// MarkerLibTest.cpp : Defines the entry point for the console application.

//

#include "stdafx.h"

#include <vector>

#include "commondefs.h"

#include "LoopingSegment.h"

#include "lec\_public\_exports.h"

const wchar\_t \*wsIPAddress=L"127.0.0.1";

//const wchar\_t \*wcsDefaultDeviceID=L"00:50:c2:c8:c1:3e";

//const wchar\_t \*wcsDefaultDeviceID=L"00:50:c2:c8:c0:00";

const wchar\_t \*wcsDefaultLANDeviceID=L"00:50:c2:4f:a2:f9";

const wchar\_t \*wcsDefaultPCIDeviceID=L"\\?\pci#ven\_1597&dev\_0301&subsys\_00011597&rev\_00#0000000101000a3500#{b74cfec2-9366-454a-ba71-7c27b51470a4}";

wchar\_t wcsDeviceID[120];

int seqNum = 0;

using namespace std;

const int MAC\_ADDR\_LEN=17;

const int PCI\_ID\_LEN=106;

void \_\_stdcall DeviceDetectionCallback(LanmarkControls::HardwareModel AModel, unsigned char\* ADeviceBytes, int AByteSize)

{

}

void \_\_stdcall JobEventCallback (unsigned \_\_int64 AJobRef,

unsigned \_\_int64 AObjRef,

LanmarkControls::LECJobEvents AEventCode,

LanmarkControls::LECResponse AResponseCode,

int APayload2,

int APayload3,

int APayload4)

{

}

void \_\_stdcall ActionEventCallback (unsigned \_\_int64 AJobRef,

LanmarkControls::UndoRedoState AUndo,

LanmarkControls::UndoRedoState ARedo)

{

}

void \_\_stdcall DeviceEventCallback(wchar\_t\* ADeviceID, unsigned int AHighPayload, unsigned int ALowPayload)

{

}

int ListCmdErrorMsg(int xRet, int seq)

{

if (xRet != LanmarkControls::Success)

switch(seq)

{

case 1:

printf("Failed to store the Application\_Event command. Error Code:%s\n",(LanmarkControls::LECResponse)xRet);

break;

case 2:

printf("Failed to store the Begin\_Job\_Event command. Error Code:%d\n",xRet);

break;

case 3:

printf("Failed to store the Enable\_MOTF command. Error Code:%d\n",xRet);

break;

case 4:

printf("Failed to store the Enable\_Wobble command. Error Code:%d\n",xRet);

break;

case 5:

printf("Failed to store the End\_Job\_Event command. Error Code:%d\n",xRet);

break;

case 6:

printf("Failed to store the Job\_Marker command. Error Code:%d\n",xRet);

break;

case 7:

printf("Failed to store the Jump\_Abs command. Error Code:%d\n",xRet);

break;

case 8:

printf("Failed to store the Laser\_On command. Error Code:%d\n",xRet);

break;

case 9:

printf("Failed to store the Laser\_Signal\_Off command. Error Code:%d\n",xRet);

break;

case 10:

printf("Failed to store the Laser\_Signal\_On command. Error Code:%d\n",xRet);

break;

case 11:

printf("Failed to store the Long\_Delay command. Error Code:%d\n",xRet);

break;

case 12:

printf("Failed to store the Mark\_Abs command. Error Code:%d\n",xRet);

break;

case 13:

printf("Failed to store the MOTF\_Reset\_Jump command. Error Code:%d\n",xRet);

break;

case 14:

printf("Failed to store the Set\_Active\_Correction\_Table command. Error Code:%d\n",xRet);

break;

case 15:

printf("Failed to store the Set\_Correction\_Table command. Error Code:%d\n",xRet);

break;

case 16:

printf("Failed to store the Set\_Enable\_Laser command. Error Code:%d\n",xRet);

break;

case 17:

printf("Failed to store the Set\_Galvo\_DAC\_Config command. Error Code:%d\n",xRet);

break;

case 18:

printf("Failed to store the Set\_Jump\_Speed command. Error Code:%d\n",xRet);

break;

case 19:

printf("Failed to store the Set\_Laser\_Enable\_Delay command. Error Code:%d\n",xRet);

break;

case 20:

printf("Failed to store the Set\_Laser\_Enable\_Timeout command. Error Code:%d\n",xRet);

break;

case 21:

printf("Failed to store the Set\_Laser\_FPK command. Error Code:%d\n",xRet);

break;

case 22:

printf("Failed to store the Set\_Laser\_Mod\_Delay command. Error Code:%d\n",xRet);

break;

case 23:

printf("Failed to store the Set\_Laser\_Mode\_Config command. Error Code:%d\n",xRet);

break;

case 24:

printf("Failed to store the Set\_Laser\_Off\_Delay command. Error Code:%d\n",xRet);

break;

case 25:

printf("Failed to store the Set\_Laser\_On\_Delay command. Error Code:%d\n",xRet);

break;

case 26:

printf("Failed to store the Set\_Laser\_Pipeline\_Delay command. Error Code:%d\n",xRet);

