ISC NIRScan WinForms SDK GUI User's Guide

Contents

List of Figures

List of Tables

Revision History

GUI Version	Date	Description
V3.7.14	2022/10/19	1. Page 10: Add oversampling information.
v3.7.7	2022/01/21	1. Page 9: Modify the behavior of scan average. 2. Page 11: Add reading *.csv files and “Average Scan Data” function.
v3.7.6	2021/11/23	1. Page 49: Add device status and error status description.
	2021/09/06	1. Page 9: Add “Apply to Config” to update scan average numbers. 2. Page 13: Add fan control for specific model.
v3.7.4	2021/06/25	1. Page 9: Add lamp warm-up timer and generate a report to record system information. 2. Page 11: Add “Convert to CSV” function to convert data from .dat file. 3. Page 13: Add logs control, and default is on. 4. Page 20: Add C# API Functions Description.

1.1 Introduction

Upon launching the ISC WinForms SDK GUI, the application checks for the ISC NIRScan enumerating through USB and displays the connected information shown in Figure 1-1. The GUI is divided into three sections:

- The title displays the connected device type and wavelength range.
- The main window includes the following three pages:
  - ***Scan Page***: main functions for scan
  - ***Utility Page***: device related functions
  - ***About Page***: InnoSpectra information
- The status bar displays the connected state, model name and serial number of ISC NIRScan on the bottom-left side, and the device error status displays behind the serial number if some errors happened. The error status can be cleared by the button that displays on the bottom-right of the scan setting of the scan page.

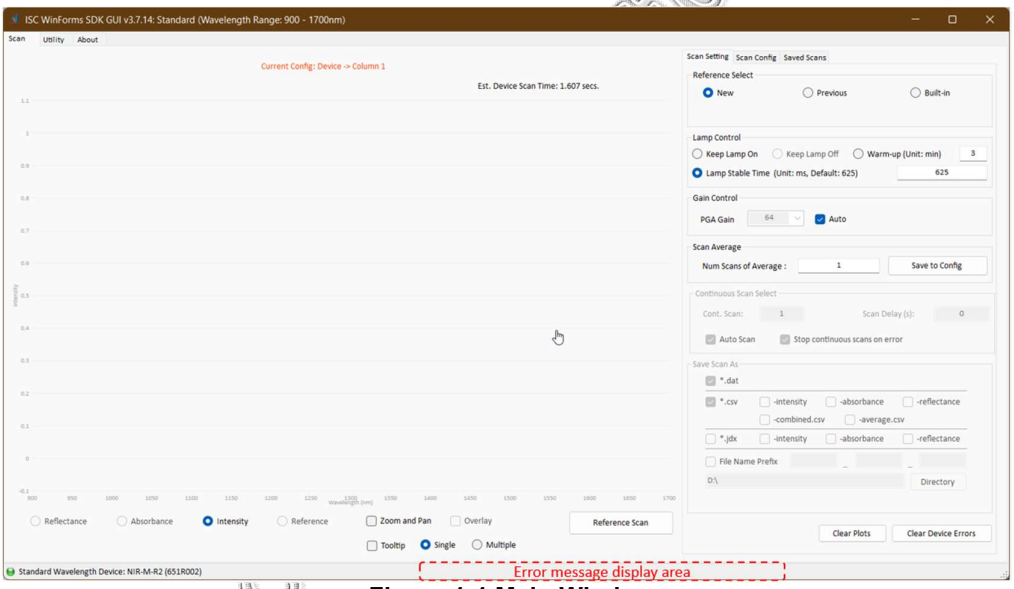


Figure 1-1 Main Window

1.1.1 Scan Page

The scan page is the main window and divided two sections:

- The most area is scan plot area that draws the scan result.
  - The orange word displays the current scan configuration.
  - On the right of current scan configuration is estimated device scan time.
  - On the left of current scan configuration is total scan time that includes USB transaction time.
  - Four radio buttons to redraw the reflectance, absorbance, intensity, or reference.
  - The overlay option can be ticked for multi-scans.
  - The tooltip and zoom and pan options can be ticked to view more detailed data of scan result.
  - The scan button is pressed to start a new scan or reference scan. The reference scan is only useful for a new scan configuration.
- The right section contains scan setting, scan configuration and saved scans, which will be introduced in the 1.1.1.1 to 1.1.1.3.

1.1.1.1 Scan Setting

Figure 1-2 shows the scan setting which includes reference select, lamp control, gain control, save average, continuous scan select, save scan as, and clear all errors button:

- **Reference Selection**: Allows the user to choose the reference for the absorbance or reflectance graph. The reference options include:
  - New: Place a highly reflective material like a metal coated with Spectralon on the sample window and perform a scan. This new scan is stored on the PC and can then be selected with the "Previous" reference radio button.
  - Previous: Choose the reference from the previous use of the "New" option.
  - Built-In: Interpolates the reference stored on TIVA EEPROM at the factory to match the current scan configuration parameters.
- **Lamp Control**: Controls lamp on/off and lamp stable time. When "Lamp Stable Time" is selected, user can set lamp stable time to extend lamp stabilization. This allows the user to avoid any lamp stability issues and reduce lamp wear caused by turning on and off the lamps, as well as the additional time needed to wait for the lamps to stabilize before executing a scan. The "Warm-up" option is used to start lamp-on for certain time. When time's up, the option will be changed to "Lamp Stable Time". During the warm-up period, if user pressed "Cancel" on the Lamp Warm-up UI, then stop the warm-up and do with each function. After the lamp warm-up function is completed, a report will be generated, which stores the system temperature, system humidity, Tiva temperature and lamp ADCs.
- **Gain Control**: Allows the user to choose the gain setting for scan.
  - Auto: System will calculate a suitable gain value.
  - Fixed: User select one gain value.
- **Scan Average**: Allows the user to change the average time. The configuration is automatically corrected after the user completes the changes. The system will determine if the reference needs to be rescanned.
- **Continuous Scan**: Allows the user to do auto repeat scan.
- **Save Scan As**: Allows the user to save which kind of file and where to store them.

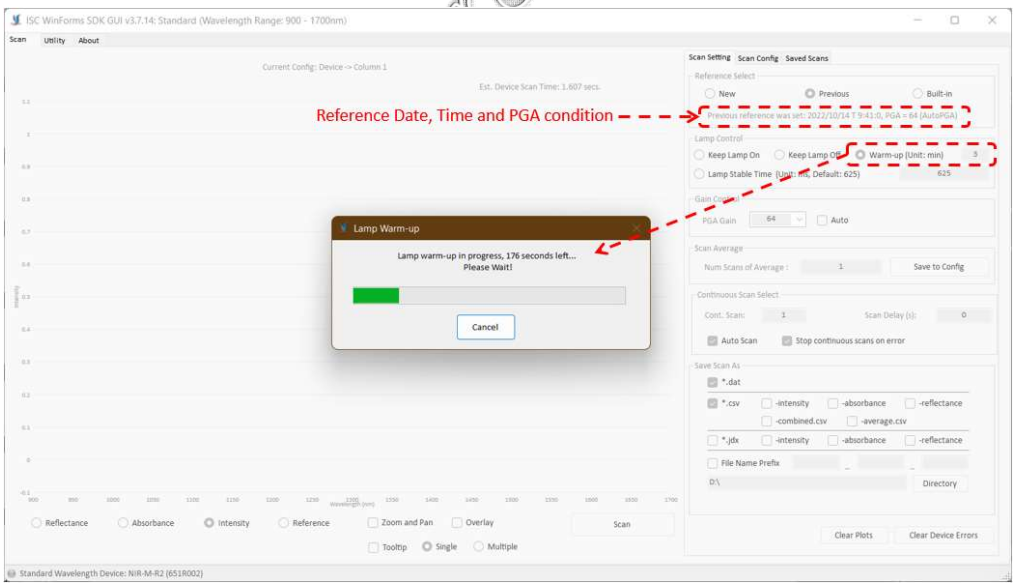


Figure 1-2 Scan Setting

1.1.1.2 Scan Config

The scan config is divided three sections shown in Figure 1-3:

- The top section is the quick selection of scan configuration:
  - Local configuration saved to the PC. Device configuration saved on the device at most 20 sets.
  - Built-in configurations: Column 1, Hadamard 1.
  - *Italic* is the system boot-up configuration which can be set from "Set Device Default Config" button.

- The "Copy" and "Move" buttons allow copying or moving scan configurations stored on the PC to the device or from the device to the PC. (Multiple configurations copy and move in the scan config tab.)
- Single click one configuration that can display data to the Details block. The area of selected configuration will be light blue color filled.
- Double click one configuration that can set to the device, and display with orange color.

- The middle section is the detail contents of the selected scan configuration. If user wants to add a new configuration, the Details block will be empty to edit.
  - **Name**: Configuration name which display to the list.
  - **Number of Scans to Average**: This is the repeated continuous scans that are averaged together.
  - **Number of Sections**: A scan can be broken up into 1 ~ 5 sections. Each section can have individual set of the following parameters:
  - **Scan Type**:
    - Column: Selects one wavelength at a time.
    - Hadamard: Creates a set with several wavelengths multiplexed at a time and then decodes the individual wavelengths.
  - **Spectral Range (nm)**: Start and End wavelengths or spectral range of interest for the scan between 900 nm to 1700 nm or 1350 nm to 2150 nm.
  - **Width (nm)**: This number selects the width of the groups of pixels in the generated Column or Hadamard patterns.
  - **Exposure Time (ms)**: The exposure time can be individually set for each section in the range of 0.635ms to 60.960ms.
  - **Digital Resolution**: This number defines how many wavelength points are captured across the defined spectral range. Each wavelength point corresponds to a pattern that is displayed on the DMD.
  - **Oversampling**: This number displays the rate of sampling patterns. It is calculated by the used digital resolution devided by the number of wavelength range divided by pattern width (rounded to integer). For example in the figure 1-3 will be 228 / ((1700 – 900) / 8) = 2.28. The results shown is rounded to the first decimal digital, so it is 2.3. The software limits the maximum oversampling rate to 4.5 and recommands the rate to be between 2 to 3.

- The bottom section is the buttons to edit the scan configuration:
  - "New" button can create a configuration.
  - "Edit" button can edit the selected configuration.
  - "Delete" button can delete the selected configuration.
  - "Save" button can save editing to local or device.
  - "Cancel" button can quit editing without saving.

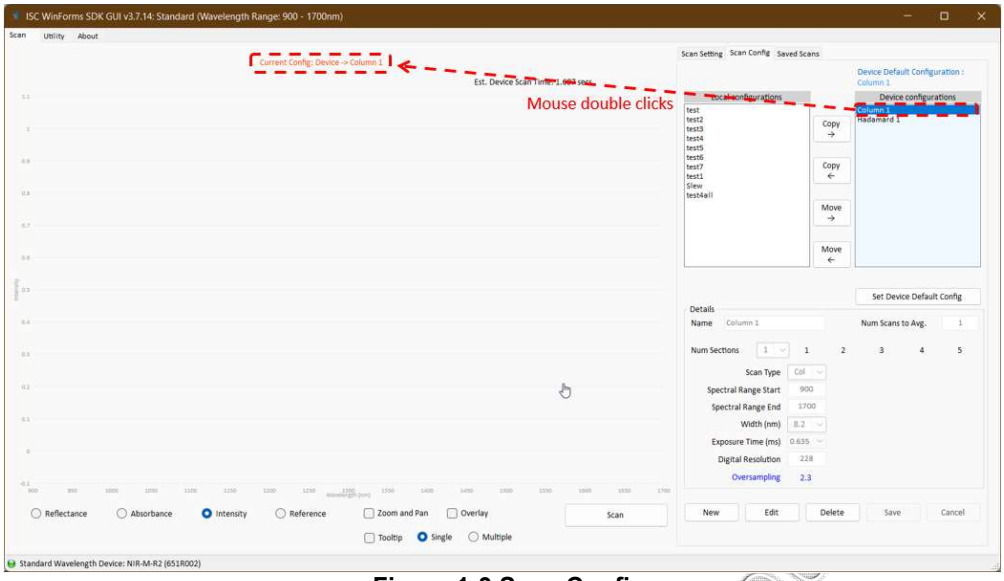


Figure 1-3 Scan Config

1.1.1.3 Saved Scans

The "Saved Scans" tab can read the file online or offline. It supports CSV file format and DAT file format. The scan files are stored with the name of the scan configuration and date and time of the scan. The folder is displayed in the directory, and you can select the file save path.

