CPC - API guide

Aurea Technology

Created on : Octorber, 2021

Last updated : July, 2022

Contents

Introduction

The Compact Photons Counter (CPC) can be controlled and used with a computer through the USB connection. To do that we provide an API (Application Programming Interface) based on a library developed in C/C++ language for all operating systems (Windows, MacOS, Linux).

Through it, you can develop your own CPC control interface. In order to help you, we provide the library files, and some examples in C++ and Python for all operating systems.

CHAPTER 1

Software Installation Guide

The following section describes how to install the CPC software on Windows, MacOS and Linux operating systems.

1.1 Windows

1.1.1 Operating Systems Requirements

- Windows 7 or higher
- Application and examples are working on 32bit and 64bit systems.

1.1.2 Installation Step

1. Run the setup file locate in provided directory.
2. Connect CPC device to your computer with the USB cable.
3. Start Aurea-CPC application or start Aurea-Launcher and then click on your device to use the software.

1.2 MacOS

- Aurea-Launcher Installation :
  1. Double click on Aurea-Launcher.dmg file.
  2. Drag Aurea-Launcher in the Applications folder.
- Aurea-CPC Installation :
  1. Double click on Aurea-CPC.dmg file.
  2. Drag Aurea-CPC in the Applications folder
  3. Connect CPC device to your computer with the USB cable.
  4. Launch Aurea-CPC or Aurea-Launcher application by clicking on it.

1.3 Linux

- Aurea-Launcher Installation :
  1. Unzip Aurea-Launcher-package.zip
  2. Go to Aurea-Launcher-package/Aurea-Launcher and double-click on Aurea-Launcher-Installer.
  3. Follow the installer instructions and make sure to install all Aurea Technology software in the same directory.
- Aurea-CPC Installation :
  1. Unzip Aurea-CPC-package.zip
  2. Go to Aurea-CPC-package/Aurea-CPC and double-click on Aurea-CPC-Installer.
  3. Follow the installer instructions and make sure to install all Aurea Technology software in the same directory.
  4. Connect CPC device to your computer with the USB cable.
  5. Launch Aurea-CPC or Aurea-Launcher application by executing the following command in the installation directory.

```
./Aurea-Launcher.sh
```

```
./Aurea-CPC.sh
```

CHAPTER 2

Custom Application

The following section guides you through the developement of your own application.

2.1 Windows

2.1.1 Requirements

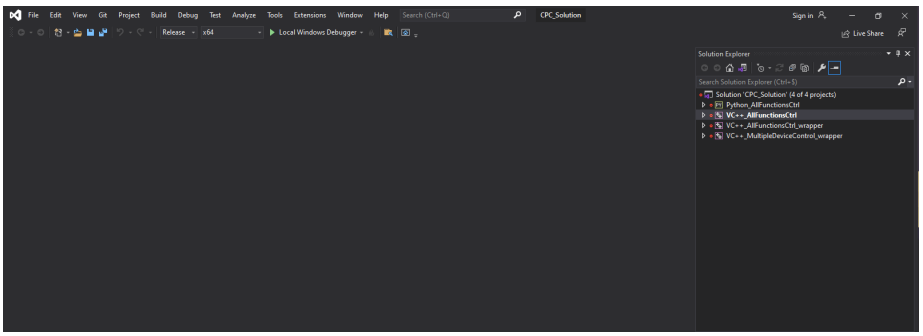
Software : Visual Studio (2019)

Visual Studio extensions :

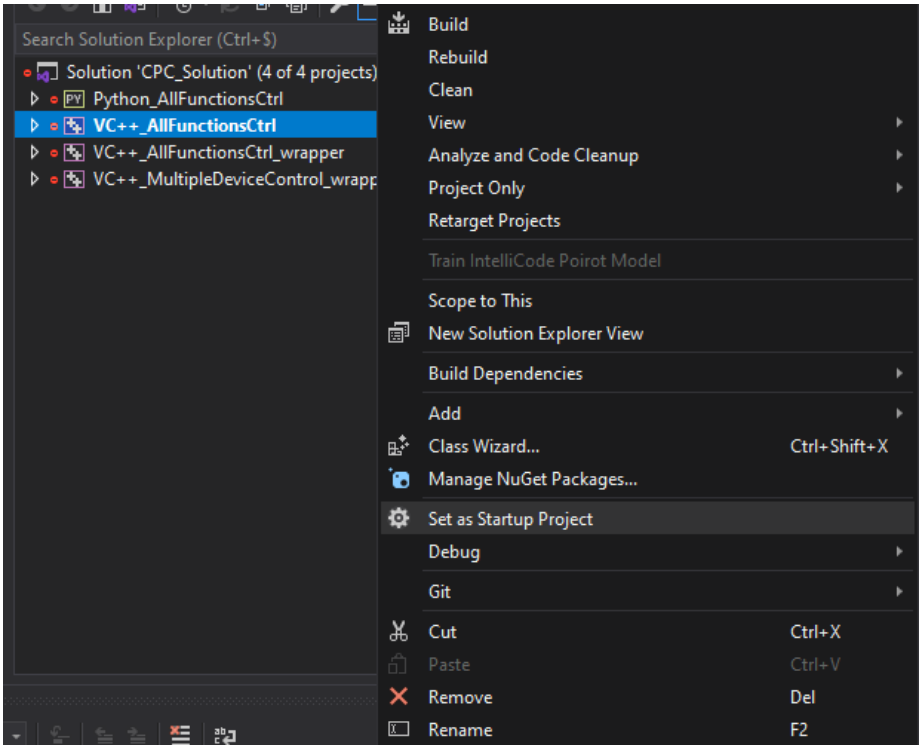
- Desktop Development C++
- Development Python

2.1.2 Create C++ application using the existing Visual Studio project

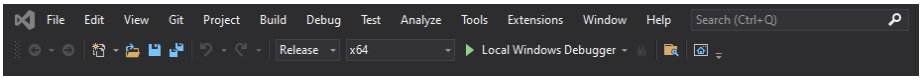
- Locate the CPC-API folder and go to “CPC-API/Applications/”.
- Open “CPC_Solution.sln”.



