Getting Started with the MIRcatSDK

Table of Contents

SDK Overview

The Daylight Solutions MIRcatSDK is the core component that facilitates high level communication with the MIRcat Laser. The MIRcatSDK 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 MIRcat Laser 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 MIRcat 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 MIRcat laser via the MIRcatSDK.

Requirements

- Windows OS
- USB <-> Serial Driver

Configuring the Visual Studio Environment

1. Create new Win32 Console Application.
2. Copy MIRcatSDK directory to root project directory.
3. Enter `Alt` + `Enter` to bring up Properties Page.
4. Navigate to `Linker` tab and select `Input`.
5. Add file path of `MIRcatSDK.lib` to `Additional Dependencies`.
6. Add `#include "MIRcatSDK\MIRcatSDK.h"` to classes referring to MIRcatSDK.
7. Copy files `MIRcatSDK.dll`, `QtCore4.dll` & `QtGui4.dll` to the `Debug` folder in the root project directory.

You can now successfully build and run.

Basic Operations

Getting SDK API Version

```
uint16_t major, minor, patch;
MIRcatSDK_GetAPIVersion(&major, &minor, &patch);
printf("API Version: %d.%d.%d\n", major, minor, patch);
```

Initialize MIRcatSDK & Connect to MIRcat laser

Important: It is required that you initialize the API before making any subsequent calls to the SDK.

```
uint32_t ret;
printf("Attempting to Initialize MIRcat API\n");
ret = MIRcatSDK_Initialize();
if (ret == MIRcatSDK_RET_SUCCESS)
{
    printf(" Successfully Connected to MIRcat\n");
}
else
{
    printf(" Failure to Initialize API. \t Error Code: %d\n", ret);
}
```

Setting Up and Running Scans

Quick Note:
The MIRcat I/O has a polling frequency of about 4hz. This means that you need to sleep the system for 250 milliseconds before querying system statuses.

Helper Function for Arming laser

This is a helper function that is not included in the SDK. It arms the laser and then waits for the QCL TECs to get to their safe operating temperatures. This function needs to be called if the system has not yet been armed or if the user has disarmed the system and wants to make a subsequent call to either tune the laser or begin a scan.

```
bool ArmAndWaitForTemp(int numQcls)
{
       bool atTemp = false, IsArmed = false;
       uint32_t ret = MIRcatSDK_IsLaserArmed(&IsArmed);
       printf(" Test Result: ret = %d \tIsArmed = %s\n",
              ret, IsArmed ? "true" : "false");
       if (!IsArmed)
       {
              ret = MIRcatSDK_ArmDisarmLaser();
              printf(" Test Result: ret = %d\n", ret);
              printf( "=========================================================\n");
              printf( "Test : Is Laser Armed\n");
       }

    while ( !IsArmed )
    {
        ret = MIRcatSDK_IsLaserArmed(&IsArmed);
        printf(" Test Result: ret = %d \tIsArmed = %s\n",
                    ret, IsArmed ? "true" : "false");
```

```
        ::Sleep(1000);
    }

       //Wait until at temperature to do any tune/scan
       printf( "=========================================================\n");
       printf( "Test : TEC Temperature Status\n");
       ret = MIRcatSDK_AreTECsAtSetTemperature(&atTemp);
       printf(" Test Result: ret = %d \tatTemp = %s\n",
              ret, atTemp ? "true" : "false");
       uint16_t tecCur = 0;
       float qclTemp = 0;
       while ( !atTemp )
       {
              for( uint8_t i = 1; i <= numQcls; i++ )
              {
                     ret = MIRcatSDK_GetQCLTemperature( i, &qclTemp );
                     printf(" Test Result: %d, QCL%u Temp: %.3f C\n", ret, i, qclTemp );
                     ret = MIRcatSDK_GetTecCurrent( i, &tecCur );
                     printf(" Test Result: %d, TEC%u Current: %u mA\n", ret, i, tecCur );
              }

              ret = MIRcatSDK_AreTECsAtSetTemperature(&atTemp);
              printf("TECs at Temperature: ret = %d \tatTemp = %s\n",
                     ret, atTemp ? "true" : "false");
              ::Sleep(1000);
       }

       return atTemp;
}
```

Scan Initialization Structure

These are the basic steps to follow before beginning tuning the laser or beginning a scan. These steps MUST be executed before tuning or scanning or else the system will propagate errors.

```
// Step 1: Get the number of installed QCLs
MIRcatSDK_GetNumInstalledQcls(&numQcls);

// Step 2: Check for Interlock Status
printf("=========================================================\n");
printf("Test : InterLocked Status\n");
ret = MIRcatSDK_IsInterlockedStatusSet(&bIlockSet);
printf(" Test Result: ret = %d \tbIslockSet = %s\n",
       ret, bIlockSet ? "true" : "false");

// Step 3: Check for Key Switch Status
printf("=========================================================\n");
printf("Test : Key Switch Status\n");
ret = MIRcatSDK_IsKeySwitchStatusSet(&bKeySwitchSet);
printf(" Test Result: ret = %d \tbKeySwitchSet = %s\n",
       ret, bKeySwitchSet ? "true" : "false");

// Step 4: Arm the Laser
printf("=========================================================\n");
printf("Test : Arm Laser\n");
// This is a helper function that is not included in the SDK. It is included in the
getting started guide.
ArmAndWaitForTemp(numQcls); // blocks until all QCLs are at temp
```

Single Tune Scan

Important: You MUST cancel Manual Tune Mode before performing another type of scan or else you will get returned an error.

```
printf("Starting Single tune test ... \n");
bool bIlockSet, bKeySwitchSet, IsArmed, atTemp, isEmitting, lightValid, isTuned = false;
float startTrig, stopTrig, trigSpacing, tunedWW, actualWW;;
uint16_t numWlTrigs, totalWlTrigs;
uint8_t units, preferredQCL, wlTrigUnits;
uint8_t numQcls = 0;

// Step 5: Tune the Laser
printf("============================================================\n");
printf("Test : Tune to WW 6.80\n");
ret = MIRcatSDK_TuneToWW(6.80, MIRcatSDK_UNITS_MICRONS, 1);
printf(" Test Result: ret = %d\n", ret);
// Check Tuned Wavelength
printf("============================================================\n");
printf("Test : Check Tuned Wavelength\n");
ret = MIRcatSDK_GetTuneWW(&tunedWW, &units, &preferredQCL);
printf(" Test Result: ret = %d \tTuned Wavelenth: %.3f \tUnits: %u \tPreferred QCL:
%u\n",
       ret, tunedWW, units, preferredQCL);
printf("============================================================\n");
printf("Test : isTuned\n");
while (!isTuned)
{
       // Check Tuning Status
       ret = MIRcatSDK_IsTuned(&isTuned);
       printf(" Test Result: ret = %d \tisTuned = %s\n",
              ret, isTuned ? "true" : "false");
       // Check Actual Wavelength
       ret = MIRcatSDK_GetActualWW(&actualWW, &units, &lightValid);
       printf(" Test Result: ret = %d \tActual Wavelenth: %.3f \tUnits: %u \tValid:
%s\n",
              ret, actualWW, units, lightValid ? "true" : "false");
       ret = MIRcatSDK_IsEmissionOn(&isEmitting);
       printf(" Test Result: ret = %d \tisEmitting: \t%s\n\n",
              ret, isEmitting ? "true" : "false");
       ::Sleep(50);
}

// Step 6: Enable Laser Emission
printf("============================================================\n");
printf("Test : Enable Laser Emission\n");
ret = MIRcatSDK_TurnEmissionOn();
printf(" Test Result: ret = %d\n", ret);
isEmitting = false;
printf("\nTest : Is laser emitting?\n");
while (!isEmitting)
{
       ret = MIRcatSDK_IsEmissionOn(&isEmitting);
       printf(" Test Result: ret = %d \tisEmitting: \t%s\n",
              ret, isEmitting ? "true" : "false");
       ::Sleep(100);
}
```

```
// Step 7: Disable Laser Emission
printf("=========================================================\n");
printf("Test : Disable Laser Emission\n");
ret = MIRcatSDK_TurnEmissionOff();
printf(" Test Result: ret = %d\n", ret);
ret = MIRcatSDK_IsEmissionOn(&isEmitting);
printf(" Test Result: ret = %d \tisEmitting: \t%s\n",
       ret, isEmitting ? "true" : "false");
MIRcatSDK_DisarmLaser();
```

After completing manually tuned scans, you must first disable manual tune mode before you can move on to another scan mode.

```
// Step 8: Disable Manual Tune Mode
// If you are going to perform a different type of scan, you must complete this step.
printf("*********************************************************\n");
printf("Test: Cancel Manual Scan\n");
ret = MIRcatSDK_CancelManualTuneMode();
printf(" Test Result: \t%d\n", ret);
```