- To plot a file, select one of the files as shown in Figure 1-4. The selected file will show the scan configuration used at that time. It also supports multiple selection and overlay functions. If selecting multiple scan files, the scan configuration only displays the latest selected information.
- With the increase of files, a file name filter is provided to facilitate searching for specific files. "Clear" button can clear enter the filter name. The list will be refreshed according to the file name filter.
- It supports sorting. "File name" button can files sorted by file name. "Time" button can files sorted by time.
- In addition, it provides mouse right button functions, including delete files, convert to CSV files and average scan data. The "Delete" function deletes all related scan data such as .dat, .csv, etc.

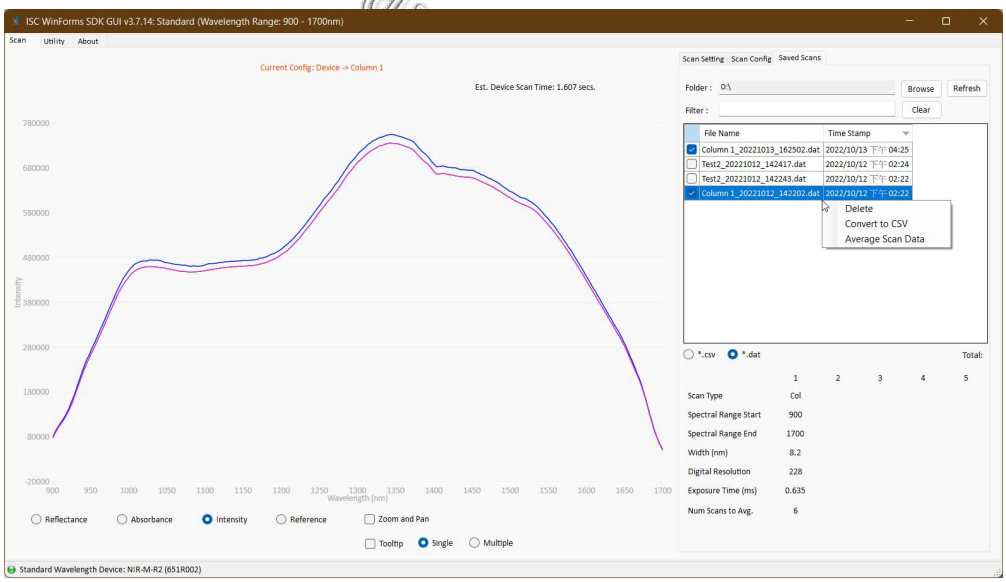


Figure 1-4 Saved Scans

1.1.2 Utility Page

The utility page includes the device related functions and device information shown in Figure 1-5.

- **Model Name**: Allows regex "a-zA-Z0-9_-" to set, and ≤ 15 characters.
- **Serial Number**: Allows regex "a-zA-Z0-9_-" to set, and ≤ 8 characters.
- **Date and Time**: Because there is no RTC battery in the device, the system time is written when the GUI is initialized.
- **Lamp Usage**: According to the module to determine whether the lamp usage can be read or write.
- **TIVA Firmware Update**: Binary File for main board.
- **DLPC150 Firmware Update**: Image File for detector board.
- **Device Information**: Display all information about firmware and hardware. A tooltip is added for double-clicking a message to copy device information.
- **Sensors**:
  - **Battery Charger Status** and **Battery Capacity**: If a Lithium-Ion or Lithium polymer single cell battery is connected.
  - **System Humidity / System Temp**: Reads by the HDC1010 in the Main Board.
  - **Tiva Temp**: Reads by the Tiva internal sensor in the Main Board.
  - **Lamp Intensity / Lamp Voltage and Lamp Current**: Reads the value of the lamp output if the main board version is ≤ D. Read the voltages and currents of the lamp output if the main board version is ≥ F.
- **Calibration Coefficients**:
  - **Calibration Coefficient Parameter Mapping**
    - Pixel to Wavelength: $Wavelength = a \times Pixel^2 + b \times Pixel + c$
    - Shift Vector: $Y = d \times X^2 + e \times X + f$
  - **Read Coeffs**: Read coefficients from the device.
  - When "**Write Enable**" checked, user can set the coefficients to the device.
    - **Write Generic**: Set the default coefficients to the device.
    - **Restore Factory Calibration Data**: The three conditions should be reached.
      (a) The Tiva version of device ≥ 2.1.0.67.
      (b) The device is activated.
      (c) The factory calibration data has saved in the device.
    - **Write Coeffs**: Write coefficients to the device.
- **Activation Key**:
  - **Key Activated Functions**: Lamp Usage Set/Get, Restore Default Calibration Coefficients, Bluetooth LE Advertising Name Set/Get, Button Status Lock/Unlock
  - **Key Not Activated**: None
- **Bluetooth LE Advertising Name**: Sets to the default advertising name, sets the customized advertising name to the device, or gets the current advertising name of the device.
- **Device**:
  - **Reset System**: Reset firmware and application software.
  - **Update Reference Data**: Replace factory reference data to customized reference data.
  - **Restore Factory Reference**: This function only restores the factory reference data, which cannot be performed without backing up the data. The factory reference data is restored from the PC.
  - **Button Lock/Unlock**: Lock or unlock the button on the device.
  - **Fan Enable/Disable**: Enable or disable fan on the device. This function only supports for the NIR-M-R11 model with fan, and the Tiva needs to ≥ v3.5.0.
- **Log File Enable/Disable**: Enable or disable to record logs for the GUI control, and default is enable.

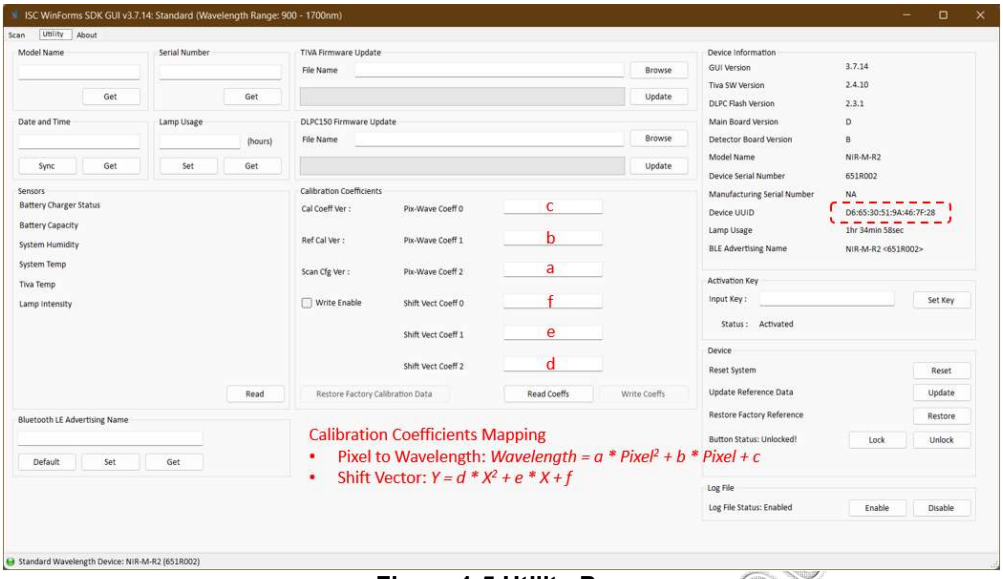


Figure 1-5 Utility Page

1.1.3 About Page

The about page includes current software version, ISC software license agreement and about us shown in Figure 1-6. The about us will link to Inno-Spectra website. The Disable UAC alert writes the registry to get permanent disk write access.

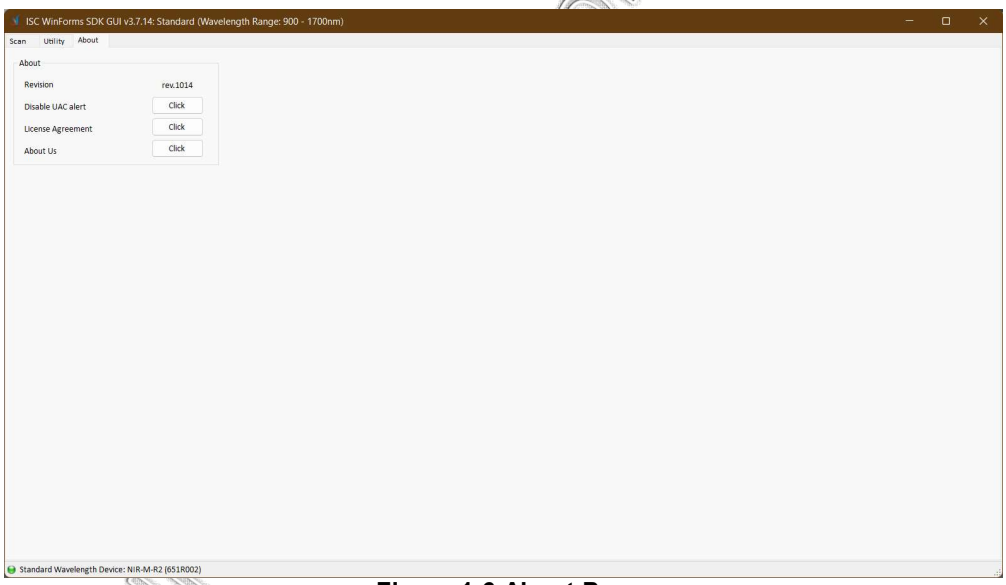


Figure 1-6 About Page

1.2 Performing a Scan

To perform a successful scan, a set of actions must be finished to select a scan configuration, perform a scan and get scan results. If user wants to use a unique scan configuration, the first step is to create a scan configuration. The scan configuration can be used directly when the GUI is opened next time.

1.2.1 Create a Scan Configuration

To create a scan configuration, the operation process is as follows:

(1) Select one of the local or device configurations. The background color of selected configuration list will be set to blue color.

(2) Click "New" button.

(3) Enter the configuration name.

(4) Enter the number of scans to average for corresponding back-to-back scans averaged together.

(5) Enter the number of sections. The section number doesn't exceed 5 sections. Sections can overlap in start and end wavelengths.

(6) For each section:

  (a) Select the scan type: column or hadamard.

  (b) Type in the desired spectral range between 900 to 1700 nm or 1350 nm to 2150 nm.

  (c) Select the width that corresponds to the smallest wavelength content that you want to resolve.

  (d) Enter the desired exposure time.

  (e) Enter the desired digital resolution which is number of wavelength points captured across the spectral range.

(7) After saving the configuration, it will synchronize to Configuration List.

1.2.2 Scan a Local Reference

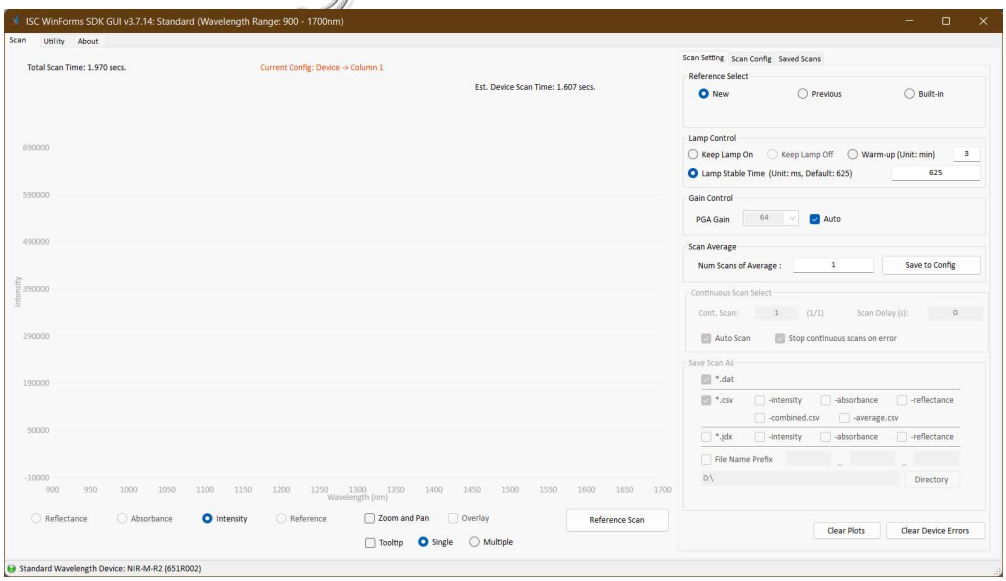
To perform a local reference scan, the operation process is as follows, and the scan result shown in Figure 1-7:

(1) Select a configuration and double click to set to the device.

