SpectroMOST-SDK Edition: Description of SWS62221's Software Development Kit

1 Introduction

SWS62221 is a MEMS based FT-IR spectral sensor module developed by Si-Ware that enables measuring materials' NIR spectra.
SpectroMOST is a software application that has two editions:

- SpectroMOST - Basic Edition: This is a Graphical User Interface (GUI) software that enables plotting, saving, and loading NIR spectra measured by SWS62221. This edition is used for demonstration and evaluation purposes.

- SpectroMOST - SDK Edition: This is a Software Development Kit (SDK) that enables the direct interface with the SWS62221 via a set of APIs. This edition is used to control the SWS62221 and to build end-usage application software.

The SDK and Demo editions of SpectroMOST share the same libraries.

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

1.1 Operating systems

SpectroMOST SDK can operate at the following platforms:

- Microsoft Windows XP (both x86 and x64).
- Microsoft Windows Vista (both x86 and x64).
- Microsoft Windows 7 (both x86 and x64).
- Microsoft Windows 8 (both x86 and x64).
- Ubuntu 12.04 (both x86 and x64).

1.2 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 32bit platform
  - spectromost: contains the source code of SpectroMOST for demonstration purpose.

1.3 Installation

SpectroMOST should be installed before proceeding with the SDK installation steps.
After downloading the SDK package the following steps should be performed in Eclipse 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:\SDKv5.0)
- 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 (Spectromost/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:\SDKv5.0\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:

For Windows platforms:

-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:/SDKv5.0/bin_winx64***" -Dswing.defaultlaf=com.sun.java.swing.plaf.nimbus.NimbusLookAndFeel

For Linux based platforms:

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

- Under “Working Directory” section, click on “Other”. Then click on “Workspace”.
- Select the bin folder under the SDK
- Click the “Run” button. The SpectroMOST software will be built and launched.

1.4 Software Architecture

SpectroMOST SDK has the components described below. The interfaces between these components are as shown in Figure 1:

- Application software
  - spectromost.jar: The source code of SpectroMOST Basic Edition is delivered as for reference. This component should be replaced by the end-use application software.
  - 3rd party modules used by spectromost.jar:
    - jcommon-1.0.21.jar
    - jfreechart-1.0.17.jar
    - log4j-1.2.17.jar
    - miglayout15-swing.jar
- Spectrometer driver:
  - p2AppManager.jar (which is the only component from which spectromost.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

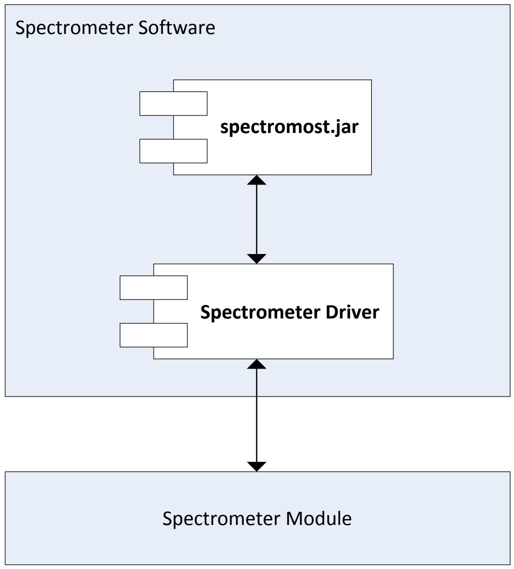

Figure 1: SDK Components

1.5 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

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:

- Conf_Files:
    - Temperature
        - Resolution1 (8nm)
        - Resolution2 (16 nm)
    - savedOpticalSettings