break;

case 27:

printf("Failed to store the Set\_Laser\_Power command. Error Code:%d\n",xRet);

break;

case 28:

printf("Failed to store the Set\_Laser\_Power\_Change\_Delay command. Error Code:%d\n",xRet);

break;

case 29:

printf("Failed to store the Set\_Laser\_Pulse command. Error Code:%d\n",xRet);

break;

case 30:

printf("Failed to store the Set\_Laser\_Standby command. Error Code:%d\n",xRet);

break;

case 31:

printf("Failed to store the Set\_Laser\_Timing command. Error Code:%d\n",xRet);

break;

case 32:

printf("Failed to store the Set\_Mark\_Speed command. Error Code:%d\n",xRet);

break;

case 33:

printf("Failed to store the Set\_Microvector\_Delay command. Error Code:%d\n",xRet);

break;

case 34:

printf("Failed to store the Set\_MOTF\_Direction command. Error Code:%d\n",xRet);

break;

case 35:

printf("Failed to store the Set\_Microvector\_Delay command. Error Code:%d\n",xRet);

break;

case 36:

printf("Failed to store the Set\_MOTF\_Direction command. Error Code:%d\n",xRet);

break;

case 37:

printf("Failed to store the Set\_MOTF\_Mode command. Error Code:%d\n",xRet);

break;

case 38:

printf("Failed to store the Set\_PM\_FieldOffset command. Error Code:%d\n",xRet);

break;

case 39:

printf("Failed to store the Set\_PM\_FieldOrientation command. Error Code:%d\n",xRet);

break;

case 40:

printf("Failed to store the Set\_PM\_LaserPowerPct command. Error Code:%d\n",xRet);

break;

case 41:

printf("Failed to store the Set\_PM\_MarkSpeedPct command. Error Code:%d\n",xRet);

break;

case 42:

printf("Failed to store the Set\_PM\_PulsePeriodPct command. Error Code:%d\n",xRet);

break;

case 43:

printf("Failed to store the Set\_PM\_PulseWidthPct command. Error Code:%d\n",xRet);

break;

case 44:

printf("Failed to store the Set\_Servo\_Config command. Error Code:%d\n",xRet);

break;

case 45:

printf("Failed to store the Set\_Wobble command. Error Code:%d\n",xRet);

break;

case 46:

printf("Failed to store the Wait\_For\_IO command. Error Code:%d\n",xRet);

break;

case 47:

printf("Failed to store the Wait\_For\_MOTF\_Count command. Error Code:%d\n",xRet);

break;

case 48:

printf("Failed to store the Write\_Analog command. Error Code:%d\n",xRet);

break;

case 49:

printf("Failed to store the Write\_Digital command. Error Code:%d\n",xRet);

break;

case 50:

printf("Failed to store the Z\_Abs command. Error Code:%d\n",xRet);

break;

}

return 0;

}

/\*

int test1(vector<LECBaseDevice \*> \*pDevices)

{

int nRet=0;

//Load Data

char \*str1="Lanmark Controls Inc";

char \*str2="Wenlong";

char \*str3="Data";

char \*str4="Test";

LoadSegment(0,(unsigned char\*)str1,strlen(str1));

LoadSegment(2,(unsigned char\*)str2,strlen(str2));

LoadSegment(3,(unsigned char\*)str3,strlen(str3));

LoadSegment(4,(unsigned char\*)str4,strlen(str4));

//Set the connection details

CONNECTDETAILS details;

details.devicemodel=LanmarkControls::LEC2\_USB;

details.clienttype=LanmarkControls::ds\_DeviceUSB;

//use the first available device

wcscpy\_s(details.deviceid,DEVICE\_ID\_LENGTH,(\*(pDevices->begin()))->FDeviceID);

printf("Execute segments\n");

nRet=ExecuteSegment(details,0);

if (nRet==LanmarkControls::Success)

printf("Succeeded in sending data to the driver\n");

else

printf("Failed to send data to the driver, Error Code:%d\n",nRet);

getchar();

//test looping

//Create a list of looping segments

LoopingSegment \*p1=new LoopingSegment();

p1->SetIndex(0);

//Then we loop on "Control Inc"

p1->SetLoopPoint(8);

LoopingSegment \*p2=new LoopingSegment();

p2->SetIndex(2);

//Then we loop on "long"

p2->SetLoopPoint(3);

LoopingSegment \*p3=new LoopingSegment();

p3->SetIndex(3);

//Then we loop on "Liu"

p3->SetLoopPoint(0);

LoopingSegment \*p4=new LoopingSegment();

p4->SetIndex(4);

//Then we loop on "Jason"

p4->SetLoopPoint(0);

//Link those segments

p1->SetNext(p2);

p2->SetNext(p3);

p3->SetNext(p4);

p4->SetNext(NULL);

printf("\*\*\*\*\*\*\*\*\*Looping\*\*\*\*\*\*\*\*\*\n");

LoopSegment(details,p1);