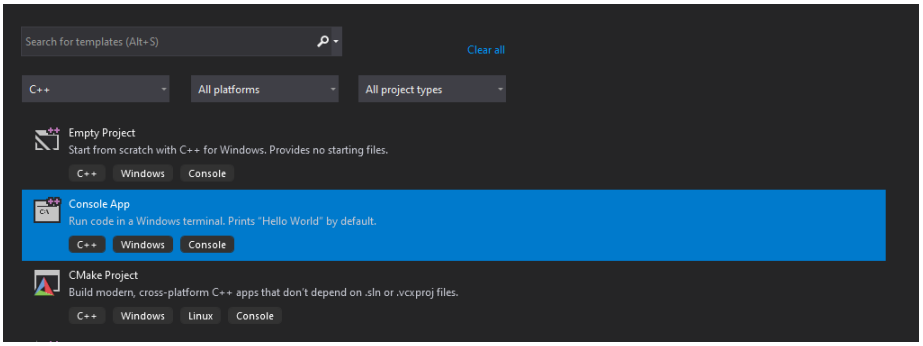
- Three differents C++ programs have been developed to help you, to change the selected applications right click on the desired example and select “Set as Startup Project”.



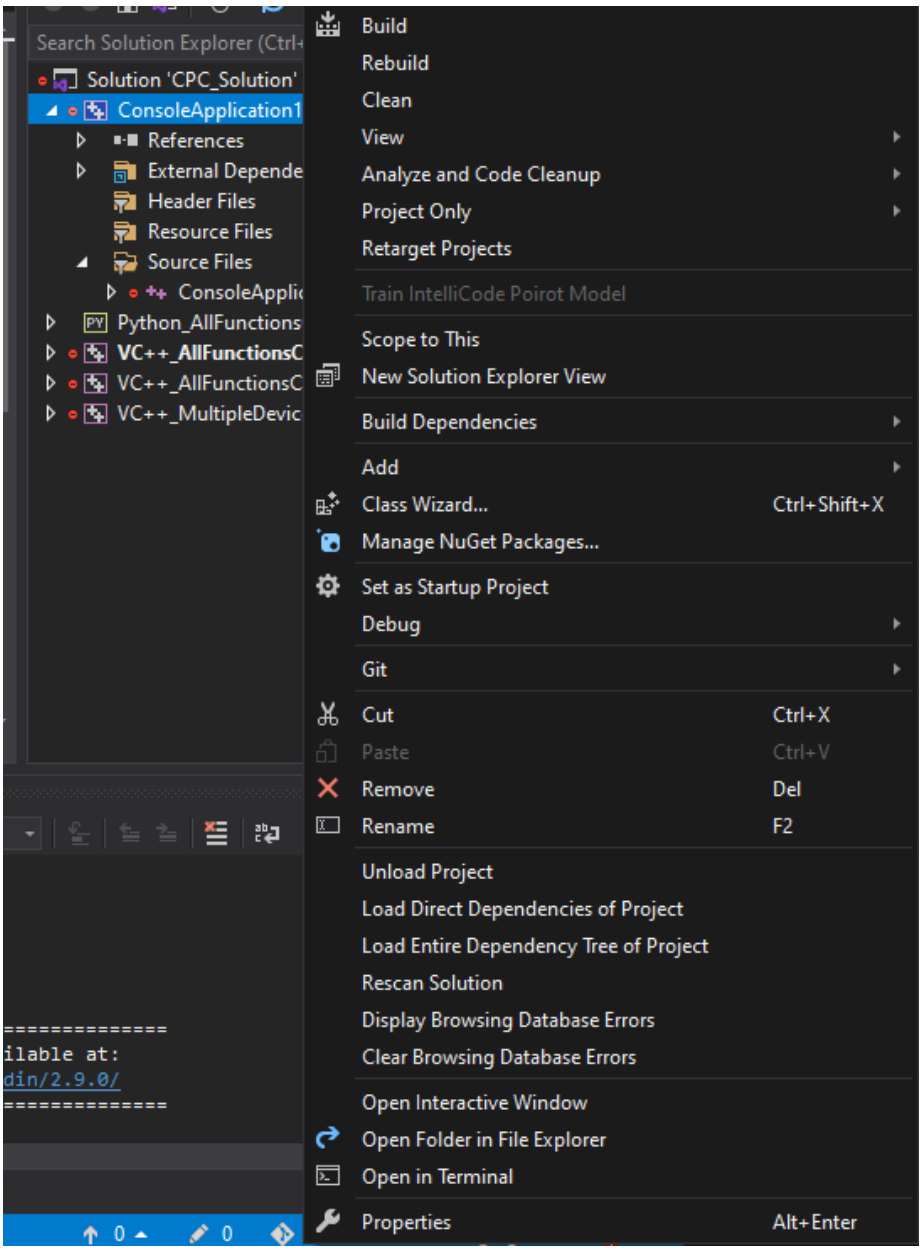
- To test those examples, make sure you have selected “Release” and “x64”, and click on “Run”.



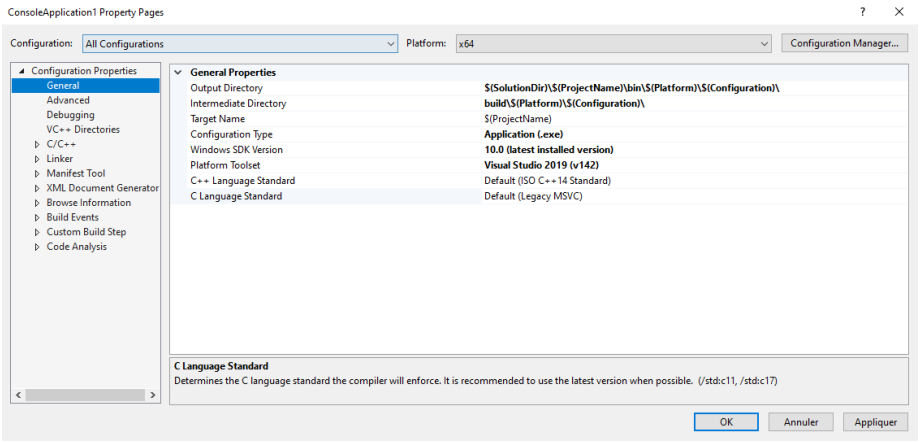
- To create your own application click on “File” > “New” > “Project...” Then select “Console App”, choose an application name, select “Add to solution” and click on “Create”.