(2) Select "New" reference to perform a scan. This scan result is stored on the local PC as a "Local Reference" and then you can select it with the "Previous" reference radio button for sample scan.

(3) The scan plot will draw the intensity of reference scan result.

(4) The "New" reference doesn't provide continuous scan selection.



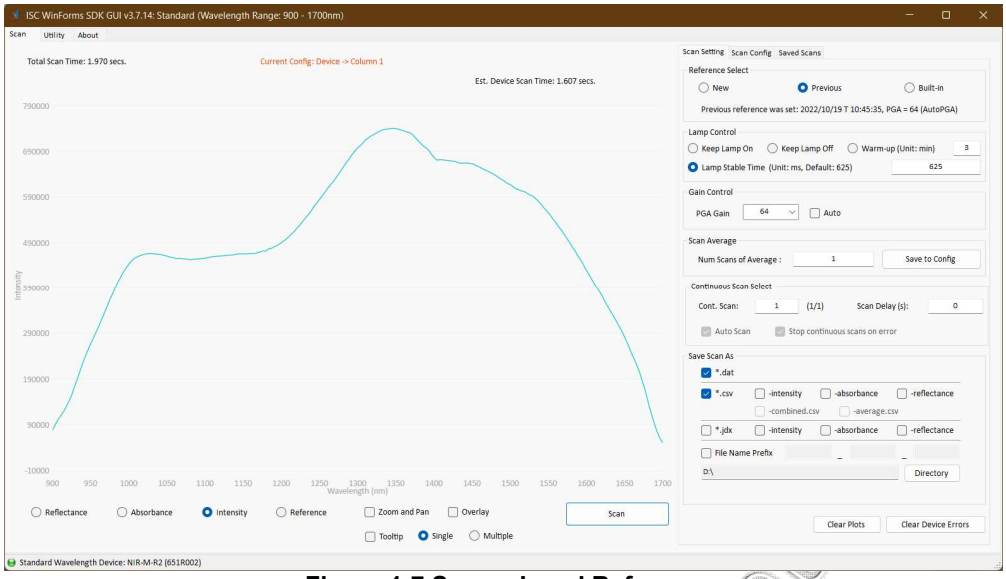


Figure 1-7 Scan a Local Reference

1.2.3 Scan a Sample

To perform a scan, the operation process is as follows, and the scan result shown in Figure 1-8:

(1) Select a configuration on the local or device configuration area and double click on it to set as current scan config. Another way to set configuration is selecting the scan config directly from the current scan configuration label by mouse click.

(2) Select the reference from built-in or previous. Select "Built-in" will use the factory made reference (SRS99) as sample scan reference.

(3) Lamp control and gain control can be set before scanning.

(4) The number scans of average can be Individual adjustment after the scan configuration is set.

(5) The location of the scan is saved under the "Save Scan As."

(6) Click "Scan" button to perform a new scan.

(7) The scan result will be plotted by one of the reflectance, absorbance, intensity or reference selection.

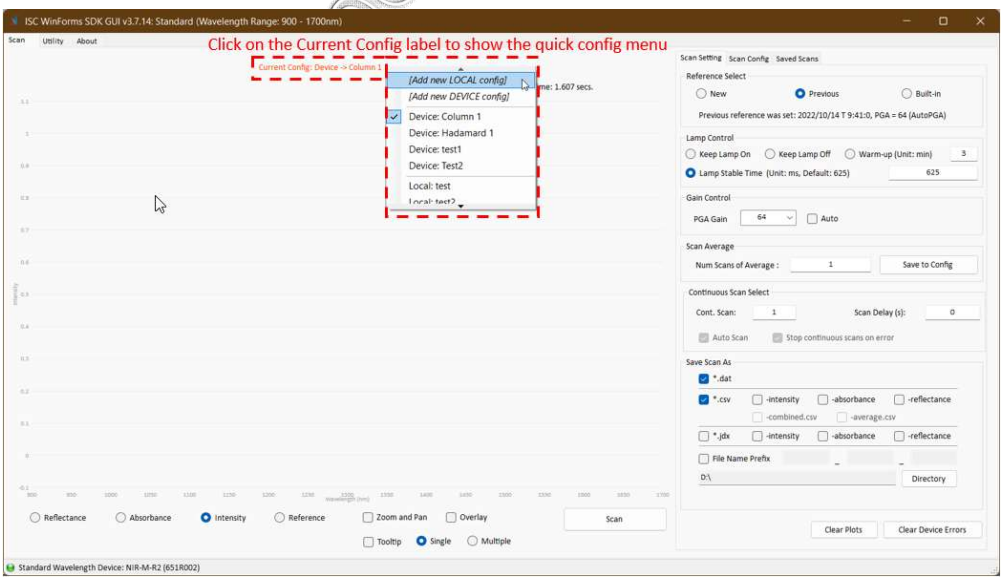


Figure 1-8 Scan a Sample

1.2.4 Continuous Scan

In addition to a single scan also provides continuous scanning, and can overlay the scan results to view trends shown

in Figure 1-9:

(1) Input the number of Continuous Scans and Scan Delay Time.

(2) Click "Scan" button to perform scans.

(3) Press "Cancel" to stop continuous scan if user wants. The user can decide whether to continue the continuous scan, if not, the remaining times will be reset to default.

(4) Set stop continuous scans on error, the spectrum of the scan is abnormal; the continuous scan will automatically stop.

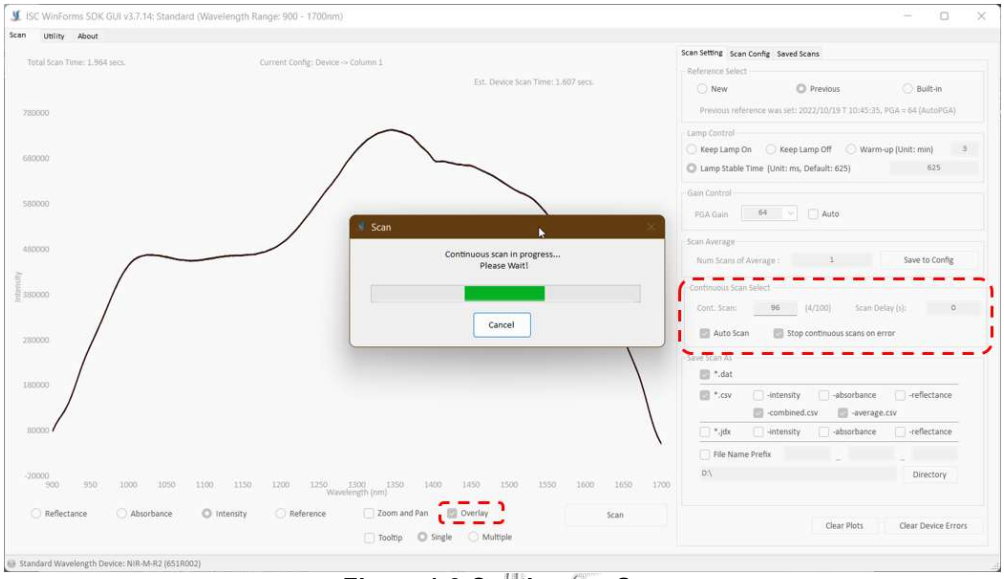


Figure 1-9 Continuous Scan

1.2.5 Scan Workflow

A scan performance requires several preparations to adjust parameters for a scan. Parameters such as scan configuration, repeat times, PGA Value, and reference type are manual input set by user through other API functions, which should be set before scan started.

Once scan performance finished, the scan results are available in lists when Scan.GetScanResult() function called in Scan.PerformScan() function. In addition, the last scan result is always saved in the device if the power is kept on, and it is also applicable by Scan.GetScanResult() function.

Figure 1-10 shows the scan workflow, including single scan and continuous scan. For operations, refer to 1.2.2 to 1.2.4. API functions will be introduced in Chapter 2.

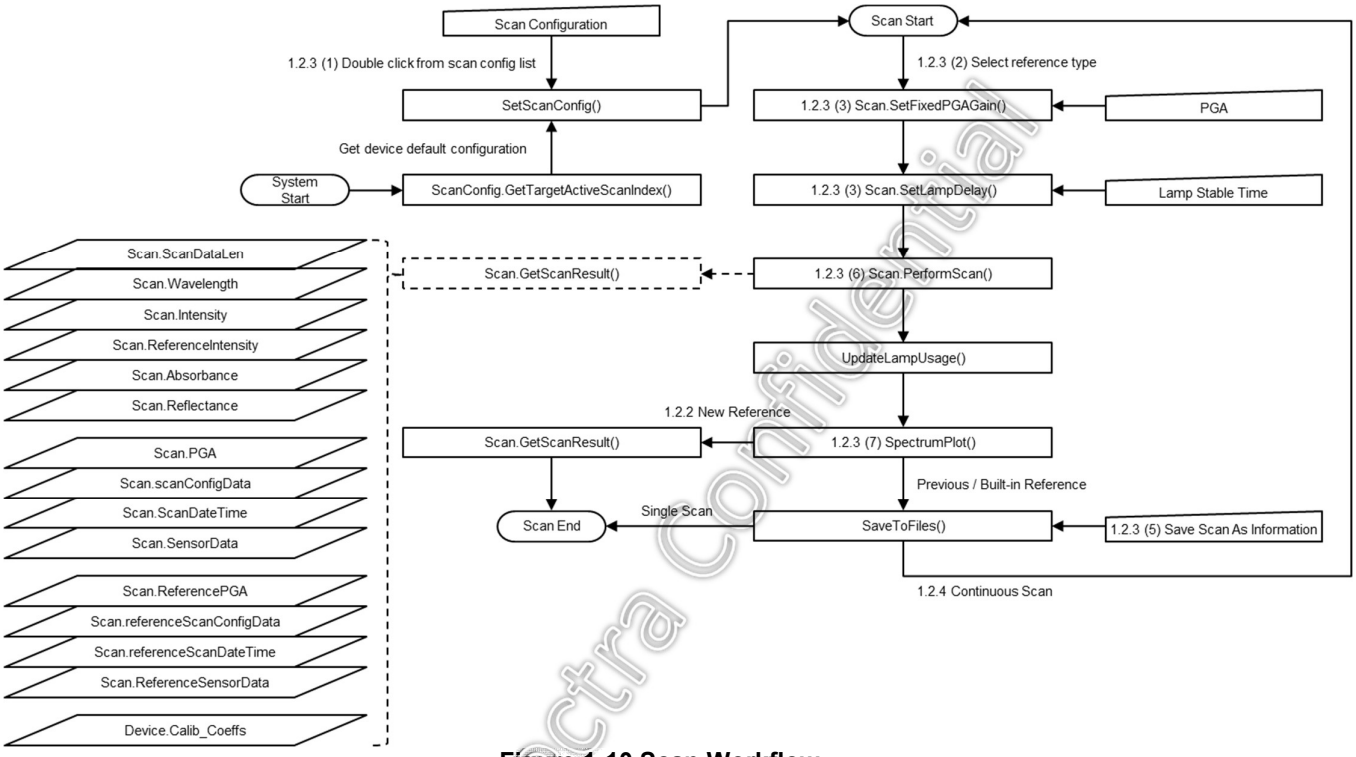


Figure 1-10 Scan Workflow

1.3 Update Reference Data

Before replacing stored reference data, preparing a highly reflective material. A 99% reflective material can be created by coating a metal with Spectralon®.

