Getting Started with the SidekickSDK

Table of Contents

SDK Overview

The Daylight Solutions SidekickSDK is the core component that facilitates high level communication with the Sidekick Controller. The SidekickSDK is extremely versatile and robust by supporting the use of C++, C#, Python, and MATLAB. You no longer have to be a software developer to integrate the Sidekick Controller into any existing experiments or software packages. With the help of the Getting Started Guide, it is easier than ever for scientists, software developers and researchers to get started with the scientific tooling that the Sidekick provides.

For additional support, please contact our dedicated team of engineers at:

scientificsupport@daylightsolutions.com.

Working with C++

In this example, we are going to be using Visual Studio 2017 to develop a test program that interacts with the Sidekick Controller via the SidekickSDK.

Requirements

- Windows OS
- USB <-> Serial Driver
- C++ Compiler

Configuring the Visual Studio Environment

1. Create new Win32 Console Application.
2. Enter `Alt` + `Enter` to bring up Properties Page.
3. Navigate to `C/C++` Menu and select `General`. Add Sidekick SDK directory to `Additional Include Directories`
4. Navigate to `Linker` tab and select `Input`. Add file path of `MIRcatSDK.lib` to `Additional Dependencies`.
5. Add `#include "SidekickSDK.h"` to classes referring to SidekickSDK.
6. Copy file `Sidekick.dll` to the `Debug` folder in the root project directory.

Basic Operations

Getting SDK API Version

```
printf("#*****************************************************************#\n");
printf("Test: Get API Version\n");
ret = SidekickSDK_GetAPIVersion(&major, &minor, &patch);
printf(" API Version: %d.%d.%d\n", major, minor, patch);
```

Initializing the Sidekick Controller

```
printf("#*****************************************************************#\n");
printf("Test: Initialize Sidekick SDK\n");
ret = SidekickSDK_Initialize();
printf(" Test Result: %d\n", ret);
```

Searching and Connecting to a Device

```
// Search for Devices
DLS_SCI_DEVICE_INFO deviceInfoList[10];
uint16_t numberOfDevices = 0;
int i = 0;
// Search over both USB and network transports
ret = SidekickSDK_SearchForDevices("239.255.101.224", 8383);
if (SIDEKICK_SDK_RET_SUCCESS == ret)
{
	ret = SidekickSDK_GetNumOfDevices(&numberOfDevices);
	if ((SIDEKICK_SDK_RET_SUCCESS == ret) && numberOfDevices > 0)
	{

	printf("#*****************************************************************#\n");
		// Enumerate device list
		for (i = 0; i < numberOfDevices; i++)
		{
			SidekickSDK_GetDeviceInfo(i, &deviceInfoList[i]);
			SidekickSDK_printDeviceInfo(&deviceInfoList[i]);
		}
	}
}

// Connect to a Device
// No connected device should have the NULL handle
DLS_SCI_DEVICE_HANDLE handle = DLS_SCI_DEVICE_NULL_HANDLE;
uint32_t StatusWord = 0;
uint16_t WarningWord = 0, ErrorWord = 0;

// Note that the address of handle is passed (pass by reference)
ret = SidekickSDK_ConnectToDeviceNumber(&handle, 0); // connect to first device found
if (SIDEKICK_SDK_RET_SUCCESS == ret)
{
	printf("#*****************************************************************#\n");
	// Connection success, read status
	// Note that the handle is now passed by value instead of reference
	ret = SidekickSDK_ReadStatusMask(handle, &StatusWord, &ErrorWord, &WarningWord);
	printf("Status Word: %d\tError Word: %d\tWarning Word: %d\n",
		StatusWord, ErrorWord, WarningWord);
}
```

Scan Initialization Structure

```
// Step 1: Check for QCL
printf("#*********************************************************************#\n");
printf("Test: Is QCL Installed & Detected?\n");
bool isQCLAvailible;
SidekickSDK_ReadAdminQclParams(handle, 0);
ret = SidekickSDK_AdminQclIsAvailable(handle, &isQCLAvailible);
printf(" Result\tret: %d\tIsQCLAvailible: %s\n",
       ret, isQCLAvailible ? "True" : "False");


// Step 2: Check for Interlock Status
printf("#*********************************************************************#\n");
printf("Test: Is Interlock Set\n");
bool isInterlockSet = false;
ret = SidekickSDK_isInterlockedStatusSet(handle, &isInterlockSet);
printf(" Result\tret: %d\tInterlock Set: %s\n",
       ret, isInterlockSet ? "True" : "False");
if (!isInterlockSet)
{
       printf("\nInterlock not set. Quitting\n");
       return 0;
}

// Step 3: Check for Keylock Status
printf("#*********************************************************************#\n");
printf("Test: Is Keylock Set\n");
bool isKeySwitchSet = false;
ret = SidekickSDK_isKeySwitchStatusSet(handle, &isKeySwitchSet);
printf(" Result\tret: %d\tInterlock Set: %s\n",
       ret, isKeySwitchSet ? "True" : "False");
if (!isKeySwitchSet)
{
       printf("\nKeySwitch not set. Quitting\n");
       return 0;
}

// Step 4: Arm the laser
printf("#*********************************************************************#\n");
printf("Test: Arm the laser\n");
ret = SidekickSDK_SetLaserArmDisarm(handle, true);
printf(" Result\tret: %d\n", ret);
printf("#*********************************************************************#\n");
printf("Test: Write Command to Controller\n");
ret = SidekickSDK_ExecLaserArmDisarm(handle);
printf(" Result\tret: %d\n", ret);

// Is the laser armed?
printf("#*********************************************************************#\n");
printf("Test: Is the laser armed?\n");
bool isLaserArmed = false;
while (!isLaserArmed)
{
```

```
	// Update SDK with latest data from Sidekick before checking laser arming status
	SidekickSDK_ReadInfoStatusMask(handle);
	ret = SidekickSDK_isLaserArmed(handle, &isLaserArmed);
	printf(" Test Results: Status:%d \tIs Armed: %s\n",
		ret, isLaserArmed ? "True" : "False");
	::Sleep(1150);
}

// Step 5: Wait for TECs to get to safe operating temperature
printf("#*****************************************************************#\n");
printf("Test: Are TECs at Temp?\n");
bool isTempStatusSet = false;
while (!isTempStatusSet)
{
	SidekickSDK_ReadInfoStatusMask(handle);
	ret = SidekickSDK_isTempStatusSet(handle, &isTempStatusSet);
	printf(" Test Results: Status:%d \tAt temp: %s\n",
		ret, isTempStatusSet ? "True" : "False");
	::Sleep(1000);
}

// Optional Step: Get Number of QCLs Installed
printf("#*****************************************************************#\n");
printf("Test: How many QCLs are Installed?\n");
SidekickSDK_ReadAdminSysParams(handle);
uint8_t numQCLs = 0;
ret = SidekickSDK_AdminGetNumQcls(handle, &numQCLs);
printf(" Result\tret: %d\tNum of QCLs: %d\n", ret, numQCLs);

// Optional Step: Get Wavelength Range
printf("#*****************************************************************#\n");
printf("Test: What is the Range of the QCL?\n");
float minWW, maxWW;
uint8_t units;
ret = SidekickSDK_AdminQclGetWavelengthLimits(handle, &(minWW), &(maxWW), &(units));
printf(" Test Result:\tret: %d \tminWW: %f \tmaxWW: %f \tunits: %d\n",
	ret, minWW, maxWW, units);
```

Single Tune Scan

```
bool isTuned = false;
bool isManualTuning = true;
uint8_t lightStatus, currentWWUnit, currentQCL;
float currentWW;

// Step 1: Set Desired Wavelength
printf("#*****************************************************************#\n");
printf("Test: Tune to Wavelength 6.55 Microns\n");
ret = SidekickSDK_SetTuneToWW(handle, SIDEKICK_SDK_UNITS_MICRONS, float(6.5500), 0);
printf(" Test Results: %d\n", ret);
// Step 2: Write Command to Sidekick
printf("Test: Send Tune Command to Sidekick\n");
ret = SidekickSDK_ExecTuneToWW(handle);
printf(" Test Results: %d\n", ret);
// Step 3: Check if the Sidekick is Tuned
printf("#*****************************************************************#\n");
printf("Test: Is Tuned?\n");
while (isManualTuning)
{
	ret = SidekickSDK_ReadStatusMask(handle, &StatusWord, &ErrorWord, &WarningWord);
	printf(" Status Mask: %d \tStatus Word: %d \tError Word: %d \tWarning Word: %d\n",
		ret, StatusWord, ErrorWord, WarningWord);
	ret = SidekickSDK_isTuned(handle, &isTuned);
	SidekickSDK_isManualTuning(handle, &isManualTuning);
	printf(" Test Results: %d \tIs Tuned: %s \tIs Manually Tuning: %s\n",
		ret, isTuned ? "True" : "False", isManualTuning ? "True" : "False");
	SidekickSDK_ReadInfoLight(handle);
	ret = SidekickSDK_GetInfoLight(handle, &lightStatus,
		&currentWW, &currentWWUnit, &currentQCL);
	printf(" Results: %d \tLight Status: %d \tCurrentWW: %f \tUnits:%d\t QCL:%d\n",
		ret, lightStatus, currentWW, currentWWUnit, currentQCL);
	::Sleep(500);

}
// Step 4: Enable Laser Emission
printf("#*****************************************************************#\n");
printf("Test: Configure Laser Emission\n");
bool isLaserFiring = false;
ret = SidekickSDK_SetLaserOnOff(handle, 0, true);
printf(" Test Result: %d\n", ret);
ret = SidekickSDK_ExecLaserOnOff(handle);
printf(" Execute Command: %d\n", ret);

// Check if laser is firing
printf("#*****************************************************************#\n");
printf("Test: Is Laser Firing?\n");
while (!isLaserFiring)
{
	ret = SidekickSDK_ReadStatusMask(handle, &(StatusWord),
		&(ErrorWord), &(WarningWord));
	printf(" Status Mask: %d \tStatus Word: %d \tError Word: %d \tWarning Word: %d\n",
		ret, StatusWord, ErrorWord, WarningWord);
	ret = SidekickSDK_isEmissionOn(handle, &(isLaserFiring));
	printf(" Result: %d \tLaser Firing: %s\n", ret, isLaserFiring ? "True" : "False");
	SidekickSDK_ReadInfoLight(handle);
	ret = SidekickSDK_GetInfoLight(handle, &lightStatus,
```

```
		&currentWW, &currentWWUnit, &currentQCL);
	printf(" Results: %d \tLight Status: %d \tCurrentWW: %f \tUnits:%d\t QCL:%d\n",
		ret, lightStatus, currentWW, currentWWUnit, currentQCL);
	::Sleep(500);
}
// Step 5: Disable Laser Emission
printf("#****************************************************************#\n");
printf("Test: Disable Laser Emission\n");
isLaserFiring = true;
ret = SidekickSDK_SetLaserOnOff(handle, 0, false);
printf(" Test Result: %d\n", ret);
ret = SidekickSDK_ExecLaserOnOff(handle);
printf(" Execute Command: %d\n", ret);

printf("#****************************************************************#\n");
printf("Test: Is Laser Firing?\n");
while (lightStatus == 1)
{
	SidekickSDK_ReadInfoLight(handle);
	ret = SidekickSDK_GetInfoLight(handle, &lightStatus,
		&currentWW, &currentWWUnit, &currentQCL);
	printf(" Results: %d \tLight Status: %d \tCurrentWW: %f \tUnits:%d\t QCL:%d\n",
		ret, lightStatus, currentWW, currentWWUnit, currentQCL);
	::Sleep(500);
}
```