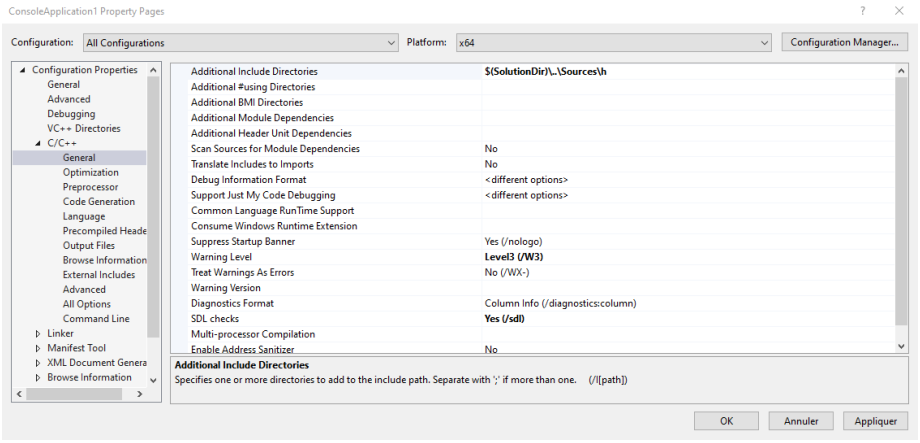
- Now, you need to configure your project, right click on it and then click on “Properties”.



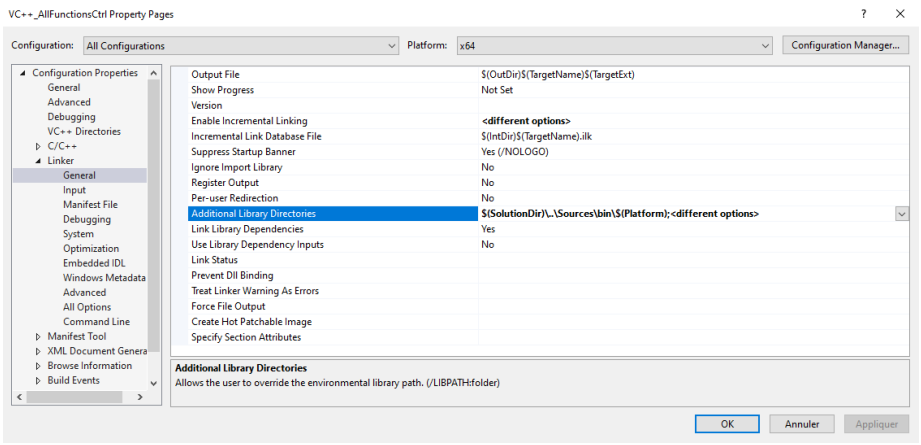
- Then make sure Configuration section is set to “All Configurations” and Platform is set to “x64”.
- Go to “Configuration Properties” > “General” > “General Properties” and set the “Output Directory” to “$(SolutionDir)\$(ProjectName)\bin\$(Platform)\$(Configuration)\”. Then set the “Intermediate Directory” to “build\$(Platform)\$(Configuration)\”.



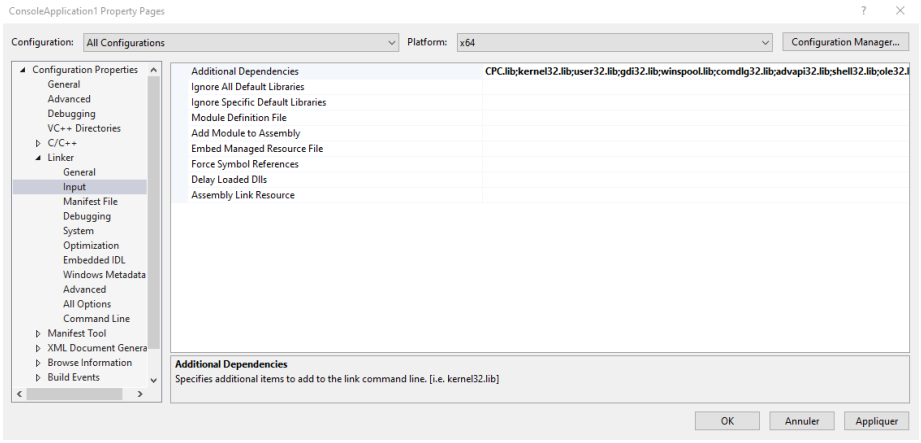
- Go to “Configuration Properties” > “C/C++” > “General” and set the “Additional Include Directories” to “$(SolutionDir)\..\Sources\h”.



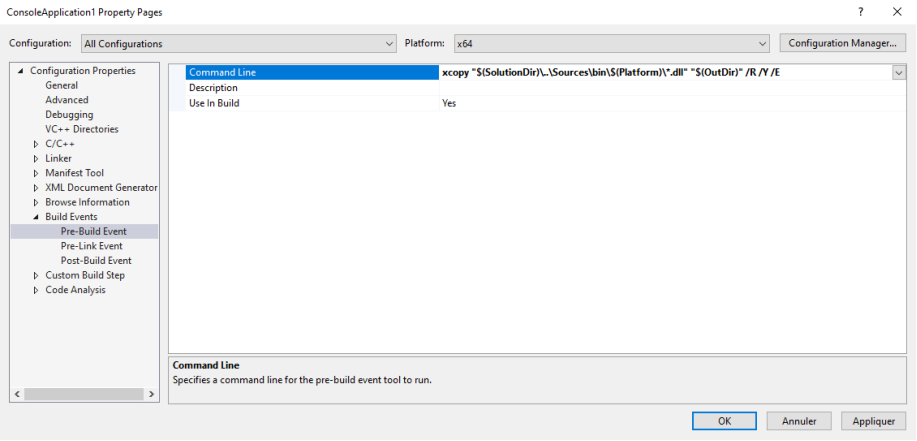
- Go to “Configuration Properties” > “Linker” > “General” and set the “Additional Library Directories” to “$(SolutionDir)\..\Sources\bin\$(Platform)”.



- Go to “Configuration Properties” > “Linker” > “Input” and set the “Additional Dependencies” to “CPC.lib”.



- Finally, in the goal to locally run the application, add the copy of the library on the output folder. Go to “Configuration Properties” > “Build Events” > “Pre-Build Event” and set the “Command Line” to “xcopy “$(SolutionDir)\..\Sources\bin\$(Platform)*.dll” “$(OutDir)” /R /Y /E”.

