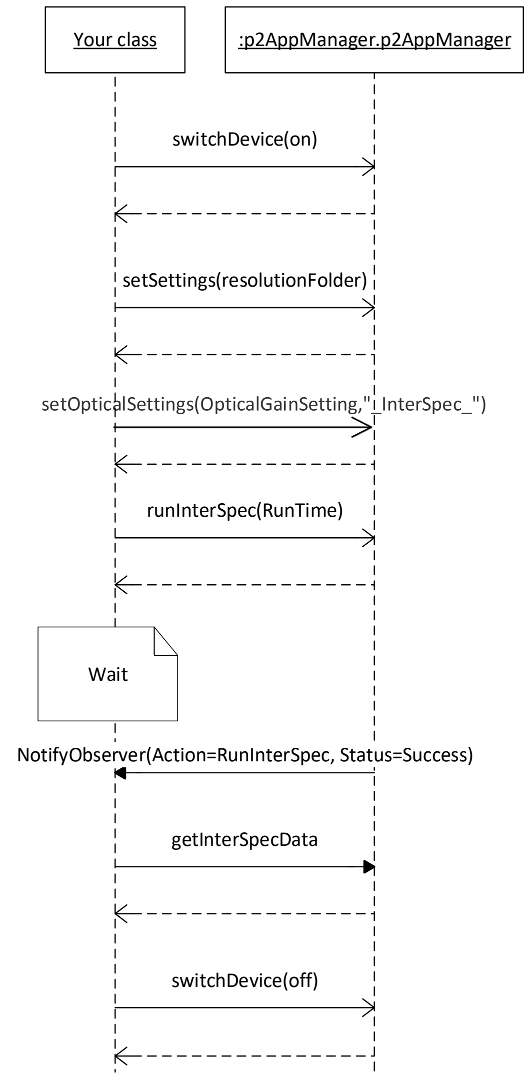

Figure 4: Interferogram and PSD Run Sequence

Spectrum Run

The Spectrum scenario consists of the following steps:

1. Switch on the module through calling `switchDevice(on=true)`
2. Set the calibration folder through calling `setSettings(resolutionFolder=`<selected calibration folder>`)`
3. Set the optical settings through calling setOpticalSettings(opticalGainSettings,"_Spec_")
4. Start the background run procedure through calling `runSpec(RunTime, isSample=false)`
5. Wait for finishing background run
6. Your class will be notified when the background run is finished
7. Start the sample run procedure through calling `runSpec(RunTime, isSample=true)`
8. Wait for finishing sample run
9. Your class will be notified when the sample run is finished
10. Get the data through calling `getSpecData()`
11. Switch off the device through calling `switchDevice(off=false)`