Sweep Scan

Important: Sweep Scan preserves the state of laser emission. You must manually enable and disable laser emission with Sweep Scan Mode. It is only Single Tune Scan mode and Sweep Mode where you must manually enable and disable laser emission.

```
printf("#*****************************************************************#\n");
printf("Test: Sweep Scan\n");
printf("Starting Sweep mode scan from 5.7 to 6.55 um with a speed 100 microns\n");
bool scanInProgress = false;
uint8_t progressMask;
uint16_t scanNum;
uint16_t scanPercent;
ret = SidekickSDK_SetSweepParams(handle, SIDEKICK_SDK_UNITS_MICRONS,
       float(5.700), float(6.55), float(100), 15, 0, 0);
printf(" Set Sweep Params: %d\n", ret);
ret = SidekickSDK_ReadWriteSweepParams(handle, true);
printf(" Write Sweep Params: %d\n", ret);
ret = SidekickSDK_SetScanOperation(handle, SIDEKICK_SDK_SCAN_START_SWEEP);
printf(" Set Operation Mode: %d\n", ret);
ret = SidekickSDK_ExecuteScanOperation(handle);
printf(" Execute Scan Operation: %d\n", (ret));
SidekickSDK_ReadInfoStatusMask(handle);
SidekickSDK_isScanningSet(handle, &(scanInProgress));
while (scanInProgress)
{
       SidekickSDK_ReadInfoStatusMask(handle);
       SidekickSDK_isScanningSet(handle, &(scanInProgress));
       SidekickSDK_ReadScanProgress(handle);
       ret = SidekickSDK_GetScanProgress(handle, &(progressMask),
              &(scanNum), &(scanPercent));
       printf(" Result: %d \tprogressMask: %d \tscanNum: %d \tscanPercent: %d\n",
              ret, progressMask, scanNum, scanPercent);
       SidekickSDK_ReadInfoLight(handle);
       ret = SidekickSDK_GetInfoLight(handle, &(lightStatus),
              &(currentWW), &(currentWWUnit), &(currentQCL));
       printf(" Results: %d \tLight Status: %d \t CurrentWW: %f \t Units:%d \t QCL:%d\n",
              ret, lightStatus, currentWW, currentWWUnit, currentQCL);
       ::Sleep(250);
}
```

Step-Measure Scan

```
printf("#*****************************************************************#\n");
printf("Test: Step-Measure Scan\n");
printf("Starting Step-Measure scan from 8.01 to 8.30 um with a speed 5 microns\n");
SidekickSDK_SetStepMeasureParams(handle, SIDEKICK_SDK_UNITS_MICRONS,
	float(8.01), float(8.30), float(0.05), 1, 1, 0, 1000, 1000);
SidekickSDK_ReadWriteStepMeasureParams(handle, true);
ret = SidekickSDK_SetScanOperation(handle, SIDEKICK_SDK_SCAN_START_STEP_MEASURE);
printf(" Set Operation Mode: %d\n", (ret));
SidekickSDK_ExecuteScanOperation(handle);
printf(" Execute Scan Operation: %d\n", (ret));
SidekickSDK_ReadInfoStatusMask(handle);
SidekickSDK_isScanningSet(handle, &(scanInProgress));
while (scanInProgress)
{
	SidekickSDK_ReadInfoStatusMask(handle);
	SidekickSDK_isScanningSet(handle, &(scanInProgress));
	SidekickSDK_ReadScanProgress(handle);
	ret = SidekickSDK_GetScanProgress(handle, &(progressMask),
		&(scanNum), &(scanPercent));
	printf(" Result: %d \tprogressMask: %d \tscanNum: %d \tscanPercent: %d\n",
		ret, progressMask, scanNum, scanPercent);
	SidekickSDK_ReadInfoLight(handle);
	ret = SidekickSDK_GetInfoLight(handle, &(lightStatus),
		&(currentWW), &(currentWWUnit), &(currentQCL));
	printf(" Results: %d \tLight Status: %d \t CurrentWW: %f \t Units:%d \t QCL:%d\n",
		ret, lightStatus, currentWW, currentWWUnit, currentQCL);
	::Sleep(250);
}
```

Multi-Spectral Scan

```
printf("#*****************************************************************#\n");
printf("Test: Multi-Spectral Scan\n");

SidekickSDK_SetProcessTrigParams(handle, SIDEKICK_SDK_TRIG_INTERNAL);
SidekickSDK_ReadWriteProcessTrigParams(handle, true);

// Step 1: Set Multi-Spectral Params
printf("Test: Configure Multi-Spectral Scan\n");
ret = SidekickSDK_SetMultiSpectralParams(handle,
       SIDEKICK_SDK_UNITS_MICRONS, 3, 10, false);
printf(" Result: Set Params: %d\n", (ret));
ret = SidekickSDK_ReadWriteMultiSpectralParams(handle, true);
printf(" Result: Write Params: %d\n", (ret));

// Step 2: Add Multi-Spectral Elements
printf("Test: Add Multi-Spectral Elements\n");
SidekickSDK_SetMultiSpectralElement(handle, 0, float(5.0942), 100, 500, false);
SidekickSDK_SetMultiSpectralElement(handle, 1, float(6.0942), 100, 500, false);
SidekickSDK_SetMultiSpectralElement(handle, 2, float(6.54), 100, 500, false);
ret = SidekickSDK_ReadWriteMultiSpectralElementParams(handle, true);
printf(" Result: Write Elements: %d\n", (ret));

// Step 3: Start the Multi-Spectral Scan
printf("Test: Begin Scan\n");
ret = SidekickSDK_SetScanOperation(handle, SIDEKICK_SDK_SCAN_START_MULTI_SPECTRAL);
printf(" Set Operation Mode: %d\n", (ret));
ret = SidekickSDK_ExecuteScanOperation(handle);
printf(" Execute Scan Operation: %d\n", (ret));

printf("Test: Scan Status\n");
SidekickSDK_ReadInfoStatusMask(handle);
SidekickSDK_isScanningSet(handle, &(scanInProgress));
while (scanInProgress)
{
       SidekickSDK_ReadInfoStatusMask(handle);
       SidekickSDK_isScanningSet(handle, &(scanInProgress));
       SidekickSDK_ReadScanProgress(handle);
       ret = SidekickSDK_GetScanProgress(handle, &(progressMask),
              &(scanNum), &(scanPercent));
       printf(" Result: %d \tprogressMask: %d \tscanNum: %d \tscanPercent: %d\n",
              ret, progressMask, scanNum, scanPercent);
       SidekickSDK_ReadInfoLight(handle);
       ret = SidekickSDK_GetInfoLight(handle, &(lightStatus), &(currentWW),
              &(currentWWUnit), &(currentQCL));
       printf(" Result: %d \tLight Status: %d \t CurrentWW: %f \t Units:%d \t QCL:%d\n",
              ret, lightStatus, currentWW, currentWWUnit, currentQCL);
       ::Sleep(100);
}
```

Disarming & Disconnecting

```
// Disarm the laser
printf("#*****************************************************************#\n");
printf("Test: Disarming the laser\n");
ret = SidekickSDK_SetLaserArmDisarm(handle, false);
printf(" Result\tret: %d\n", ret);
printf("#*****************************************************************#\n");
printf("Test: Write Command to Controller\n");
ret = SidekickSDK_ExecLaserArmDisarm(handle);
printf(" Result\tret: %d\n", ret);

// Is the laser armed?
printf("#*****************************************************************#");
printf("Test: Is the laser armed?\n");
isLaserArmed = true;
while (isLaserArmed)
{
	// Update SDK with latest data from Sidekick before checking laser arming status
	SidekickSDK_ReadInfoStatusMask(handle);
	ret = SidekickSDK_isLaserArmed(handle, &isLaserArmed);
	printf(" Test Results: Status:%d \tIs Armed: %s\n",
		ret, isLaserArmed ? "True" : "False");
	::Sleep(1150);
}

printf("#*****************************************************************#\n");
printf("Test: Disconnect from Sidekick\n");
ret = SidekickSDK_Disconnect(handle);
printf(" Test Result: %d", (ret));
```

Working with C#

In this example, we are going to be using Visual Studio 2017 to develop a test program that interacts with the Sidekick Controller via the SidekickSDK.

Requirements

- Windows OS
- USB <-> Serial Driver
- C# Compiler

Configuring the Visual Studio Environment

1. Create new Console App (.Net Framework).
2. Copy `**SidekickSDKC#**` directory to root project directory.
3. Add Existing Item to project by pressing `**Alt**` + `**Shift**` + `**A**` and selecting `**SidekickSDK.cs**`.
4. Add `**using static Sidekick_Control.SidekickSDK**;` & `**using static Sidekick_Control.SDKConstants;**` to files working with SidekickSDK.
5. Change Solution Platform from `**Any CPU**` to `**x86**`.
6. Open the Project's Properties Page and Enable "**Allow Unsafe Code**" in the "Build" properties page.
7. Copy `**SidekickSDK.dll**` to `**{Project Root Directory}/x86/Debug/**`

Basic Operations

Getting SDK API Version

```
Console.WriteLine("#*****************************************************************#");
UInt32 ret;
UInt16 major = 0, minor = 0, patch = 0;
ret = SidekickSDK_GetAPIVersion(ref major, ref minor, ref patch);
Console.WriteLine("SDK Version: {0}.{1}.{2}", major, minor, patch);
```

Initialize Sidekick SDK

Important: You MUST initialize the Sidekick SDK before making any subsequent calls to SDK functions.

```
Console.WriteLine("#*****************************************************************#");
Console.WriteLine("Test: Initialize Sidekick SDK");
ret = SidekickSDK_Initialize();
Console.WriteLine(" Test Result: {0}", ret);
```

Searching and Connecting to a Device

```
// Search for Devices
DLS_SCI_DEVICE_INFO[] deviceInfoList = new DLS_SCI_DEVICE_INFO[10];
UInt16 numberOfDevices = 0;
```

```
int i = 0;
// Search over both USB and network transports
ret = SidekickSDK_SearchForDevices(new StringBuilder("239.255.101.224"), 8383);
if ((uint)SIDEKICK_SDK_RET_SUCCESS == ret)
{
	ret = SidekickSDK_GetNumOfDevices(ref numberOfDevices);
	if (((uint)SIDEKICK_SDK_RET_SUCCESS == ret) && numberOfDevices > 0)
	{

	Console.WriteLine("#************************************************************#");
		// Enumerate device list
		for (i = 0; i < numberOfDevices; i++)
		{
			SidekickSDK_GetDeviceInfo((ushort)i, ref deviceInfoList[i]);
			//printDeviceInfo(ref deviceInfoList[i]);
		}
	}
}
// Connect to a Device

uint handle = 0;
UInt32 StatusWord = 0;
UInt16 WarningWord = 0, ErrorWord = 0;

// Note that the address of handle is passed (pass by reference)
ret = SidekickSDK_ConnectToDeviceNumber(ref handle, 0); // connect to first device found
if ((uint)SIDEKICK_SDK_RET_SUCCESS == ret)
{
	Console.WriteLine("#************************************************************#");
	// Connection success, read status
	// Note that the handle is now passed by value instead of reference
	ret = SidekickSDK_ReadStatusMask(handle, ref StatusWord,
		ref ErrorWord, ref WarningWord);
	Console.WriteLine("Status: {0} \tError: {1} \tWarning: {2}",
		StatusWord, ErrorWord, WarningWord);
}
```