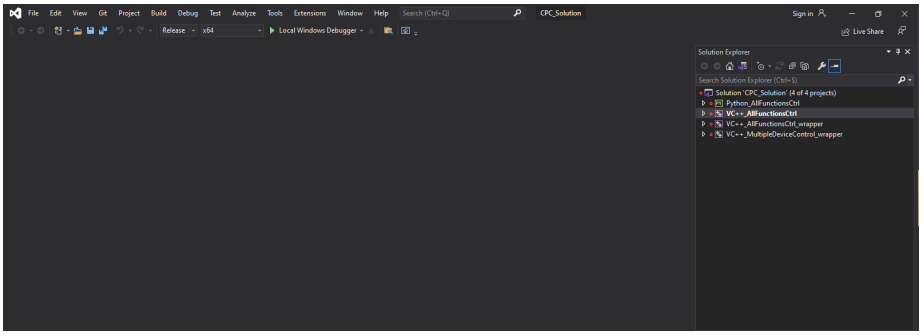


Note: If you use windows 7, the xcopy command may give you an error. To solve this issue, add this path to your environement path : "C:WindowsSystem32". You also can remove the xcopy command and manually copy the CPC.dll file in the same directory as your application executable file.

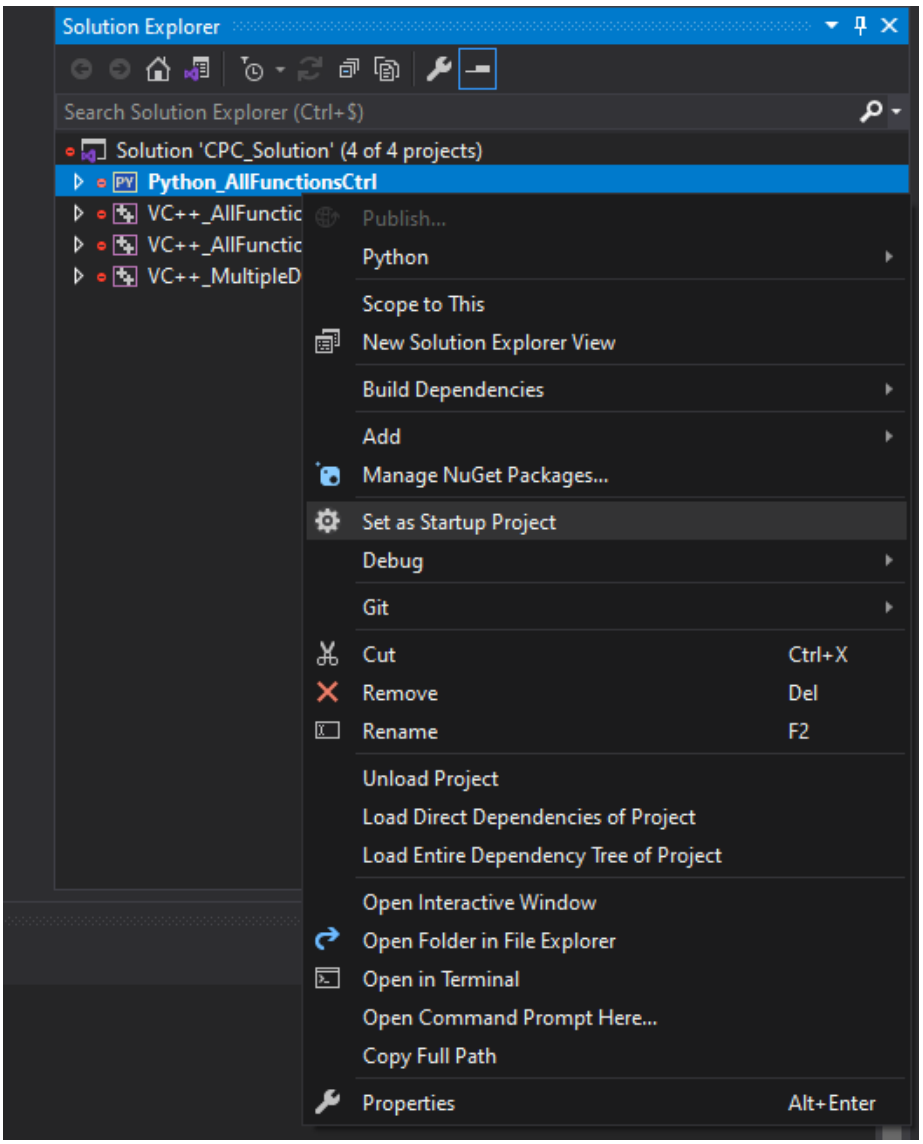
Your C++ application is now ready to use. Please refer to Section *Code Examples* to access basic code.

2.1.3 Create Python application using the existing Visual Studio project

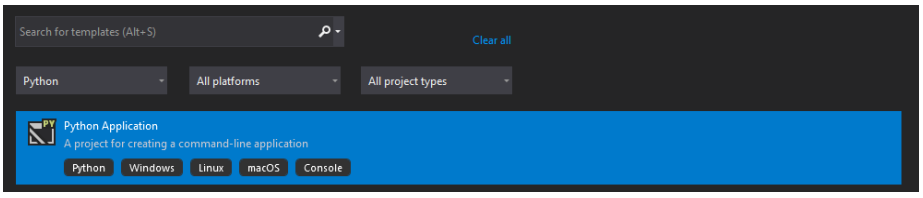
- Locate the CPC-API folder and go to "CPC-API/Applications/".
- Open "CPC_Solution.sln".