Sweep Scan

```
printf("============================================================\n");
// Step 5: Start Sweep Scan
printf("Starting Sweep mode scan from 6.7 to 7.2 um with a speed 100 microns\n");
ret = MIRcatSDK_StartSweepScan(6.7, 7.2, 100, MIRcatSDK_UNITS_MICRONS, 1, true, 1);
printf(" Test Result: ret = %d\n", ret);
if (ret == MIRcatSDK_RET_SUCCESS)
{
	bool bIsScanInProgress = false;
	bool bIsScanActive = false;
	bool bIsScanPaused = false;
	uint16_t wCurScanNum = 0;
	uint16_t wCurrentScanPercent;
	float fCurrentWW;
	uint8_t bUnits;
	bool bIsTECInProgress, bIsMotionInProgress;
	// Step 6: Check Scan Status
	do
	{
		printf("\n============================================================\n");
		printf(" Test Get Scan Status\n");

		ret = MIRcatSDK_GetScanStatus(&bIsScanInProgress, &bIsScanActive,
			&bIsScanPaused, &wCurScanNum, &wCurrentScanPercent,
			&fCurrentWW, &bUnits, &bIsTECInProgress, &bIsMotionInProgress);

		printf(" Test Result: ret = %d, IsScanInProgress = %d, IsScanActive = %d,"
			" bIsScanPaused = %d wCurScanNum = %d\n",
			ret, bIsScanInProgress, bIsScanActive, bIsScanPaused, wCurScanNum);
		printf(" Test Result:  currentWW= %f, Units = %d, CurrentScanPercent= %d,"
			" bIsTECInProgress= %d bIsMotionInProgress= %d\n",
			fCurrentWW, bUnits, wCurrentScanPercent,
			bIsTECInProgress, bIsMotionInProgress);
		ret = MIRcatSDK_GetActualWW(&actualWW, &units, &lightValid);
		printf(" Test Result: ret = %d, WL: %.3f, U: %u, Valid: %s",
			ret, actualWW, units, lightValid ? "TRUE" : "FALSE");

		::Sleep(300);
	} while (bIsScanInProgress);
}
```

Step-Measure Scan

```
// Step 5: Start Step-Measure Scan
ret = MIRcatSDK_StartStepMeasureModeScan(float(6.7), float(7.0), float(0.25),
MIRcatSDK_UNITS_MICRONS, 1);
printf(" Test Result: ret = %d\n", ret);
::Sleep(1000);
if (ret == MIRcatSDK_RET_SUCCESS)
{
	bool bIsScanInProgress = false;
	bool bIsScanActive = false;
	bool bIsScanPaused = false;
	uint16_t wCurScanNum = 0;
	uint16_t wCurrentScanPercent;
	float fCurrentWW;
	uint8_t bUnits;
	bool bIsTECInProgress;
	bool bIsMotionInProgress;
	// Step 6: Check Scan Status
	do
	{
		printf("\n=========================================================\n");
		printf(" Test Get Scan Status\n");

		ret = MIRcatSDK_GetScanStatus(&bIsScanInProgress, &bIsScanActive,
			&bIsScanPaused, &wCurScanNum, &wCurrentScanPercent,
			&fCurrentWW, &bUnits, &bIsTECInProgress, &bIsMotionInProgress);

		printf(" Test Result: ret = %d, IsScanInProgress = %d, IsScanActive = %d,"
			" bIsScanPaused = %d wCurScanNum = %d\n",
			ret, bIsScanInProgress, bIsScanActive, bIsScanPaused, wCurScanNum);
		printf(" Test Result:  currentWW= %f, Units = %d, CurrentScanPercent= %d,"
			" bIsTECInProgress= %d bIsMotionInProgress= %d\n",
			fCurrentWW, bUnits, wCurrentScanPercent,
			bIsTECInProgress, bIsMotionInProgress);
		ret = MIRcatSDK_GetActualWW(&actualWW, &units, &lightValid);
		printf(" Test Result: ret = %d, WL: %.3f, U: %u, Valid: %s",
			ret, actualWW, units, lightValid ? "TRUE" : "FALSE");

		::Sleep(300);
	} while (bIsScanInProgress);
}
```

Multi-Spectral Scan

```
// Step 5: Set the amount of Multi-Spectral Elements
ret = MIRcatSDK_SetNumMultiSpectralElements(10);
printf(" Test Result: ret = %d\n", ret);
printf("============================================================\n");
float fScanWW = 5.7;

// Step 6: Add Multi-Spectral Elements
for (float i = 0; i < 5.0; i += 0.5)
{
	printf("Test : AddMultiSpectralElement\n");
	ret = MIRcatSDK_AddMultiSpectralElement((fScanWW + i),
		MIRcatSDK_UNITS_MICRONS, 1000, 1000);
	printf(" Test Result: ret = %d\n", ret);
}
printf("Test : StartMultiSpectralModeScan\n");

// Step 7: Start the Multi-Spectral Scan
ret = MIRcatSDK_StartMultiSpectralModeScan(1);
printf(" Test Result: ret = %d\n", ret);
if (MIRcatSDK_RET_SUCCESS == ret)
{
	bool bIsScanInProgress = false;
	bool bIsScanActive = false;
	bool bIsScanPaused = false;
	uint16_t wCurScanNum = 0;
	uint16_t wCurrentScanPercent;
	float fCurrentWW;
	uint8_t bUnits;
	bool bIsTECInProgress;
	bool bIsMotionInProgress;
	// Step 8: Check Scan Status
	do
	{
		printf("\n============================================================\n");
		printf(" Test Get Scan Status\n");

		ret = MIRcatSDK_GetScanStatus(&bIsScanInProgress, &bIsScanActive,
			&bIsScanPaused, &wCurScanNum, &wCurrentScanPercent,
			&fCurrentWW, &bUnits, &bIsTECInProgress, &bIsMotionInProgress);

		printf(" Test Result: ret = %d, IsScanInProgress = %d, IsScanActive = %d,"
			" bIsScanPaused = %d wCurScanNum = %d\n",
			ret, bIsScanInProgress, bIsScanActive, bIsScanPaused, wCurScanNum);
		printf(" Test Result:  currentWW= %f, Units = %d, CurrentScanPercent= %d,"
			" bIsTECInProgress= %d bIsMotionInProgress= %d\n",
			fCurrentWW, bUnits, wCurrentScanPercent,
			bIsTECInProgress, bIsMotionInProgress);
		ret = MIRcatSDK_GetActualWW(&actualWW, &units, &lightValid);
		printf(" Test Result: ret = %d, WL: %.3f, U: %u, Valid: %s",
			ret, actualWW, units, lightValid ? "TRUE" : "FALSE");

		::Sleep(300);
	} while (bIsScanInProgress);
}
```

Disarming & Disconnecting from MIRcat

```
printf("\n=========================================================\n");
printf("Attempting to Disarm Laser...\n");
ret = MIRcatSDK_DisarmLaser();
printf(" Test Result: ret = %d\n", ret);

printf("\nChecking Arming Status...\n");
ret = MIRcatSDK_IsLaserArmed(&IsArmed);
printf(" Test Result: ret = %d \tIs the laser armed: %s", ret, IsArmed ? "true" :
"false");
::Sleep(5000);

printf("\n=========================================================\n");
printf("Attempting to De-Initialize MIRcatSDK...\n");
ret = MIRcatSDK_DeInitialize();
switch (ret)
{
case MIRcatSDK_RET_SUCCESS:
       printf("Successfully Disconnected from MIRcat\n");
       break;
case  MIRcatSDK_RET_NOT_INITIALIZED:
       printf("Attempt to De-initialize before controller instantiation.");
       break;
default:
       printf("Fatal Error. \t Error Code: %d\n", ret);
       break;
}
printf("=========================================================\n");
```

Working with C#

In this example, we are going to be using Visual Studio 2017 to develop a test program that interacts with the MIRcat laser via the MIRcatSDK.

Requirements

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

Configuring the Visual Studio Environment

1. Create new Console App (.Net Framework).
2. Copy MIRcatSDK directory to root project directory.
3. Add Existing Item to project by pressing `Alt` + `Shift` + `A` and selecting `MIRcatSDK.cs`.
4. Add `using static MIRcat_Ctrl.MIRcatSDK;` & `using static MIRcat_Ctrl.SDKConstants;` to files working with MIRcatSDK.
5. Change Solution Platform from `Any CPU` to `x86`
6. Copy files `MIRcatSDK.dll`, `QtCore4.dll` & `QtGui4.dll` to the `{project root}/bin/x86/Debug/` folder in the root project directory.
7. You can now successfully build and run.