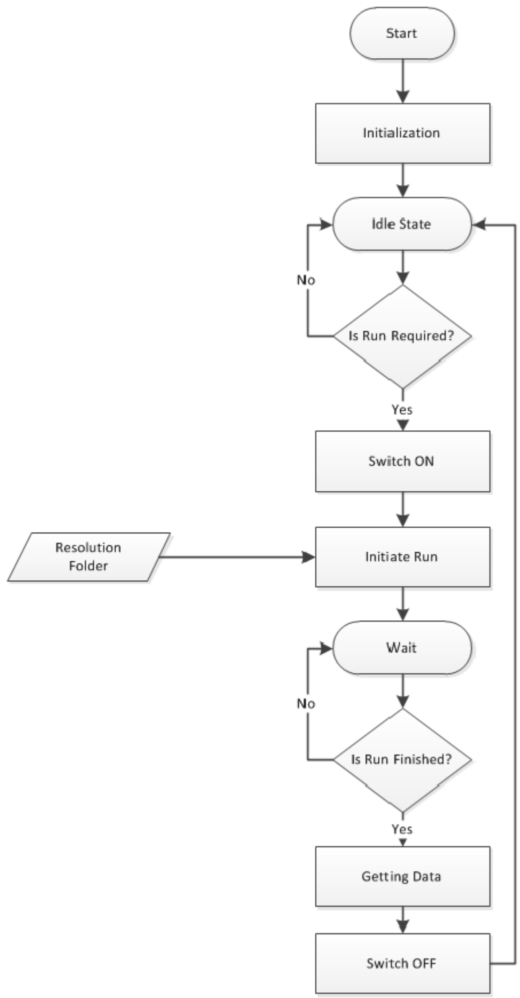

Figure 2: Operation flowchart

2 APIs

2.1 p2AppManager.jar

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

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

2.2 p2AppManager APIs

The p2AppManager component has the following APIs:

1. **Interface: P2AppManagerImpl()**
Description: Component Constructor

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

2. **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.

3. **Interface: getDeviceId()**
Description: Gets the ID of the connected spectrometer module.

Inputs	Outputs	Return	Type
-	String deviceID	Spectrometer ID	Sync

4. **Interface: initializeCore()**
Description: Begin initializing the connected board

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

5. **Interface: setSettings()**
Description: Set the relative path of the resolution folder to be used during the upcoming runs

Inputs	Outputs	Return	Type
- String resolutionFolder:	-	-	Sync

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			

6. 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	-	-	Sync

7. 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	-	p2AppManagerStatus: see Table 3	Async

8. Interface: getSpecData()

Description: Get data corresponding to runSpec function

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

9. 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)	-	p2AppManagerStatus: see Table 3	Async

See Table 1			

10. Interface: getInterSpecData()
Description: Get data corresponding to runInterSpec or runInterSpecInjectedData commands

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

11. Interface: runInterSpecRawData()
Description: Acquire intermediate interferogram signal from neoSpectra. To be processed by "runInterSpecInjectedData" function afterwards.

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

12. Interface: getInjectionData()
Description: Get intermediate interferogram signal corresponding to runInterSpecRawData command

Inputs	Outputs	Return	Type
-		double[][]	Sync

13. Interface: getConfigurationData()
Description: Get configuration data that will be used to configure the system before calling "runInterSpecInjectedData" function afterwards.

Inputs	Outputs	Return	Type
-		double[][]	Sync

14. Interface: setInjectedData()
Description: Set the intermediate interferogram signal corresponding to runInterSpecRawData command

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

15. Interface: setConfigurationData()
Description: Set configuration data that will be used to configure the system before calling "runInterSpecInjectedData" function afterwards.

Inputs	Outputs	Return	Type

- double[][] data	-	-	Sync

16. Interface: runInterSpecInjectedData()
Description: Generate Interferogram and Power Spectral Density from injected intermediate interferogram signal (can be called without a connected neoSpectra).

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

17. Interface: checkDeviceStatus()
Description: Check the current status of the connected device

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

18. 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

19. 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

20. 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[a]: name of the sample to be used - String apodization - String zeroPadding See Table 1	-	p2AppManagerStatus: see Table 3	Async

21. 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

22. 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	Async

23. 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	Async

24. 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

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

25. Interface: getGainAdjustInterSpecData()

Description: Get the gain settings corresponding to runInterSpecGainAdj()

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

26. 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

27. 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

28. Interface: getGainAdjustSpecData()

Description: Get gain settings corresponding to runSpecGainAdjBG()

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

29. Interface: burnSettings()

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

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

30. 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	-	p2AppManagerStatus: see Table 3	Async

gain settings to burn - String updateCorrection: flag if set to true it saves the correction values to the module.			