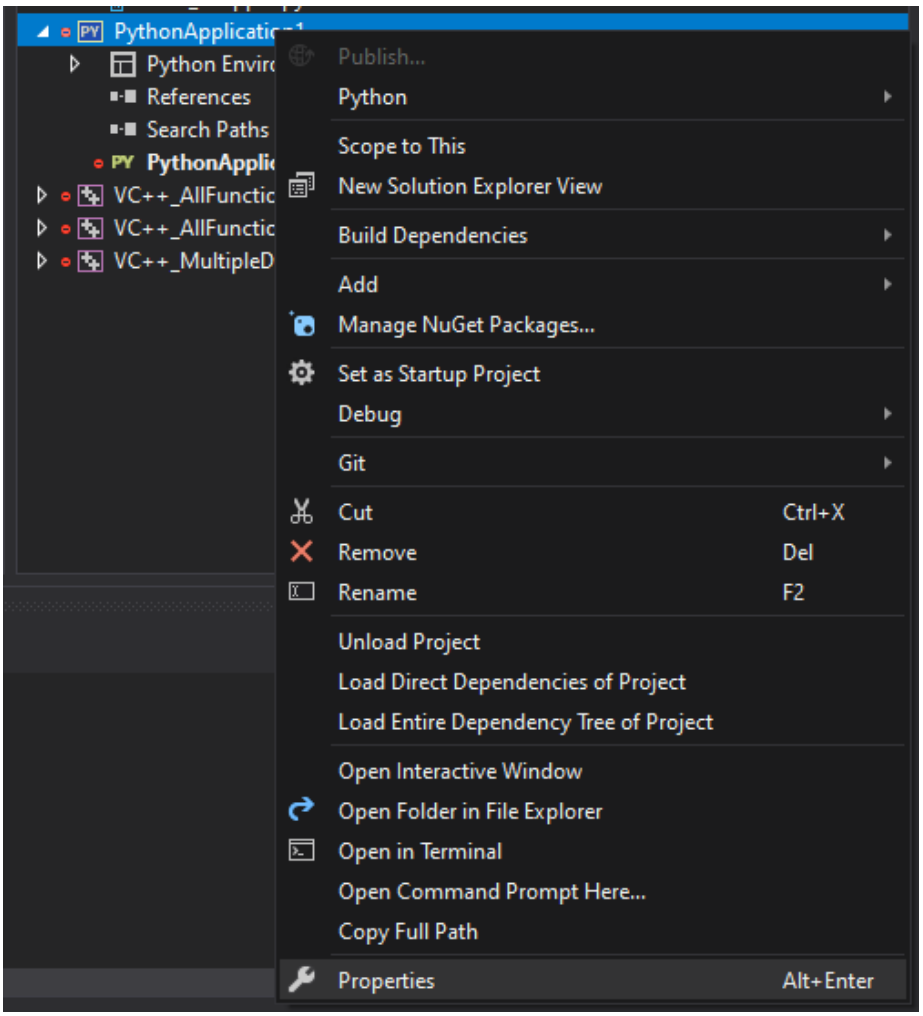
- One Python program has been developed to help you, to change the selected applications right click on the desired example and select "Set as Startup Project".



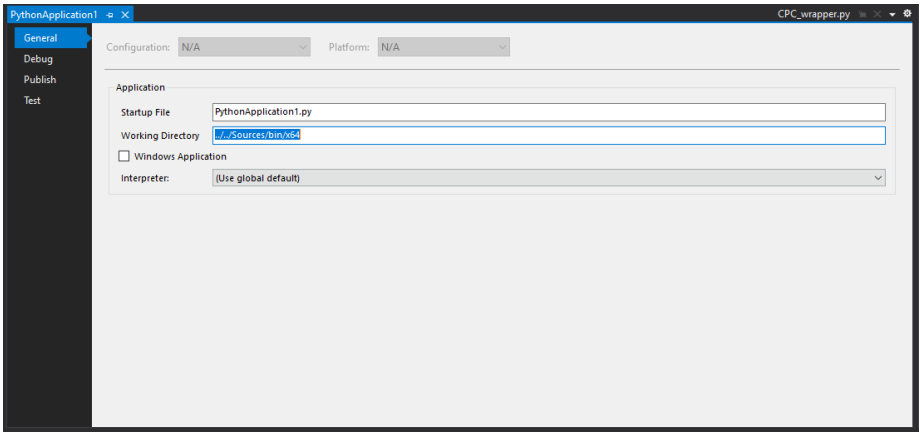
- To create your own application click on “File” > “New” > “Project...” then select “Python Application”, choose an application name, select “Add to solution” and click on “Create”.



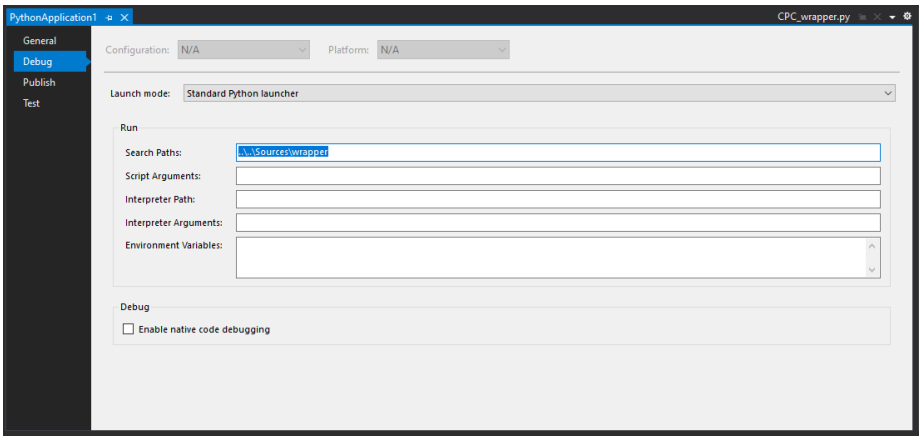
- Now, you need to configure your project, right click on it and then click on “Properties”.



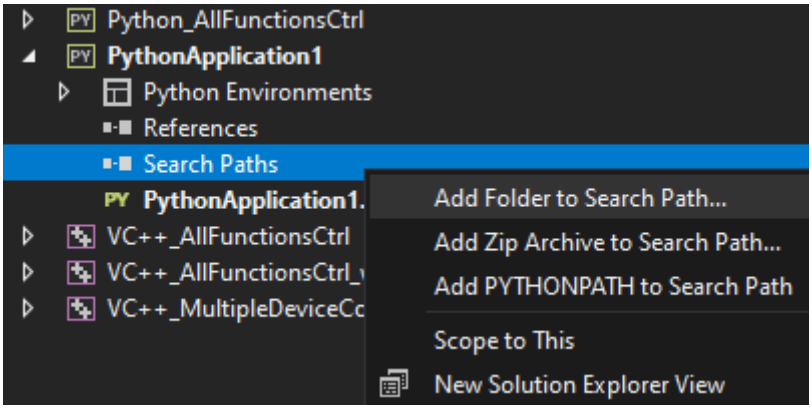
- In order to not locally copy the library, you can adjust the "Working Directory" with the library path "../../Sources/bin/x64"



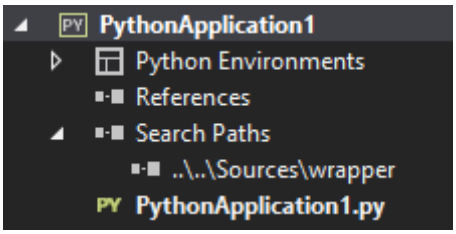
- In “Debug” set the “Search Paths” to “..\..\Sources\wrapper” allowing to specify the wrapper package location



- Finally, right click on “Search Paths” and select “Add Folder to Search Path…”.