//sleep and abort

::Sleep(1000);

AbortList(details);

//wait for abort

::Sleep(1000);

return nRet;

}\*/

//This is a really meaningful sample

/\*

int test2(vector<wchar\_t\*> \*pDeviceIDList, MarkerLibraryRT\* pMarkerLibrary)

{

int nRet=0;

//Load Data

char \*str1="Lanmark Controls Inc";

char \*str2="Long String!Long String!Long String!Long String!Long String!Long String!Long String!";

char \*str3="Data";

char \*str4="Test";

//pMarkerLibrary.LoadSegment(0,(unsigned char\*)str1,strlen(str1));

//pMarkerLibrary.LoadSegment(2,(unsigned char\*)str2,strlen(str2));

//pMarkerLibrary.LoadSegment(3,(unsigned char\*)str3,strlen(str3));

//pMarkerLibrary.LoadSegment(4,(unsigned char\*)str4,strlen(str4));

//Set the connection details

CONNECTDETAILS details;

details.devicemodel=LanmarkControls::LEC2\_USB;

details.clienttype=LanmarkControls::ds\_DeviceUSB;

//use the first available device

wcscpy\_s(details.deviceid,DEVICE\_ID\_LENGTH,(\*(pDeviceIDList->begin())));

printf("Execute segments:%ws\n",details.deviceid);

nRet=ExecuteList(details.deviceid,0);

if (nRet==LanmarkControls::Success)

printf("Succeeded in sending data to the driver\n");

else

printf("Failed to send data to the driver, Error Code:%d\n",nRet);

getchar();

//test looping

//Create a list of looping segments

LoopingSegment \*p1=new LoopingSegment();

p1->SetIndex(0);

//Then we loop on "Control Inc"

p1->SetLoopPoint(8);

LoopingSegment \*p2=new LoopingSegment();

p2->SetIndex(2);

//Then we loop on "long"

p2->SetLoopPoint(8);

LoopingSegment \*p3=new LoopingSegment();

p3->SetIndex(3);

//Then we loop on "Data"

p3->SetLoopPoint(0);

LoopingSegment \*p4=new LoopingSegment();

p4->SetIndex(4);

//Then we loop on "Test"

p4->SetLoopPoint(0);

//Link those segments

p1->SetNext(p2);

p2->SetNext(p3);

p3->SetNext(p4);

p4->SetNext(p2);

printf("\*\*\*\*\*\*\*\*\*Looping\*\*\*\*\*\*\*\*\*\n");

//LoopSegment(details,p1);

nRet=LoopList(details.deviceid, p1);

//sleep and abort

::Sleep(1000);

//AbortList(details);

//wait for abort

::Sleep(1000);

return nRet;

return 0;

}

\*/

int jumpStepSize = 100;

int jumpStepPeriod = 300;

int markStepSize = 10;

int markStepPeriod = 300;

int indexZero = 0;

int indexOne = 1;

int indexTwo = 2;

int indexThree = 3;

int indexFour = 4;

int indexFive = 5;

int length = 1000;

int delay = 1;

int laserOnPeriod = 10000;

int opDelay = 500;

int laserEnDelay = 50;

int laserEnTimeout = 500;

int pwrChgDelay = 100;

int laserPower = 100;

int laserPowerPct = 1;

int markSpeedPct = 1;

int laserSignalIDP = 1;

int laserSignalIDW = 1;

int laserNumSetPulse = 1;

int laserPulseWidth = 20;

int laserPulsePeriod = 100;

int pulsePeriodPct = 1;

int pulseWidthPct = 1;

int laserWidthPct = 1;

int laserNumSetStandby = 1;

int laserStandbyPulseWidth = 20;

int laserStandbyPulsePeriod = 100;

int orientation = 0;

int xOffset = 0;

int yOffset = 0;

int zOffset = 0;

int MOTFMode = 1;

int zCoordinate = 0;

int tickUnit = 10;

//This tests LEC1 LAN device

int test3(vector<wchar\_t\*> \*pDeviceIDList)

{

int nRet=0;

for (vector<wchar\_t\*>::iterator iter=pDeviceIDList->begin();

iter!=pDeviceIDList->end();++iter)

{

printf("%ws\n",(\*iter));

//wchar\_t \*deviceID=new wchar\_t[50];

//wcscpy\_s(deviceID,50,wcsDeviceID);

//Segment 1:

// Square:5000\*5000

Set\_Jump\_Speed(0,jumpStepPeriod,jumpStepSize);

Set\_Mark\_Speed(0,markStepPeriod,markStepSize);

Jump\_Abs(0,-5000,-5000,100,length,delay);

Mark\_Abs(0,-5000,5000,100,length,delay);

Mark\_Abs(0,5000,5000,100,length,delay);

Mark\_Abs(0,5000,-5000,100,length,delay);

Mark\_Abs(0,-5000,-5000,100,length,delay);

//Segment 2:

// Square:7500\*7500

Set\_Jump\_Speed(indexOne,jumpStepPeriod,jumpStepSize);