Before replacing stored reference data, user needs to read User Agreements to agree to bear the consequences shown in Figure 1-11.

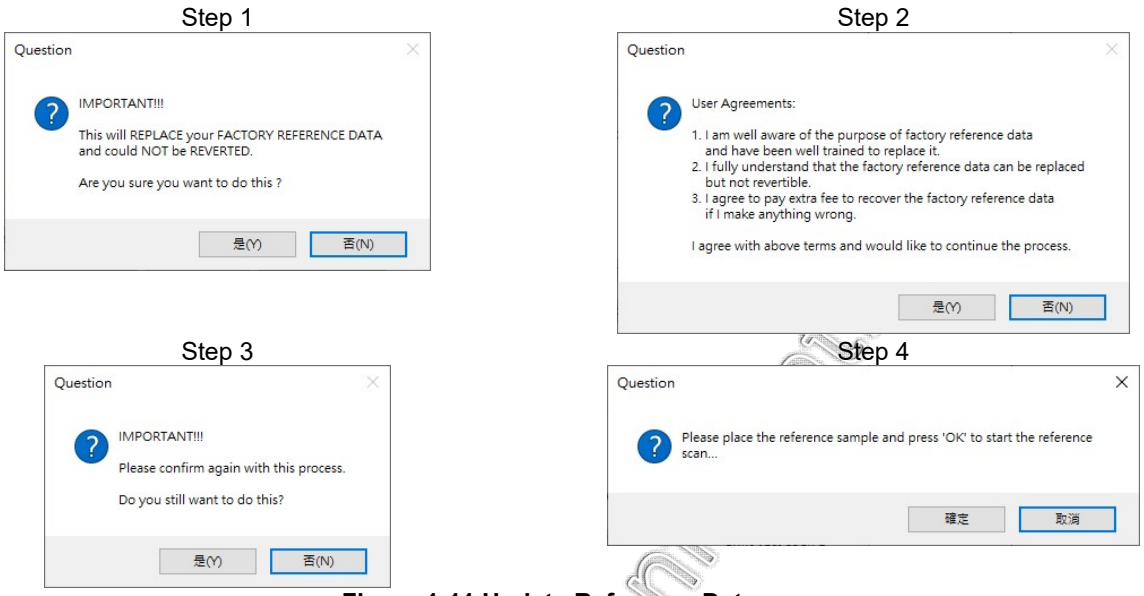


Figure 1-11 Update Reference Data

1.4 Firmware Update

ISC NIRScan includes two firmwares: TIVA and DLPC. If ISC publishes the latest version on the website, users can update them by themselves. If the user is updating TIVA for the first time, please refer to Appendix B.1.

1.4.1 TIVA Firmware Update

Figure 1-12 shows the interface of TIVA Firmware Update. To update the TIVA firmware, click the "Browse" button to search for the TIVA FW file (for example, \\ISC-NIRScan-Tiva-Release-v2.4.7.bin). Then, click the "Update" button. The firmware will be flashed on the TIVA internal Flash while the progress bar indicates the update process. If the TIVA firmware update fails, it will display the corresponding error message.

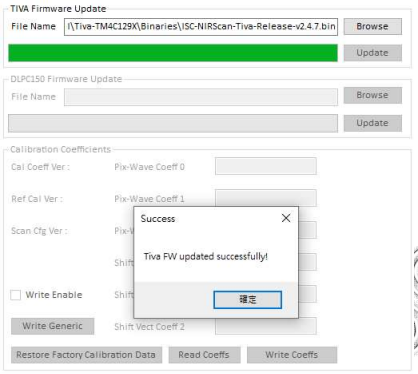


Figure 1-12 TIVA Firmware Update

1.4.2 DLPC150 Firmware Update

Figure 1-13 shows the interface of DLPC150 Firmware Update. To update the DLPC150 firmware, click the "Browse" button to search for the DLPC150 firmware file (for example, \\DLPR150PROM_2.2.0.img). Then, click the "Update" button. The firmware will be flashed to the board while the progress bar indicates the update process. If the DLPC150 firmware update fails, it will display the corresponding error message.

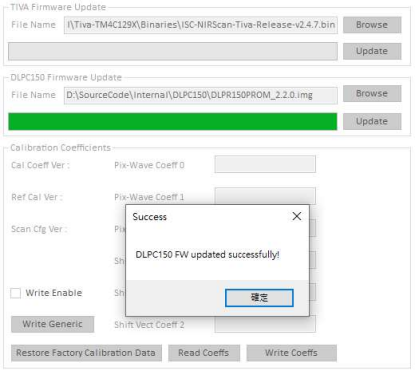


Figure 1-13 DLPC Firmware Update

C# API Functions Description

2.1 Introduction

This API user guide is for InnoSpectra Corporation WinForms SDK GUI executed on Windows in “C#” language.

The API provides a set of “C#” functions to transmit instructions and data between ISC NIRScan and computer through the GUI application. These API functions are also applicable to develop on other use with ISC NIRScan devices.

2.2 API Functions Summary

This section includes function overview in tables, and details for each function are available following by API Functions Summary.

The API functions include the following different class:

- ***Scan Class***
- ***Scan Config Class***
- ***Device Class***
- ***Helper Class***
- ***Generic Class***
- ***Debug Class***

2.2.1 Scan Class

Table 2-1 Methods of Scan Class

Method	Description
SetLamp	Sets lamp status.
SetLampDelay	Sets lighting time of the lamp for a scan.
GetEstimatedScanTime	Estimates cost of time for a scan, transmit time not included.
SetScanNumRepeats	Sets repeat times for a scan.
SetFixedPGAGain	Sets PGA value for a scan. (Supports in Tiva v2.1.0 and above.)
GetPGAGain	Gets PGA value.
SetPGAGain	Sets PGA value for a scan. (Supports under Tiva v2.1.0, not including v2.1.0.)
PerformScan	Executes a scan and performs the results.
GetScanResult	Gets scan results by reading Lists data.
SaveRefernceScan	Saves scan results as reference data.
SaveScanResultToBinFile	Saves scan results as binary file on local space.
ReadScanResultFromBinFile	Reads saved scan results from local space.

Table 2-2 Types of Scan Class

Type	Description
SCAN_REF_TYPE	Options of reference data for scan.
LAMP_CONTROL	Parameters of lamp status for scan.

Table 2-3 Properties of Scan Class

Property	Description
UInt32 ScanDataVersion { get }	Version of scanned data.
String ScanSerialNumber { get }	Serial number of scanned data
Int32 ScanDataLen { get }	Length of scanned data.
ScanConfig.SlewScanConfig ScanConfigData { get }	Configuration details of scanned data.
Byte[ ] ScnDateTiem { get }	Date and time of scanned data.
Double[ ] SensorData { get }	Information about system temperature, sensor temperature, device humity, and lamp condition of scanned data.
Byte PGA { get }	PGA Gain value of scanned data.
List<Double> WaveLength { get }	Wavelength values in nm unit.
List<Int32> Intensity { get }	Intensity values for each point on wavelength.
List<Double> Absorbance { get }	Absorbance values for each point on wavelength.
List<Double> Reflectance { get }	Reflectance values for each point on wavelength.
UInt32 ReferenceScanDataVersion { get }	Version of reference data.
String ReferenceScanSerialNumber { get }	Serial number of reference data
ScanConfig.SlewScanConfig ReferenceScanConfigData { get }	Configuration details of reference data.
Byte[ ] ReferenceScanDateTime { get }	Date and time of reference data.
Double[ ] ReferenceSensorData { get }	Information about system temperature, sensor temperature, device humidity, and lamp condition of reference data.
Byte ReferencePGA { get }	PGA Gain value for of reference data.
List<Int32> ReferenceIntensity { get }	Intensity values for each point on wavelength of reference data.

2.2.2 Scan Config Class

Table 2-4 Methods of Scan Config Class

Method	Description
GetTargetCfgListNum	Gets the number of configurations saved in the device.
GetConfigList	Reads configurations saved in the device and saves to TargetConfig.
SetConfigList	Sets all configurations to the device from TargetConfig.
SetTargetActiveScanIndex	Sets boot-up default configuration.
GetTargetActiveScanIndex	Gets boot-up default configuration.
GetMaxResolutions	Gets the maximum resolution of the target section in selected configuration.
GetHadamardUsedPatterns	Gets the Hadamard patterns used of the target section in selected configuration.
SetScanConfig	Sets configuration to scan.
GetCurrentConfig	Gets current configuration of the device.

Table 2-5 Structures of Scan Config Class

Structure	Description
SlewScanSection	Contains parameters and values set in each section.
SlewScanConfigHead	Contains general titles and values set in each configuration.
SlewScanConfig	Contains section and head structures in each configuration.

Table 2-6 Properties of Scan Config Class

Property	Description
List<SlewScanConfig> TargetConfig	Scan configurations saved in the device.

2.2.3 Device Class

Table 2-7 Methods of Device Class

Method	Description
Init	Initializes setting before USB connection built.
IsConnected	Returns connection status of device.
Open	Reads information saved in the device once connection built.
Close	Cancels the USB connection with device.
Exit	Terminates the USB functionality and clears cache.
IsDFUConnected	Returns connection status of Device Firmware Update.
Enumerate	Lists connected devices.
Information	Reads device information with DeviceInfo variable.
ResetTiva	Resets the system to default setting.
ReadDeviceStatus	Reads device status with DeviceStatus variable.
ReadErrorStatusAndCode	Reads error code when device is in error status.
ResetErrorStatus	Clears error status on the device by reset.
SetBluetooth	Sets Bluetooth status.
ChkBleExist	Checks if hardware for Bluetooth exists.
SetModelName	Sets model name to the device.
ReadModelName	Reads model name of the device.
SetSerialNumber	Sets serial number to the device.
GetSerialNumber	Reads serial number of the device.
SetDataTime	Sets date and time to the device.
GetDateTime	Reads date and time of the device.
WriteLampUsage	Writes time of lamp usage to the device.
ReadLampUsage	Reads time of lamp usage of the device.
ReadSensorsData	Reads sensor information with DevSensors variable.
ReadLampRampUpData	Reads the information of the lamp ramp up after turning on the lamp.
ReadLampRepeatedScanData	Reads the information of the lamp driver between scans.
ReadLampAdcTimeStamp	Reads the time stamp information of the lamp driver.
ReadLampParam	Reads lamp parameters of the device.
GetCalibStruct	Reads calibration coefficients with Calib_Coeffs variable.
SendCalibStruct	Sets calibration coefficients to the device.
SetGenericCalibStruct	Sets generic calibration coefficients to the device.
RestoreDefaultCalibStruct	Restores calibration coefficients with default data to the device.
WriteBleDispName	Writes BLE broadcast name.
ReadBleDispName	Reads BLE broadcast name.
SetButtonLock	Sets the button lock status to the device.
GetButtonLockStatus	Gets the button lock status of the device.
DLPC_SetImageSize	Sets file size for DLPC update.
DLPC_CheckSignature	Checks signature file for DLPC update.
DPLC_FW_Update_WriteData	Updates DLPC firmware with input file and value.
DLPC_Get_Checksum	Reads validation value after DLPC firmware updated.
Set_Tiva_To_Bootloader	Accesses to Tiva system to enable boot loader.
Tiva_FW_Update	Updates Tiva firmware with input file.
Backup_Factory_Reference	Backs up reference data provided from factory.
Restore_Factory_Reference	Resotres reference data provided from factory.
SetActivationKey	Sets activation key code to device.
FetchActivationResult	Gets activation status from the device.
GetActivationResult	Checks application status of activation key.

Table 2-8 Structures of Device Class

Structure	Description
UsbDevice	Contains information about USB connection data, which includes product name and serial number of the device.
DeviceInfo	Contains information about firmware/software data of the device.
DeviceDateTime	Contains information about date and time data of the device.
DeviceSensors	Contains information about sensor data of the device.
CalibCoeffs	Contains wavelength calibration coefficients data of the device.