- Then locate and select “CPC-API/Sources/wrapper”.



Your Python application is now ready to use. Please refer to Section *Code Examples* to access basic code.

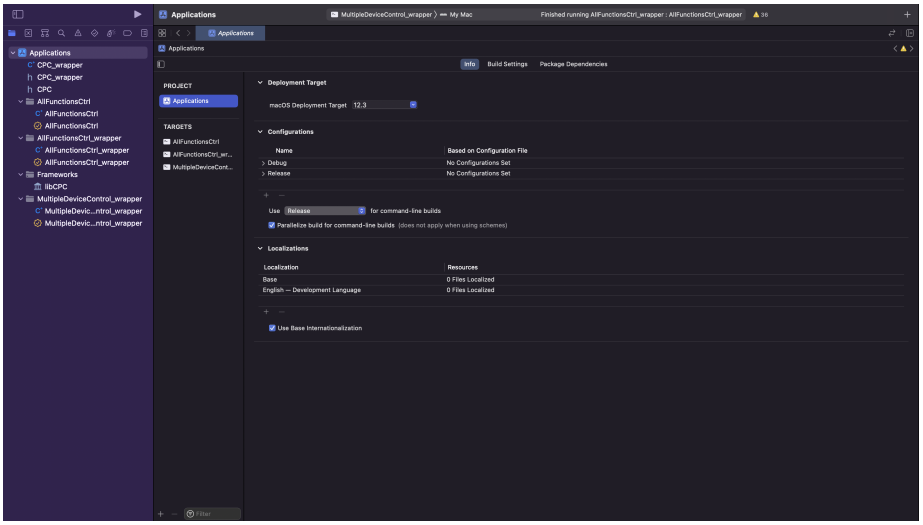
2.2 MacOS

2.2.1 Requirements

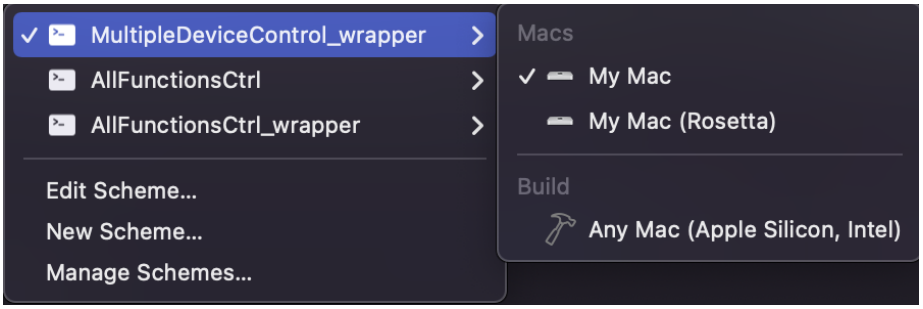
Software : XCode.

2.2.2 Create C++ application using the existing Xcode project

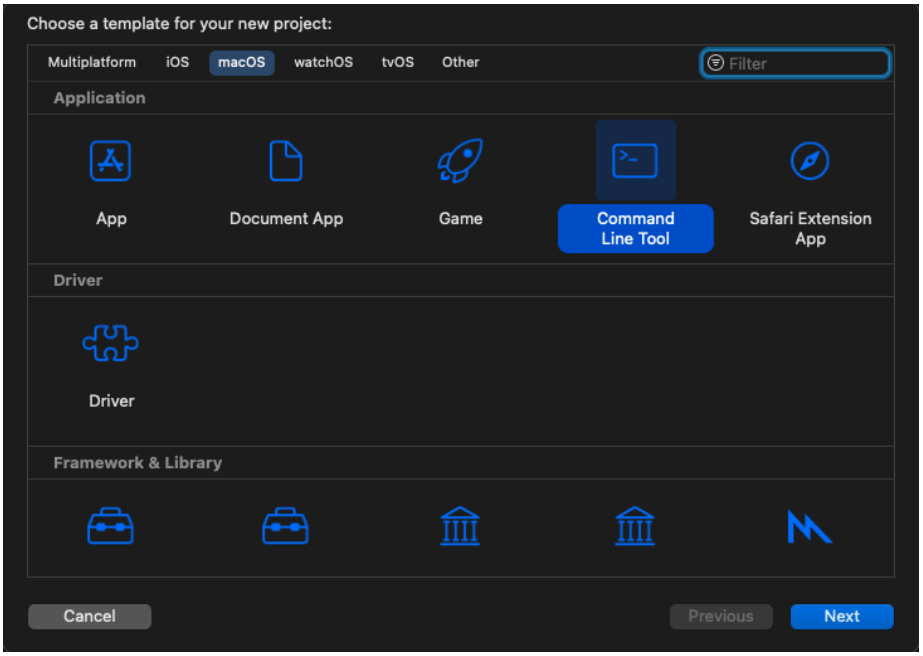
- Locate the CPC-API folder and go to “CPC-API/Applications/C++/”.
- Open “Applications.xcodeproj”.



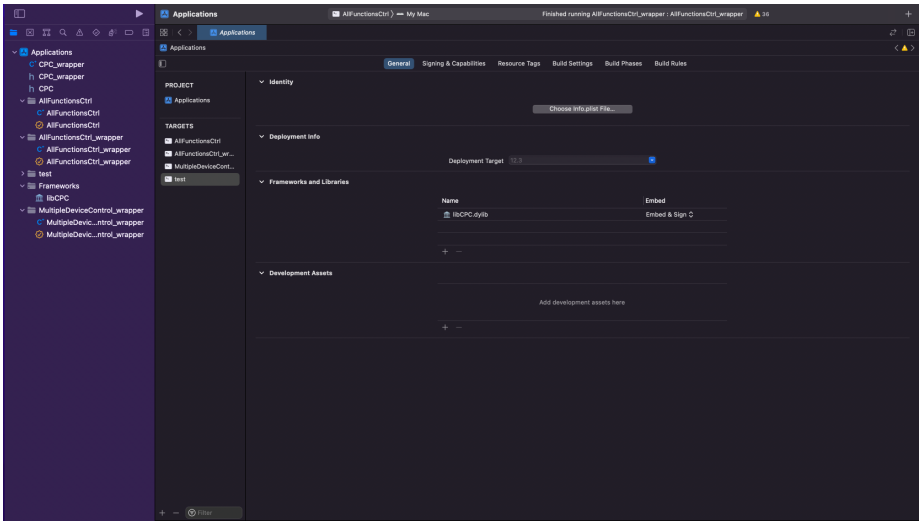
- Three differents programs have been developed to help you, to change the selected applications click on the list at the top.