Setting up and Running Scans

Scan Initialization Structure

```
// Step 1: Check for QCL
Console.WriteLine("#*****************************************************************#");
Console.WriteLine("Test: Is QCL Installed & Detected?");
bool isQCLAvailible = false;
SidekickSDK_ReadAdminQclParams(handle, 0);
ret = SidekickSDK_AdminQclIsAvailable(handle, ref isQCLAvailible);
Console.WriteLine(" Result\tret: {0} \tIsQCLAvailible: {1}", ret, isQCLAvailible);

// Step 2: Check for Interlock Status
Console.WriteLine("#*****************************************************************#");
Console.WriteLine("Test: Is Interlock Set");
bool isInterlockSet = false;
ret = SidekickSDK_isInterlockedStatusSet(handle, ref isInterlockSet);
Console.WriteLine(" Result\tret: {0} \tInterlock Set: {1}", ret, isInterlockSet);
```

```
if (!isInterlockSet)
{
	Console.WriteLine("\nInterlock not set. Quitting\n");
	return;
}

// Step 3: Check for Keylock Status
Console.WriteLine("#*****************************************************************#");
Console.WriteLine("Test: Is Keylock Set");
bool isKeySwitchSet = false;
ret = SidekickSDK_isKeySwitchStatusSet(handle, ref isKeySwitchSet);
Console.WriteLine(" Result\tret: {0} \tInterlock Set: {1}", ret, isKeySwitchSet);
if (!isKeySwitchSet)
{
	Console.WriteLine("\nKeySwitch not set. Quitting\n");
	return;
}

// Step 4: Arm the laser
Console.WriteLine("#*****************************************************************#");
Console.WriteLine("Test: Arm the laser");
ret = SidekickSDK_SetLaserArmDisarm(handle, true);
Console.WriteLine(" Result\tret: {0}", ret);
Console.WriteLine("#*****************************************************************#");
Console.WriteLine("Test: Write Command to Controller");
ret = SidekickSDK_ExecLaserArmDisarm(handle);
Console.WriteLine(" Result\tret: {0}", ret);

// Is the laser armed?
Console.WriteLine("#*****************************************************************#");
Console.WriteLine("Test: Is the laser armed?");
bool isLaserArmed = false;
while (!isLaserArmed)
{
	// Update SDK with latest data from Sidekick before checking laser arming status
	SidekickSDK_ReadInfoStatusMask(handle);
	ret = SidekickSDK_isLaserArmed(handle, ref isLaserArmed);
	Console.WriteLine(" Test Results: Status:{0} \tIs Armed: {1}", ret, isLaserArmed);
	Thread.Sleep(1150);
}

// Step 5: Wait for TECs to get to safe operating temperature
Console.WriteLine("#*****************************************************************#");
Console.WriteLine("Test: Are TECs at Temp?");
bool isTempStatusSet = false;
while (!isTempStatusSet)
{
	SidekickSDK_ReadInfoStatusMask(handle);
	ret = SidekickSDK_isTempStatusSet(handle, ref isTempStatusSet);
	Console.WriteLine(" Test Results: Status:{0} \tAt temp: {1}",
		ret, isTempStatusSet);
	Thread.Sleep(1000);
}

// Optional Step: Get Number of QCLs Installed
Console.WriteLine("#*****************************************************************#");
Console.WriteLine("Test: How many QCLs are Installed?");
SidekickSDK_ReadAdminSysParams(handle);
```

```
byte numQCLs = 0;
ret = SidekickSDK_AdminGetNumQcls(handle, ref numQCLs);
Console.WriteLine(" Result\tret: {0} \tNum of QCLs: {1}", ret, numQCLs);

// Optional Step: Get Wavelength Range
Console.WriteLine("#*****************************************************************#");
Console.WriteLine("Test: What is the Range of the QCL?");
float minWW = 0.00F, maxWW = 0.00F;
byte units = 0;
ret = SidekickSDK_AdminQclGetWavelengthLimits(handle, ref minWW, ref maxWW, ref units);
Console.WriteLine(" Test Result:\tret: {0} \tminWW: {1} \tmaxWW: {2} \tunits: {3}",
       ret, minWW, maxWW, units);
```

Single Tune Scan

```
bool isTuned = false;
bool isManualTuning = true;
byte lightStatus = 0, currentWWUnit = 0, currentQCL = 0;
float currentWW = 0.0F;

Console.WriteLine("#*****************************************************************#");
// Step 1: Set Desired Wavelength
Console.WriteLine("Test: Tune to Wavelength 8.87 Microns");
ret = SidekickSDK_SetTuneToWW(handle, (byte)SIDEKICK_SDK_UNITS_MICRONS, (float)8.87, 0);
Console.WriteLine(" Test Results: {0}", ret);

// Step 2: Write Command to Sidekick
Console.WriteLine("Test: Send Tune Command to Sidekick");
ret = SidekickSDK_ExecTuneToWW(handle);
Console.WriteLine(" Test Results: {0}", ret);

// Step 3: Check if the Sidekick is Tuned
Console.WriteLine("#*****************************************************************#");
Console.WriteLine("Test: Is Tuned?");
while (isManualTuning)
{
	ret = SidekickSDK_ReadStatusMask(handle,
		ref StatusWord, ref ErrorWord, ref WarningWord);
	Console.WriteLine(" Status Mask: {0} \tStatus: {1} \tError: {2} \tWarning: {3}",
		ret, StatusWord, ErrorWord, WarningWord);
	ret = SidekickSDK_isTuned(handle, ref isTuned);
	SidekickSDK_isManualTuning(handle, ref isManualTuning);
	Console.WriteLine(" Test Results: {0} \tIs Tuned: {1} \tIs Manually Tuning: {2}",
		ret, isTuned, isManualTuning);
	SidekickSDK_ReadInfoLight(handle);
	ret = SidekickSDK_GetInfoLight(handle, ref lightStatus,
		ref currentWW, ref currentWWUnit, ref currentQCL);
	Console.WriteLine(" Results:{0} \tLight: {1} \tWW: {2} \tUnits:{3} \t QCL:{4}",
		ret, lightStatus, currentWW, currentWWUnit, currentQCL);
	Thread.Sleep(500);

}
// Step 4: Enable Laser Emission
Console.WriteLine("#*****************************************************************#");
Console.WriteLine("Test: Configure Laser Emission");
bool isLaserFiring = false;
ret = SidekickSDK_SetLaserOnOff(handle, 0, true);
Console.WriteLine(" Test Result: {0}", ret);
ret = SidekickSDK_ExecLaserOnOff(handle);
Console.WriteLine(" Execute Command: {0}", ret);
// Check if laser is firing
Console.WriteLine("#*****************************************************************#");
Console.WriteLine("Test: Is Laser Firing?");
while (!isLaserFiring)
{
	ret = SidekickSDK_ReadStatusMask(handle,
		ref StatusWord, ref ErrorWord, ref WarningWord);
	Console.WriteLine(" Status Mask: {0} \tStatus: {1} \tError: {2} \tWarning: {3}",
		ret, StatusWord, ErrorWord, WarningWord);
	ret = SidekickSDK_isEmissionOn(handle, ref isLaserFiring);
	Console.WriteLine(" Result: {0} \tLaser Firing: {1}", ret, isLaserFiring);
```

```
        SidekickSDK_ReadInfoLight(handle);
        ret = SidekickSDK_GetInfoLight(handle, ref lightStatus,
                ref currentWW, ref currentWWUnit, ref currentQCL);
        Console.WriteLine(" Results: {0} \tLight : {1} \tCurWW: {2} \tUnits:{3} \t
QCL:{4}",
                ret, lightStatus, currentWW, currentWWUnit, currentQCL);
        Thread.Sleep(500);
}
// Step 5: Disable Laser Emission
Console.WriteLine("#*****************************************************************#");
Console.WriteLine("Test: Disable Laser Emission");
isLaserFiring = true;
ret = SidekickSDK_SetLaserOnOff(handle, 0, false);
Console.WriteLine(" Test Result: {0}", ret);
ret = SidekickSDK_ExecLaserOnOff(handle);
Console.WriteLine(" Execute Command: {0}", ret);

Console.WriteLine("#*****************************************************************#");
Console.WriteLine("Test: Is Laser Firing?");
while (lightStatus == 1)
{
        SidekickSDK_ReadInfoLight(handle);
        ret = SidekickSDK_GetInfoLight(handle, ref lightStatus,
                ref currentWW, ref currentWWUnit, ref currentQCL);
        Console.WriteLine(" Test Results: {0} \tLight Status: {1}", ret, lightStatus);
        Thread.Sleep(500);
}
```

Sweep Scan

Important: Sweep Scan preserves the state of laser emission. You must manually enable and disable laser emission with Sweep Scan Mode. It is only Single Tune Scan mode and Sweep Mode where you must manually enable and disable laser emission.

```
Console.WriteLine("#*****************************************************************#");
Console.WriteLine("Test: Sweep Scan");
Console.WriteLine("Starting Sweep mode scan from 8.9 to 10.0 um with a speed 100
microns");
bool scanInProgress = false;
byte progressMask = 0;
UInt16 scanNum = 0, scanPercent = 0;
// Step 1: Configure Sweep Scan Parameters
ret = SidekickSDK_SetSweepParams(handle, (byte)SIDEKICK_SDK_UNITS_MICRONS,
       (float)8.900, (float)10.0, (float)110, 15, 0, 0);
Console.WriteLine(" Set Sweep Params: {0}", ret);
// Step 2: Write the Parameters to the Sidekick
ret = SidekickSDK_ReadWriteSweepParams(handle, true);
Console.WriteLine(" Write Sweep Params: {0}", ret);
// Step 3: Set the Scan Operation
ret = SidekickSDK_SetScanOperation(handle, (byte)SIDEKICK_SDK_SCAN_START_SWEEP);
Console.WriteLine(" Set Operation Mode: {0}", ret);
// Step 4: Execute the Scan Operation
ret = SidekickSDK_ExecuteScanOperation(handle);
Console.WriteLine(" Execute Scan Operation: {0}", (ret));
SidekickSDK_ReadInfoStatusMask(handle);
SidekickSDK_isScanningSet(handle, ref scanInProgress);
// Check Scan Progress
while (scanInProgress)
{
       SidekickSDK_ReadInfoStatusMask(handle);
       SidekickSDK_isScanningSet(handle, ref scanInProgress);
       SidekickSDK_ReadScanProgress(handle);
       ret = SidekickSDK_GetScanProgress(handle,
              ref progressMask, ref scanNum, ref scanPercent);
       Console.WriteLine(" Result:{0} \tprogMask: {1} \tscanNum: {2} \tscanPercent: {3}",
              ret, progressMask, scanNum, scanPercent);
       SidekickSDK_ReadInfoLight(handle);
       ret = SidekickSDK_GetInfoLight(handle, ref lightStatus,
              ref currentWW, ref currentWWUnit, ref currentQCL);
       Console.WriteLine(" Result: {0} \tLight: {1} \t WW: {2} \t Units:{3} \t QCL:{4}",
              ret, lightStatus, currentWW, currentWWUnit, currentQCL);
       Thread.Sleep(250);
}
```

Step-Measure Scan

```
Console.WriteLine("#*****************************************************************#");
Console.WriteLine("Test: Step-Measure Scan");
Console.WriteLine("Starting Step-Measure scan from 8.01 to 8.30 um with a speed 5
microns");
// Step 1: Set Step-Measure Parameters
SidekickSDK_SetStepMeasureParams(handle, (byte)SIDEKICK_SDK_UNITS_MICRONS, 8.01F, 8.30F,
0.05F, 1, 1, 0, 1000, 1000);
// Step 2: Write Step-Measure Parameters
SidekickSDK_ReadWriteStepMeasureParams(handle, true);
// Step 3: Set Scan Operation
ret = SidekickSDK_SetScanOperation(handle, (byte)SIDEKICK_SDK_SCAN_START_STEP_MEASURE);
Console.WriteLine(" Set Operation Mode: {0}", ret);
// Step 4: Execute Scan Operation
SidekickSDK_ExecuteScanOperation(handle);
Console.WriteLine(" Execute Scan Operation: {0}", ret);
SidekickSDK_ReadInfoStatusMask(handle);
SidekickSDK_isScanningSet(handle, ref scanInProgress);
// Check Scan Progress
while (scanInProgress)
{
       SidekickSDK_ReadInfoStatusMask(handle);
       SidekickSDK_isScanningSet(handle, ref scanInProgress);
       SidekickSDK_ReadScanProgress(handle);
       ret = SidekickSDK_GetScanProgress(handle,
              ref progressMask, ref scanNum, ref scanPercent);
       Console.WriteLine(" Result:{0} \tprogMask: {1} \tscanNum: {2} \tscanPercent: {3}",
              ret, progressMask, scanNum, scanPercent);
       SidekickSDK_ReadInfoLight(handle);
       ret = SidekickSDK_GetInfoLight(handle, ref lightStatus,
              ref currentWW, ref currentWWUnit, ref currentQCL);
       Console.WriteLine(" Result:{0} \tLight: {1} \t WW: {2} \t Units:{3} \t QCL:{4}",
              ret, lightStatus, currentWW, currentWWUnit, currentQCL);
       Thread.Sleep(250);
}
```