Table 2-9 Properties of Device Class

Property	Description
UInt32 DeviceStatus	Current status of the device.
UInt32 ErrStatus	Error status of the device.
Byte[ ] ErrCode	Error code to describe error status of the device.
DeviceInfo DevInfo	Information about firmware/software of the device.
DeviceDateTime DevDateTime	Information about date and time of the device.
UInt64 LampUsage	The time the lamp has been used in ms unit.
DeviceSensors DevSensors	Information about sensor of the device.
UInt32[ ] LampADC { get }	Information about lamp driver real-time ADC.
UInt16[ ] LampRampUpADC { get }	Information about lamp driver ramp up ADC.
UInt16[ ] LampRepeatedScanADC { get }	Information about lamp driver ADC between scans.
UInt32[ ] LampAdcTimeStamp { get }	Information about time stamp of lamp driver between scans.
CalibCoeffs Calib_Coeffs	Wavelength calibration coefficients of the device.
Int32 DeviceCounts	The number of devices connected by USB.
UsbDevice[ ] DeviceFound	The number of devices found by USB.

2.2.4 Helper Class

Table 2-10 Methods of Helper Class

Method	Description
ScanTypeIndexToMode	Translates scan type from index to mode.
CfgWidthItemsCount	Determines number of items based on width pattern.
CfgWidthIndexToPixel	Translates configuration width pattern from index to pixel.
CfgWidthIndexToNM	Translates configuration width pattern from index to nm.
CfgWidthPixelToIndex	Translates configuration width pattern from pixel to index.
CfgWidthPixelToNM	Translates configuration width pattern from pixel to nm.
CfgExpItemsCount	Determines number of items based on exposure time.
CfgExpIndexToTime	Translates exposure time from index to time.
CheckRegex	Filterlizes invalid characters in string.
CheckRegex_Chinese	Filterlizes invalid characters in string, Chinese is valid.

2.2.5 Generic Class

Table 2-11 Events of Generic Class

Event	Description
OnBeginConnectingDevice	Takes action to connect to device.
OnDeviceConnected	Takes action when device connected.
OnDeviceConnectionLost	Takes action when device disconnected.
OnDeviceFound	Takes action when connectable device found.
OnDeviceError	Takes action when data transmit error happens on device.
OnErrorStatusFound	Takes action when error status known of the device.
OnBeginScan	Takes action when scan started.
OnScanCompleted	Takes action when scan completed.
OnUSBConnectionBusy	Takes action when USB connection is busy.
OnButtonScan	Takes action when the scan is from the button of the device.

Table 2-12 Properties of Generic Class

Property	Description
RETURN_PASS { get }	Case passed.
RETURN_FAIL { get }	Case failed.
IsConnectionChecking { get, set }	Checks if USB connection set.
IsUsbConnectionBusy { get }	Check if USB connection is busy.
ConnectionCheckInterval { get, set }	Sets checking interval for USB connection every specific time in ms unit, the minimum interval is one second.
DeviceOpenDeley { get, set }	Sets the time for USB connection delay in ms unit, the minimum delay is 200 ms.
IsEnableNotify { get, set }	Notifies the GUI if to execute actions such as connect to the device, device found, and scan start/complete.
AutoSearch	Set true to automatically search for device connections, or set false to stop searching.

2.2.6 Debug Class

Table 2-13 Methods of Debug Class

Method	Description
WriteLine	Prints console to check execution status.
Enable_CPP_Console	Enables CPP console functionality.

2.3 Scan Class

2.3.1 SetLamp

The SetLamp function is used to set up lamp status with LAMP_CONTROL control value.

Declare	**void SetLamp(LAMP_CONTROL control)**	
Parameters	LAMP_CONTROL control	Parameters for lamp status set up, more details are available in LAMP_CONTROL.
Return Values	None	
Comment	None	
See Also	***LAMP_CONTROL***	

2.3.2 SetLampDelay

The SetLampDelay function is used to set the lamp lighting time for a scan to maintain stability of the light.

Declare	**Int32 SetLampDelay(UInt32 ms)**	
Parameters	UInt32 ms	Use millisecond as the unit of waiting time.
Return Values	0	PASS
	< 0	FAIL
Comment	None	
See Also	None	

2.3.3 GetEstimatedScanTime

The GetEstimatedScanTime function is used to estimate the time a scan costs, which does not include the time for data transmit.

Declare	**Double GetEstimatedScanTime( )**	
Parameters	None	
Return Values	> 0	Returns the estimated time.
	≤ 0	FAIL
Comment	None	
See Also	None	

2.3.4 SetScanNumRepeats

The SetScanNumRepeats function is used to set the number of repeat times. According to the repeat times parameter num, a scan would scan `num` times and average results to avoid noise.

Declare	**Int32 SetScanNumRepeats(UInt16 num)**	
Parameters	UInt16 num	It determines how many times to scan and averaged by for each scan.
Return Values	0	PASS
	< 0	FAIL
Comment	None	
See Also	None	

2.3.5 SetFixedPGAGain

The SetFixedPGAGain function is used to check if PGA gain is a constant and set the value. If the value is a constant, set the value with gainVal, or the PGA gain value would be computed by the device.

Declare	**Int32 SetFixedPGAGain(Boolean isFixed, Byte gainVal)**	
Parameters	Boolean isFixed	The value can be true or false.
	Byte gainVal	The value can be 0, 1, 2, 4, 8, 16, 32, or 64. 0 indicates the PGA gain value should be computed. 1 to 64 indicates the PGA gain value is available.
Return Values	0	PASS
	< 0	FAIL
Comment	The PGA value determines the scale of result on the plot, the scaled values in auto PGA mode would not be out of range on the plot. Example: The following are supported in Tiva v2.1.0 and above. • Auto PGA: SetFixedPGAGain(true, 0) • Fixed PGA: SetFixedPGAGain(true, 64) The following are only supported under Tiva v2.1.0, not including v2.1.0. • Auto PGA: SetFixedPGAGain(false, 0) • Fixed PGA: SetPGAGain(64)	
See Also	***GetPGAGain, SetPGAGain***	

2.3.6 GetPGAGain

The GetPGAGain function is used to get the PGA gain value in order to execute a scan.

Declare	**Int32 GetPGAGain( )**	
Parameters	None	
Return Values	> 0	Returns valid PGA Gain value.
	≤ 0	FAIL
Comment	None	
See Also	***SetFixedPGAGain, SetPGAGain***	

2.3.7 SetPGAGain

The SetPGAGain function is used to set the PGA gain value in order to execute a scan.

Declare	**Int32 SetPGAGain(Byte gainVal)**	
Parameters	Byte gainVal	The value can be 1, 2, 4, 8, 16, 32, or 64.
Return Values	0	PASS
	< 0	FAIL
Comment	This function is only supported under Tiva v2.1.0, not including v2.1.0.	
See Also	***SetFixedPGAGain, GetPGAGain***	

2.3.8 PerformScan

The PerformScan function is used to execute a scan based on set reference type then perform the scan result, which is read from Lists of Wavelength, Intensity, Absorbance, Reflectance, Reference Intensity, and Scan Data Length.

Declare	**Int32 PerformScan(SCAN_REF_TYPE ref_sel)**	
Parameters	SCAN_REF_TYPE ref_sel	Parameter to determine reference type for a scan, more details is available in SCAN_REF_TYPE.
Return Values	0	PASS
	< 0	FAIL
Comment	None	
See Also	***GetScanResult, SCAN_REF_TYPE***	

2.3.9 GetScanResult

The GetScanResult function is used to read scan results and write data into Lists.

Declare	**void GetScanResult( )**	
Parameters	None	
Return Values	None	
Comment	None	
See Also	***PerformScan, CalibCoeffs, Properties of Scan Class***	

2.3.10 GetRefTime

The GetRefTime function is used get the reference time of build-in reference and previous reference.

Declare	**Int32 GetRefTime(SCAN_REF_TYPE ref_sel)**	
Parameters	SCAN_REF_TYPE ref_sel	Parameter to determine reference type, more details is available in SCAN_REF_TYPE.
Return Values	0	PASS
	< 0	FAIL
Comment	None	
See Also	***SCAN_REF_TYPE***	

2.3.11 SaveRefernceScan

The SaveRefernceScan function is used when the user determines to replace the reference data saved in the device with new scan, then the scan result would be used when user scan with SCAN_REF_BUILT_IN parameter.

Declare	**Int32 SaveReferenceScan( )**	
Parameters	None	
Return Values	0	PASS
	< 0	FAIL
Comment	The function can be processed only if the PerformScan( ) has been completed to avoid saving error.	
See Also	***PerformScan***	

2.3.12 SaveScanResultToBinFile

The SaveScanResultToBinFile is used to save scan results as binary file to local space.

Declare	**Int32 SaveScanResultToBinFile(String FileName)**	
Parameters	String FileName	The file name must contain the whole path to save the file.
Return Values	0	PASS
	-1	FAIL
Comment	None	
See Also	***ReadScanResultFromBinFile***	

2.3.13 ReadScanResultFromBinFile

The ReadScanResultFromBinFile function is used to read saved binary file of scan results from local space.

Declare	**Int32 ReadScanResultFromBinFile(String FileName)**	
Parameters	String FileName	The file name must contain the whole path of saved file.
Return Values	0	PASS
	-1	FAIL
Comment	None	
See Also	***SaveScanResultToBinFile***	

2.3.14 SCAN_REF_TYPE

The SCAN_REF_TYPE type is used to determine which type of reference data the user want to use to scan.

Declare	enum SCAN_REF_TYPE	
Parameters	SCAN_REF_BUILT_IN = 0	The built in reference data saved in the device.
	SCAN_REF_PREV = 1	The previous scanned data saved on local space.
	SCAN_REF_NEW = 2	The following scan would be set as reference data.
Comment	The SCAN_REF_PREV parameter can be set only if a scan with SCAN_REF_NEW parameter has been processed.	
See Also	***PerformScan***	

2.3.15 LAMP_CONTROL

The LAMP_CONTROL type is used to determine lamp status.

Declare	enum LAMP_CONTROL	
Parameters	AUTO = 0	The light would be turned on and off automatically.
	ON_SCAN = 1	The light would be turned and kept on.
	OFF_SCAN = 2	The light would be turned and kept off.
Comment	None	
See Also	***SetLamp***	

2.4 Scan Config Class

2.4.1 GetTargetCfgListNum

The GetTargetCfgListNum function is used to read the number of saved configurations in the device.

Declare	**Int32 GetTargetCfgListNum( )**	
Parameters	None	
Return Values	≥ 0	Returns the number of configurations.
	-1	FAIL
Comment	None	
See Also	***GetConfigList***	

2.4.2 GetConfigList

The GetConfigList function is used to get configurations saved in device, and then copy the data to TargetConfig.

Declare	**Int32 GetConfigList( )**	
Parameters	None	
Return Values	0	PASS
	< 0	FAIL
Comment	This function works to read configuration data from the device if return value from GetTargetCfgListNum() function > 0.	
See Also	***GetTargetCfgListNum***	

2.4.3 SetConfigList

The SetConfigList function is used to set configuration data from TargetConfig to device since the logic is to write all configurations to the device once, and the variable TargetConfig is used to save temporary data.

Declare	**Int32 SetConfigList( )**	
Parameters	None	
Return Values	0	PASS
	< 0	FAIL
Comment	None	
See Also	None	

2.4.4 SetTargetActiveScanIndex

The SetTargetActiveScanIndex function is used to set default configuration with index when the device boot-up.

Declare	**Int32 SetTargetActiveScanIndex(Int32 index)**	
Parameters	Int32 index	The index of default configuration.
Return Values	0	PASS
	< 0	FAIL
Comment	None	
See Also	***GetTargetActiveScanIndex***	

2.4.5 GetTargetActiveScanIndex

The GetTargetActiveScanIndex function is used to get index of default configuration when the device boot-up.

Declare	**Int32 GetTargetActiveScanIndex( )**	
Parameters	None	
Return Values	≥ 0	Returns index of default configuration.
	< 0	FAIL
Comment	None	
See Also	***SetTargetActiveScanIndex***	

2.4.6 GetMaxResolutions

The GetMaxResolutions function is used to get digital resolution value in specific section of selected configuration.