Basic Operations

Getting SDK API Version

```
UInt16 major = 0, minor = 0, patch = 0;
MIRcatSDK_GetAPIVersion(ref major, ref minor, ref patch);
Console.WriteLine("\nAPI Version: {0}.{1}.{2}", major, minor, patch);
```

Initialize MIRcatSDK & Connect to MIRcat laser

```
Console.WriteLine("\nAttempting to Initialize MIRcat API");
ret = MIRcatSDK_Initialize();
if ((UInt32) MIRcatSDK_RET_SUCCESS == ret)
{
	Console.WriteLine(" Successfully Connected to MIRcat");
}
else
{
	Console.WriteLine(" Failure to Initialize API. \t Error Code: {0}", ret);
}
```

Setting Up and Running Scans

Quick Note:
The MIRcat I/O has a polling frequency of about 4hz. This means that you need to sleep the system for 250 milliseconds before querying system statuses.

Helper Function for Arming laser

This is a helper function that is not included in the SDK. It arms the laser and then waits for the QCL TECs to get to their safe operating temperatures. This function needs to be called if the system has not yet been armed or if the user has disarmed the system and wants to make a subsequent call to either tune the laser or begin a scan.

```
static bool ArmAndWaitForTemp(int numQcls)
{
	bool atTemp = false, IsArmed = false;
	UInt32 ret = MIRcatSDK_IsLaserArmed(ref IsArmed);
	if (!IsArmed)
	{
		ret = MIRcatSDK_ArmDisarmLaser();
		Console.WriteLine(" Test Result: ret = {0}\n", ret);

	Console.WriteLine("============================================================\n");
		Console.WriteLine("Test : Is Laser Armed\n");
	}

	while (!IsArmed)
	{
		ret = MIRcatSDK_IsLaserArmed(ref IsArmed);
		Console.WriteLine(" Test Result: ret = {0} \tIsArmed = {1}\n",
			ret, IsArmed);
		Thread.Sleep(1000);
	}

	// Wait until TECs are at temperature before doing any tuning/scanning
	// Note: This can take a while depending on how the laser is cooled.
	Console.WriteLine("============================================================\n");
	Console.WriteLine("Test : TEC Temperature Status\n");
	ret = MIRcatSDK_AreTECsAtSetTemperature(ref atTemp);
	Console.WriteLine(" Test Result: ret = {0} \tatTemp = {1}\n", ret, atTemp);
	UInt16 tecCur = 0;
	float qclTemp = 0;
	while (!atTemp)
	{
		for (byte i = 1; i <= numQcls; i++)
		{
			ret = MIRcatSDK_GetQCLTemperature(i, ref qclTemp);
			Console.WriteLine(" Test Result: {0}, QCL{1} Temp: {2} C",
				ret, i, qclTemp);
			ret = MIRcatSDK_GetTecCurrent(i, ref tecCur);
			Console.WriteLine(" Test Result: {0}, TEC{1} Current: {2} mA",
				ret, i, tecCur);
		}

		ret = MIRcatSDK_AreTECsAtSetTemperature(ref atTemp);
		Console.WriteLine("TECs at Temperature: ret = {0} \tatTemp = {1}\n",
			ret, atTemp);
```

```
                Thread.Sleep(1000);
        }

        return atTemp;
}
```

Scan Initialization Structure

These are the basic steps to follow before beginning tuning the laser or beginning a scan. These steps MUST be executed before tuning or scanning or else the system will propagate errors.

```
bool bIlockSet = false, bKeySwitchSet = false;
byte numQcls = 0;

// Step 1: Get the number of installed QCLs
MIRcatSDK_GetNumInstalledQcls(ref numQcls);

// Step 2: Check for Interlock Status
Console.WriteLine("=========================================================");
Console.WriteLine("Test : InterLocked Status");
ret = MIRcatSDK_IsInterlockedStatusSet(ref bIlockSet);
Console.WriteLine(" Test Result: ret = {0} \tbIslockSet = {1}", ret, bIlockSet);

// Step 3: Check for Key Switch Status
Console.WriteLine("=========================================================");
Console.WriteLine("Test : Key Switch Status\n");
ret = MIRcatSDK_IsKeySwitchStatusSet(ref bKeySwitchSet);
Console.WriteLine(" Test Result: ret = {0} \tbKeySwitchSet = {1}", ret, bKeySwitchSet);

// Step 4: Arm the Laser
Console.WriteLine("=========================================================");
Console.WriteLine("Test : Arm Laser");
// This is a helper function that is not included in the SDK. It is included in the
getting started guide.
ArmAndWaitForTemp(numQcls); // blocks until all QCLs are at temp
```

Single Tune Scan

Important: You MUST cancel Manual Tune Mode before performing another type of scan or else you will get returned an error.

```
Console.WriteLine("Starting Single tune test ...");
bool IsArmed, isEmitting, lightValid, isTuned = false;
float tunedWW, actualWW;
byte units, preferredQCL;

// Step 5: Tune the Laser
Console.WriteLine("=========================================================");
Console.WriteLine("Test : Tune to WW 6.80\n");
ret = MIRcatSDK_TuneToWW((float)6.80, (byte) MIRcatSDK_UNITS_MICRONS, 1);
Console.WriteLine(" Test Result: ret = {0}\n", ret);
// Check Tuned Wavelength
```

```
Console.WriteLine("============================================================");
Console.WriteLine("Test : Check Tuned Wavelength");
tunedWW = 0;
units = 0;
preferredQCL = 0;
ret = MIRcatSDK_GetTuneWW(ref tunedWW, ref units, ref preferredQCL);
Console.WriteLine(" Test Result: ret = {0} \tTuned Wavelenth: {1} \tUnits: {2}
\tPreferred QCL: {3}",
       ret, tunedWW, units, preferredQCL);
Console.WriteLine("============================================================");
Console.WriteLine("Test : isTuned\n");
while (!isTuned)
{
       // Check Tuning Status
       ret = MIRcatSDK_IsTuned(ref isTuned);
       Console.WriteLine(" Test Result: ret = {0} \tisTuned = {1}\n", ret, isTuned);
       // Check Actual Wavelength
       actualWW = 0;
       lightValid = false;
       ret = MIRcatSDK_GetActualWW(ref actualWW, ref units, ref lightValid);
       Console.WriteLine(" Test Result: ret = {0} \tActual Wavelenth: {1} \tUnits: {2}
\tValid: {3}",
              ret, actualWW, units, lightValid);
       Thread.Sleep(50);
}

// Step 6: Enable Laser Emission
Console.WriteLine("\n============================================================\n");
Console.WriteLine("Test : Enable Laser Emission");
ret = MIRcatSDK_TurnEmissionOn();
Console.WriteLine(" Test Result: ret = {0}\n", ret);
isEmitting = false;
Console.WriteLine("\nTest : Is laser emitting?");
while (!isEmitting)
{
       ret = MIRcatSDK_IsEmissionOn(ref isEmitting);
       Console.WriteLine(" Test Result: ret = {0} \tisEmitting: \t{1}", ret, isEmitting
);
       Thread.Sleep(100);
}

// Step 7: Disable Laser Emission
Console.WriteLine("\n============================================================");
Console.WriteLine("Test : Disable Laser Emission\n");
ret = MIRcatSDK_TurnEmissionOff();
Console.WriteLine(" Test Result: ret = {0}", ret);
ret = MIRcatSDK_IsEmissionOn(ref isEmitting);
Console.WriteLine(" Test Result: ret = {0} \tisEmitting: \t{1}", ret, isEmitting);

// Step 8: Disable Manual Tune Mode
// If you are going to perform a different type of scan, you must complete this step.
Console.WriteLine("************************************************************");
Console.WriteLine("Test: Cancel Manual Scan");
ret = MIRcatSDK_CancelManualTuneMode();
Console.WriteLine(" Test Result: \t{0}", ret);
```