Multi-Spectral Scan

```
Console.WriteLine("#*****************************************************************#");
Console.WriteLine("Test: Multi-Spectral Scan");

SidekickSDK_SetProcessTrigParams(handle,
(byte)Sidekick_Control.TriggerModes.SIDEKICK_SDK_TRIG_INTERNAL);
SidekickSDK_ReadWriteProcessTrigParams(handle, true);

//Step 1 Set Multi-Spectral Params
Console.WriteLine("Test: Configure Multi-Spectral Scan");
ret = SidekickSDK_SetMultiSpectralParams(handle,
       (byte)SIDEKICK_SDK_UNITS_MICRONS, 3, 10, false);
Console.WriteLine(" Result: Set Params: {0}", ret);
ret = SidekickSDK_ReadWriteMultiSpectralParams(handle, true);
Console.WriteLine(" Result: Write Params: {0}", ret);

// Step 2: Add Multi-Spectral Elements
Console.WriteLine("Test: Add Multi-Spectral Elements");
SidekickSDK_SetMultiSpectralElement(handle, 0, (float)8.0942, 100, 500, false);
SidekickSDK_SetMultiSpectralElement(handle, 1, (float)9.0942, 100, 500, false);
SidekickSDK_SetMultiSpectralElement(handle, 2, (float)10.0942, 100, 500, false);
ret = SidekickSDK_ReadWriteMultiSpectralElementParams(handle, true);
Console.WriteLine(" Result: Write Elements: {0}", ret);

// Step 3: Start the Multi-Spectral Scan
Console.WriteLine("Test: Begin Scan");
ret = SidekickSDK_SetScanOperation(handle,
       (byte)SIDEKICK_SDK_SCAN_START_MULTI_SPECTRAL);
Console.WriteLine(" Set Operation Mode: {0}", ret);
ret = SidekickSDK_ExecuteScanOperation(handle);
Console.WriteLine(" Execute Scan Operation: {0}", ret);

Console.WriteLine("Test: Scan Status");
SidekickSDK_ReadInfoStatusMask(handle);
SidekickSDK_isScanningSet(handle, ref scanInProgress);
while (scanInProgress)
{
       SidekickSDK_ReadInfoStatusMask(handle);
       SidekickSDK_isScanningSet(handle, ref scanInProgress);
       SidekickSDK_ReadScanProgress(handle);
       ret = SidekickSDK_GetScanProgress(handle,
              ref progressMask, ref scanNum, ref scanPercent);
       Console.WriteLine(" Result: {0} \tprogMask: {1} \tscan#: {2} \tscanPercent: {3}",
              ret, progressMask, scanNum, scanPercent);
       SidekickSDK_ReadInfoLight(handle);
       ret = SidekickSDK_GetInfoLight(handle, ref lightStatus,
              ref currentWW, ref currentWWUnit, ref currentQCL);
       Console.WriteLine(" Result:{0} \tLight : {1} \t WW: {2} \t Units:{3} \t QCL:{4}",
              ret, lightStatus, currentWW, currentWWUnit, currentQCL);
       Thread.Sleep(100);
}
```

Disarming & Disconnecting

```
// Disarm the laser
Console.WriteLine("#*****************************************************************#");
Console.WriteLine("Test: Disarming the laser");
ret = SidekickSDK_SetLaserArmDisarm(handle, false);
Console.WriteLine(" Result\tret: {0}", ret);
Console.WriteLine("#*****************************************************************#");
Console.WriteLine("Test: Write Command to Controller");
ret = SidekickSDK_ExecLaserArmDisarm(handle);
Console.WriteLine(" Result\tret: {0}", ret);

// Is the laser armed?
Console.WriteLine("#*****************************************************************#");
Console.WriteLine("Test: Is the laser armed?");
isLaserArmed = true;
while (isLaserArmed)
{
       // Update SDK with latest data from Sidekick before checking laser arming status
       SidekickSDK_ReadInfoStatusMask(handle);
       ret = SidekickSDK_isLaserArmed(handle, ref isLaserArmed);
       Console.WriteLine(" Test Results: Status:{0} \tIs Armed: {1}", ret, isLaserArmed);
       Thread.Sleep(1150);
}

// Disconnect from Sidekick
Console.WriteLine("#*****************************************************************#");
Console.WriteLine("Test: Disconnect from Sidekick");
ret = SidekickSDK_Disconnect(handle);
Console.WriteLine(" Test Result: {0}", ret);
Console.WriteLine("\n#*****************************************************************#");
```

Working with MATLAB

In this example, we are going to be using MATLAB R2017a to develop a test program that interacts with the Sidekick Controller via the SidekickSDK.

Important Notes:

1. Familiarize yourself with the MATLAB documentation regarding converting c-types to MATLAB types.
2. MATLAB does not expose which variables are required for each function. You must view the `**SidekickSDK.h**` file or official SDK documentation to see what is required for each function call.
3. You should call `**unloadlibrary SidekickSDK;**` at the end of your script or where a possible error can occur. This is done in order to avoid MATLAB interpretation errors.

Requirements

- Windows OS
- USB <-> Serial Driver
- MATLAB License
- mingw x64 c/c++ compiler

Configuring the MATLAB Environment

1. Change the MATLAB working directory to the location of your `SidekickSDK` directory.
2. Ensure required mingw x64 c/c++ compiler from the Mathworks website is installed.
3. Before you can get started writing code you must first load the constants from the **`SidekickSDKconstants.mat`** file and load the C++ Library into the MATLAB workspace.

```
dll = 'libs/SidekickSDK';
hfile = 'SidekickSDK/SidekickSDK.h';
[notfound, warnings] = loadlibrary(dll, hfile, 'alias', 'SidekickSDK');

load('SidekickSDKconstants.mat'); % Load the constants from the SDK
```

Basic Operations

The basic call structure is as follows:

```
% ret is the return value
% Function_Name is the name of the SidekickSDK function to call.
%     For example, `SidekickSDK_isEmissionOn`
% Variables are comma separated and are listed after the function name.
ret = calllib('SidekickSDK','Function_Name', variable1, variable2);
```

You can display all the available SDK Functions with the command:

```
libfunctions('SidekickSDK')
```

Getting SDK API Version

```
fprintf('=====================================================\n');
fprintf('Quering API Version ... ');

% Create your variables and Pointers if necessary.
major = uint16(0);
majorPtr = libpointer('uint16Ptr', major);
minor = uint16(0);
minorPtr = libpointer('uint16Ptr', minor);
patch = uint16(0);
patchPtr = libpointer('uint16Ptr', patch);

% Call your function
ret = calllib('SidekickSDK','SidekickSDK_GetAPIVersion', ...
    majorPtr, minorPtr, patchPtr);

%Check to see if function call was Successful
if SIDEKICK_SDK_RET_SUCCESS == ret
    fprintf('Successful.\n' );
else
    % If the operation fails, unload the library and raise an error.
    unloadlibrary SidekickSDK;
    error('Error! Code: %d', ret);
end

% Convert the pointer values to the original variables.
major = majorPtr.value;
minor = minorPtr.value;
patch = patchPtr.value;

fprintf(' API Version: %d.%d.%d\n', major, minor, patch);
```

Initialize Sidekick SDK

```
% Initialize Sidekick SDK
fprintf('=====================================================\n');
fprintf('Initialize Sidekick SDK ... ' );
ret = calllib('SidekickSDK', 'SidekickSDK_Initialize');
if SIDEKICK_SDK_RET_SUCCESS == ret
    fprintf(' Successful.\n' );
else
    unloadlibrary SidekickSDK;
    error('Error! Unable to Initialize SDK. Code: %d', ret);
end
```

Searching and Connecting to a Device

```
% Search for devices
fprintf('=====================================================\n');
fprintf('Searching for USB devices ... ' );

% Call the function
```

```
ret = calllib('SidekickSDK','SidekickSDK_SearchForUsbDevices');

% Check to see if the function was successful
if SIDEKICK_SDK_RET_SUCCESS == ret
    fprintf(' Search successful. \n' );
elseif SIDEKICK_SDK_RET_SEARCH_ERROR == ret
    unloadlibrary SidekickSDK;
    error(' Error! Error occurred while searching for USB devices. \tCode:
%d', ret);
else
    unloadlibrary SidekickSDK;
    error(' Error! An unknown error occurred while searching for USB devices.
Code:%d', ...
        ret);
end


% Find number of USB devices
fprintf('=============================================================\n');
fprintf('Find number of USB devices ... ');

% Declare Variable and Pointer
numDevices = uint16(0);
numDevicesPtr = libpointer('uint16Ptr',numDevices);

% Call the function
ret = calllib('SidekickSDK','SidekickSDK_GetNumUsbDevices',numDevicesPtr);

% Check to see if the function was successful
if SIDEKICK_SDK_RET_SUCCESS == ret
    fprintf(' Successful.\n' );
else
    unloadlibrary SidekickSDK;
    error('Error! Unable to Query Amount of USB Devices. Code: %d', ret);
end

numDevices = numDevicesPtr.value;
fprintf(' Result: Number of devices found: %u\n',numDevices);


% Connnect to a device if there is only one installed
fprintf('=============================================================\n');

% Declare Variable and Pointer
handle = DLS_SCI_DEVICE_HANDLE;
handlePtr = libpointer('uint32Ptr', handle);

% Check to see that there is only 1 device installed
if numDevices == 1
    fprintf('Connecting to USB device ... ');
    % Call function to connect to the first device
    ret = calllib('SidekickSDK','SidekickSDK_ConnectToDeviceNumber',...
        handlePtr, numDevices - 1); % devices are indexed from 0
```

```
    % Check to see if the device connection was successful
    if SIDEKICK_SDK_RET_SUCCESS == ret
        fprintf(' Connection successful.\n' );
        % Don't forget to convert the pointer to the variable.
        % You will continue to use the handle for function calls
        handle = handlePtr.value;
    else
        unloadlibrary SidekickSDK
        error(' Error! Unable to Connect to Device. \tCode: %d', ret);
    end
elseif numDevices < 1
    unloadlibrary SidekickSDK
    error(' Error! No USB Devices are installed.');
else
    unloadlibrary SidekickSDK
    error(' Error! There is more than 1 USB Device installed.');
end
```