The value can't exceed the maximum value, or it would lead to error during a scan.

Declare	Int32 GetMaxResolutions(SlewScanConfig scanCfg, Int32 section)	
Parameters	SlewScanConfig scanCfg	The selected configuration.
	Int32 section	The target section of selected configuration.
Return Values	≥ 0	Returns the maximum pattern value of the section.
	< 0	FAIL
Comment	None	
See Also	***SlewScanConfig***	

2.4.7 GetHadamardUsedPatterns

The GetHadamardUsedPatternsfunction is used to get Hadamard pattern value in specific section of selected configuration.

Declare	Int32 GetHadamardUsedPatterns(SlewScanConfig scanCfg, Int32 section)	
Parameters	SlewScanConfig scanCfg	The selected configuration.
	Int32 section	The target section of selected configuration.
Return Values	≥ 0	Returns the Hadamard pattern value of the section.
	< 0	FAIL
Comment	None	
See Also	***SlewScanConfig***	

2.4.8 SetScanConfig

The SetScanConfig function is used to set selected configuration to scan.

Declare	Int32 SetScanConfig(SlewScanConfig scanCfg)	
Parameters	SlewScanConfig scanCfg	The selected configuration used to set.
Return Values	≥ 0	PASS
	< 0	FAIL
Comment	None	
See Also	***SlewScanConfig***	

2.4.9 GetCurrentConfig

The GetCurrentConfig function is used to get current configuration of the device.

Declare	SlewScanConfig GetCurrentConfig( )	
Parameters	None	
Return Values	SlewScanConfig	Returns the current scan configuration.
Comment	None	
See Also	***SlewScanConfig***	

2.4.10 SlewScanSection

The SlewScanSection structure is used to record variables and values in each section.

Declare	struct SlewScanSection	
Parameters	Byte section_scan_type	The scan mode for each section could be set as Column/Hadamard.
	Byte width_px	The width for each pattern in pixel unit. Unit translate function is available in Helper.cs.
	UInt16 wavelength_start_nm	The wavelength start point in each section in nm unit.
	UInt16 wavelength_end_nm	The wavelength end point in each section in nm unit.
	UInt16 num_patterns	The total amount of patterns in each section.
	Exposure_time	The exposure time index value, which can be translated in Helper.cs.
Comment	None	
See Also	***SlewScanConfig, SlewScanConfigHead, Helper Class***	

2.4.11 SlewScanConfigHead

The SlewScanConfigHead structure is used to record titles and values in each configuration.

Declare	struct SlewScanConfigHead	
Parameters	Byte scan_type	The scan type.
	UInt16 scanConfigIndex	The index of configuration in the device, and the maximum amount of configuration is 20.
	String ScanConfig_serial_number	The serial number composed of letters or numbers with length = 8.
	String config_name	The set name of configuration composed of letters or numbers with maximum length 40.
	UInt16 num_repeats	The number of repeat and average times. The averaged data with multiple scans helps reduce noise but costs additional time to scan.
	Byte num_sections	The number of sections set for scan with maximum number of sections 5. Parameters in each section would be recorded in SlewScanSection.
Comment	None	
See Also	***SlewScanConfig, SlewScanSection***	

2.4.12 SlewScanConfig

The SlewScanConfig structure is used to record head and section details of configuration.

Declare	struct SlewScanConfig	
Parameters	SlewScanConfigHead head	Contains head information, see more in SlewScanConfigHead.
	SlewScanSection[ ] section	Contains details in each section, see more in SlewScanSection.
Comment	None	
See Also	***SlewScanConfigHead, SlewScanConfig***	

2.5 Device Class

2.5.1 Init

The Init function is used to initialize all setting before making USB connection.

Declare	**Int32 Init( )**	
Parameters	None	
Return Values	0	PASS
	-1	FAIL
Comment	None	
See Also	None	

2.5.2 IsConnected

The IsConnected function is used to check if USB connection made successfully.

Declare	**bool IsConnected( )**	
Parameters	None	
Return Values	True	Connected
	False	Disconnected
Comment	None	
See Also	None	

2.5.3 Open

The Open function is used to get information on the device such as device information, calibration coefficients, and configurations saved in the device after the device is connected through USB with specific serial number.

Declare	**void Open(String serNum)**	
Parameters	String serNum	Serial number of the device.
Return Values	None	
Comment	None	
See Also	None	

2.5.4 Close

The Close function is used to disconnect the USB connection of the device. The light would be turned off if it was on before disconnecting.

Declare	**Int32 Close( )**	
Parameters	None	
Return Values	0	PASS
	-1	FAIL
Comment	None	
See Also	None	

2.5.5 Exit

The Exit function is used to terminate the USB functionality and clear setting and cache.

Declare	**Int32 Exit( )**	
Parameters	None	
Return Values	0	PASS
	< 0	FAIL
Comment	None	
See Also	None	

2.5.6 IsDFUConnected

The IsDFUConnectedfunction is used to check the USB device is in Device Firmware Update mode or not.

Declare	**bool IsDFUConnected( )**

Parameters	None	
Return Values	True	Connected
	False	Not connected
Comment	None	
See Also	None	

2.5.7 Enumerate

The Enumerate function is used to enumerate all connected devices.

Declare	**bool Enumerate( )**	
Parameters	None	
Return Values	0	PASS
	< 0	FAIL
Comment	None	
See Also	None	

2.5.8 Information

The Information function is used to read information on device, and the content of information is available in DeviceInfo.

Declare	**Int32 Information( )**	
Parameters	None	
Return Values	0	PASS
	-1	FAIL
Comment	None	
See Also	***DeviceInfo***	

2.5.9 ResetTiva

The ResetTiva function is used to reset the system to default setting when the reset action is asked.

Declare	**Int32 ResetTiva(bool isFWupdate)**	
Parameters	bool isFWupdate	The parameter is true if the reset action is asked from firmware update, or it is asked by user.
Return Values	0	PASS
	< 0	FAIL
Comment	None	
See Also	None	

2.5.10 ReadDeviceStatus

The ReadDeviceStatus function is used to read status of the device, information is available from DeviceStatus.

Declare	**Int32 ReadDeviceStatus( )**	
Parameters	None	
Return Values	0	PASS
	< 0	FAIL
Comment	None	
See Also	DeviceStatus	

2.5.11 ReadErrorStatusAndCode

The ReadErrorStatusAndCode function is used to read error status and error code of the device from ErrStatus and ErrCode variables.

Declare	**Int32 ReadErrorStatusAndCode( )**	
Parameters	None	
Return Values	0	PASS
	< 0	FAIL
Comment	None	
See Also	ErrStatus, ErrCode	

2.5.12 ResetErrorStatus

The ResetErrorStatus function is used to clear error status of the device by resetting.

Declare	**Int32 ResetErrorStatus(UInt32 field)**	
Parameters	UInt32 field	Set field = 0 to clear all error status, or set specific field to clear specific error status. The value is as follows: 0x00000001: Scan Error 0x00000002: ADC Error 0x00000004: SD Card Error 0x00000008: EEPROM Error 0x00000010: BLE Error 0x00000020: Spectrum Library Error 0x00000040: Hardware Error 0x00000080: TMP Sensor Error 0x00000100: HDC Sensor Error 0x00000200: Battery Error 0x00000400: Insufficient Memory Error 0x00000800: UART Error 0x00001000: System Error
Return Values	0	PASS
	< 0	FAIL
Comment	None	
See Also	None	

2.5.13 SetBluetooth

The SetBluetooth function is used to set Bluetooth status on the device.

Declare	**Int32 SetBluetooth(Boolean Enable)**	
Parameters	Boolean Enable	The Bluetooth is on if Enable variable is true, or it is off.
Return Values	0	PASS
	< 0	FAIL
Comment	None	
See Also	None	

2.5.14 ChkBleExist

TheChkBleExist function is used to ensure if the Bluetooth board is attached on the device.

Declare	**Int32 ChkBleExist( )**	
Parameters	None	
Return Values	1	Bluetooth board attached.
	≤ 0	Bluetooth board non-exists.
Comment	None	
See Also	None	

2.5.15 SetModelName

The SetModelName function is used to set new model name with input value on the device.

Declare	Int32 SetModelName(String Name)	
Parameters	String Name	The input of new model name to set, the maximum length is 15 characters.
Return Values	0	PASS
	< 0	FAIL
Comment	None	
See Also	*ReadModelName*	

2.5.16 ReadModelName

The ReadModelName function is used to read model name from the device.

Declare	Int32 ReadModelName(StringBuilder Name)	
Parameters	StringBuilder Name	The output of model name of the device.
Return Values	0	PASS
	< 0	FAIL
Comment	None	
See Also	*SetModelName*	

2.5.17 SetSerialNumber

The SetSerialNumber function is used to set new serial number on the device.

Declare	Int32 SetSerialNumber(String Number)	
Parameters	String Number	The input of new serial number to set, the maximum length is 8 digits.
Return Values	0	PASS
	< 0	FAIL
Comment	None	
See Also	*GetSerialNumber*	

2.5.18 GetSerialNumber

The GetSerialNumber function is used to read serial number from the device.

Declare	Int32 GetSerialNumber(StringBuilder Number)	
Parameters	StringBuilder Number	The output of serial number of the device.
Return Values	0	PASS
	< 0	FAIL
Comment	None	
See Also	*SetSerialNumber*	

2.5.19 SetDataTime

The SetDataTime function is used to set current date and time on the device.

Declare	Int32 SetDateTime(DeviceDateTime DevDateTime)	
Parameters	DeviceDateTime DevDateTime	The input of current date and time information to set. If the value is blank, the default value would be 2018/01/01 7:00 A.M.
Return Values	0	PASS
	< 0	FAIL
Comment	None	
See Also	*GetDateTime, DeviceDateTime*	

2.5.20 GetDateTime

The GetDateTime function is used to get the date and time information from the device with DevDateTime variable.

Declare	**Int32 GetDateTime( )**	
Parameters	None	
Return Values	0	PASS
	< 0	FAIL
Comment	None	
See Also	***SetDataTime, DeviceDateTime***	

2.5.21 WriteLampUsage

The WriteLampUsage function is used to write the duration the lamp has been used.

Declare	**Int32 WriteLampUsage(UInt64 Usage)**	
Parameters	UInt64 Usage	The duration of lamp usage in ms unit.
Return Values	0	PASS
	< 0	FAIL
Comment	The functionality is only available once activation key has been applied.	
See Also	***ReadLampUsage***	

2.5.22 ReadLampUsage

The ReadLampUsage function is used to read the value LampUsage to get the duration of lamp usage. The unit of the value is ms.

Declare	**Int32 ReadLampUsage( )**	
Parameters	None	
Return Values	0	PASS
	< 0	FAIL
Comment	None	
See Also	***WriteLampUsage***	

2.5.23 ReadSensorsData

The ReadSensorsData function is used to read related information about the sensor through DevSensors variable.

Declare	**Int32 ReadSensorsData( )**	
Parameters	None	
Return Values	0	PASS
	< 0	FAIL
Comment	None	
See Also	***DeviceSensors***	

2.5.24 ReadLampRampUpData

The ReadLampRampUpData function is used to read related information about the lamp driver that rises after the lamp is turned on through LampRampUpADC variable.

Declare	**Int32 ReadLampRampUpData( )**	
Parameters	None	
Return Values	0	PASS
	< 0	FAIL
Comment	None	
See Also	LampRampUpADC	

2.5.25 ReadLampRepeatedScanData

The ReadLampRepeatedScanData function is used to read related information about the lamp driver between scans through LampRepeatedScanADC variable.

Declare	Int32 ReadLampRepeatedScanData( )	
Parameters	None	
Return Values	0	PASS
	< 0	FAIL
Comment	None	
See Also	LampRepeatedScanADC	

2.5.26 ReadLampAdcTimeStamp

The ReadLampAdcTimeStamp function is used to read time stamp information about the lamp driver through LampAdcTimeStamp variable.