Sweep Scan

```
bool bIsScanInProgress = false, bIsScanActive = false, bIsScanPaused = false;
bool bIsTECInProgress = false, bIsMotionInProgress = false;
UInt16 wCurScanNum = 0;
UInt16 wCurrentScanPercent = 0;
float fCurrentWW = 0;
byte bUnits = 0;

Console.WriteLine("\n\n\n**********************************************************");
Console.WriteLine("Starting Sweep Test ...");
Console.WriteLine("**********************************************************");

Console.WriteLine("==========================================================");
Console.WriteLine("Starting Sweep mode scan from 6.7 to 7.2 um with a speed 100
microns");
// Step 5: Start Sweep Scan
ret = MIRcatSDK_StartSweepScan((float)6.7, (float)7.2, 100,
       (byte)MIRcatSDK_UNITS_MICRONS, 1, true, 1);
Console.WriteLine(" Test Result: ret = {0}", ret);
if (ret == (UInt32)MIRcatSDK_RET_SUCCESS)
{

       actualWW = 0;
       lightValid = false;
       do
       {

              Console.WriteLine("\n================================================");
              Console.WriteLine(" Test Get Scan Status");

              ret = MIRcatSDK_GetScanStatus(ref bIsScanInProgress, ref bIsScanActive,
                     ref bIsScanPaused, ref wCurScanNum, ref wCurrentScanPercent,
                     ref fCurrentWW, ref bUnits, ref bIsTECInProgress,
                     ref bIsMotionInProgress);

              Console.WriteLine(" Test Result: ret = {0}, bIsScanInProgress = {1},
bIsScanActive = {2}, bIsScanPaused = {3} wCurScanNum = {4}",
                     ret, bIsScanInProgress, bIsScanActive, bIsScanPaused, wCurScanNum);
              Console.WriteLine(" Test Result:  fCurrentWW= {0}, bUnits = {1},
wCurrentScanPercent= {2}, bIsTECInProgress= {3}, bIsMotionInProgress= {4}",
                     fCurrentWW, bUnits, wCurrentScanPercent, bIsTECInProgress,
bIsMotionInProgress);
              Thread.Sleep(250);
       } while (bIsScanInProgress);}
```

Step-Measure Scan

```
Console.WriteLine("\n\n\n*********************************************************");
Console.WriteLine(" Starting Step Measure Test ...");
Console.WriteLine("*********************************************************");
bIsScanInProgress = false;
bIsScanActive = false;
bIsScanPaused = false;
wCurScanNum = 0;
wCurrentScanPercent = 0;
fCurrentWW = 0;
bUnits = 0;
bIsTECInProgress = false;
bIsMotionInProgress = false;


ret = MIRcatSDK_StartStepMeasureModeScan((float)6.7, (float)7.0, (float)0.25,
       (byte)MIRcatSDK_UNITS_MICRONS, 1);
Console.WriteLine(" Test Result: ret = {0}", ret);
Thread.Sleep(1000);
if (ret == (UInt32)MIRcatSDK_RET_SUCCESS)
{

       do
       {
              Console.WriteLine("\n================================================");
              Console.WriteLine(" Test Get Scan Status");

              ret = MIRcatSDK_GetScanStatus(ref bIsScanInProgress, ref bIsScanActive,
                     ref bIsScanPaused, ref wCurScanNum, ref wCurrentScanPercent,
                     ref fCurrentWW, ref bUnits, ref bIsTECInProgress,
                     ref bIsMotionInProgress);

              Console.WriteLine(" Test Result: ret = {0}, bIsScanInProgress = {1},
bIsScanActive = {2}, bIsScanPaused = {3} wCurScanNum = {4}",
                     ret, bIsScanInProgress, bIsScanActive, bIsScanPaused, wCurScanNum);
              Console.WriteLine(" Test Result:  fCurrentWW= {0}, bUnits = {1},
wCurrentScanPercent= {2}, bIsTECInProgress= {3}, bIsMotionInProgress= {4}",
                     fCurrentWW, bUnits, wCurrentScanPercent, bIsTECInProgress,
bIsMotionInProgress);
              Thread.Sleep(250);
       } while (bIsScanInProgress);
}
```

Multi-Spectral Scan

```
bIsScanInProgress = false;
bIsScanActive = false;
bIsScanPaused = false;
wCurScanNum = 0;
wCurrentScanPercent = 0;
fCurrentWW = 0;
bUnits = 0;
bIsTECInProgress = false;
bIsMotionInProgress = false;

Console.WriteLine("\n\n\n************************************************************");
Console.WriteLine(" Starting Multi Spectral Test ...");
Console.WriteLine("************************************************************");
Console.WriteLine("Test : SetNumMultiSpectralElements");
ret = MIRcatSDK_SetNumMultiSpectralElements(10);
Console.WriteLine(" Test Result: ret = {0}", ret);
Console.WriteLine("============================================================");
float fScanWW = 5.7F;
for (double i = 0; i < 5.0; i += 0.5)
{
       Console.WriteLine("Test : AddMultiSpectralElement\n");
       ret = MIRcatSDK_AddMultiSpectralElement((fScanWW + (float)i),
(byte)MIRcatSDK_UNITS_MICRONS, 1000, 1000);
       Console.WriteLine(" Test Result: ret = {0}", ret);
}
Console.WriteLine("Test : StartMultiSpectralModeScan\n");

ret = MIRcatSDK_StartMultiSpectralModeScan(1);
Console.WriteLine(" Test Result: ret = {0}", ret);
if ((byte)MIRcatSDK_RET_SUCCESS == ret)
{
       do
       {
              Console.WriteLine("================================================");
              Console.WriteLine(" Test Get Scan Status\n");

              ret = MIRcatSDK_GetScanStatus(ref bIsScanInProgress, ref bIsScanActive, ref
bIsScanPaused,
                     ref wCurScanNum, ref wCurrentScanPercent,
                     ref fCurrentWW, ref bUnits,
                     ref bIsTECInProgress, ref bIsMotionInProgress);

              Console.WriteLine(" Test Result: ret = {0}, bIsScanInProgress = {1},
bIsScanActive = {2}, bIsScanPaused = {3}, wCurScanNum = {4}",
                     ret, bIsScanInProgress, bIsScanActive, bIsScanPaused, wCurScanNum);
              Console.WriteLine(" Test Result:  fCurrentWW= {0}, bUnits = {1},
wCurrentScanPercent= {2}, bIsTECInProgress= {3} bIsMotionInProgress= {4}",
                     fCurrentWW, bUnits, wCurrentScanPercent, bIsTECInProgress,
bIsMotionInProgress);
              Thread.Sleep(250);
       } while (bIsScanInProgress);
}
```

Disarming & Disconnecting from MIRcat

```
IsArmed = false;

Console.WriteLine("\n=========================================================");
Console.WriteLine("Attempting to Disarm Laser...");
ret = MIRcatSDK_DisarmLaser();
Console.WriteLine(" Test Result: ret = {0}", ret);
Console.WriteLine("\nChecking Arming Status...");
ret = MIRcatSDK_IsLaserArmed(ref IsArmed);
Console.WriteLine(" Test Result: ret = {0} \tIs the laser armed: {1}", ret, IsArmed);
Thread.Sleep(5000);
Console.WriteLine("\n=========================================================");
Console.WriteLine("Attempting to De-Initialize MIRcatSDK...");
ret = MIRcatSDK_DeInitialize();
switch (ret)
{
	case (UInt32)MIRcatSDK_RET_SUCCESS:
		Console.WriteLine(" Successfully Disconnected from MIRcat");
		break;
	case (UInt32)MIRcatSDK_RET_NOT_INITIALIZED:
		Console.WriteLine(" Attempt to De-initialize before controller
instantiation.");
		break;
	default:
		Console.WriteLine("Fatal Error. \t Error Code:{0}", ret);
		break;
}
Console.WriteLine("=========================================================");
```

Working with MATLAB

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

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 `**MIRcatSDK.h**` file or official SDK documentation to see what is required for each function call.
3. You should call `**unloadlibrary MIRcatSDK;**` 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 7+
- USB <-> Serial Driver
- Mingw x64 c/c++ compiler
- MATLAB License

Configuring the MATLAB Environment

1. Change the MATLAB working directory to the location of your `MIRcatSDK` 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 **`MIRcatSDKconstants.mat`** file and load the C++ Library into the MATLAB workspace.

```
hfile = 'MIRcatSDK.h'; % Denote the location of the headerfile
[notfound, warnings] = loadlibrary('libs/MIRcatSDK', hfile, 'alias', 'MIR-
catSDK');

load('MIRcatSDKconstants.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 MIRcatSDK function you would like to call.
%     For example, `MIRcatSDK_StartSweepScan`
% Variables are comma separated and are listed after the function name.
ret = calllib('MIRcatSDK','Function_Name', variable1, variable2);
```

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

```
libfunctions('MIRcatSDK')
```

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 the function
ret = calllib('MIRcatSDK','MIRcatSDK_GetAPIVersion', majorPtr, minorPtr,
patchPtr);
% Check to see if function call was Successful
if MIRcatSDK_RET_SUCCESS == ret
    fprintf('Successful\n' );
else
    % If the operation fails, unload the library and raise an error.
    unloadlibrary MIRcatSDK;
    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);
```

Connecting to MIRcat

```
fprintf('=========================================================\n');
fprintf('Initializing MIRcat ... ');
% Call your function
ret = calllib('MIRcatSDK','MIRcatSDK_Initialize');
% Check to see if function call was Successful
if MIRcatSDK_RET_SUCCESS == ret
    fprintf('Successful\n' );
else
    % If the operation fails, unload the library and raise an error.
    unloadlibrary MIRcatSDK;
    error('Error! Code: %d', ret);
end
```

Setting Up and Running Scans

Quick Notes:

The MIRcat I/O has a polling frequency of about 4hz. This means that you need to sleep the system for 250 milliseconds before querying system statuses.

Scan Initialization Procedure

These are the basic steps to follow before beginning tuning the laser or beginning a scan. These steps MUST be executed before tuning or scanning or else the system will raise errors.