Setting up and Running Scans

Scan Initialization Structure

```
% Step 1: Check for installed QCL
fprintf('=============================================================\n');
fprintf('Test: Is QCL Installed & Detected?\n');
isQCLAvailable = false;
isQCLAvailablePtr = libpointer('bool',isQCLAvailable);
calllib('SidekickSDK','SidekickSDK_ReadAdminQclParams', handle, 0);
calllib('SidekickSDK','SidekickSDK_AdminQclIsAvailable',...
    handle, isQCLAvailablePtr);
isQCLAvailable = isQCLAvailablePtr.value;
fprintf(' Result: Is Installed?: ... ');
if logical(isQCLAvailable)
    fprintf(' True\n');
else
    fprintf(' False\n');
    error(' There is not QCL dected or installed. ');
end


% Step 2: Check for Interlock Status
fprintf('=============================================================\n');
fprintf('Test: Is Interlock Set?\n');
isInterlockSet = false;
isInterlockSetPtr = libpointer('bool', isInterlockSet);
calllib('SidekickSDK', 'SidekickSDK_isInterlockedStatusSet', ...
    handle, isInterlockSetPtr);
isInterlockSet = isInterlockSetPtr.value;
fprintf(' Result: \tIsInterlockSet: ');
if logical(isInterlockSet)
    fprintf(' True\n');
else
    fprintf(' False\n');
```

```
    calllib('SidekickSDK','SidekickSDK_Disconnect', handle);
    unloadlibrary SidekickSDK;
    error(' Interlock is not set. ');
end


% Step 3: Check for Key Switch Status
fprintf('=============================================================\n');
fprintf('Test: Is Keyswitch Set?\n');
isKeyswitchSet = false;
isKeyswitchSetPtr = libpointer('bool', isKeyswitchSet);
calllib('SidekickSDK', 'SidekickSDK_isKeySwitchStatusSet', ...
    handle, isKeyswitchSetPtr);
isKeyswitchSet = isKeyswitchSetPtr.value;
fprintf(' Result:\tIsKeyswitchSet: ');
if logical(isKeyswitchSet)
    fprintf(' True\n');
else
    fprintf(' False\n');
    calllib('SidekickSDK','SidekickSDK_Disconnect', handle);
    unloadlibrary SidekickSDK;
    error(' Key Switch is not set. Enable the Key Switch to continue.');
end


% Step 4: Arm the laser
fprintf('=============================================================\n');
fprintf('Test: Arm the laser ... ');
ret = calllib('SidekickSDK', 'SidekickSDK_SetLaserArmDisarm', handle, true);
if ret == SIDEKICK_SDK_RET_SUCCESS
    fprintf(' Successfully Set.\n' );
else
    calllib('SidekickSDK','SidekickSDK_Disconnect', handle);
    unloadlibrary SidekickSDK;
    error(' Error! Unable to Set Laser Arming. \tCode: %d', ret);
end

fprintf('=============================================================\n');
fprintf('Test: Send Arming Command ...');
ret = calllib('SidekickSDK', 'SidekickSDK_ExecLaserArmDisarm', handle);
if ret == SIDEKICK_SDK_RET_SUCCESS
    fprintf(' Successfully Sent.\n' );
else
    calllib('SidekickSDK','SidekickSDK_Disconnect', handle);
    unloadlibrary SidekickSDK;
    error(' Error! Unable to Send Command to Arm Laser. \tCode: %d', ret);
end

fprintf('=============================================================\n');
fprintf('Test: Is the Laser Armed? \n');
isLaserArmed = false;
isLaserArmedPtr = libpointer('bool', isLaserArmed);
while (~isLaserArmed) % Lasers take 5 seconds to Arm.
    calllib('SidekickSDK', 'SidekickSDK_ReadInfoStatusMask', handle);
    calllib('SidekickSDK', 'SidekickSDK_isLaserArmed', handle,
isLaserArmedPtr);
```

```
    isLaserArmed = isLaserArmedPtr.value;
    if logical(isLaserArmed)
        fprintf(' True\n');
    else
        fprintf(' False\n');
    end
    pause(1.0);
end


% Step 5: Wait for the TECs to get to a safe working temperature.
fprintf('=============================================================\n');
fprintf('Test: Are TECs at Temp? \n');
isTempSet = false;
isTempSetPtr = libpointer('bool', isTempSet);
% The time required to reach a working temp depends on environment's ambient
temp.
while (~isTempSet)
    calllib('SidekickSDK', 'SidekickSDK_ReadInfoStatusMask', handle);
    calllib('SidekickSDK', 'SidekickSDK_isTempStatusSet', handle,
isTempSetPtr);
    isTempSet = isTempSetPtr.value;
    if logical(isTempSet)
        fprintf(' True\n');
    else
        fprintf(' False\n');
    end
    pause(1.0);
end

% Optional Step: Check the QCL Wavelength Limits
fprintf('=============================================================\n');
fprintf('Test: Get QCL Wavelength Limits\n');

minWW = single(0);
minWWPtr = libpointer('singlePtr', minWW);
maxWW = single(0);
maxWWPtr = libpointer('singlePtr', maxWW);
units = uint8(0);
unitsPtr = libpointer('uint8Ptr', units);

calllib('SidekickSDK', 'SidekickSDK_AdminQclGetWavelengthLimits', ...
    handle, minWWPtr, maxWWPtr, unitsPtr);

minWW = minWWPtr.value;
maxWW = maxWWPtr.value;
units = unitsPtr.value;

fprintf(' Result:\tMinWW: %.3f\tMaxWW: %.3f\tUnits: %u\n',...
    minWW, maxWW, units);
```

Single Tune Scan

```
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
%                                Manual Tuning                                 %
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
fprintf('============================================================\n');
fprintf('============================================================\n');
fprintf(' *\t\t\tTune to Wavelength 8.87 Microns\t\t\t*\n');
fprintf('============================================================\n');
fprintf('============================================================\n');
% Step 1: Set the WW to Tune to.
fprintf('Test: Set Tuning Parameters ... ');
ret = calllib('SidekickSDK', 'SidekickSDK_SetTuneToWW',...
    handle, SIDEKICK_SDK_UNITS_MICRONS, 8.87, 0);
if ret == SIDEKICK_SDK_RET_SUCCESS
    fprintf(' Successfully Set.\n' );
else
    calllib('SidekickSDK','SidekickSDK_Disconnect', handle);
    error(' Error! Unable to Set Tuning Parameters to SDK. \tCode: %d', ret);
end
% Step 2: Send Tuning Command
fprintf('============================================================\n');
fprintf('Test: Send Tuning Command ... ');
ret = calllib('SidekickSDK', 'SidekickSDK_ExecTuneToWW', handle);
if ret == SIDEKICK_SDK_RET_SUCCESS
    fprintf(' Successfully Sent.\n' );
else
    calllib('SidekickSDK','SidekickSDK_Disconnect', handle);
    error(' Error! Unable to Send Tuning Command. \tCode: %d', ret);
end
fprintf('============================================================\n');
isTuned = false;
isTunedPtr = libpointer('bool', isTuned);
isManualTuning = true;
isManualTuningPtr = libpointer('bool', isManualTuning);
% Step 3: Check Tuning Status
while (isManualTuning)
    fprintf('\tTest: ReadInfoStatusMask ... ');
    ret = calllib('SidekickSDK','SidekickSDK_ReadInfoStatusMask', handle);
    if ret == SIDEKICK_SDK_RET_SUCCESS
        fprintf(' Successful.\n' );
    else
        calllib('\tSidekickSDK','SidekickSDK_Disconnect', handle);
        unloadlibrary SidekickSDK;
        error('\t Error! \tCode: %d', ret);
    end

    fprintf('\tTest: Read Info Light ... ');
    ret = calllib('SidekickSDK','SidekickSDK_ReadInfoLight', handle);
    if ret == SIDEKICK_SDK_RET_SUCCESS
        fprintf(' Successful.\n' );
    else
        calllib('SidekickSDK','SidekickSDK_Disconnect', handle);
        unloadlibrary SidekickSDK;
        error(' Error! \tCode: %d', ret);
    end
```

```
    fprintf('\tTest: GetTuning Status ... ');
    ret = calllib('SidekickSDK','SidekickSDK_isTuned', handle, isTunedPtr);
    if ret == SIDEKICK_SDK_RET_SUCCESS
        fprintf(' Successful.\n' );
    else
        calllib('SidekickSDK','SidekickSDK_Disconnect', handle);
        unloadlibrary SidekickSDK;
        error(' Error! \tCode: %d', ret);
    end

    fprintf('\tTest: Is the Laser Tuned? ... ');
    isTuned = isTunedPtr.value;
    if logical(isTuned)
        fprintf(' \tTrue\n');
    else
        fprintf(' \tFalse\n');
    end

    fprintf('\tTest: Is Manual Tune in Progress? ... ');
    calllib('SidekickSDK','SidekickSDK_isManualTuning', ...
        handle, isManualTuningPtr);
    isManualTuning = isManualTuningPtr.value;
    if logical(isManualTuning)
        fprintf(' \tTrue\n');
    else
        fprintf(' \tFalse\n');
    end

    pLightStatus = libpointer('uint8Ptr', uint8(0));
    pCurWW = libpointer('singlePtr', single(0));
    pUnits = libpointer('uint8Ptr', uint8(0));
    pCurQCL = libpointer('uint8Ptr',uint8(0));
    calllib('SidekickSDK','SidekickSDK_GetInfoLight', ...
        handle, pLightStatus, pCurWW, pUnits, pCurQCL);
    fprintf('\tCurrent Wavelength: \t%.3f\n', pCurWW.value);
    fprintf('\n');
    pause(1.0);
end


% Step 4: Enable Laser Emission
calllib('SidekickSDK','SidekickSDK_SetLaserOnOff', handle, 0, true);
calllib('SidekickSDK','SidekickSDK_ExecLaserOnOff', handle);
isLaserFiring = false;
isLaserFiringPtr = libpointer('bool', isLaserFiring);

fprintf('=========================================================\n');

while (~isLaserFiring)
    fprintf('\tTest: ReadInfoStatusMask ... ');
    ret = calllib('SidekickSDK','SidekickSDK_ReadInfoStatusMask', handle);
    if ret == SIDEKICK_SDK_RET_SUCCESS
        fprintf(' Successful.\n' );
    else
```

```
        calllib('\tSidekickSDK','SidekickSDK_Disconnect', handle);
        error('\t Error! \tCode: %d', ret);
    end

    fprintf('\tTest: Read Info Light ... ');
    ret = calllib('SidekickSDK','SidekickSDK_ReadInfoLight', handle);
    if ret == SIDEKICK_SDK_RET_SUCCESS
        fprintf(' Successful.\n' );
    else
        calllib('SidekickSDK','SidekickSDK_Disconnect', handle);
        unloadlibrary SidekickSDK;
        error(' Error! \tCode: %d', ret);
    end

    calllib('SidekickSDK','SidekickSDK_isEmissionOn', ...
        handle, isLaserFiringPtr);
    isLaserFiring = isLaserFiringPtr.value;
    fprintf('\tTest: Is Laser Firing? ... ');
    if logical(isLaserFiring)
        fprintf(' \tTrue\n');
    else
        fprintf(' \tFalse\n');
    end

    fprintf('\n');
    pause(0.5);

end

% Step 5: Disable Laser Emission
fprintf('Disabling Laser Emissison\n');
calllib('SidekickSDK','SidekickSDK_SetLaserOnOff', handle, 0, false); % Set
the command
calllib('SidekickSDK','SidekickSDK_ExecLaserOnOff', handle); % Send the
command
calllib('SidekickSDK','SidekickSDK_ReadInfoLight', handle); % Query light
status
calllib('SidekickSDK','SidekickSDK_GetInfoLight', ...
    handle, pLightStatus, pCurWW, pUnits, pCurQCL);
fprintf('\tTest: Is Laser Firing? ... ');
if logical(pLightStatus.value)
    fprintf(' \tTrue\n');
else
    fprintf(' \tFalse\n');
end
```