Set\_Mark\_Speed(indexOne,markStepPeriod,markStepSize);

Jump\_Abs(indexOne,-7500,-7500,100,length,delay);

Mark\_Abs(indexOne,-7500,7500,100,length,delay);

Mark\_Abs(indexOne,7500,7500,100,length,delay);

Mark\_Abs(indexOne,7500,-7500,100,length,delay);

Mark\_Abs(indexOne,-7500,-7500,100,length,delay);

//Segment 3:

//Square:10000\*10000

Set\_Jump\_Speed(indexTwo,jumpStepPeriod,jumpStepSize);

Set\_Mark\_Speed(indexTwo,markStepPeriod,markStepSize);

Jump\_Abs(indexTwo,-10000,-10000,100,length,delay);

Mark\_Abs(indexTwo,-10000,10000,100,length,delay);

Mark\_Abs(indexTwo,10000,10000,100,length,delay);

Mark\_Abs(indexTwo,10000,-10000,100,length,delay);

Mark\_Abs(indexTwo,-10000,-10000,100,length,delay);

//Segment 4:

//Triangle

Set\_Jump\_Speed(indexThree,jumpStepPeriod,jumpStepSize);

Set\_Mark\_Speed(indexThree,markStepPeriod,markStepSize);

Jump\_Abs(indexThree,0,5000,100,length,delay);

Mark\_Abs(indexThree,5000,0,100,length,delay);

Mark\_Abs(indexThree,-5000,0,100,length,delay);

Mark\_Abs(indexThree,0,5000,100,length,delay);

//Segment 5:

//Triangle

Set\_Jump\_Speed(indexFour,jumpStepPeriod,jumpStepSize);//8

Set\_Mark\_Speed(indexFour,markStepPeriod,markStepSize);//8

Jump\_Abs(indexFour,0,20000,100,length,delay);//16

Mark\_Abs(indexFour,20000,0,100,length,delay);//16

Mark\_Abs(indexFour,-20000,0,100,length,delay);//16

Mark\_Abs(indexFour,0,20000,100,length,delay);//16

printf("Execute segments\n");

Sleep(1000);

int index=0;

while (index <=4)

{

nRet=ExecuteList(\*iter,index);

if (nRet==LanmarkControls::Success)

printf("Succeeded in sending data to the driver\n");

else

printf("Failed to send data to the driver, Error Code:%d\n",nRet);

index++;

Sleep(50);

}

}

Sleep(10000);

//return 0;

//getchar();

//test looping

//Create a list of looping segments

LoopingSegment \*p1=new LoopingSegment();

p1->SetIndex(0);

p1->SetLoopPoint(0);

LoopingSegment \*p2=new LoopingSegment();

p2->SetIndex(1);

p2->SetLoopPoint(0);

LoopingSegment \*p3=new LoopingSegment();

p3->SetIndex(2);

p3->SetLoopPoint(0);

LoopingSegment \*p4=new LoopingSegment();

p4->SetIndex(3);

p4->SetLoopPoint(0);

LoopingSegment \*p5=new LoopingSegment();

p5->SetIndex(4);

p5->SetLoopPoint(0);

//Link those segments

p1->SetNext(p2);

p2->SetNext(p3);

p3->SetNext(p4);

p4->SetNext(p5);

p5->SetNext(p1);

printf("\*\*\*\*\*\*\*\*\*Looping\*\*\*\*\*\*\*\*\*\n");

vector<wchar\_t\*>::iterator iterloop=pDeviceIDList->begin();

Sleep(1000);

nRet=LoopList(\*iterloop, p1);

if (nRet==LanmarkControls::Success)

printf("Succeeded in Looping data\n");

else

printf("Failed to Loop data, Error Code:%d\n",nRet);

//getchar();

vector<wchar\_t\*>::iterator iterAbort = pDeviceIDList->begin();

AbortList(\*iterAbort);

//wait for abort

::Sleep(1000);

//delete [] deviceID;

delete [] \*iterAbort;

return nRet;

}

int test4(vector<wchar\_t\*> \*pDeviceIDList)

{

int nRet = 0;

int lsRet = 0;

int xRet = 0;

unsigned int listSize = 0;

for (vector<wchar\_t\*>::iterator iter = pDeviceIDList->begin();

iter != pDeviceIDList->end();++iter)

{

printf("%ws\n",(\*iter));

//Segment 1:

//Square:5000\*5000

//Start of a job

xRet = Begin\_Job\_Event(0);//4

ListCmdErrorMsg(xRet, 2);

//Set the laser active state

xRet = Set\_Enable\_Laser(0, 1);//4

ListCmdErrorMsg(xRet, 16);

//Set the delay from laser enable to laser beam on

xRet = Set\_Laser\_Enable\_Delay(0, laserEnDelay);//4

ListCmdErrorMsg(xRet, 19);

//Set the delay from laser beam off to laser enable off

xRet = Set\_Laser\_Enable\_Timeout(0, laserEnTimeout);//4

ListCmdErrorMsg(xRet, 20);

//Turn the laser on for the duration period

xRet = Laser\_On(0, laserOnPeriod);//8

ListCmdErrorMsg(xRet, 8);

/\*

//Here can not use this again since you have already set the enbale delay.