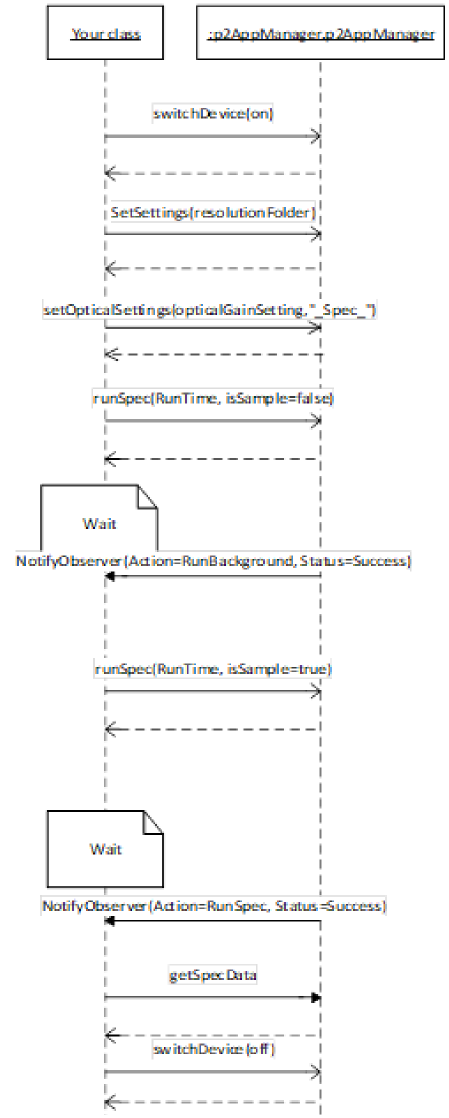

Figure 5: Spectrum Run Sequence

Adding Gain Settings for the Interferogram

Adding new gain settings for the Interferogram consists of the following steps:

1. Switch on the module through calling `switchDevice(on=true)`
2. Set the calibration folder through calling `setSettings(resolutionFolder=<selected calibration folder>)`
3. Start adjusting the gain using background by calling `runInterSpecGainAdj(RunTime)`
4. Wait for finishing background run
5. Your class will be notified when the background run is finished
6. Get the new gain settings by calling `getGainAdjustInterSpecData()`
7. Save the gain settings by calling `saveInterSpecGainSettings(optionName, result)`
8. To burn the gain settings to the module, call the function `burnSettings()`
9. To restore the default gain settings from the module, call the function `restoreDefaultSettings()`

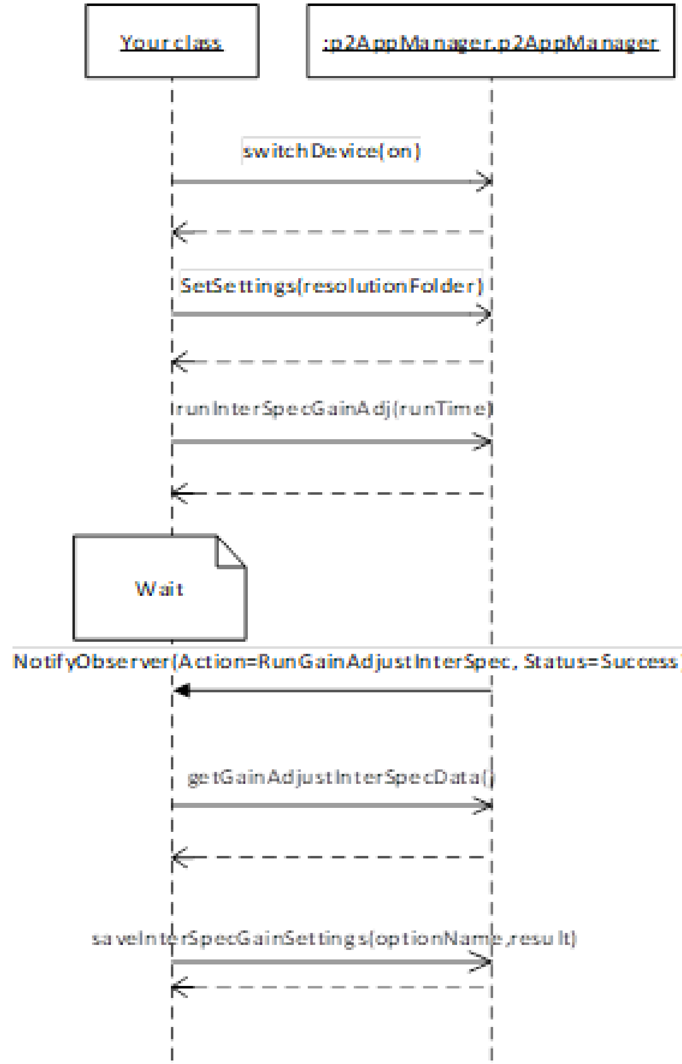

Figure 6: Interferogram Gain Adjustment

Adding Gain Settings for the Spectrum

Adding new gain settings the Spectrum consists of the following steps:

1. Switch on the module through calling `switchDevice(on=true)`
2. Set the calibration folder through calling `setSettings(resolutionFolder=<selected calibration folder>)`
3. Start adjusting the gain first using background by calling `runSpecGainAdjBG(RunTime)`
4. Wait for finishing background run, your class will be notified when the sample run is finished
5. Get the new gain settings by calling `getGainAdjustSpecData()`
6. Save the gain settings by calling `saveSpecGainSettings(optionName, result)`
7. To burn the gain settings to the module, call the function `burnSettings()`
8. To restore the default gain settings from the module, call the function `restoreDefaultSettings()`

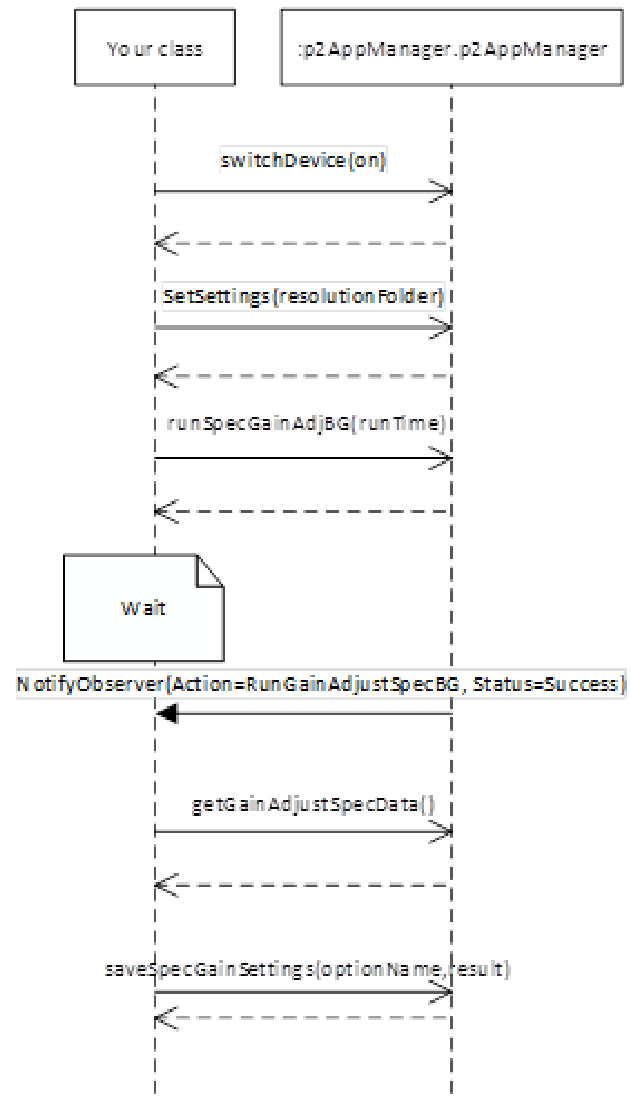

Figure 7: Spectrum Gain Adjustment

Perform Correction

Correction can be done using one of two techniques:

Perform Self-Correction

1. Switch on the module through calling `switchDevice(on=true)`
2. Set the calibration folder through calling `setSettings(resolutionFolder=<two_points_corr folder>)`
3. Set the optical settings through calling setOpticalSettings(opticalGainSettings,"_Spec_")
4. Start the correction using `runCalibCorr()` with a background reading
5. Wait for finishing background run
6. To burn the correction to the module, call the function `burnSettings()`

Note: `burnSettings()` writes the gain settings and the correction settings to the module