Sweep Scan

Important: Sweep Scan preserves the state of laser emission. You must manually enable and disable laser emission with Sweep Scan Mode. It is only Single Tune Scan mode and Sweep Mode where you must manually enable and disable laser emission.

```
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
%                                Sweep Scan                                %
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
fprintf('============================================================\n');
fprintf('============================================================\n');
fprintf('============================================================\n');
fprintf('\t\tTest: Sweep Scan \n');
fprintf('============================================================\n');
fprintf('============================================================\n');
fprintf('Starting Sweep mode scan from 8.7 to 9.2 um with a speed 100
microns\n');
scanInProgress = false;
scanInProgressPtr = libpointer('bool', scanInProgress);
progressMask = uint8(0);
progressMaskPtr = libpointer('uint8Ptr', progressMask);
scanNum = uint16(0);
scanNumPtr = libpointer('uint16Ptr', scanNum);
scanPercent = uint16(0);
scanPercentPtr = libpointer('uint16Ptr', scanPercent);
% Step 1: Set the Sweep Parameters
fprintf('============================================================\n');
fprintf('\tTest: Set Sweep Params ... ');
ret = calllib('SidekickSDK','SidekickSDK_SetSweepParams', ...
    handle, SIDEKICK_SDK_UNITS_MICRONS, single(8.7), single(9.2), ...
    single(100), 15, 0, 0);
if ret == SIDEKICK_SDK_RET_SUCCESS
    fprintf(' Successful.\n' );
else
    calllib('SidekickSDK','SidekickSDK_Disconnect', handle);
    unloadlibrary SidekickSDK;
    error(' Error! \tCode: %d', ret);
end
% Step 2: Write the parameters
fprintf('============================================================\n');
fprintf('\tTest: Write Sweep Params ... ');
ret = calllib('SidekickSDK','SidekickSDK_ReadWriteSweepParams', ...
    handle, true);
if ret == SIDEKICK_SDK_RET_SUCCESS
    fprintf(' Successful.\n' );
else
    calllib('SidekickSDK','SidekickSDK_Disconnect', handle);
    unloadlibrary SidekickSDK;
    error(' Error! \tCode: %d', ret);
end
% Step 3: Set the Operation Mode
fprintf('============================================================\n');
fprintf('\tTest: Set Operation Mode ... ');
ret = calllib('SidekickSDK','SidekickSDK_SetScanOperation', ...
    handle, SIDEKICK_SDK_SCAN_START_SWEEP);
if ret == SIDEKICK_SDK_RET_SUCCESS
    fprintf(' Successful.\n' );
```

```
else
    calllib('SidekickSDK','SidekickSDK_Disconnect', handle);
    unloadlibrary SidekickSDK;
    error(' Error! \tCode: %d', ret);
end

% Step 4: Execute the scan operation
fprintf('=====================================================\n');
fprintf('\tTest: Execute Scan Operation ... ');
ret = calllib('SidekickSDK','SidekickSDK_ExecuteScanOperation', handle);
if ret == SIDEKICK_SDK_RET_SUCCESS
    fprintf(' Successful.\n' );
else
    calllib('SidekickSDK','SidekickSDK_Disconnect', handle);
    unloadlibrary SidekickSDK;
    error(' Error! \tCode: %d', ret);
end

% Check Scan Progress
calllib('SidekickSDK','SidekickSDK_ReadInfoStatusMask', handle);
calllib('SidekickSDK','SidekickSDK_isScanningSet', handle,
scanInProgressPtr);
lightStatusPtr = libpointer('uint8Ptr', uint8(0));
curWWPtr = libpointer('singlePtr', single(0));
unitsPtr = libpointer('uint8Ptr', uint8(0));
curQCLPtr = libpointer ('uint8Ptr', uint8(0));
while logical(scanInProgressPtr.value)
    calllib('SidekickSDK','SidekickSDK_ReadInfoStatusMask', handle);
    calllib('SidekickSDK','SidekickSDK_isScanningSet', handle,
scanInProgressPtr);
    calllib('SidekickSDK','SidekickSDK_ReadScanProgress', handle);
    calllib('SidekickSDK','SidekickSDK_GetScanProgress', ...
        handle, progressMaskPtr, scanNumPtr, scanPercentPtr);
    fprintf('\tProg Mask: %d \tScanNum: %d \tScanPercent: %.3f\n', ...
        progressMaskPtr.value, scanNumPtr.value, scanPercentPtr.value);
    calllib('SidekickSDK','SidekickSDK_ReadInfoLight', handle); % Query light
status
    calllib('SidekickSDK','SidekickSDK_GetInfoLight', ...
        handle, lightStatusPtr, curWWPtr, unitsPtr, curQCLPtr);
    fprintf('\tLight Status: %d \tCurWW: %.3f \tUnits: %u\n', ...
        lightStatusPtr.value, curWWPtr.value, unitsPtr.value);
    pause(0.25);
end
```

Step-Measure Scan

```
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
%                          Step-Measure Scan                                   %
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
fprintf('============================================================\n');
fprintf('============================================================\n');
fprintf('============================================================\n');
fprintf('\tTest: Step-Measure Scan \n');
fprintf('============================================================\n');
fprintf('============================================================\n');
fprintf('Starting Step-Measure scan from 8.01 to 8.30 um with a speed 5
microns\n');
scanInProgress = false;
scanInProgressPtr = libpointer('bool', scanInProgress);
progressMask = uint8(0);
progressMaskPtr = libpointer('uint8Ptr', progressMask);
scanNum = uint16(0);
scanNumPtr = libpointer('uint16Ptr', scanNum);
scanPercent = uint16(0);
scanPercentPtr = libpointer('uint16Ptr', scanPercent);

% Step 1: Set Step-Measure Parameters
fprintf('============================================================\n');
fprintf('\tTest: Set Step-Measure Params ... ');
ret = calllib('SidekickSDK','SidekickSDK_SetStepMeasureParams', ...
    handle, SIDEKICK_SDK_UNITS_MICRONS, single(8.01), single(8.30), ...
    single(0.05), 1, 1, 0, 1000, 1000);
if ret == SIDEKICK_SDK_RET_SUCCESS
    fprintf(' Successful.\n' );
else
    calllib('SidekickSDK','SidekickSDK_Disconnect', handle);
    error(' Error! \tCode: %d', ret);
end
% Step 2: Write Step-Measure Parameters
fprintf('============================================================\n');
fprintf('\tTest: Write Step-Measure Params ... ');
ret = calllib('SidekickSDK','SidekickSDK_ReadWriteStepMeasureParams', ...
    handle, true);
if ret == SIDEKICK_SDK_RET_SUCCESS
    fprintf(' Successful.\n' );
else
    calllib('SidekickSDK','SidekickSDK_Disconnect', handle);
    error(' Error! \tCode: %d', ret);
end
% Step 3: Set Operation Mode
fprintf('============================================================\n');
fprintf('\tTest: Set Operation Mode ... ');
ret = calllib('SidekickSDK','SidekickSDK_SetScanOperation', ...
    handle, SIDEKICK_SDK_SCAN_START_STEP_MEASURE);
if ret == SIDEKICK_SDK_RET_SUCCESS
    fprintf(' Successful.\n' );
else
    calllib('SidekickSDK','SidekickSDK_Disconnect', handle);
    error(' Error! \tCode: %d', ret);
end
```

```
% Step 4: Execute Scan Operation
fprintf('=============================================================\n');
fprintf('\tTest: Execute Scan Operation ... ');
ret = calllib('SidekickSDK','SidekickSDK_ExecuteScanOperation', handle);
if ret == SIDEKICK_SDK_RET_SUCCESS
    fprintf(' Successful.\n' );
else
    calllib('SidekickSDK','SidekickSDK_Disconnect', handle);
    error(' Error! \tCode: %d', ret);
end

% Check Scan Progress
calllib('SidekickSDK','SidekickSDK_ReadInfoStatusMask', handle);
calllib('SidekickSDK','SidekickSDK_isScanningSet', handle,
scanInProgressPtr);
lightStatusPtr = libpointer('uint8Ptr', uint8(0));
curWWPtr = libpointer('singlePtr', single(0));
unitsPtr = libpointer('uint8Ptr', uint8(0));
curQCLPtr = libpointer ('uint8Ptr', uint8(0));
while logical(scanInProgressPtr.value)
    calllib('SidekickSDK','SidekickSDK_ReadInfoStatusMask', handle);
    calllib('SidekickSDK','SidekickSDK_isScanningSet', handle,
scanInProgressPtr);
    calllib('SidekickSDK','SidekickSDK_ReadScanProgress', handle);
    calllib('SidekickSDK','SidekickSDK_GetScanProgress', ...
        handle, progressMaskPtr, scanNumPtr, scanPercentPtr);
    fprintf('\tProg Mask: %d \tScanNum: %d \tScanPercent: %.3f\n', ...
        progressMaskPtr.value, scanNumPtr.value, scanPercentPtr.value);
    calllib('SidekickSDK','SidekickSDK_ReadInfoLight', handle); % Query light
status
    calllib('SidekickSDK','SidekickSDK_GetInfoLight', ...
        handle, lightStatusPtr, curWWPtr, unitsPtr, curQCLPtr);
    fprintf('\tLight Status: %d \tCurWW: %.3f \tUnits: %u\n', ...
        lightStatusPtr.value, curWWPtr.value, unitsPtr.value);
    pause(0.25);
end
```

Multi-Spectral Scan

```
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
%                          Multi-Spectral Scan                            %
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
fprintf('============================================================\n');
fprintf('============================================================\n');
fprintf('============================================================\n');
fprintf('\tTest: Multi-Spectral Scan \n');
fprintf('============================================================\n');
fprintf('============================================================\n');
scanInProgress = false;
scanInProgressPtr = libpointer('bool', scanInProgress);
progressMask = uint8(0);
progressMaskPtr = libpointer('uint8Ptr', progressMask);
scanNum = uint16(0);
scanNumPtr = libpointer('uint16Ptr', scanNum);
scanPercent = uint16(0);
scanPercentPtr = libpointer('uint16Ptr', scanPercent);

% Step 1: Set & Write Process Trigger Params
calllib('SidekickSDK','SidekickSDK_SetProcessTrigParams', ...
    handle, SIDEKICK_SDK_TRIG_INTERNAL);
calllib('SidekickSDK','SidekickSDK_ReadWriteProcessTrigParams', ...
    handle, true);

% Step 2: Set & Write Multi-Spectral Params
fprintf('============================================================\n');
fprintf('\tTest: Set Multi-Spectral Params ... ');
ret = calllib('SidekickSDK','SidekickSDK_SetMultiSpectralParams', ...
    handle, SIDEKICK SDK UNITS MICRONS, 3, 10, false);
if ret == SIDEKICK_SDK_RET_SUCCESS
    fprintf(' Successful.\n' );
else
    calllib('SidekickSDK','SidekickSDK_Disconnect', handle);
    error(' Error! \tCode: %d', ret);
end

fprintf('============================================================\n');
fprintf('\tTest: Write Multi-Spectral Params ... ');
ret = calllib('SidekickSDK','SidekickSDK_ReadWriteMultiSpectralParams', ...
    handle, true);
if ret == SIDEKICK_SDK_RET_SUCCESS
    fprintf(' Successful.\n' );
else
    calllib('SidekickSDK','SidekickSDK_Disconnect', handle);
    unloadlibrary SidekickSDK;
    error(' Error! \tCode: %d', ret);
end
% Step 3: Add & Write Multi-Spectral Elements
fprintf('============================================================\n');
fprintf('\tTest: Add Multi-Spectral Elements ... \n');
calllib('SidekickSDK','SidekickSDK_SetMultiSpectralElement', ...
    handle, 0, single(8.0942), 100, 500, false);
calllib('SidekickSDK','SidekickSDK_SetMultiSpectralElement', ...
```

```
    handle, 1, single(9.0942), 100, 500, false);
calllib('SidekickSDK','SidekickSDK_SetMultiSpectralElement', ...
    handle, 2, single(10.0942), 100, 500, false);

fprintf('=============================================================\n');
fprintf('\tTest: Write Multi-Spectral Elements ... ');
ret =
calllib('SidekickSDK','SidekickSDK_ReadWriteMultiSpectralElementParams', ...
    handle, true);
if ret == SIDEKICK_SDK_RET_SUCCESS
    fprintf(' Successful.\n' );
else
    calllib('SidekickSDK','SidekickSDK_Disconnect', handle);
    unloadlibrary SidekickSDK;
    error(' Error! \tCode: %d', ret);
end
% Step 4: Set & Write Operation Mode
fprintf('=============================================================\n');
fprintf('\tTest: Set Operation Mode ... ');
ret = calllib('SidekickSDK','SidekickSDK_SetScanOperation', ...
    handle, SIDEKICK_SDK_SCAN_START_MULTI_SPECTRAL);
if ret == SIDEKICK_SDK_RET_SUCCESS
    fprintf(' Successful.\n' );
else
    calllib('SidekickSDK','SidekickSDK_Disconnect', handle);
    unloadlibrary SidekickSDK;
    error(' Error! \tCode: %d', ret);
end

fprintf('=============================================================\n');
fprintf('\tTest: Execute Scan Operation ... ');
ret = calllib('SidekickSDK','SidekickSDK_ExecuteScanOperation', handle);
if ret == SIDEKICK_SDK_RET_SUCCESS
    fprintf(' Successful.\n' );
else
    calllib('SidekickSDK','SidekickSDK_Disconnect', handle);
    unloadlibrary SidekickSDK;
    error(' Error! \tCode: %d', ret);
end

% Check Scan Progress
calllib('SidekickSDK','SidekickSDK_ReadInfoStatusMask', handle);
calllib('SidekickSDK','SidekickSDK_isScanningSet', handle,
scanInProgressPtr);
lightStatusPtr = libpointer('uint8Ptr', uint8(0));
curWWPtr = libpointer('singlePtr', single(0));
unitsPtr = libpointer('uint8Ptr', uint8(0));
curQCLPtr = libpointer ('uint8Ptr', uint8(0));
while logical(scanInProgressPtr.value)
    calllib('SidekickSDK','SidekickSDK_ReadInfoStatusMask', handle);
    calllib('SidekickSDK','SidekickSDK_isScanningSet', handle,
scanInProgressPtr);
    calllib('SidekickSDK','SidekickSDK_ReadScanProgress', handle);
    calllib('SidekickSDK','SidekickSDK_GetScanProgress', ...
        handle, progressMaskPtr, scanNumPtr, scanPercentPtr);
    fprintf('\tProg Mask: %d \tScanNum: %d \tScanPercent: %.3f\n', ...
```

```
        progressMaskPtr.value, scanNumPtr.value, scanPercentPtr.value);
    calllib('SidekickSDK','SidekickSDK_ReadInfoLight', handle); % Query light
status
    calllib('SidekickSDK','SidekickSDK_GetInfoLight', ...
        handle, lightStatusPtr, curWWPtr, unitsPtr, curQCLPtr);
    fprintf('\tLight Status: %d \tCurWW: %.3f \tUnits: %u\n', ...
        lightStatusPtr.value, curWWPtr.value, unitsPtr.value);
    pause(0.25);
end
```