//Turns laser on immediately

xRet = Laser\_Signal\_On(0);//4

ListCmdErrorMsg(xRet, 10);

\*/

//Set the mark speed, mark speed percentage, jump speed

xRet = Set\_Mark\_Speed(0, jumpStepPeriod, markStepSize);//8

ListCmdErrorMsg(xRet, 32);

xRet = Set\_Jump\_Speed(0, markStepPeriod, jumpStepSize);//8

ListCmdErrorMsg(xRet, 18);

//Jump to the start point

xRet = Jump\_Abs(0, -5000, -5000, 100, length, delay);//16

ListCmdErrorMsg(xRet,7);

//Start marking

xRet = Mark\_Abs(0, -5000, 5000, 100, length, delay);//16

ListCmdErrorMsg(xRet, 12);

xRet = Mark\_Abs(0, 5000, 5000, 100, length, delay);//16

ListCmdErrorMsg(xRet, 12);

xRet = Mark\_Abs(0, 5000, -5000, 100, length, delay);//16

ListCmdErrorMsg(xRet, 12);

xRet = Mark\_Abs(0, -5000, -5000, 100, length, delay);//16

ListCmdErrorMsg(xRet, 12);

//Set the laser inactive state

xRet = Set\_Enable\_Laser(0, 0);//4

ListCmdErrorMsg(xRet, 16);

//End of the job

xRet = End\_Job\_Event(0);//4

ListCmdErrorMsg(xRet, 5);

/\* //Start of a job

xRet = Begin\_Job\_Event(0);//4

ListCmdErrorMsg(xRet, 2);

//Try to test turn the laser on for the duration period

xRet = Laser\_On(0, laserOnDelay);//8

ListCmdErrorMsg(xRet, 8);

//Turns laser on immediately

xRet = Laser\_Signal\_On(0);//4

ListCmdErrorMsg(xRet, 10);

//Set the laser active state

xRet = Set\_Enable\_Laser(0, 1);//4

ListCmdErrorMsg(xRet, 16);

//Set the delay from laser enable to laser beam on

xRet = Set\_Laser\_Enable\_Delay(0, laserEnDelay);//4

ListCmdErrorMsg(xRet, 19);

//Set laser pulse width and pulse period, percentages, standby pulse, power, power percentage,

//and power change delay

xRet = Set\_Laser\_Pulse(0, laserNumSetPulse, laserPulseWidth, laserPulsePeriod);//12

ListCmdErrorMsg(xRet, 29);

xRet = Set\_PM\_PulsePeriodPct(0, pulsePeriodPct, laserSignalIDP);//8

ListCmdErrorMsg(xRet, 42);

xRet = Set\_PM\_PulseWidthPct(0, pulseWidthPct, laserSignalIDW);//8

ListCmdErrorMsg(xRet, 43);

xRet = Set\_Laser\_Standby(0, laserNumSetStandby, laserStandbyPulseWidth, laserStandbyPulsePeriod);//12

ListCmdErrorMsg(xRet, 30);

xRet = Set\_Laser\_Power(0, laserPower);//4

ListCmdErrorMsg(xRet,27);

xRet = Set\_PM\_LaserPowerPct(0, laserPowerPct);//8

ListCmdErrorMsg(xRet, 40);

xRet = Set\_Laser\_Power\_Change\_Delay(0, pwrChgDelay);//8

ListCmdErrorMsg(xRet, 28);

xRet = Set\_Laser\_Timing(0, tickUnit);//4

ListCmdErrorMsg(xRet, 31);

//88

//Set the mark speed, mark speed percentage, jump speed

xRet = Set\_Mark\_Speed(0, stepPeriod, stepSize);//8

ListCmdErrorMsg(xRet, 32);

xRet = Set\_PM\_MarkSpeedPct(0, markSpeedPct);//8

ListCmdErrorMsg(xRet, 41);

xRet = Set\_Jump\_Speed(0, stepPeriod, stepSize);//8

ListCmdErrorMsg(xRet, 18);

//Marking Field Orientation

// Set the basic orientation of the coordinate.

xRet = Set\_PM\_FieldOrientation(0, orientation);//4

ListCmdErrorMsg(xRet, 39);

// Set the field offset to be applied to all vectors.

xRet = Set\_PM\_FieldOffset(0, xOffset, yOffset, zOffset);//8

ListCmdErrorMsg(xRet, 38);

//124

//MOTF

// Set the MOTF operational mode

xRet = Set\_MOTF\_Mode(0, MOTFMode);//4

ListCmdErrorMsg(xRet, 37);

// Set the MOTF operational state, first enable it and then disable it

xRet = Enable\_MOTF(0,1);//4

ListCmdErrorMsg(xRet,3);

xRet = Enable\_MOTF(0,0);//4

ListCmdErrorMsg(xRet,3);