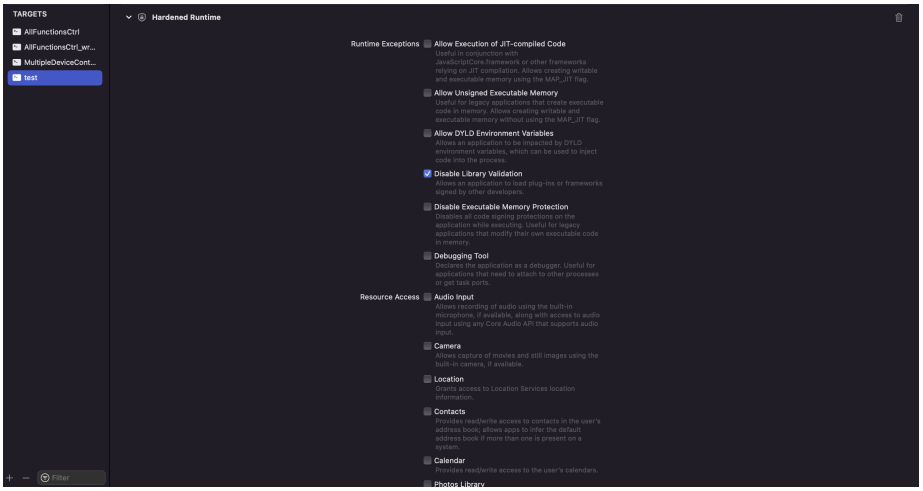
- To create your own application click on “File” > “New” > “Target...”” Then select Command Line Tools, choose an application name and click on “Finish”.



- Now, you need to configure your target, click on it, click on “General”, on “Frameworks and Libraries” and click “+”.
- Click on “Add Ohter…” and select “Add Files”. Then add libCPC.dylib that is locate in “CPC-API/Sources/bin/”.



- To avoid library issue, click on your target, click on “Signing & Capabilities”, check box “Disable Library Validation”.



Your C++ application is now ready to use. Please refer to Section *Code Examples* to access basic code.

2.2.3 Create Python application

- Locate the CPC-API folder and go to "CPC-API/Applications/Python/".
- An Example has been developed to help you, to develop your own, we advise you to copy AllFunctionsCrtl.py, rename it and make your modifications. Please make sure you have CPC software installed and you are using CPC_wrapper.py file.

Your Python application is now ready to use. Please refer to Section *Code Examples* to access basic code.

2.3 Linux

2.3.1 Requirements

Package : build-essential, libudev-dev

To install these packages, please execute the following commands :

```
sudo apt update
sudo apt upgrade
sudo apt build-essential
sudo apt install libudev-dev
sudo apt install libusb-1.0-0-dev
```

Note: If you have installed the Aurea-CPS Software, you may already have the necessary packages.

2.3.2 Makefile Example

- Locate the CPC-API folder and go to "CPC-API/Applications/C++".
- Three differents programs have been developed to help you, To create your own application, we advise you to copy the AllFunctionsCrtl folder. Then rename the folder and the AllFunctionCrtl.cpp file. Finally replace the target name by your application name in the Makefile.

```
CC = g++
CFLAGS  = -Wall -pthread

# The build target
TARGET = AllFunctionsCtrl
INCLUDE = -I../../../Sources/h/

.PHONY: all
all: ${TARGET}

$(TARGET): $(TARGETPATH)$(TARGET).cpp
    $(CC) $(INCLUDE) $(CFLAGS) -o $(TARGET) $(TARGET).cpp -lCPC

.PHONY: clean
clean:
    -${RM} ${TARGET}
```

- Finally edit the cpp file to develop your application.

2.3.3 Create Python application

- Locate the CPC-API folder and go to "CPC-API/Applications/Python/".
- An Example has been devloped to help you, to develop your own, we advise you to copy AllFunctionsCrtl.py, rename it and make your modifications. Please make sure you have CPC software installed and you are using CPC_wrapper.py file.

Your Python application is now ready to use. Please refer to Section *Code Examples* to access basic code.

CHAPTER 3

Code Examples

The following section present simple codes in C++ and Python to use CPC device.

3.1 Communication

This first example shows how to list all CPC connected to a computer and how to open and close USB communication. Device information is also recovered in this example.

3.1.1 C++ program

```
#include <iostream>
using namespace std;

#include "CPC.h"

int main(int argc, const char* argv[]) {
    short iDev = 0;
    short ret;
    char* devicesList[10];
    short numberDevices;
    char* pch;
    char* next_pch = NULL;
    char version[64];
    char versionParam[3][32];
    char systemName[6];
```

(continues on next page)

(continued from previous page)

```
    memset(version, ' ', 64);
    memset(systemName, '\0', 6);

    /*  listDevices function    */
    // List Aurea Technology devices: MANDATORY BEFORE EACH␣
↪OTHER ACTION ON THE SYSTEM
    if (CPC_listDevices(devicesList, &numberDevices) == 0) {
        if (numberDevices == 0) {
            cout << endl << "   Please connect device !" << endl
↪<< endl;
            do {
                delay(500);
                CPC_listDevices(devicesList, &numberDevices);
            } while (numberDevices == 0);
        }
    }

    if (numberDevices > 1) {                              // If␣
↪more 1 device is present, list devices available
        cout << endl << "Device(s) available:" << endl << endl;

        for (int i = 0; i < numberDevices; i++) {
            printf(" -%u: %s\n", i, devicesList[i]);
        }
        cout << endl << "Select device to drive: ";
        cin >> iDev;

        if (CPC_openDevice(iDev) != 0) {                          /
↪/ Open and initialize device: MANDATORY BEFORE EACH OTHER␣
↪ACTION ON THE SYSTEM
            cout << "Failed to open CPC" << endl;
        }
    }
    else {                                                        /
↪/ Open by default only the device connected
        iDev = 0;
        if (CPC_openDevice(iDev) != 0) {                          /
↪/ Open and initialize device: MANDATORY BEFORE EACH OTHER␣
↪ACTION ON THE SYSTEM
            cout << "Failed to open CPC" << endl;
        }
    }

    // System version recovery
    if (CPC_getSystemVersion(iDev, version) == 0) {              ␣
↪       // Recovery of the system version
```

(continues on next page)