```
% Step 1: Get the number of installed QCLs
fprintf('========================================================\n');
fprintf('Test: How many QCLs are installed? ... ');
numQCLs = uint8(0);
numQCLsPtr = libpointer('uint8Ptr', numQCLs);
calllib('MIRcatSDK','MIRcatSDK_GetNumInstalledQcls', numQCLsPtr);
numQCLs = numQCLsPtr.value;
fprintf(' %d\n', numQCLs);
```

```
% Step 2: Check for Interlock Status
fprintf('========================================================\n');
fprintf('Test: Is Interlock Set ... ');
isInterlockSet = false;
isInterlockSetPtr = libpointer('bool', isInterlockSet);
ret = calllib('MIRcatSDK','MIRcatSDK IsInterlockedStatusSet',
isInterlockSetPtr);
isInterlockSet = isInterlockSetPtr.value;
if logical(isInterlockSet)
    fprintf(' Yes\n' );
else
    % If the operation fails, unload the library and raise an error.
    fprintf(' NO\n' );
    calllib('MIRcatSDK','MIRcatSDK_DeInitialize');
    unloadlibrary MIRcatSDK;
    error('Error! Interlock is not set. Code: %d', ret);
end

% Step 3: Check for Key Switch Status
fprintf('========================================================\n');
fprintf('Test: Is Key Switch Set ... ');
isKeySwitchSet = false;
isKeySwitchSetPtr = libpointer('bool', isKeySwitchSet);
ret = calllib('MIRcatSDK','MIRcatSDK_IsKeySwitchStatusSet',
isKeySwitchSetPtr);
isKeySwitchSet = isKeySwitchSetPtr.value;
if logical(isKeySwitchSet)
    fprintf(' Yes\n' );
else
    % If the operation fails, unload the library and raise an error.
    fprintf(' NO\n' );
    calllib('MIRcatSDK','MIRcatSDK_DeInitialize');
    unloadlibrary MIRcatSDK;
    error('Error! KeySwitch is not set. Code: %d', ret);
end


% Step 4: Arm the laser
fprintf('========================================================\n');
```

```
fprintf('Test: Arm Laser ... ');
isArmed = false;
isArmedPtr = libpointer('bool', isArmed);
calllib('MIRcatSDK','MIRcatSDK_IsLaserArmed', isArmedPtr);
isArmed = isArmedPtr.value;

if ~isArmed
    ret = calllib('MIRcatSDK','MIRcatSDK_ArmDisarmLaser');
    if MIRcatSDK_RET_SUCCESS == ret
        fprintf(' Successful\n' );

fprintf('=======================================================\n');
        fprintf('Test: Is Laser Armed?\n');
    else
        % If the operation fails, unload the library and raise an error.
        fprintf(' Failure\n');
        unloadlibrary MIRcatSDK;
        error('Error! Code: %d', ret);
    end
else
    fprintf(' Already Armed\n ');
end

while ~isArmed
    calllib('MIRcatSDK','MIRcatSDK_IsLaserArmed', isArmedPtr);
    isArmed = isArmedPtr.value;
    if logical(isArmed)
        fprintf('\tTrue\n' );
    else
        fprintf('\tFalse\n');
    end
    pause(1.0);
end

% Step 5: Wait for TECs to arrive at safe operating temperature
fprintf('=======================================================\n');
fprintf('Are TECs at Safe Operating Temp? ... \n');
atTemp = false;
atTempPtr = libpointer('bool', atTemp);
calllib('MIRcatSDK','MIRcatSDK_AreTECsAtSetTemperature', atTempPtr);
atTemp = atTempPtr.value;
if logical(atTemp)
    fprintf('\tTrue\n' );
end

while ~atTemp
    calllib('MIRcatSDK','MIRcatSDK_AreTECsAtSetTemperature', atTempPtr);
    atTemp = atTempPtr.value;
    if logical(atTemp)
        fprintf('\tTrue\n' );
    else
        fprintf('\tFalse\n');
    end
    pause(1);
end
```

Single Tune Scan

Important: - You MUST cancel Manual Tune Mode before performing another type of scan or else an error will be returned.

```
fprintf('=========================================================\n');
fprintf('Starting Single Tune Test\n\n');
fprintf('=========================================================\n');
fprintf('Test: Tune to WW 8.835 Microns ... ');
ret = calllib('MIRcatSDK','MIRcatSDK_TuneToWW', ...
    single(10.835), MIRcatSDK_UNITS_MICRONS, 1);
if MIRcatSDK_RET_SUCCESS == ret
    fprintf(' Successful\n' );
else
    % If the operation fails, unload the library and raise an error.
    fprintf(' Failure\n' );
    calllib('MIRcatSDK','MIRcatSDK_DeInitialize');
    unloadlibrary MIRcatSDK;
    error('Error! Code: %d', ret);
end



% Check the laser tuning
isTuned = false;
isTunedPtr = libpointer('bool', isTuned);
actualWW = single(0);
actualWWPtr = libpointer('singlePtr', actualWW);
units = uint8(0);
unitsPtr = libpointer('uint8Ptr', units);
lightValid = false;
lightValidPtr = libpointer('bool', lightValid);

fprintf('Test: Is Tuned? ... \n');
calllib('MIRcatSDK','MIRcatSDK_IsTuned', isTunedPtr);
isTuned = isTunedPtr.value;
if isTuned
    fprintf('\t True\n');
end
while ~isTuned
    % Check Tuning Status
    calllib('MIRcatSDK','MIRcatSDK_IsTuned', isTunedPtr);
    isTuned = isTunedPtr.value;
    if logical(isTuned)
        fprintf('\tTrue');
    else
        fprintf('\tFalse');
    end
    % Check Actual Wavelength
    calllib('MIRcatSDK','MIRcatSDK_GetActualWW', actualWWPtr, unitsPtr,
lightValidPtr);
    actualWW = actualWWPtr.value;
    units = unitsPtr.value;
    fprintf('\tActual WW: %.3f \tunits: %u\n', actualWW, units);
    pause(0.1);

end
```

```
% Enable Laser Emission
fprintf('=========================================================\n');
isEmitting = false;
isEmittingPtr = libpointer('bool', isEmitting);
calllib('MIRcatSDK','MIRcatSDK_IsEmissionOn', isEmittingPtr);
isEmitting = isEmittingPtr.value;
if ~isEmitting
    fprintf('Enable Laser Emission... ');
    ret = calllib('MIRcatSDK','MIRcatSDK_TurnEmissionOn');
    if MIRcatSDK_RET_SUCCESS == ret
        fprintf(' Successful\n' );
    else
        % If the operation fails, unload the library and raise an error.
        fprintf('Failure\n' );
        calllib('MIRcatSDK','MIRcatSDK_DeInitialize');
        unloadlibrary MIRcatSDK;
        error('Error! Code: %d', ret);
    end
end

% Check for Laser Emission
fprintf('=========================================================\n');
fprintf('Is Laser Emitting? ... ');
calllib('MIRcatSDK','MIRcatSDK_IsEmissionOn', isEmittingPtr);
isEmitting = isEmittingPtr.value;
if isEmitting
    fprintf(' True\n');
end
fprintf('\n');
while ~isEmitting
    calllib('MIRcatSDK','MIRcatSDK_IsEmissionOn', isEmittingPtr);
    isEmitting = isEmittingPtr.value;
    if isEmitting
        fprintf('\tTrue\n');
    else
        fprintf('\tFalse\n');
    end
    pause(0.5);
end
```

After completing manually tuned scans, you must first disable manual tune mode before you can move on to another scan mode.