//Enable wobble and then disable it

xRet = Enable\_Wobble(0, 1);//4

ListCmdErrorMsg(xRet, 4);

xRet = Enable\_Wobble(0, 0);//4

ListCmdErrorMsg(xRet, 4);

//Jump to the start point

xRet = Jump\_Abs(0, -5000, -5000, 100, length, delay);//16

ListCmdErrorMsg(xRet,7);

//Delay all operations for the duration provided

xRet = Long\_Delay(0, opDelay);//8

ListCmdErrorMsg(xRet, 11);

//Start marking

xRet = Mark\_Abs(0, -5000, 5000, 100, length, delay);//16

ListCmdErrorMsg(xRet, 12);

xRet = Mark\_Abs(0, 5000, 5000, 100, length, delay);//16

ListCmdErrorMsg(xRet, 12);

xRet = Mark\_Abs(0, 5000, -5000, 100, length, delay);//16

ListCmdErrorMsg(xRet, 12);

xRet = Mark\_Abs(0, -5000, -5000, 100, length, delay);//16

ListCmdErrorMsg(xRet, 12);

xRet = Z\_Abs(0, zCoordinate, delay);//8

ListCmdErrorMsg(xRet, 50);

//Set the delay from laser beam off to laser enable off

xRet = Set\_Laser\_Enable\_Timeout(0, laserEnTimeout);//4

ListCmdErrorMsg(xRet, 20);

//Set the laser to inactive state

xRet = Set\_Enable\_Laser(indexOne, 0);//4

ListCmdErrorMsg(xRet, 16);

//Turn off the laser immediately

xRet = Laser\_Signal\_Off(0);//4

ListCmdErrorMsg(xRet, 9);

//End of the job

xRet = End\_Job\_Event(0);//4

ListCmdErrorMsg(xRet, 5);

\*/

//Segment 2:

//Triangle

//OriginalListSize=80

//PackedListSize=112

/\*xRet = Begin\_Job\_Event(indexOne);

ListCmdErrorMsg(xRet,2);

xRet = Laser\_On(laserOnDelay);

ListCmdErrorMsg(xRet,8);

xRet = Laser\_Signal\_On(indexOne);

ListCmdErrorMsg(xRet,10);

xRet = Set\_Enable\_Laser(indexOne,1);

ListCmdErrorMsg(xRet, 16);

xRet = Set\_PM\_MarkSpeedPct(indexOne,markSpeedPct);

ListCmdErrorMsg(xRet,41);

xRet = Set\_Laser\_Enable\_Delay(indexOne,laserEnDelay);

ListCmdErrorMsg(xRet,19);

xRet = Set\_PM\_LaserPowerPct(indexOne,laserPowerPct);

ListCmdErrorMsg(xRet,40);

xRet = Set\_PM\_PulsePeriodPct(indexOne,pulsePeriodPct,laserSignalID);

ListCmdErrorMsg(xRet,42);

xRet = Set\_PM\_PulseWidthPct(indexOne,pulseWidthPct,laserSignalIDW);

ListCmdErrorMsg(xRet,43);

xRet = Enable\_MOTF(indexOne,1);

ListCmdErrorMsg(xRet,3);

xRet = Enable\_MOTF(indexOne,0);

ListCmdErrorMsg(xRet,3);

xRet = Enable\_Wobble(indexOne,1);

ListCmdErrorMsg(xRet,4);

xRet = Enable\_Wobble(indexOne,0);

ListCmdErrorMsg(xRet,4);

xRet = Set\_Jump\_Speed(indexOne,stepPeriod,stepSize);

ListCmdErrorMsg(xRet,18);

xRet = Set\_Mark\_Speed(indexOne,stepPeriod,stepSize);

ListCmdErrorMsg(xRet,32);

xRet = Set\_Laser\_Power\_Change\_Delay(indexOne,pwrChgDelay);

ListCmdErrorMsg(xRet,28);

xRet = Jump\_Abs(indexOne,0,5000,100,length,delay);

ListCmdErrorMsg(xRet,7);

xRet = Long\_Delay(opDelay);

listCmdErrorMsg(xRet,11);

xRet = Mark\_Abs(indexOne,5000,0,100,length,delay);

ListCmdErrorMsg(xRet,12);

xRet = Set\_Laser\_Power(indexOne, laserPower);

ListCmdErrorMsg(xRet,27);

xRet = Mark\_Abs(indexOne,-5000,0,100,length,delay);

ListCmdErrorMsg(xRet,12);

xRet = Set\_Laser\_Enable\_Timeout(indexOne,laserEnTimeout);

ListCmdErrorMsg(xRet,20);

xRet = Set\_Enable\_Laser(indexOne,0);

ListCmdErrorMsg(xRet, 16);

xRet = Laser\_Signal\_Off(indexOne);

ListCmdErrorMsg(xRet,9);

xRet = End\_Job\_Event(indexOne);

ListCmdErrorMsg(xRet,5);\*/

printf("Execute segments\n");

Sleep(1000);