Declare	Int32 ReadLampAdcTimeStamp( )	
Parameters	None	
Return Values	0	PASS
	< 0	FAIL
Comment	None	
See Also	LampAdcTimeStamp	

2.5.27 ReadLampParam

The ReadLampParam function is used to read lamp ADC to calculate voltage and current. Different hardware has different displays.

Declare	Int32 ReadLampParam( )	
Parameters	None	
Return Values	0	PASS
	< 0	FAIL
Comment	None	
See Also	LampADC	

2.5.28 GetCalibStruct

The GetCalibStruct function is used to get wavelength calibration coefficients through build-in variable, Calib_Coeffs.

Declare	Int32 GetCalibStruct( )	
Parameters	None	
Return Values	0	PASS
	< 0	FAIL
Comment	None	
See Also	***SendCalibStruct, CalibCoeffs***	

2.5.29 SendCalibStruct

The SendCalibStruct function is used to set wavelength calibration coefficients on the device.

Declare	Int32 SendCalibStruct(CalibCoeffs pCalibResult)	
Parameters	CalibCoeffs pCalibResult	Coefficients of wavelength calibration.
Return Values	0	PASS
	< 0	FAIL
Comment	None	
See Also	***GetCalibStruct, CalibCoeffs***	

2.5.30 SetGenericCalibStruct

The SetGenericCalibStruct function is used to set generic wavelength calibration coefficients, these coefficients are built-in data in the device.

Declare	Int32 SetGenericCalibStruct( )

Parameters	None	
Return Values	0	PASS
	< 0	FAIL
Comment	None	
See Also	None	

2.5.31 RestoreDefaultCalibStruct

The RestoreDefaultCalibStruct function is used to restore calibration coefficients provided by the factory, and the coefficients are built-in data in the device. This functionality is only available if the Tiva version is later than v2.0.22 on the device originally.

Declare	**Int32 RestoreDefaultCalibStruct( )**	
Parameters	None	
Return Values	0	PASS
	< 0	FAIL
Comment	None	
See Also	None	

2.5.32 WriteBleDispName

The WriteBleDispName function is used to set the display name of the Bluetooth broadcast.

Declare	**Int32 WriteBleDispName(String Name)**	
Parameters	String Name	Bluetooth broadcast name
Return Values	0	PASS
	< 0	FAIL
Comment	None	
See Also	***ReadBleDispName***	

2.5.33 ReadBleDispName

The ReadBleDispName function is used to read display name of the Bluetooth broadcast.

Declare	**Int32 ReadBleDispName(StringBuilder Name)**	
Parameters	StringBuilder Name	The output of Bluetooth broadcast name.
Return Values	0	PASS
	< 0	FAIL
Comment	None	
See Also	***WriteBleDispName***	

2.5.34 SetButtonLock

The SetButtonLock function is used to set the button lock or unlock on the device.

Declare	**Int32 SetButtonLock(Boolean Enable)**	
Parameters	Boolean Enable	The button is locked if Enable variable is true, or it is unlocked.
Return Values	0	PASS
	< 0	FAIL
Comment	None	
See Also	***GetButtonLockStatus***	

2.5.35 GetButtonLockStatus

The GetButtonLockStatus function is used to read button status on the device.

Declare	**Int32 GetButtonLockStatus()**	
Parameters	None	
Return Values	1	Lock
	0	Unlock
	< 0	FAIL

Comment	None
See Also	***SetButtonLock***

2.5.36 DLPC_SetImageSize

The DLPC_SetImageSize function is used to set file size for DLPC update.

Declare	**Int32 DLPC_SetImageSize(Int32 imgSize)**	
Parameters	None	
Return Values	≥ 0	PASS
	< 0	FAIL
Comment	None	
See Also	None	

2.5.37 DLPC_CheckSignature

The DLPC_CheckSignature function is used to examine if the signature file of DLPC may cause error.

Declare	**bool DLPC_CheckSignature(byte [ ] imgDataArray)**	
Parameters	byte [ ] imgDataArray	The file size.
Return Values	1	Valid DLPC signature file.
	0	Invalid DLPC signature file.
Comment	None	
See Also	None	

2.5.38 DPLC_FW_Update_WriteData

The DLPC_FW_Update_WriteData function is used to update DLPC firmware with input file.

Declare	**Int32 DLPC_FW_Update_WriteData(byte [ ] dataByteArray, Int32 dataLen)**	
Parameters	byte [ ] dataByteArray	The input file used to update DLPC firmware.
	Int32 dataLen	The file size of input file.
Return Values	≥ 0	The real file size of written file.
	< 0	FAIL
Comment	None	
See Also	None	

2.5.39 DLPC_Get_Checksum

The DLPC_Get_Checksum function is used to read validation value after the DLPC firmware updated.

Declare	**Int32 DLPC_Get_Checksum( )**	
Parameters	None	
Return Values	≥ 0	The validation value for DLPC firmware update.
	< 0	FAIL
Comment	None	
See Also	None	

2.5.40 Set_Tiva_To_Bootloader

The Set_Tiva_To_Bootloader function is used to access to Tiva bootloader, so the Tiva firmware can be updated.

Declare	**Int32 Set_Tiva_To_Bootloader( )**
Parameters	None
Return Values	None
Comment	None
See Also	***Tiva_FW_Update***

2.5.41 Tiva_FW_Update

The Tiva_FW_Update function is used to update Tiva firmware with input file.

Declare	Int32 Tiva_FW_Update(String tivaFilePath)	
Parameters	String tivaFilePath	The target file with whole path and file name used to update Tiva firmware.
Return Values	0	PASS
	< 0	FAIL
Comment	None	
See Also	***Set_Tiva_To_Bootloader***	

2.5.42 Backup_Factory_Reference

The Backup_Factory_Reference function is used to back up the factory reference calibration data stored in the Tiva to local. If the factory reference data has been covered, then the backup would be failed.

Declare	Int32 Backup_Factory_Reference(String serNum)	
Parameters	Stirng serNum	The serial number of the device.
Return Values	0	PASS
	-1	Failure caused by insufficient local memory size.
	-2	Failure caused by read/write error on the file.
	-3	Failure caused by transmits error to the device.
	-4	Failure caused by invalid reference data.
Comment	This functionality is enabling with Tiva version released later than v2.1.0.50.	
See Also	***Restore_Factory_Reference***	

2.5.43 Restore_Factory_Reference

The Restore_Factory_Reference function is used to restore the reference data with data from the factory. The action failed if backup data doesn't exist.

Declare	Int32 Restore_Factory_Reference(String serNum)	
Parameters	Stirng serNum	The serial number of the device.
Return Values	0	PASS
	-1	Failure caused by insufficient local memory size.
	-2	Failure caused by invalid path of the backup file.
	-3	Failure caused by read/write error on the file.
	-4	Failure caused by reference data destroyed.
	-5	Failure caused by transmits error to the device.
	-6	Failure caused by invalid reference data.
Comment	This functionality is enabling with Tiva version released later than v2.1.0.50.	
See Also	***Backup_Factory_Reference***	

2.5.44 SetActivationKey

The SetActivationKey function is used to write activation key to the device with key value. The key is provided from ISC.

Declare	int SetActivationKey(byte [ ] key)	
Parameters	byte [ ] key	The key value is composed of 12-byte-length code provided from ISC.
Return Values	0	PASS
	< 0	FAIL
Comment	This functionality is enabling with hardware version D and Tiva version released later than v2.1.0.50.	
See Also	*FetchActivationResult, GetActivationResult*	

2.5.45 FetchActivationResult

The FetchActivationResult is used to check activation status from device if the activation key has been applied.

Declare	void FetchActivationResult( )	
Parameters	None	
Return Values	1	The key has been activated.
	0	The key is inactivated.
Comment	This functionality is enabling with hardware version D and Tiva version released later than v2.1.0.50.	
See Also	*SetActivationKey, GetActivationResult*	

2.5.46 GetActivationResult

The GetActivationResult is used to get current activation status.

Declare	int GetActivationResult( )	
Parameters	None	
Return Values	1	The key has been activated.
	0	The key is inactivated.
Comment	This functionality is enabling with hardware version D and Tiva version released later than v2.1.0.50.	
See Also	*SetActivationKey, FetchActivationResult*	

2.5.47 UsbDevice

The UsbDevice structure is used to save information related to USB connection of the device, which including model name and serial number.

Declare	struct UsbDevice	
Parameters	String ProductString	The model name of the device.
	String SerialNumber	The serial number of the device.
Comment	None	
See Also	None	

2.5.48 DeviceInfo

The DeviceInfo structure is used to save information related to firmware/hardware in the device.

Declare	struct DeviceInfo	
Parameters	String ModelName	Model name of the device.
	String SerialNumber	Serial number of the device.
	Byte[ ] DeviceUUID	Identification code of the device.
	String HardwareRev	Hardware version of the device.
	Byte[ ] TivaRev	Tiva firmware version of the device.
	Byte[ ] DLPCRev	DLPC firmware version of the device..
	Byte[ ] SpecLibRev	The version of database for spectra computation.
	Byte CalRev	The data construction version of calibration coefficients.
	Byte CfgRev	The data construction version of configuration.
	String Manufacturing_SerialNumber	The manufacturing serial number of the device.
	UInt16 MaxWavelength	The maximum wavelength of the device.
	UInt16 MinWavelength	The minimum wavelength of the device.
	String ModelType	The model type of the device.
Comment	None	
See Also	None	

2.5.49 DeviceDateTime

The DeviceDateTime structure is used to check date and time information of the device.

Declare	struct DeviceDateTime	
Parameters	Int32 Year	Year, only save the last two numbers of year, eg. 20xx.
	Int32 Month	Month.
	Int32 Day	Day.
	Int32 DayOfWeek	Day of week.
	Int32 Hour	Hour.
	Int32 Minute	Minute.
	Int32 Second	Second.
Comment	None	
See Also	None	

2.5.50 DeviceSensors

The DeviceSensors structure is used to save information related to sensor in the device.

Declare	sturct DeviceSensors	
Parameters	String BattStatus	Status of battery.
	Double BattCapicity	Capacity of battery.
	Double Humidity	System humidity.
	Double HDCTemp	System temperature.
	Double TiveTemp	Temperature of Tiva.
	Int32 PhotoDetector	Photo detector.
Comment	None	
See Also	None	

2.5.51 CalibCoeffs

The CalibCoeffs structure is used to save wavelength calibration coefficients of the device.

Declare	**struct CalibCoeffs**	
Parameters	Double[ ] ShiftVectorCoeffs	Calibration coefficients of shift vector.
	Double[ ] PixelToWavelengthCoeffs	Calibration coefficients for pixel to wavelength translation.
Comment	None	
See Also	None	

2.6 Helper Class

2.6.1 ScanTypeIndexToMode

The ScanTypeIndexToMode function is used to translate scan type from index value to corresponding scan mode.

Declare	**IntPtr ScanTypeIndexToMode(Int32 Index)**	
Parameters	Int32 Index	The index value could be 1, 2, and 3.
Return Values	Returns Column, Hadmard, or Slew.	
Comment	None	
See Also	None	

2.6.2 CfgWidthItemsCount

The CfgWidthItemsCount function is used to set total amount of items according to configuration pattern width.

Declare	**Int32 CfgWidthItemsCount( )**
Parameters	None
Return Values	Returns the amount of items.
Comment	None
See Also	None

2.6.3 CfgWidthIndexToPixel

The CfgWidthIndexToPixel function is used to translate configuration pattern width from index to pixel.

Declare	**Int32 CfgWidthIndexToPixel(Int32 Index)**	
Parameters	Int32 Index	The index used to translate to pixel.
Return Values	Returns pixel value of configuration pattern.	
Comment	None	
See Also	None	

2.6.4 CfgWidthIndexToNM

The CfgWidthIndexToNM function is used to translate configuration pattern width from index to nm.

Declare	**Double CfgWidthIndexToNM(Int32 Index)**	
Parameters	Int32 Index	The width index used to translate to nm.
Return Values	Returns nm value of configuration pattern.	
Comment	None	
See Also	None	

2.6.5 CfgWidthPixelToIndex

The CfgWidthPixelToIndex function is used to translate configuration pattern width from pixel to index.