```
% IMPORTANT: Disable Manual Scan Tune before starting another scan
fprintf('Test: Disable Manual Tune Mode ... ');
ret = calllib('MIRcatSDK','MIRcatSDK_CancelManualTuneMode');
if MIRcatSDK_RET_SUCCESS == ret
    fprintf(' Successful\n' );
else
    % If the operation fails, unload the library and raise an error.
    fprintf(' Failure\n' );
    calllib('MIRcatSDK','MIRcatSDK_DeInitialize');
    unloadlibrary MIRcatSDK;
    error('Error! Code: %d', ret);
end
```

Sweep Scan

```
fprintf('=========================================================\n');
fprintf('Starting Sweep mode scan from 6.7 to 7.2 um with a speed 100 microns\n');
ret = calllib('MIRcatSDK','MIRcatSDK_StartSweepScan', ...
    single(6.7), single(7.2), single(0.1), ...
    MIRcatSDK_UNITS_MICRONS, uint16(1), true, uint8(1));
if MIRcatSDK_RET_SUCCESS == ret
    fprintf(' Successful\n' );
else
    % If the operation fails, unload the library and raise an error.
    fprintf(' Failure\n' );
    calllib('MIRcatSDK','MIRcatSDK_DeInitialize');
    unloadlibrary MIRcatSDK;
    error('Error! Code: %d', ret);
end

isScanInProgress = true;
isScanInProgressPtr = libpointer('bool', isScanInProgress);
isScanActive = false;
isScanActivePtr = libpointer('bool', isScanActive);
isScanPaused = false;
isScanPausedPtr = libpointer('bool', isScanPaused);
curScanNum = uint16(0);
curScanNumPtr = libpointer('uint16Ptr', curScanNum);
curScanPercent = uint16(0);
curScanPercentPtr = libpointer ('uint16Ptr', curScanPercent);
curWW = single(0);
curWWPtr = libpointer('singlePtr', curWW);
isTECinProgress = false;
isTECinProgressPtr = libpointer('bool', isTECinProgress);
isMotionInProgress = false;
isMotionInProgressPtr = libpointer('bool', isMotionInProgress);

fprintf('=========================================================\n');
fprintf('Test: Get Scan Status\n');
while isScanInProgress
    calllib('MIRcatSDK','MIRcatSDK_GetScanStatus', ...
        isScanInProgressPtr, isScanActivePtr, isScanPausedPtr, ...
        curScanNumPtr, curScanPercentPtr, curWWPtr, unitsPtr, ...
        isTECinProgressPtr, isMotionInProgressPtr);
    isScanInProgress = isScanInProgressPtr.value;
    isScanActive = isScanActivePtr.value;
    isScanPaused = isScanPausedPtr.value;
    curScanNum = curScanNumPtr.value;
    curScanPercent = curScanPercentPtr.value;
    curWW = curWWPtr.value;
    units = unitsPtr.value;
    isTECinProgress = isTECinProgressPtr.value;
    isMotionInProgress = isMotionInProgressPtr.value;
    fprintf(['\tIsScanInProgress: %d \tIsScanActive: %d \tisScanPaused: %d', ...
        '\tcurScanNum: %d \tcurWW: %.3f \tunits: %u \tcurScanPercent: %.2f', ...
        '\tisTECinProgress: %d \tisMotionInProgress: %d\n'], ...
        isScanInProgress, isScanActive, isScanPaused, curScanNum, curWW, ...
        units, curScanPercent, isTECinProgress, isMotionInProgress);
    pause(0.3);
end
```

Step-Measure Scan

```
fprintf('=======================================================\n');
fprintf('Starting Step-Measure Scan ... ');
ret = calllib('MIRcatSDK','MIRcatSDK_StartStepMeasureModeScan', ...
    single(6.7), single(7.0), single(0.25), MIRcatSDK_UNITS_MICRONS, uint8(1));
if MIRcatSDK_RET_SUCCESS == ret
    fprintf(' Successful\n' );
else
    % If the operation fails, unload the library and raise an error.
    fprintf(' Failure\n' );
    calllib('MIRcatSDK','MIRcatSDK_DeInitialize');
    unloadlibrary MIRcatSDK;
    error('Error! Code: %d', ret);
end

isScanInProgress = true;
isScanInProgressPtr = libpointer('bool', isScanInProgress);
isScanActive = false;
isScanActivePtr = libpointer('bool', isScanActive);
isScanPaused = false;
isScanPausedPtr = libpointer('bool', isScanPaused);
curScanNum = uint16(0);
curScanNumPtr = libpointer('uint16Ptr', curScanNum);
curScanPercent = uint16(0);
curScanPercentPtr = libpointer ('uint16Ptr', curScanPercent);
curWW = single(0);
curWWPtr = libpointer('singlePtr', curWW);
isTECinProgress = false;
isTECinProgressPtr = libpointer('bool', isTECinProgress);
isMotionInProgress = false;
isMotionInProgressPtr = libpointer('bool', isMotionInProgress);

fprintf('=======================================================\n');
fprintf('Test: Get Scan Status\n');
while isScanInProgress
    calllib('MIRcatSDK','MIRcatSDK_GetScanStatus', ...
        isScanInProgressPtr, isScanActivePtr, isScanPausedPtr, ...
        curScanNumPtr, curScanPercentPtr, curWWPtr, unitsPtr, ...
        isTECinProgressPtr, isMotionInProgressPtr);
    isScanInProgress = isScanInProgressPtr.value;
    isScanActive = isScanActivePtr.value;
    isScanPaused = isScanPausedPtr.value;
    curScanNum = curScanNumPtr.value;
    curScanPercent = curScanPercentPtr.value;
    curWW = curWWPtr.value;
    units = unitsPtr.value;
    isTECinProgress = isTECinProgressPtr.value;
    isMotionInProgress = isMotionInProgressPtr.value;
    fprintf(['\tIsScanInProgress: %d \tIsScanActive: %d \tisScanPaused: %d', ...
        '\tcurScanNum: %d \tcurWW: %.3f \tunits: %u \tcurScanPercent: %.2f', ...
        '\tisTECinProgress: %d \tisMotionInProgress: %d\n'], ...
        isScanInProgress, isScanActive, isScanPaused, curScanNum, curWW, ...
        units, curScanPercent, isTECinProgress, isMotionInProgress);
    pause(0.3);
end
```

Multi-Spectral Scan

```
fprintf('=======================================================\n');
fprintf('Starting Multi-Spectral Scan ... ');

fprintf('=======================================================\n');
fprintf('Test: Set Amount of Multi-Specral Elements ... ');
ret = calllib('MIRcatSDK','MIRcatSDK_SetNumMultiSpectralElements', 10);
if MIRcatSDK_RET_SUCCESS == ret
    fprintf(' Successful\n' );
else
    % If the operation fails, unload the library and raise an error.
    fprintf(' Failure\n' );
    unloadlibrary MIRcatSDK;
    error('Error! Code: %d', ret);
end

fprintf('=======================================================\n');
fprintf('Test: Add Multi-Specral Elements ... ');
startWW = single(5.7);
for i = 0.0 : 0.5 : 4.5
    fprintf('\tTest: Add Multi-Specral Element ... ');
    ret = calllib('MIRcatSDK','MIRcatSDK_AddMultiSpectralElement', ...
        single(startWW + i), MIRcatSDK_UNITS_MICRONS, 1000, 1000);
    if MIRcatSDK_RET_SUCCESS == ret
        fprintf(' Successful\n' );
    else
        % If the operation fails, unload the library and raise an error.
        fprintf(' Failure\n' );
        unloadlibrary MIRcatSDK;
        error('Error! Code: %d', ret);
    end
end

fprintf('=======================================================\n');
fprintf('Test: Start Multi-Spectral Scan ... ');
ret = calllib('MIRcatSDK','MIRcatSDK_StartMultiSpectralModeScan', 1);
if MIRcatSDK_RET_SUCCESS == ret
    fprintf(' Successful\n' );
else
    % If the operation fails, unload the library and raise an error.
    fprintf(' Failure\n' );
    unloadlibrary MIRcatSDK;
    error('Error! Code: %d', ret);
end

isScanInProgress = true;
isScanInProgressPtr = libpointer('bool', isScanInProgress);
isScanActive = false;
isScanActivePtr = libpointer('bool', isScanActive);
isScanPaused = false;
isScanPausedPtr = libpointer('bool', isScanPaused);
curScanNum = uint16(0);
curScanNumPtr = libpointer('uint16Ptr', curScanNum);
curScanPercent = uint16(0);
curScanPercentPtr = libpointer ('uint16Ptr', curScanPercent);
curWW = single(0);
curWWPtr = libpointer('singlePtr', curWW);
isTECinProgress = false;
isTECinProgressPtr = libpointer('bool', isTECinProgress);
```