DWORD index = 0;

//unsigned int index1 = 0;

unsigned int \*lsPtr = NULL;

lsPtr = &listSize;

//while (index <= 1)

while (index <= 0)

{

nRet = ExecuteList(\*iter, index);

//lsRet = GetListSize(index, lsPtr);

//lsRet = MarkerLibraryRT::GetSegmentSize(index, lsPtr);

if (nRet == LanmarkControls::Success)

{

printf("Succeeded in sending data to the driver\n");

/\*if (lsRet = LanmarkControls::Success)

printf("Succeed in getting list size.");

else

printf("Failed in getting list size.");\*/

}

else

printf("Failed to send data to the driver, Error Code:%d\n",nRet);

index++;

//index1++;

Sleep(50);

}

}

vector<wchar\_t\*>::iterator iterAbort = pDeviceIDList->begin();

AbortList(\*iterAbort);

//wait for abort

::Sleep(1000);

int dRet = 0;

DisconnectLECDevice(\*iterAbort);

if (dRet == LanmarkControls::Success)

printf("Succeeded in disconnecting LEC device.\n");

else

printf("Failed in disconnecting LEC device, Error Code:%d\n",dRet);

//delete [] deviceID;

delete [] \*iterAbort;

return nRet;

}

int TestControls(vector<wchar\_t\*>& vecDevIDs)

{

const int EVENT\_NAME\_LEN=100;

printf("Test Controls:\n");

for (vector<wchar\_t\*>::iterator iter=vecDevIDs.begin();

iter!=vecDevIDs.end();++iter)

{

printf("Device ID:%ls\n",\*iter);

//Print out some of the registers

{

//reg MecFPGAReg\_CmdFifoPeek

MecFPGAReg reg(MecFPGAReg\_CmdFifoPeek);

unsigned int val;

if (LanmarkControls::Success==ReadRegister(\*iter,(unsigned int)reg,&val))

printf(" Register MecFPGAReg\_CmdFifoPeek:%x\n",val);

//reg MecFPGAReg\_Status

reg=MecFPGAReg\_Status;

if (LanmarkControls::Success==ReadRegister(\*iter,(unsigned int)reg,&val))

printf(" Register MecFPGAReg\_Status:%x\n",val);

//

}

/\*

//Message Event Name

wchar\_t wcsEventName[EVENT\_NAME\_LEN];

if (LanmarkControls::Success==GetJobMsgName(\*iter,wcsEventName,EVENT\_NAME\_LEN))

printf\_s("\nMessage Event Name:%ws\n",wcsEventName,EVENT\_NAME\_LEN);

//Exception Event Name

if (LanmarkControls::Success==GetExceptionMsgName(\*iter,wcsEventName,EVENT\_NAME\_LEN))

printf\_s("Exception Event Name:%ws\n",wcsEventName,EVENT\_NAME\_LEN);

//Driver status

printf("\n");

\*/

}

return 0;