Declare	**Int32 CfgWidthPixelToIndex(Int32 Pixel)**	
Parameters	Int32 Pixel	The pixel value used to translate to index.
Return Values	Returns index value of configuration pattern.	
Comment	None	
See Also	None	

2.6.6 CfgWidthPixelToNM

The CfgWidthPixelToNM function is used to translate configuration pattern width from pixel to nm.

Declare	**Double CfgWidthPixelToNM(Int32 Pixel)**	
Parameters	Int32 Pixel	The pixel value used to translate to nm.
Return Values	Returns nm value of configuration pattern.	
Comment	None	
See Also	None	

2.6.7 CfgExpItemsCount

The CfgExpItemsCount function is used to set total amount of items according to exposure time in configuration.

Declare	**Int32 CfgExpItemsCount( )**
Parameters	None
Return Values	Returns total amount of items.
Comment	None
See Also	None

2.6.8 CfgExpIndexToTime

The CfgExpIndexToTime function is used to translate exposure time from index to time.

Declare	**Double CfgExpIndexToTime(Int32 Index)**	
Parameters	Int32 Index	The index value used to translate to time.
Return Values	Returns exposure time.	
Comment	None	
See Also	None	

2.6.9 CheckRegex

The CheckRegex function is used to delete character out of range in the input string. The range of acceptable characters are letters (a-z, A-Z), numbers (0-9), under line (_), space ( ), and dash (-).

Declare	**String CheckRegex(String input)**	
Parameters	String input	The input string for checking.
Return Values	Returns valid string after computation.	
Comment	None	
See Also	None	

2.6.10 CheckRegex_Chinese

The CheckRegex_Chinese function is used to delete character out of range in the input string. The range of acceptable characters are Chinese, letters (a-z, A-Z), numbers (0-9), under line (_), space ( ), and dash (-).

Declare	**String CheckRegex_Chinese(String input)**	
Parameters	String input	The input string for checking.
Return Values	Returns valid string after computation.	
Comment	None	
See Also	None	

2.7 Generic Class

2.7.1 OnBeginConnectingDevice

The OnBeginConnectingDevice event is used to connect to device, and the receiver would execute the following example code segment for required action. If the variable IsEnableNotify = false, the action would not be taken.

Event	SDK.OnBeginConnectingDevice += new Action(Connecting_Device);

C# Sample	void Connecting_Device( )
	{
	…
	}

Action	Action OnBeginConnectingDevice = null

2.7.2 OnDeviceConnected

The OnDeviceConnected event is used when device connected, and the receiver would execute the example code segment below for required action. If the variable IsEnableNotify = false, the action would not be taken. The serial number would be known after connection built.

Event	SDK.OnDeviceConnected += new Action<String>(Device_Connected_Handler);

C# Sample	void Device_Connected_Handler(String SerialNumber)
	{
	…
	}

Action	Action<String> OnDeviceConnected = null

2.7.3 OnDeviceConnectionLost

The OnDeviceConnectionLost event is used when device disconnected, and the receiver would execute the example code segment below for required action. If the variable IsEnableNotify = false, the action would not be taken. The reason of disconnection would be recorded with error variable, if error = true, the disconnection is not caused by expectable reason.

Event	SDK.OnDeviceConnectionLost += new Action<bool>(Device_Disconnected_Handler);

C# Sample	void Device_Disconnected_Handler(bool error)
	{
	…
	}

Action	Action<bool> OnDeviceConnectionLost = null

2.7.4 OnDeviceFound

The OnDeviceFound event is used when connectable device found, and the receiver would execute the example code segment below for required action. If the variable IsEnableNotify = false, the action would not be taken.

Event	SDK.OnDeviceFound += new Action(Device_Found_Handler)

C# Sample	void Device_Found_Handler( )
	{
	…
	}

Action	Action OnDeviceFound = null

2.7.5 OnDeviceError

The OnDeviceError event is uesd when error happens during data transmit with device, and the receiver would execute the example code segment below for required action. If the variable IsEnableNotify = false, the action would not be taken.

Event	SDK.OnDeviceError += new Action<String>(Device_Error_Handler);

C# Sample	void Device_Error_Handler(string error)
	{
	…
	}

Action	Action<String> OnDeviceError = null

2.7.6 OnErrorStatusFound

The OnErrorStatusFound event is used when the error status is found of the device, and the receiver would execute

the example code segment below for required action. If the variable IsEnableNotify = false, the action would not be taken.

Event	SDK.OnErrorStatusFound += new Action(RefreshErrorStatus);

C# Sample	void RefreshErrorStatus( )
	{
	…
	}

Action	Action OnErrorStatusFound = null

2.7.7 OnBeginScan

The OnBeginScan event is used when scan action starts, and the receiver would execute the example code segment below for required action. If the variable IsEnableNotify = false, the action would not be taken.

Event	SDK.OnBeginScan += new Action(BeginScan);

C# Sample	void BeginScan( )
	{
	…
	}

Action	Action OnBeginScan = null

2.7.8 OnScanCompleted

The OnScanCompleted event is used when scan completed, and the receiver would execute the example code segment below for required action. If the variable IsEnableNotify = false, the action would not be taken.

Event	SDK.OnScanCompleted += new Action(ScanCompleted);

C# Sample	void ScanCompleted( )
	{
	…
	}

Action	Action OnScanCompleted = null

2.7.9 OnUSBConnectionBusy

The OnUSBConnectionBusy event is used when the USB connection is busy, and the receiver would execute the example code segment below for required action. If the variable IsEnableNotify = false, the action would not be taken.

Event	SDK.OnUSBConnectionBusy += new Action(USBIsBusy);

C# Sample	void USBIsBusy( )
	{
	…
	}

Action	Action OnUSBConnectionBusy = null

2.7.10 OnButtonScan

The OnButtonScan event is used when scan is from the button of the device, and the receiver would execute the example code segment below for required action. If the variable IsEnableNotify = false, the action would not be taken.

Event	SDK.OnButtonScan += new Action(StartButtonScan);

C# Sample	void StartButtonScan( )
	{
	…
	}

Action	Action OnButtonScan = null

2.8 Debug Class

2.8.1 WriteLine

The WriteLine function is used to print console with input message and parameters, the printed line includes file name of the function and execution time.

Declare	**void WriteLine(String msg, params object [ ] p)**	
Parameters	String msg	String to print, including parameters.
	params [ ] p	Parameters attached in string.
Return Values	None	
Comment	None	
See Also	None	

2.8.2 Enable_CPP_Console

The Enable_CPP_Console function used to enable CPP console functionality to help debug.

Declare	**void Enable_CPP_Console( )**
Parameters	None
Return Values	None
Comment	None
See Also	None

Introduction to Device Status and Error Status

A.1 Device Status

The device status responded by TIVA software is shown in Table A-1. The statuses in blue are where the ISC adds and modifies the original statuses. The statuses in gray are not used by ISC.

Table A-1 Device Status

Device Status	Definitions
1	Tiva Active
2	Scan In Progress
4	SD Card Present
8	SD Card I/O Access In Progress
16	Bluetooth Active
32	Bluetooth Connected
64	Scan Interpretation In Progress
128	Scan Button Pressed
256	Battery In Charge

A.2 Error Status

The error status responded by TIVA software is shown in Table A-2, and Table A-3 to Table A-9 describe the detailed error codes of each error status. The statuses in blue are where the ISC adds and modifies the original statuses. The statuses in gray are not used by ISC.

Table A-2 Error Status

Error Status	Definitions
1	Scan Error
2	ADC Error
4	SD Card Error
8	EEPROM Error
16	Bluetooth Error
32	Spectrum Library Error
64	Hardware Error
128	TMP Sensor Error (EXT Version Only)
256	HDC Sensor Error
512	Battery Error
1024	Memory Error
2048	UART Error
4096	System Error

Table A-3 Scan Error Codes

Error Codes	Definitions
1	DLPC150 Boot Error
2	DLPC150 Initial Error
4	DLPC150 Lamp Driver Error
8	DLPC150 Crop Image Failed
16	ADC Data Error
32	Scan Configuration Invalid
64	Scan Pattern Streaming Error
128	DLPC150 Read Error

Table A-4 ADC Error Codes

Error Codes	Definitions
1	Timeout Error
2	Power Down Error
3	Power Up Error
4	Standby Error
5	Wake Up Error
6	Read Register Error
7	Write Register Error
8	Configure Error
9	Set Buffer Error
10	Command Error
11	Set PGA Error

Table A-5 Hardware Error Codes

Error Codes	Definitions
1	DLPC150 Error
2	Read UUID Error
3	Flash Initial Error

Table A-6 TMP Sensor Error Codes

Error Codes	Definitions
1	Invalid Manufacturing ID
2	Invalid Device ID
3	Reset Error
4	Read Register Error
5	Write Register Error
6	Timeout Error
7	I2C Error

Table A-7 HDC Sensor Error Codes

Error Codes	Definitions
1	Invalid Manufacturing ID
2	Invalid Device ID
3	Reset Error
4	Read Register Error
5	Write Register Error
6	Timeout Error
7	I2C Error

Table A-8 Battery Error Codes

Error Codes	Definitions
1	Battery Low

Table A-9 System Error Codes

Error Codes	Definitions
1	Unstable Lamp ADC
2	Unstable Peak Intensity
4	ADS1255 Error
8	Auto PGA Error
16	Unstable Scan in Repeated times

Troubleshooting

B.1 First Time to Update TIVA Firmware

If user is the first time to update TIVA firmware, checking the device status as follows:

1 Download DFU Driver from ISC.

2 Download and execute WinForms SDK GUI.

3 Connect ISC NIRScan through USB, turn it on its power switch and the ISC NIRScan and GUI are ready for operation.

4 Go to Utility page and select TIVA firmware.

5 Press “Update” to enter bootloader mode, and the PC will check the driver.

6 Open the device manager of the PC, located at \\Control Panel\System and Security\System\Device Manager.

7 If USB driver automatic installation is failed, then there will be an unknown device named “Tiva Device Firmware Update” on the PC’s device manager shown in Figure B-1.

8 Right click on the “Tiva Device Firmware Update” and select “Update Driver.”

9 Browse my computer for driver software, select the DFU driver folder and search for drivers in this location.

10 Update the driver.

11 After updating the driver, the GUI and ISC NIRScan need to be restarted to perform the update process again.

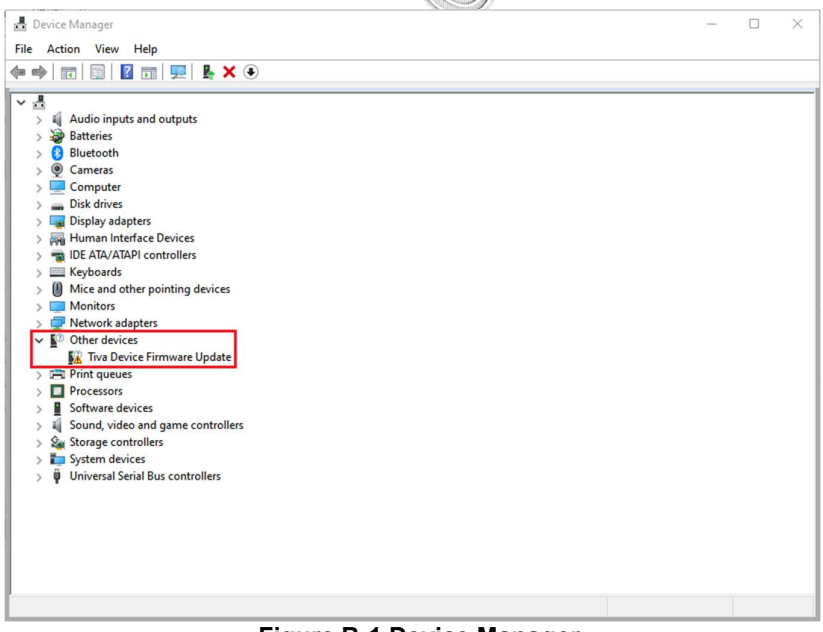


Figure B-1 Device Manager