Disarming & Disconnecting

```
fprintf('=========================================================\n');
% Set Disarm Command
calllib('SidekickSDK', 'SidekickSDK_SetLaserArmDisarm', handle, false);
% Write Disarm Command
calllib('SidekickSDK', 'SidekickSDK_ExecLaserArmDisarm', handle);

% Disconnect from Sidekick
calllib('SidekickSDK','SidekickSDK_Disconnect', handle);

% Don't forget to unload the library after you are done
unloadlibrary SidekickSDK;
```

Working with Python

In this example, we are going to be building a Python application using PyCharm Community Edition to interface with the SidekickController. We are going to be using the 64bit Python interpreter version 3.6.0.

Important Notes:

1. Python interfaces nicely with the C++ language. To use the SidekickSDK.dll, you must use the built-in library `**ctypes**`. Please familiarize yourself with the ctypes documentation before getting started.
2. Python does not expose which variables are required for each function. You must view the `**SidekickSDK.h**` file or official SDK documentation to see what is required for each function call.
3. The bit type of your Python complier determines which version of the SidekickSDK.dll you should use. For example, if you are using the 32bit Python interpreter, you should use the SidekickSDK.dll found in the `**libs/x32/**` directory.
4. Be sure to import the file ` **SidekickSDKTypes** `

Requirements

- Windows OS
- USB <-> Serial Driver
- Python Interpreter

Configuring PyCharm Environment

1. Copy SidekickSDK to Project Root Directory.
2. Import necessary libraries

```
import time
from SidekickSDKTypes import *
```

3. Create reference to SidekickSDK

```
SDK = CDLL("../SidekickSDKx64")
```

Basic Operations

Getting SDK API Version

```
major = c_uint16()
minor = c_uint16()
patch = c_uint16()

print("#*****************************************************************#")
SDK.SidekickSDK_GetAPIVersion(byref(major), byref(minor), byref(patch))
print('API Version: {0}.{1}.{2}'.format(major.value, minor.value, patch.value))
```

Initialize Sidekick SDK

```
print("#****************************************************************#")
print("Test: Initialize Sidekick SDK")
ret = SDK.SidekickSDK_Initialize()
print(" Test Result: {0}".format(ret))
```

Searching and Connecting to a Device

```
# Search for a Device
num_of_devices = c_uint16(0)
deviceInfoListType = DLS_SCI_DEVICE_INFO * 10
deviceInfoList = deviceInfoListType()

print("#****************************************************************#")
ret = SDK.SidekickSDK_SearchForDevices("239.255.101.224", c_uint(8383))
print("Test: Search for Devices \tStatus Code:{0}".format(ret))
if ret == SIDEKICK_SDK_RET_SUCCESS.value:
    ret = SDK.SidekickSDK_GetNumOfDevices(byref(num_of_devices))
    print(" Result - Number of Devices: {0}".format(num_of_devices.value))
    print("Test: Getting Device Info")
    if ret == SIDEKICK_SDK_RET_SUCCESS.value and num_of_devices.value > 0:
        for i in range(num_of_devices.value):
            print(" Result: Device {0}".format(i))
            SDK.SidekickSDK_GetDeviceInfo(i, byref(deviceInfoList[i]))
            SDK.SidekickSDK_printDeviceInfo(byref(deviceInfoList[i]))

# Connect to a Device
handle = DLS_SCI_DEVICE_NULL_HANDLE
StatusWord = c_uint32()
ret = 0
WarningWord = c_uint16(0)
ErrorWord = c_uint16(0)

# Note that the address of handle is passed (pass by reference)
ret = SDK.SidekickSDK_ConnectToDeviceNumber(byref(handle), 0)  # connect to first
device found
print("Test: Connect to first device \tStatus Code:{0}".format(ret))
if ret == SIDEKICK_SDK_RET_SUCCESS.value:
    # Connection success, read status
    # Note that the handle is now passed by value instead of reference
    ret = SDK.SidekickSDK_ReadStatusMask(handle, byref(StatusWord), byref(ErrorWord),
byref(WarningWord))
    print("Test: Read Status Mask \tStatus Code:{0}".format(ret))
    print(" Test Results: Status Word: {0},\tError Word: {1},\tWarning Word:{2}"
          .format(hex(StatusWord.value), ErrorWord.value, WarningWord.value))
```

Scan Initialization Structure

```
# Step 1: Check for installed QCL
print("#****************************************************************#")
print("Test: Is QCL Installed & Detected?")
isQCLAvailable = c_bool(False)
SDK.SidekickSDK_ReadAdminQclParams(handle, 0)
ret = SDK.SidekickSDK_AdminQclIsAvailable(handle, byref(isQCLAvailable)) # This
changes based on x32 or x64
print(" Test Result: {0} \tIs Installed: {1}".format(ret, isQCLAvailable.value))
if not isQCLAvailable:
    exit(0)

# Step Optional: Get Number of QCLs Installed
print("#****************************************************************#")
print("Test: How many QCLs are Installed?")
numQCLs = c_uint8(0)
SDK.SidekickSDK_ReadAdminSysParams(handle)
ret = SDK.SidekickSDK_AdminGetNumQcls(handle, byref(numQCLs))
print(" Test Result: {0} \tNum of QCLs: {1}".format(ret, numQCLs.value))

# Step Optional: Get Wavelength Range
print("#****************************************************************#")
print("Test: What is the Range of the QCL?")
minWW = c_float()
maxWW = c_float()
units = c_uint8()
ret = SDK.SidekickSDK_AdminQclGetWavelengthLimits(handle, byref(minWW), byref(maxWW),
byref(units))
print(" Test Result:\tret: {} \tminWW: {:.3} \tmaxWW: {:.3} \tunits: {}".format(ret,
minWW.value, maxWW.value, units.value))

# Step 2: Check for Interlock Status
print("#****************************************************************#")
print("Test: Is Interlock Set")
isInterlockSet = c_bool(False)
ret = SDK.SidekickSDK_isInterlockedStatusSet(handle, byref(isInterlockSet))
print(" Test Result: {0} \tInterlock Set: {1}".format(ret, isInterlockSet.value))
if not isInterlockSet:
    exit(0)

# Step 3: Check for Key Switch Status
print("#****************************************************************#")
print("Test: Is Key Switch Set")
isKeySwitchSet = c_bool(False)
ret = SDK.SidekickSDK_isKeySwitchStatusSet(handle, byref(isKeySwitchSet))
print(" Test Result: {0} \tKey Switch Set: {1}".format(ret, isKeySwitchSet.value))
if not isKeySwitchSet:
    exit(0)

# Step 4: Arm the laser
isLaserArmed = c_bool(False)

print("#****************************************************************#")
print("Test: Arming the laser")
# Tell the SDK you would like to arm the laser
ret = SDK.SidekickSDK_SetLaserArmDisarm(handle, c_bool(True))
print(" Test Result: {0}".format(ret))
```

```
print("#*****************************************************************#")
print("Test: Send Arm laser Command")
# Send the Arming Command to the Sidekick
ret = SDK.SidekickSDK_ExecLaserArmDisarm(handle)
print(" Test Result: {0}".format(ret))

print("#*****************************************************************#")
print("Test: Is the laser armed?")
# Check to see if the laser is armed
while not isLaserArmed.value:
    # Update SDK with latest data from Sidekick before checking laser arming status
    SDK.SidekickSDK_ReadInfoStatusMask(handle)
    ret = SDK.SidekickSDK_isLaserArmed(handle, byref(isLaserArmed))
    print(" Test Results: Status:{0} \tIs Armed: {1}".format(ret, isLaserArmed.value))
    time.sleep(1.15)

# Step 5: Wait for the TECs to get to a safe working temperature.
print("#*****************************************************************#")
print("Test: Are TECs at Temp?")
isTempStatusSet = c_bool(False)
tempSpikes = 0
# After the temperature reaches its proper temp, it often spikes, waiting for 3 spikes
ensures temp is set
# The temperatures of the laser can bounce low even after the temperature has been
set.
# By waiting to get 6 positive temperature readings, we can insure that the
temperature has stopped fluctuations
while (not isTempStatusSet.value) or tempSpikes < 6:
    SDK.SidekickSDK_ReadInfoStatusMask(handle)
    ret = SDK.SidekickSDK_isTempStatusSet(handle, byref(isTempStatusSet))
    print(" Test Results: Status:{0} \tAt temp: {1}".format(ret,
isTempStatusSet.value))
    if isTempStatusSet.value:
        tempSpikes += 1
    time.sleep(1)
```