}

const char \*errorNames[] =

{

"Success",

"JobIdle",

"Busy",

"NoJob",

"InControl",

"NotInControl",

"LicenseUnavailable",

"LicenseAccessDenied",

"BadCommand",

"BadArg",

"ArgOutOfRange", // 10

"UnknownTimeZone",

"TimeZoneError",

"BadConversion",

"RegistryError",

"TimeZoneFileError",

"ResetInterlock",

"ListNotOpen",

"ListAlreadyOpen",

"BadData",

"APIException", // 20

"JobAborting",

"FPGALoadFail",

"JobManagerInitFail",

"LaserLoadFail",

"LensLoadFail",

"PMLoadFail",

"MotionLoadFail",

"HostManagerInitFail",

"InvalidIPAddress",

"DataUnknown", // 30

"BadChecksum",

"NetworkShareNotConnected",

"NetworkConnectFail",

"UnknownNetworkError",

"APICommandTimeout",

"ExternalProcessFail",

"DLLLoadFail",

"NoAdapter",

"AddIPAddressFailure",

"BadAPIResponse", // 40

"CannotCreateSocket",

"CannotConnectSocket",

"CannotGetFPGABufInfo",

"CannotGetFPGABuf",

"CannotWriteFPGABuf",

"FPGAException",

"FTPConnectionError",

"FileAlreadyExists",

"UnknownOS",

"SocketException", // 50

"ProcessTimeout",

"DeviceNotFound",

"LoginInProgress",

"APIClientInControl",

"StreamClientInControl",

"CannotConnectToAPI",

"ReadFail",

"StreamBufferFull",

"NoConfigRecord",

"OperationCanceled", // 60

"NoData",

"InitializationError",

"FailToCreateServiceThread",

"CannotOpenDevice",

"SegmentFull",

"MarkerLibraryNotInitialized",

"RingBufferNotInitialized",

"AccessDenied",

"RequiresUACElevation",

"NotAllowed", // 70

"NoLaserConfig",

"NoLensConfig",

"OutOfMemory",

"LensTableNotFound",

"NoFilesFound", // 100

"NoDrive",

"SystemOutOfMemory",

"TooManyObjects",

"NoObject",

"JobException",

"NotInHostControl",

"WrongHostType",

"ErrorJobBusy",

"NoActiveJob",

"ErrorSoftware", // 110

"LoadFail",

"NoObjects",

"WriteFail",

"JobFileFormat",

"FileException",

"UnknownObject",

"UnknownType",

"NotSupported",

"NotAvailable",

"FPGADataFail", // 120

"FileNotFound",

"FileCreationError",

"WriteFileFail",

"PathNotFound",

"NotInCacheMode",

"NotWaitingForStartMark",

"MotionNotHomed",

"No3DModel",

"ProjectionError",

"NoProperties", // 200

"ObjectException",

"JobAbort",

"NoFontResource",

"NoOverride",

"ExternalEnableDenied",

"CannotCreatePort",

"CannotOpenPort",

"PortNotOpen",

"PortTimeout",

"WrongPortNumber", // 210

"WrongObjectType",

"AxisNotConfigured",

"TextBufferOverrun",

"InvBarcodeStringValue",

"InvBarcodeStringLength",

"InvBarcodeNarrowWidth",

"InvBarcodeWidthReduce",

"InvBarcodeECC",

"BarcodeOutOfMemory",

"BarcodeUnknownError", // 220

"BarCodeException",

"NoVectors",

"BadMotionResponse",

"MotionDriverNotFound",

"MotionDeviceNotFound",

"EncoderNotFound",

"InvStringValue",

"MotionControllerNotFound",

"MotorNotProvisioned",

"RuntimeMotionError", // 230

"ObjectOutOfBounds",

"InvVersion",

"ConfigBadFormat", // 300

"LensConfigBadFormat",

"LensTableBadFormat",

"LaserConfigBadFormat",

"MotionConfigBadFormat",

"AdminConfigBadFormat",

"ControllerConfigBadFormat",

"GlobalConfigBadFormat",

};

int \_tmain(int argc, \_TCHAR\* argv[])

{

LoadMarkerLibrary();

InitializeMarkerLibrary(NULL,NULL,NULL,DeviceDetectionCallback,NULL);

//getchar();

int nRet=InitializeBufferManager(5,2100);

vector<wchar\_t\*> vecTemp,vecDevIDs;

//The following two lines are for testing the producing of error code.

//int sample = 0;

//printf("Failed to store the Application\_Event command. Error Code:%s\n",errorNames[sample]);

bool b=true;

printf("Please choose the device that you want to test:(input the sequence number)\n");

printf("1: LAN\n");

printf("2: PCI\n");

scanf("%i",&seqNum);

printf("Please input the Device ID (mac address or device path)\n");

scanf("%ls",wcsDeviceID,sizeof(wcsDeviceID));

if(seqNum==1){

if (wcslen(wcsDeviceID)!=MAC\_ADDR\_LEN)

{

wcscpy(wcsDeviceID,wcsDefaultLANDeviceID);

printf("Use the default LAN Device ID:%ls\n",wcsDeviceID);

}

}

else if(seqNum==2)

{

if (wcslen(wcsDeviceID)!=PCI\_ID\_LEN)

{

wcscpy(wcsDeviceID,wcsDefaultPCIDeviceID);

printf("Use the default PCI Device path:%ls\n",wcsDeviceID);

}

}

if (wcslen(wcsDeviceID)==0)

//Test all devices. Wait for enumerating all devices.

::Sleep(1000);

while (b)

{

GetLECDeviceIDList(&vecTemp);

for (vector<wchar\_t\*>::iterator iter=vecTemp.begin();

iter!=vecTemp.end();++iter)

{

/\*if(seqNum==1)

{

if (wcslen(wcsDeviceID)==0 || wcscmp((\*iter),wcsDeviceID)==0)

{

//Our device has already been found

vecDevIDs.push\_back((\*iter));

b=false;

}

}

else if(seqNum==2)

{

//if (wcslen(wcsDeviceID)==0 || wcscmp((\*iter),wcsDeviceID)==0)

//{

// b=false;

// vecDevIDs.push\_back((\*iter));

}

wcscpy(wcsDeviceID, \*iter);

b=false;

vecDevIDs.push\_back((\*iter));

break;

}\*/

wcscpy(wcsDeviceID, \*iter);

b=false;

vecDevIDs.push\_back((\*iter));

break;

}

}

//test1(pDevices);

//test2(&vecDevIDs, pMarkerLibrary);

test3(&vecDevIDs);

//test4(&vecDevIDs);

//unsigned int ls=0;

//GetListSize(0,&ls);

//TestControls(vecDevIDs);

vecDevIDs.clear();

for (vector<wchar\_t\*>::iterator iter=vecTemp.begin();

iter!=vecTemp.end();++iter)

delete \*iter;

vecTemp.clear();

ReleaseLibrary();

//getchar();

//getchar();

//exit(0);

printf("End of the program:\n");

//To do : seems some threads are not terminated correctly

return 0;

}