31. 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

32. Interface: restoreDefaultSettings()
Description: Restore the default gain settings and wavenumber correction settings from the module

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

33. Interface: setWorkingDirectory()
Description: Sets the working directory of the application

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

34. Interface: getWorkingDirectory()
Description: return the current working directory of the application

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

35. Interface: getSDKVersion()
Description: return the version number of the SDK

Inputs	Outputs	Return	Type
-	-	- String : Version number	Sync

Input Data Format

Parameter	Description	Value	Description
Apodization	Shape of the window to be used to multiply the Interferogram before FFT	0	Boxcar
		1	Gaussian
		2	Happ-Genzel
		3	Lorenz
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 [b]
		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

[b] 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	Photo detector'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*	SpectroMOST does not detect any connected NeoSpectra module
3	*BOARD_NOT_INITIALIZED_ERROR*	NeoSpectra module is not initialized
4	*UNKNOWN_ERROR*	Unknown error. Contact Si-Ware Systems
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
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 NeoSpectra module initialization
43	*INVALID_BOARD_CONFIGURATION_ERROR*	Error occurred during NeoSpectra module initialization

50	*DATA_STREAMING_TAIF_ERROR*	Error occurred during streaming from NeoSpectra module
51	*DATA_STREAMING_ERROR*	Error occurred during streaming from NeoSpectra 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*	NeoSpectra 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*	NeoSpectra module is switched off
62	*ASIC_REGISTER_WRITING_ERROR*	Error occurred during writing to chip registers

Table 3: p2AppManagerStatus values

3 Sequence diagrams

3.1 Initialization

The initialization scenario should be run at least once for the connected NeoSpectra 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

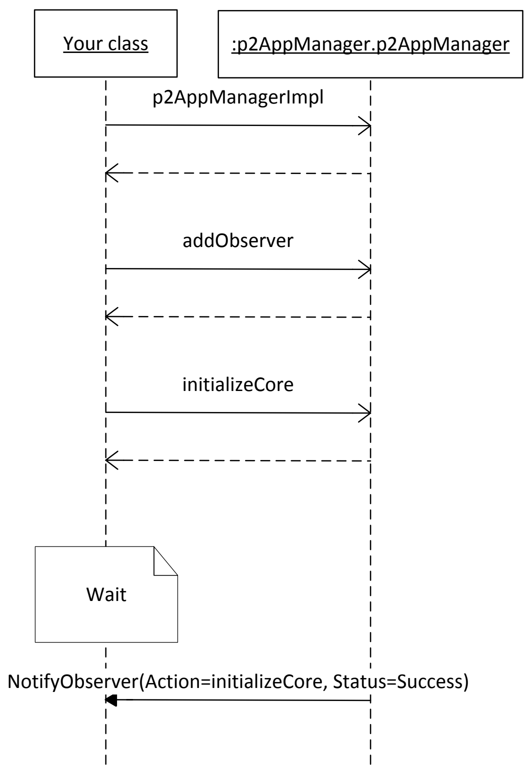

Figure 3: Initialization Sequence

3.2 Interferogram & PSD Run

The Interferogram & PSDscenario 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. Waiting for finishing run
6. Your class will be notified when the run is finished
7. Getting the data through calling getInterSpecData()
8. Switch off the module through calling switchDevice(off=false)