```
isMotionInProgress = false;
isMotionInProgressPtr = libpointer('bool', isMotionInProgress);

fprintf('=========================================================\n');
fprintf('Test: Get Scan Status\n');
while isScanInProgress
    calllib('MIRcatSDK','MIRcatSDK_GetScanStatus', ...
        isScanInProgressPtr, isScanActivePtr, isScanPausedPtr, ...
        curScanNumPtr, curScanPercentPtr, curWWPtr, unitsPtr, ...
        isTECinProgressPtr, isMotionInProgressPtr);
    isScanInProgress = isScanInProgressPtr.value;
    isScanActive = isScanActivePtr.value;
    isScanPaused = isScanPausedPtr.value;
    curScanNum = curScanNumPtr.value;
    curScanPercent = curScanPercentPtr.value;
    curWW = curWWPtr.value;
    units = unitsPtr.value;
    isTECinProgress = isTECinProgressPtr.value;
    isMotionInProgress = isMotionInProgressPtr.value;
    fprintf(['\tIsScanInProgress: %d \tIsScanActive: %d \tisScanPaused: %d', ...
        '\tcurScanNum: %d \tcurWW: %.3f \tunits: %u \tcurScanPercent: %.2f', ...
        '\tisTECinProgress: %d \tisMotionInProgress: %d\n'], ...
        isScanInProgress, isScanActive, isScanPaused, curScanNum, curWW, ...
        units, curScanPercent, isTECinProgress, isMotionInProgress);
    pause(0.3);
end
```

Disarming & Disconnecting from MIRcat

```
% Disarm Laser
fprintf('=========================================================\n');
fprintf('Disarming Laser ... ');
ret = calllib('MIRcatSDK','MIRcatSDK_DisarmLaser');
if MIRcatSDK_RET_SUCCESS == ret
    fprintf('Successful\n' );
else
    % If the operation fails, unload the library and raise an error.
    calllib('MIRcatSDK','MIRcatSDK_DeInitialize');
    unloadlibrary MIRcatSDK;
    error('Error! Code: %d', ret);
end

% Disconnect from MIRcat
fprintf('=========================================================\n');
fprintf('De-Initialize MIRcatSDK ... ');
ret = calllib('MIRcatSDK','MIRcatSDK_DeInitialize');
if MIRcatSDK_RET_SUCCESS == ret
    fprintf('Successful\n' );
else
    % If the operation fails, unload the library and raise an error.
    unloadlibrary MIRcatSDK;
    error('Error! Code: %d', ret);
end
```

Working with Python

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

Important Notes:

1. Python interfaces nicely with the C++ language. In order to use the MIRcatSDK dll, you must use the built in library `ctypes`. Please familiarize yourself with the ctypes documentation before getting started.
2. The bit type of your Python complier determines which version of the MIRcatSDK.dll you should use. For example, if you are using the 32bit Python interpreter, you should use the MIRcatSDK.dll found in the `libs/x32/` directory.
3. Be sure to import the files `MIRcatSDKConstants` and `MIRcatSDKHelpers`

Requirements

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

Configuring PyCharm Environment

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

```
import os
import time
from MIRcatSDKConstants import *
from MIRcatSDKHelpers import ArmAndWaitForTemp
```

3. Change working directory to libraries directory of MIRcatSDK

```
os.chdir("../libs/x64") # Change the working directory to reference
libraries based on version of python x32 or x64
SDK = CDLL("MIRcatSDK")
```

Basic Operations

Getting SDK API Version

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

print("Test: Get API Version")
ret = SDK.MIRcatSDK_GetAPIVersion(byref(major), byref(minor), byref(patch))
print(" Result: {0} \tVersion: {1}.{2}.{3}".format(ret, major.value, minor.value,
patch.value))
```

Initialize MIRcatSDK

```
# Initialize MIRcatSDK & Connect to MIRcat laser
print("Test: Initialize MIRcatSDK")
ret = SDK.MIRcatSDK_Initialize()
if ret == MIRcatSDK_RET_SUCCESS.value:
    print(" Successfully Connected to MIRcat")
else:
    print(" Failure to Initialize API. \tError Code: {0}".format(ret))
    exit(0)
```

Setting Up and Running Scans

The MIRcat I/O has a polling frequency of about 4hz. This means that you need to sleep the system for 250 milliseconds before querying system statuses.

Helper Function for Arming Laser

This is a function that is included in the example project and can be found in the file, `MIRcatSDKHelpers`. It serves as a template for arming the laser and waiting for the TECs to get to a safe operating temperature.

```
import time
from MIRcatSDKConstants import *


def ArmAndWaitForTemp(SDK, numQcls):  # Do not pass in numQcls by reference
    atTemp = c_bool(False)
    isArmed = c_bool(False)
    ret = SDK.MIRcatSDK_IsLaserArmed(byref(isArmed))
    if not isArmed.value:
        ret = SDK.MIRcatSDK_ArmDisarmLaser()
        print(" Test Result: \tret:{0}".format(ret))

        print("#**************************************************************#")
        print("Test: Is Laser Armed?")

    while not isArmed.value:
        ret = SDK.MIRcatSDK_IsLaserArmed(byref(isArmed))
        print(" Test Result: \tret:{0} \tIsArmed: {1}".format(ret, isArmed.value))
        time.sleep(1)

    # Wait until TECs are at temperature before doing any tuning/scanning
    # Note: This can take a while depending on how the laser is cooled.
    print("#**************************************************************#")
    print("Test : TEC Temperature Status")
    ret = SDK.MIRcatSDK_AreTECsAtSetTemperature(byref(atTemp))
    print(" Test Result: {0} \tatTemp = {1}".format(ret, atTemp.value))
    tecCur = c_uint16(0)
    qclTemp = c_float(0)
    while not atTemp.value:
        for i in range(1, numQcls.value + 1):
            ret = SDK.MIRcatSDK_GetQCLTemperature(c_uint8(i), byref(qclTemp))
            print(" Test Result: \tret:{0} \tQCL:{1} \tTemp: {2:.5} C"
                  .format(ret, i, qclTemp.value))
            ret = SDK.MIRcatSDK_GetTecCurrent(c_uint8(i), byref(tecCur))
            print(" Test Result: \tret:{0} \tTEC:{1} \tCurrent: {2} mA"
                  .format(ret, i, tecCur.value))
```

```
        ret = SDK.MIRcatSDK_AreTECsAtSetTemperature(byref(atTemp))
        print("TECs at Temperature: \tret:{0} \tatTemp = {1}"
              .format(ret, atTemp.value))
        time.sleep(.1)

    return atTemp
```

Scan Initialization Structure

```
isInterlockSet = c_bool(False)
isKeySwitchSet = c_bool(False)
numQCLs = c_uint8(0)

# Step 1: Get the number of installed QCLs
print("#*****************************************************************#")
print("Test: How many QCLs?")
SDK.MIRcatSDK_GetNumInstalledQcls(byref(numQCLs))
print(" QCLs: {0}".format(numQCLs.value))


# Step 2: Check for Interlock Status
print("#*****************************************************************#")
print("Test: Is Interlock Set?")
ret = SDK.MIRcatSDK_IsInterlockedStatusSet(byref(isInterlockSet))
if isInterlockSet.value:
    print(" Interlock Set: {0}".format(isInterlockSet.value))
else:
    print(" Interlock Set: {0} \tret:{0}".format(isInterlockSet.value, ret))
    exit(0)

# Step 3: Check for Key Switch Status
print("#*****************************************************************#")
print("Test: Is Key Switch Set?")
ret = SDK.MIRcatSDK_IsKeySwitchStatusSet(byref(isKeySwitchSet))
if isKeySwitchSet.value:
    print(" KeySwitch Set: {0}".format(isKeySwitchSet.value))
else:
    print(" KeySwitch Set: {0} \tret:{1}".format(isKeySwitchSet.value, ret))
    exit(0)

# Step 4: Arm the laser
print("#*****************************************************************#")
print("Test: Arm Laser")
ArmAndWaitForTemp(SDK, numQCLs)
```