Single Tune Scan

```
isTuned = c_bool(False)
isManualTuning = c_bool(True)
lightStatus = c_uint8()
currentWW = c_float()
currentWWUnit = c_uint8()
currentQCL = c_uint8()

# Step 1: Set Desired Wavelength
print("#****************************************************************#")
print("Test: Tune to Wavelength 8.87 Microns")
ret = SDK.SidekickSDK_SetTuneToWW(handle, SIDEKICK_SDK_UNITS_MICRONS,
                                  c_float(8.8700), 0)
print(" Test Results: {0}".format(ret))

# Step 2: Write Command to Sidekick
print("Test: Send Tune Command to Sidekick")
ret = SDK.SidekickSDK_ExecTuneToWW(handle)
print(" Test Results: {0}".format(ret))

# Step 3: Check if the Sidekick is Tuned
print("#****************************************************************#")
print("Test: Is Tuned?")
while isManualTuning.value:  # This is not the right way of doing this
    ret = SDK.SidekickSDK_ReadStatusMask(handle, byref(StatusWord),
                                         byref(ErrorWord), byref(WarningWord))
    print(" Status Mask: {0} \tStatus: {1} \tError: {2} \tWarning: {3}"
          .format(ret, hex(StatusWord.value), ErrorWord.value, WarningWord.value))
    ret = SDK.SidekickSDK_isTuned(handle, byref(isTuned))
    SDK.SidekickSDK_isManualTuning(handle, byref(isManualTuning))
    print(" Test Results: {0} \tIs Tuned: {1} \tIs Manually Tuning: {2}"
          .format(ret, isTuned.value, isManualTuning.value))
    SDK.SidekickSDK_ReadInfoLight(handle)
    ret = SDK.SidekickSDK_GetInfoLight(handle, byref(lightStatus), byref(currentWW),
                                       byref(currentWWUnit), byref(currentQCL))
    print(" Results:{0} \tLight Status: {1} \tWW: {2} \tUnits:{3}\t QCL:{4}\n"
          .format(ret, lightStatus.value, currentWW.value,
                  currentWWUnit.value, currentQCL.value))
    time.sleep(0.5)

# Step 4: Enable Laser Emission
print("#****************************************************************#")
print("Test: Configure Laser Emission")
isLaserFiring = c_bool()
ret = SDK.SidekickSDK_SetLaserOnOff(handle, 0, c_bool(True))
print(" Test Result: {0}".format(ret))
ret = SDK.SidekickSDK_ExecLaserOnOff(handle)
print(" Execute Command: {0}".format(ret))

print("#****************************************************************#")
print("Test: Is Laser Firing?")
while not isLaserFiring.value:
    ret = SDK.SidekickSDK_ReadStatusMask(handle, byref(StatusWord),
                                         byref(ErrorWord), byref(WarningWord))
    print(" Status Mask: {0} \tStatus: {1} \tError: {2} \tWarning: {3}"
          .format(ret, hex(StatusWord.value), ErrorWord.value, WarningWord.value))
    ret = SDK.SidekickSDK_isEmissionOn(handle, byref(isLaserFiring))
    print(" Result: {0} \tLaser Firing: {1}".format(ret, isLaserFiring.value))
    SDK.SidekickSDK_ReadInfoLight(handle)
    ret = SDK.SidekickSDK_GetInfoLight(handle, byref(lightStatus),byref(currentWW),
                                       byref(currentWWUnit), byref(currentQCL))
    print(" Result: {0} \tLight Status: {1} \t WW: {2} \t Units:{3} \t QCL:{4}\n"
```

```
          .format(ret, lightStatus.value, currentWW.value,
                  currentWWUnit.value, currentQCL.value))
    time.sleep(0.5)

# Step 5: Disable Laser Emission
print("#****************************************************************#")
print("Test: Disable Laser Emission")
isLaserFiring = c_bool(True)
ret = SDK.SidekickSDK_SetLaserOnOff(handle, 0, c_bool(False))
print(" Test Result: {0}".format(ret))
ret = SDK.SidekickSDK_ExecLaserOnOff(handle)
print(" Execute Command: {0}".format(ret))

print("#****************************************************************#")
print("Test: Is Laser Firing?")
while lightStatus.value == 1:
    SDK.SidekickSDK_ReadInfoLight(handle)
    ret = SDK.SidekickSDK_GetInfoLight(handle, byref(lightStatus),
                                       byref(currentWW), byref(currentWWUnit),
                                       byref(currentQCL))
    print(" Results: {0} \tLight Status: {1}".format(ret, lightStatus.value))
    time.sleep(0.5)
```

Sweep Scan

Important: Sweep Scan preserves the state of laser emission. You must manually enable and disable laser emission with Sweep Scan Mode. It is only Single Tune Scan mode and Sweep Mode where you must manually enable and disable laser emission.

```
print("#****************************************************************#")
print("Test: Sweep Scan")
print("Starting Sweep mode scan from 8.7 to 9.2 um with a speed 100 microns")
scanInProgress = c_bool(False)
progressMask = c_uint8()
scanNum = c_uint16()
scanPercent = c_uint16()

# Step 1: Set the Sweep Parameters
ret = SDK.SidekickSDK_SetSweepParams(handle, SIDEKICK_SDK_UNITS_MICRONS,
                                     c_float(8.700), c_float(9.200), c_float(100),
                                     c_uint16(15), c_uint8(0), c_uint8(0))
print(" Set Sweep Params: {0}".format(ret))

# Step 2: Write the Parameters
ret = SDK.SidekickSDK_ReadWriteSweepParams(handle, c_bool(True))
print(" Write Sweep Params: {0}".format(ret))

# Step 3: Set the Operation Mode
ret = SDK.SidekickSDK_SetScanOperation(handle, SIDEKICK_SDK_SCAN_START_SWEEP)
print(" Set Operation Mode: {0}".format(ret))

# Step 4: Execute the Scan Operation
ret = SDK.SidekickSDK_ExecuteScanOperation(handle)
print(" Execute Scan Operation: {0}".format(ret))
SDK.SidekickSDK_ReadInfoStatusMask(handle)
SDK.SidekickSDK_isScanningSet(handle, byref(scanInProgress))
# Check Scan Progress
while scanInProgress.value:
    SDK.SidekickSDK_ReadInfoStatusMask(handle)
    SDK.SidekickSDK_isScanningSet(handle, byref(scanInProgress))
    SDK.SidekickSDK_ReadScanProgress(handle)
    ret = SDK.SidekickSDK_GetScanProgress(handle, byref(progressMask),
                                          byref(scanNum), byref(scanPercent))
    print(" Result: {0} \tprogMask: {1} \tscanNum: {2} \tscanPercent: {3}"
          .format(ret, progressMask.value, scanNum.value, scanPercent.value))
    SDK.SidekickSDK_ReadInfoLight(handle)
    ret = SDK.SidekickSDK_GetInfoLight(handle, byref(lightStatus),
                                       byref(currentWW), byref(currentWWUnit),
                                       byref(currentQCL))
    print(" Results: {0} \tLight: {1} \t WW: {2} \t Units:{3} \t QCL:{4}\n"
          .format(ret, lightStatus.value, currentWW.value,
                  currentWWUnit.value, currentQCL.value))
    time.sleep(0.25)
```

Step-Measure Scan

```
print("#****************************************************************#")
print("Test: Step-Measure Scan")
print("Starting Step-Measure scan from 8.01 to 8.30 um with a speed 5 microns")
# Step 1: Set Step-Measure Parameters
SDK.SidekickSDK_SetStepMeasureParams(
    handle, SIDEKICK_SDK_UNITS_MICRONS, c_float(8.01),
    c_float(8.30), c_float(0.05), c_uint16(1), c_uint8(1),
    c_uint8(0), c_uint32(1000), c_uint32(1000))
# Step 2: Write Step-Measure Parameters
SDK.SidekickSDK_ReadWriteStepMeasureParams(handle, True)
# Step 3: Set Scan Operation Mode
ret = SDK.SidekickSDK_SetScanOperation(handle, SIDEKICK_SDK_SCAN_START_STEP_MEASURE)
print(" Set Operation Mode: {0}".format(ret))
# Step 4: Execute Scan Operation Mode
ret = SDK.SidekickSDK_ExecuteScanOperation(handle)
print(" Execute Scan Operation: {0}".format(ret))
SDK.SidekickSDK_ReadInfoStatusMask(handle)
SDK.SidekickSDK_isScanningSet(handle, byref(scanInProgress))
while scanInProgress.value:
    SDK.SidekickSDK_ReadInfoStatusMask(handle)
    SDK.SidekickSDK_isScanningSet(handle, byref(scanInProgress))
    SDK.SidekickSDK_ReadScanProgress(handle)
    ret = SDK.SidekickSDK_GetScanProgress(handle, byref(progressMask),
                                          byref(scanNum), byref(scanPercent))
    print(" Result: {0} \tprogMask: {1} \tscanNum: {2} \tscanPercent: {3}"
          .format(ret, progressMask.value, scanNum.value, scanPercent.value))
    SDK.SidekickSDK_ReadInfoLight(handle)
    ret = SDK.SidekickSDK_GetInfoLight(handle, byref(lightStatus),
                                       byref(currentWW), byref(currentWWUnit),
                                       byref(currentQCL))
    print(" Results: {0} \tLight: {1} \t WW: {2} \t Units:{3} \t QCL:{4}\n"
          .format(ret, lightStatus.value, currentWW.value,
                  currentWWUnit.value, currentQCL.value))
    time.sleep(0.25)
```

Multi-Spectral Scan

```
print("#*****************************************************************#")
print("Test: Multi-Spectral Scan")

# Step 1: Set & Write Process Trigger Params
SDK.SidekickSDK_SetProcessTrigParams(handle, SIDEKICK_SDK_TRIG_INTERNAL)
SDK.SidekickSDK_ReadWriteProcessTrigParams(handle, c_bool(True))

# Step 2: Set & Write Multi-Spectral Params
print("Test: Configure Multi-Spectral Scan")
ret = SDK.SidekickSDK_SetMultiSpectralParams(handle, SIDEKICK_SDK_UNITS_MICRONS,
c_uint16(3), c_uint16(10), c_bool(False))
print(" Result: Set Params: {0}".format(ret))
ret = SDK.SidekickSDK_ReadWriteMultiSpectralParams(handle, c_bool(True))
print(" Result: Write Params: {0}".format(ret))

# Step 3: Add & Write Multi-Spectral Elements
print("Test: Add Multi-Spectral Elements")
SDK.SidekickSDK_SetMultiSpectralElement(handle, c_uint16(0), c_float(8.0942),
c_uint32(100), c_uint32(500), c_bool(False))
SDK.SidekickSDK_SetMultiSpectralElement(handle, c_uint16(1), c_float(9.0942),
c_uint32(100), c_uint32(500), c_bool(False))
SDK.SidekickSDK_SetMultiSpectralElement(handle, c_uint16(2), c_float(10.0942),
c_uint32(100), c_uint32(500), c_bool(False))
ret = SDK.SidekickSDK_ReadWriteMultiSpectralElementParams(handle, c_bool(True))
print(" Result: Write Elements: {0}".format(ret))

# Step 4: Set & Write Operation Mode
print("Test: Begin Scan")
ret = SDK.SidekickSDK_SetScanOperation(handle, SIDEKICK_SDK_SCAN_START_MULTI_SPECTRAL)
print(" Set Operation Mode: {0}".format(ret))
ret = SDK.SidekickSDK_ExecuteScanOperation(handle)
print(" Execute Scan Operation: {0}".format(ret))

print("Test: Scan Status")
SDK.SidekickSDK_ReadInfoStatusMask(handle)
SDK.SidekickSDK_isScanningSet(handle, byref(scanInProgress))
while scanInProgress.value:
    SDK.SidekickSDK_ReadInfoStatusMask(handle)
    SDK.SidekickSDK_isScanningSet(handle, byref(scanInProgress))
    SDK.SidekickSDK_ReadScanProgress(handle)
    ret = SDK.SidekickSDK_GetScanProgress(handle, byref(progressMask),
                                          byref(scanNum), byref(scanPercent))
    print(" Result: {0} \tprogMask: {1} \tscanNum: {2} \tscanPercent: {3}"
          .format(ret, progressMask.value, scanNum.value, scanPercent.value))
    SDK.SidekickSDK_ReadInfoLight(handle)
    ret = SDK.SidekickSDK_GetInfoLight(handle, byref(lightStatus),
                                       byref(currentWW), byref(currentWWUnit),
                                       byref(currentQCL))
    print(" Results: {0} \tLight: {1} \t WW: {2} \t Units:{3} \t QCL:{4}\n"
          .format(ret, lightStatus.value, currentWW.value,
                  currentWWUnit.value, currentQCL.value))
    time.sleep(0.10)
```

Disarming & Disconnecting

```
print("#****************************************************************#")
# Disarm Laser
print("Test: Disarming the laser")
# Tell the SDK you would like to disarm the laser
ret = SDK.SidekickSDK_SetLaserArmDisarm(handle, c_bool(False))
print(" Test Result: {0}".format(ret))

print("#****************************************************************#")
print("Test: Send Disarm laser Command")
# Send the Disarming Command to the Sidekick
ret = SDK.SidekickSDK_ExecLaserArmDisarm(handle)
print(" Test Result: {0}".format(ret))

print("#****************************************************************#")
print("Test: Is the laser armed?")
while isLaserArmed.value:
    SDK.SidekickSDK_ReadInfoStatusMask(handle)
    ret = SDK.SidekickSDK_isLaserArmed(handle, byref(isLaserArmed))
    print(" Test Results: Status:{0},\t Is Armed: {1}".format(ret,
isLaserArmed.value))
    time.sleep(1)

print("#****************************************************************#")
print("Test: Disconnect from Sidekick")
ret = SDK.SidekickSDK_Disconnect(handle)
print(" Test Result: {0}".format(ret))
```