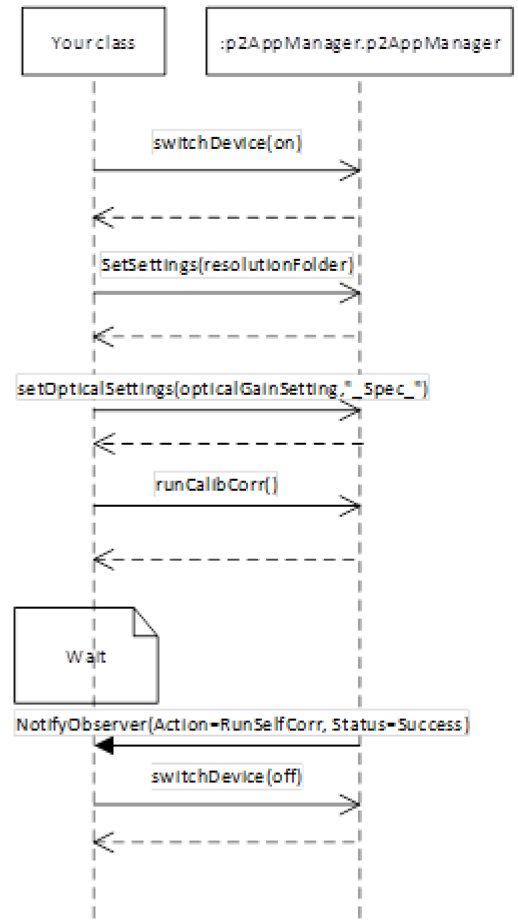

Figure 8: Self Correction

Perform Correction Using a Standard Sample

1. Switch on the module through calling `switchDevice(on=true)`
2. Set the calibration folder through calling `setSettings(resolutionFolder=`<selected calibration folder>`)`
3. Set the optical settings through calling setOpticalSettings(opticalGainSettings,"_Spec_")
4. Start the first step of correction using `wavelengthCalibrationBG()` with a background reading
5. Wait for finishing background run
6. Start the second step of the correction using `wavelengthCalibration()` with a sample reading
7. Wait for finishing the sample run
8. To burn the correction to the module, call the function `burnSettings()`

Note: `burnSettings()` writes the gain settings and the correction settings to the module

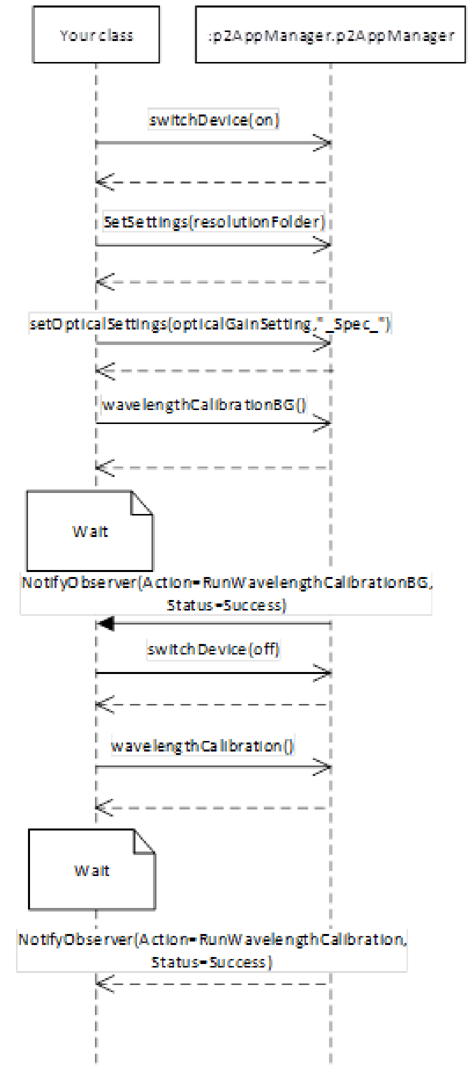

Figure 9: Correction Using Standard Sample

Unlock the Unknown

Ocean Insight exists to end guessing. We equip humanity with technology and data to make precisely informed decisions providing transformational clarity for human advancement in health, safety, and the environment.

Questions?
Chat with us at OceanInsight.com.
info@oceaninsight.com • **US** +1 727-733-2447
EUROPE +31 26-3190500 • **ASIA** +86 21-6295-6600