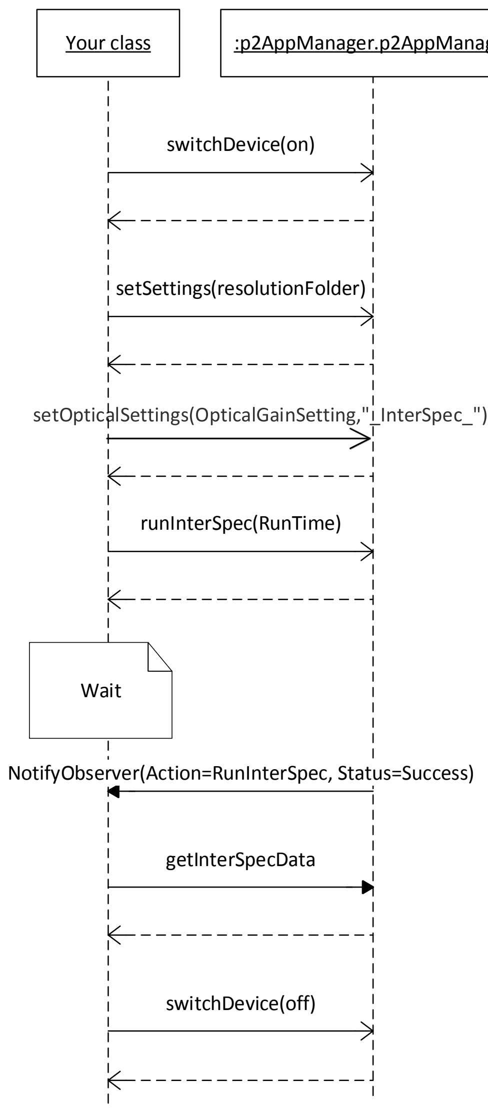

Figure 4: Interferogram & PSD Run Sequence

3.3 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. Waiting 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. Waiting for finishing sample run
9. Your class will be notified when the sample run is finished
10. Getting the data through calling getSpecData()
11. Switch off the device through calling switchDevice(off=false)

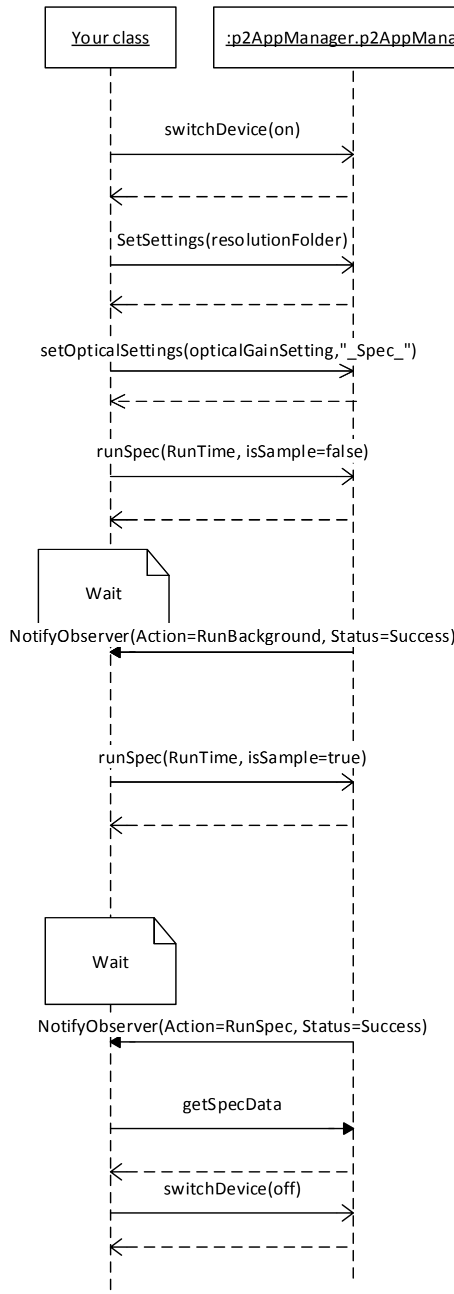

Figure 5: Spectrum Run Sequence

3.4 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. Waiting 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()