Single Tune Test

Important: You MUST cancel Manual Tune Mode before performing another type of scan or else you will get returned an error.

```
# Single Tune Test
print("#*****************************************************************#")
print("Staring single tune test\n")
print("Test: Tune to WW 8.80")
ret = SDK.MIRcatSDK_TuneToWW(c_float(8.835), MIRcatSDK_UNITS_MICRONS, c_uint8(1))
print(" Test Result: \tret:{0}".format(ret))

print("#*****************************************************************#")
```

```
print("Test: Check Tuned Wavelength")
tunedWW = c_float()
units = c_uint8()
QCL = c_uint8()
ret = SDK.MIRcatSDK_GetTuneWW(byref(tunedWW), byref(units), byref(QCL))
print(" Test Results:\tret:{0} \tWavelength: {1:.5} \tUnits: {2}
\tQCL:{3}".format(ret, tunedWW.value, units.value, QCL.value))

print("#****************************************************************#")
print("Test: Is Tuned?")
isTuned = c_bool(False)
actualWW = c_float()
lightValid = c_bool()
while not isTuned.value:
    """Check Tuning Status"""
    ret = SDK.MIRcatSDK_IsTuned(byref(isTuned))
    print(" Test Result:\tret:{0} \tisTuned: {1}".format(ret, isTuned.value))
    """Check Actual Wavelength"""
    ret = SDK.MIRcatSDK_GetActualWW(byref(actualWW), byref(units), byref(lightValid))
    print(" Test Result:\tret:{0} \tActual WW: {1:.5} \tUnits:{2} \tLight Valid:
{3}".format(ret, actualWW.value, units.value, lightValid.value))
    time.sleep(0.05)

print("#****************************************************************#")
print("Test: Enable Laser Emission")
ret = SDK.MIRcatSDK_TurnEmissionOn()
print(" Test Result: ret:{0}".format(ret))

print("#****************************************************************#")
print("Test : Is laser emitting?")
isEmitting = c_bool(False)
while not isEmitting.value:
    ret = SDK.MIRcatSDK_IsEmissionOn(byref(isEmitting))
    print(" Test Result:\tret:{0} \tIs Emitting: {1}".format(ret, isEmitting.value))
    time.sleep(0.5)

print("#****************************************************************#")
print("Test: Disable Laser")
ret = SDK.MIRcatSDK_TurnEmissionOff()
print(" Test Result:\tret:{0}".format(ret))

print("#****************************************************************#")
print("Test : Is laser emitting?")
ret = SDK.MIRcatSDK_IsEmissionOn(byref(isEmitting))
print(" Test Result:\tret:{0} \tIs Emitting: {1}".format(ret, isEmitting.value))
```

After completing manually tuned scans, you must first disable manual tune mode before you can move on to another scan mode.

```
print("#****************************************************************#")
print("Test: Cancel Manual Scan Mode")
ret = SDK.MIRcatSDK_CancelManualTuneMode()
print(" Test Result:\tret:{0}".format(ret))
```

Sweep Scan

```
print("#*****************************************************************#")
# Step 5: Start Sweep Scan
print("Starting Sweep mode scan from 6.7 to 7.2 um with a speed 100 microns")
ret = SDK.MIRcatSDK_StartSweepScan(c_float(6.7), c_float(7.2), c_float(.1),
MIRcatSDK_UNITS_MICRONS, c_uint16(1), c_bool(True), c_uint8(1))
print(" Test Result:\tret:{0}".format(ret))
if ret == MIRcatSDK_RET_SUCCESS.value:
    isScanInProgress = c_bool(True)
    isScanActive = c_bool(False)
    isScanPaused = c_bool(False)
    curScanNum = c_uint16()
    curScanPercent = c_uint16()
    curWW = c_float()
    isTECinProgress = c_bool()
    isMotionInProgress = c_bool()
    print("#*****************************************************************#")

    while isScanInProgress.value:
        print("Test: Get Scan Status")
        ret = SDK.MIRcatSDK_GetScanStatus( byref(isScanInProgress),
byref(isScanActive), byref(isScanPaused),
                                    byref(curScanNum), byref(curScanPercent),
byref(curWW),
                                    byref(units), byref(isTECinProgress),
byref(isMotionInProgress))
        print(" Test Result:\tisScanInProgress = {} \tisScanActive = {}
\tbisScanPaused = {} \tCurScanNum = {}"
              .format(isScanInProgress.value, isScanActive.value, isScanPaused.value,
curScanNum.value))
        print(" Test Result:\tCurrentWW= {0:.3} \tUnits = {1} \tCurrentScanPercent =
{2} \tisTECInProgress = {3} \tisMotionInProgress = {4}"
              .format(curWW.value, units.value, curScanPercent.value,
isTECinProgress.value, isMotionInProgress.value))
        ret = SDK.MIRcatSDK_GetActualWW(byref(actualWW), byref(units),
byref(lightValid))
        print(" Test Result:\tActual WW: {:.3} \tValid: {}\n"
              .format(actualWW.value, lightValid.value))
        time.sleep(0.3)
```

Step-Measure Scan

```
print("#****************************************************************#")
print("Starting Step-Measure Scan")
ret = SDK.MIRcatSDK_StartStepMeasureModeScan(c_float(6.7), c_float(7.0),
c_float(0.25), MIRcatSDK_UNITS_MICRONS, c_uint8(1))
print(" Test Result:\tret:{}".format(ret))
time.sleep(0.1)
if ret == MIRcatSDK_RET_SUCCESS.value:
    isScanInProgress = c_bool(True)
    print("#****************************************************************#")
    while isScanInProgress.value:
        print("Test: Get Scan Status")
        ret = SDK.MIRcatSDK_GetScanStatus(byref(isScanInProgress),
byref(isScanActive), byref(isScanPaused),
                                          byref(curScanNum), byref(curScanPercent),
byref(curWW),
                                          byref(units), byref(isTECinProgress),
byref(isMotionInProgress))
        print(" Test Result:\tisScanInProgress = {} \tisScanActive = {}
\tbisScanPaused = {} \tCurScanNum = {}"
              .format(isScanInProgress.value, isScanActive.value, isScanPaused.value,
curScanNum.value))
        print(
            " Test Result:\tCurrentWW= {0:.3} \tUnits = {1} \tCurrentScanPercent = {2}
\tisTECInProgress = {3} \tisMotionInProgress = {4}"
            .format(curWW.value, units.value, curScanPercent.value,
isTECinProgress.value, isMotionInProgress.value))
        ret = SDK.MIRcatSDK_GetActualWW(byref(actualWW), byref(units),
byref(lightValid))
        print(" Test Result:\tActual WW: {:.3} \tValid: {}\n"
              .format(actualWW.value, lightValid.value))
        time.sleep(0.3)
```

Multi-Spectral Scan

```
print("#****************************************************************#")
print("Starting Multi-Spectral Scan\n\n")
# Step 5: Set the amount of Multi-Spectral Elements
print("Test: SetNumMultiSpectralElements")
SDK.MIRcatSDK_SetNumMultiSpectralElements(10)
print(" Test Result:\tret:{0}".format(ret))
# Step 6: Add Multi-Spectral Elements
fScanWW = 5.7
i = 0.0
while i < 5.0: # Python does not easily support for-loops with floats
    print("Test : AddMultiSpectralElement")
    ret = SDK.MIRcatSDK_AddMultiSpectralElement(c_float(fScanWW + i),
MIRcatSDK_UNITS_MICRONS, c_uint32(1000), c_uint32(1000))
    print(" Test Result:\tret:{0}".format(ret))
    i += 0.5
print("#****************************************************************#")
print("Test: StartMultiSpectralModeScan")
# Step 7: Start the Multi-Spectral Scan
ret = SDK.MIRcatSDK_StartMultiSpectralModeScan(c_uint16(1))
print(" Test Result:\tret:{0}".format(ret))
if ret == MIRcatSDK_RET_SUCCESS.value:
    isScanInProgress = c_bool(True)
    print("#****************************************************************#")
    while isScanInProgress.value:
        print("Test: Get Scan Status")
        ret = SDK.MIRcatSDK_GetScanStatus(byref(isScanInProgress),
byref(isScanActive), byref(isScanPaused),
                                          byref(curScanNum), byref(curScanPercent),
byref(curWW),
                                          byref(units), byref(isTECinProgress),
byref(isMotionInProgress))
        print(" Test Result:\tisScanInProgress = {} \tisScanActive = {}
\tbisScanPaused = {} \tCurScanNum = {}"
              .format(isScanInProgress.value, isScanActive.value, isScanPaused.value,
curScanNum.value))
        print(
            " Test Result:\tCurrentWW= {0:.3} \tUnits = {1} \tCurrentScanPercent = {2}
\tisTECInProgress = {3} \tisMotionInProgress = {4}"
            .format(curWW.value, units.value, curScanPercent.value,
isTECinProgress.value, isMotionInProgress.value))
        ret = SDK.MIRcatSDK_GetActualWW(byref(actualWW), byref(units),
byref(lightValid))
        print(" Test Result:\tActual WW: {:.3} \tValid: {}\n"
              .format(actualWW.value, lightValid.value))
        time.sleep(0.3)
```

Disarming & Disconnecting from MIRcat

```
# Disarm Laser
print("#*****************************************************************#")
print("Attempting to Disarm Laser...")
ret = SDK.MIRcatSDK_DisarmLaser()
print(" Test Result: {0}".format(ret))


# Check Arming Status
print("#*****************************************************************#")
print("Test: Is Laser Armed?")
isArmed = c_bool(True)
ret = SDK.MIRcatSDK_IsLaserArmed(byref(isArmed))
print(" Test Result: {0} \tIs Armed: {1}".format(ret, isArmed.value))


# Disconnect from MIRcat
print("#*****************************************************************#")
print("Attempting to De-Initialize MIRcatSDK...")
ret = SDK.MIRcatSDK_DeInitialize()
print(" Test Result: {0}".format(ret))
```