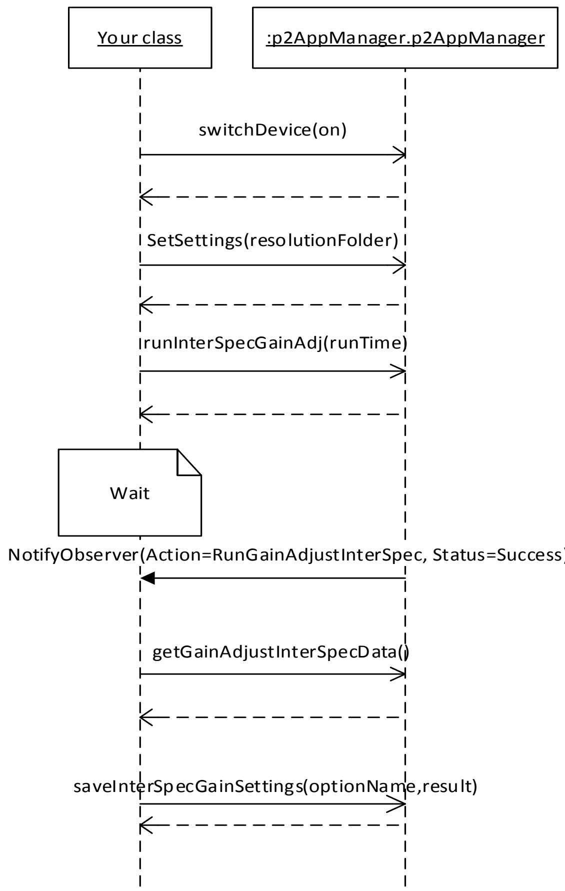

Figure 6: Interferogram Gain Adjustment

3.5 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. Waiting 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()

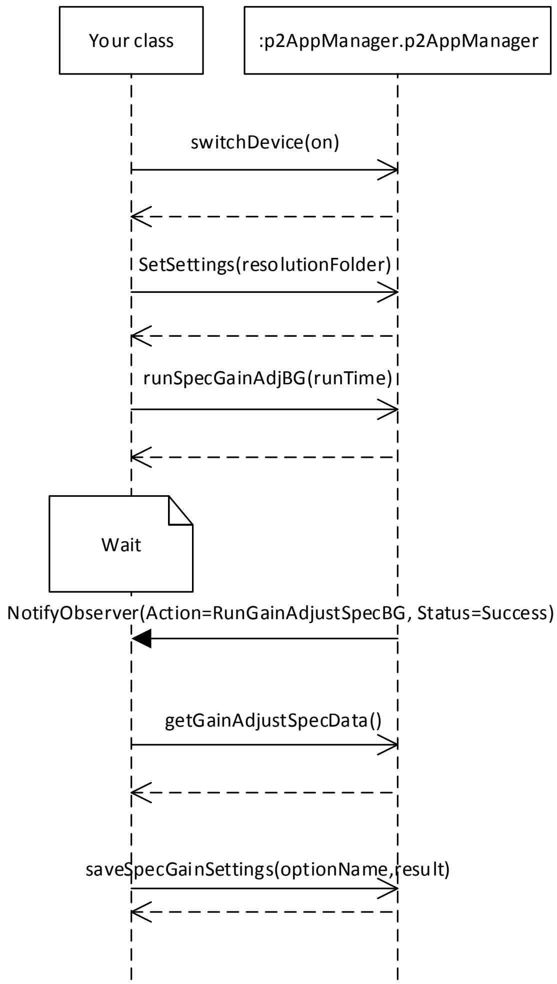

Figure 7: Spectrum Gain Adjustment

3.6 Perform Correction

Correction can be done using one of two techniques:

3.6.1 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

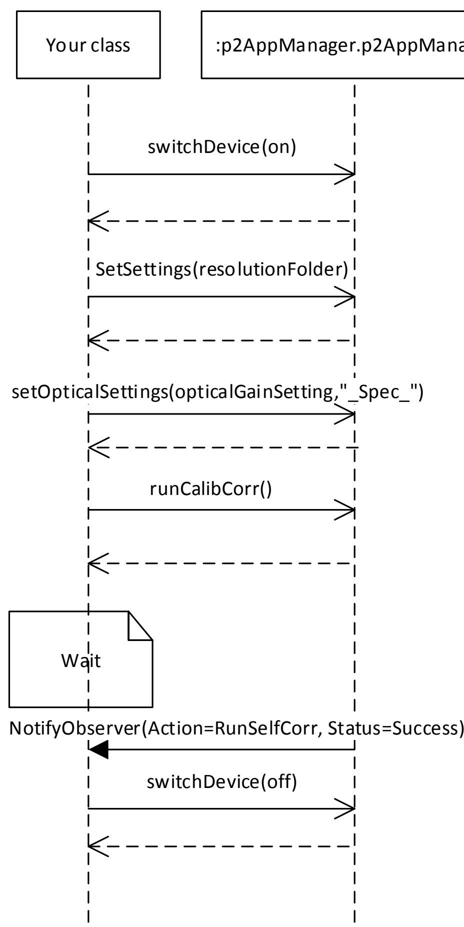

Figure 8: Self Correction

3.6.2 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

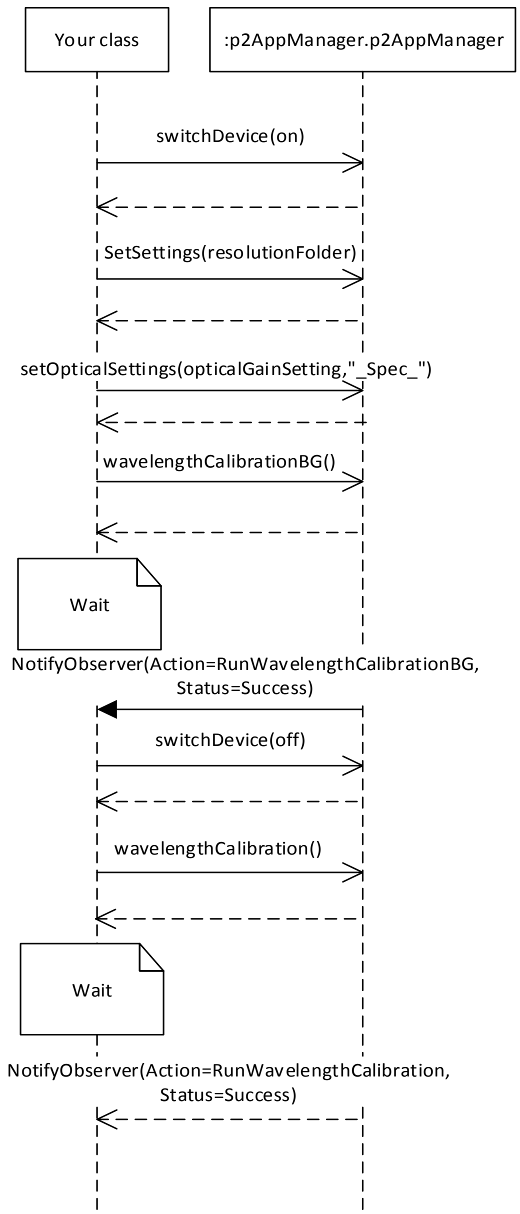

Figure 9: Correction Using Standard Sample

4 Revision History

Revision	Date	Description
1.0	03/02/2015	Initial Version. Delivered before end of development of SDK edition for early feedback.
1.1	05/03/2015	General updates
1.2	15/12/2015	Adjusting the parameters of some API functions
1.3	28/01/2016	Updates for RevB SDK v3.0
1.4	16/05/2016	Add new APIs before CS release
1.5	22/5/2016	Correcting the names of 3 functions
1.6	06/06/2016	Adding extra steps in the installation
1.7	13/06/2016	Adding extra API (getSDKVersion)
1.8	16/06/2016	Fixing the signature of some APIs
1.9	21/11/2016	Add runInterSpecRawData & runInterSpecInjectedData which separate data acquisition and DSP post processing

5 Contact Information

Cairo Office (Headquarters)	**Los Angeles Office**
Si-Ware Systems 3, Khaled Ibn Al-Waleed Street Sheraton, Heliopolis Cairo 11361 Egypt Tel.: +20 2 22 68 47 04 Fax: +20 2 22 68 47 05	Si-Ware Systems, Inc. 1150 Foothill Blvd., Suite M La Canada, CA 91011 USA Tel.: +1 818 790 1151