(continued from previous page)

```
        cout << endl << "  * System version:" << endl << endl;
    }
    else {
        cout << endl << " -> Failed to get system version" <<␣
↪endl << endl;
    }

    // Loop to extract CPC parameters
    int v = 0;
    pch = secure_strtok(version, ":", &next_pch);
    while (pch != NULL) {
        snprintf((char*)&versionParam[v][0], 32, "%s", pch);
        pch = secure_strtok(NULL, ":", &next_pch);
        v++;
    }
    if (pch != 0) { snprintf((char*)&versionParam[v][0], 32, "%s
↪", pch); }

    // Show system identity
    memcpy(systemName, (char*)&versionParam[2][0] + 3, 3);
    cout << "   AT System       : " << systemName << endl;
    cout << "   Serial number   : " << versionParam[0] << endl;
    cout << "   Product number  : " << versionParam[1] << endl;
    cout << "   Firmware version: " << versionParam[2] << endl;
    cout << endl;

    // Wait some time
    delay(2000);

    /*  CloseDevice function    */
    // Close initial device opened: MANDATORY AFTER EACH END OF␣
↪SYSTEM COMMUNICATION.
    if (CPC_closeDevice(iDev) == 0) cout << "   -> Communication␣
↪closed" << endl;
    else cout << "  -> Failed to close communication" << endl;

    return 0;
}
```

3.1.2 Python program

```
from ctypes import *
import time

# Import CPC wrapper file
import CPC_wrapper as CPC

# Application main
def main():
        key = ''
        iDev = c_short(0)
        nDev = c_short()
        devList = []

        # Scan and open selected device
        devList,nDev=CPC.listDevices()
        if nDev==0:   # if no device detected, wait
                print ("No device connected, waiting...")
                while nDev==0:
                    devList,nDev=CPC.listDevices()
                    time.sleep(1)
        elif nDev>1:  # if more 1 device detected, select target
                print("Found " + str(nDev) + " device(s) :")
                for i in range(nDev):
                    print (" -"+str(i)+": " + devList[i])
                iDev=int(input("Select device to open (0 to n):
↪"))

        # Open device
        if CPC.openDevice(iDev)<0:
                input(" -> Failed to open device, press enter to␣
↪quit !")
                return 0
        print("Device correctly opened")

        # Recover system version
        ret,version = CPC.getSystemVersion(iDev)
        if ret<0: print(" -> failed\n")
        else:print("System version = {} \n".format(version))

        # Wait some time
        time.sleep(2)

        # Close device communication
        CPC.closeDevice(iDev)

# Python main entry point
```

(continues on next page)

(continued from previous page)

```
if __name__ == "__main__":
        main()
```

3.2 Recover Data

The next example shows how to use CPC to recover clock and photon count. You can place the function CPC_getCLKCountData in a loop in order to get multiple data.

3.2.1 C++ program

```
#include <iostream>
using namespace std;

#include "CPC.h"

int main(int argc, const char* argv[]) {
    short iDev = 0;
    short ret;
    char* devicesList[10];
    short numberDevices;
    unsigned long CLK = 0, Count = 0;

    /*  listDevices function    */
    // List Aurea Technology devices: MANDATORY BEFORE EACH␣
↪OTHER ACTION ON THE SYSTEM
    if (CPC_listDevices(devicesList, &numberDevices) == 0) {
        if (numberDevices == 0) {
            cout << endl << "   Please connect device !" << endl
↪<< endl;
            do {
                delay(500);
                CPC_listDevices(devicesList, &numberDevices);
            } while (numberDevices == 0);
        }
    }

    if (numberDevices > 1) {                            // If␣
↪more 1 device is present, list devices available
        cout << endl << "Device(s) available:" << endl << endl;

        for (int i = 0; i < numberDevices; i++) {
            printf(" -%u: %s\n", i, devicesList[i]);
        }
        cout << endl << "Select device to drive: ";
```

(continues on next page)

(continued from previous page)

```
        cin >> iDev;

        if (CPC_openDevice(iDev) != 0) {                        /
↪/ Open and initialize device: MANDATORY BEFORE EACH OTHER␣
↪ACTION ON THE SYSTEM
            cout << "Failed to open CPC" << endl;
        }
    }
    else {                                                      /
↪/ Open by default only the device connected
        iDev = 0;
        if (CPC_openDevice(iDev) != 0) {                        /
↪/ Open and initialize device: MANDATORY BEFORE EACH OTHER␣
↪ACTION ON THE SYSTEM
            cout << "Failed to open CPC" << endl;
        }
    }

    // Recover Clock and Photons count
    if (CPC_getCLKCountData(iDev, &CLK, &Count) != 0)
        cout << " -> ! data not match !" << endl;
    else
        printf("\r -> Clock: %7lu       Hz      Counts: %7lu",␣
↪CLK, Count);

    // Wait some time
    delay(2000);

    /*  CloseDevice function    */
    // Close initial device opened: MANDATORY AFTER EACH END OF␣
↪SYSTEM COMMUNICATION.
    if (CPC_closeDevice(iDev) == 0) cout << "   -> Communication␣
↪closed" << endl;
    else cout << "  -> Failed to close communication" << endl;

    return 0;
}
```

3.2.2 Python program

```
from ctypes import *
import time

# Import CPC wrapper file
import CPC_wrapper as CPC

# Application main
def main():
        key = ''
        iDev = c_short(0)
        nDev = c_short()
        devList = []

        # Scan and open selected device
        devList,nDev=CPC.listDevices()
        if nDev==0:   # if no device detected, wait
                print ("No device connected, waiting...")
                while nDev==0:
                devList,nDev=CPC.listDevices()
                time.sleep(1)
        elif nDev>1:  # if more 1 device detected, select target
                print("Found " + str(nDev) + " device(s) :")
                for i in range(nDev):
                print (" -"+str(i)+": " + devList[i])
                iDev=int(input("Select device to open (0 to n):
↪"))

        # Open device
        if CPC.openDevice(iDev)<0:
                input(" -> Failed to open device, press enter to␣
↪quit !")
                return 0
        print("Device correctly opened")

        # Recover Clock and Photons Count
        time.sleep(2)
        ret,clk,det=CPC.getClockDetData(iDev)
        if ret<0: print(" -> failed\n")
        else: print(" Clock     = {} Hz \n Detection = {} cnt\s \
↪n".format(clk.value,det.value))

        # Wait some time
        time.sleep(2)

        # Close device communication
        CPC.closeDevice(iDev)
```

(continues on next page)

(continued from previous page)

```
# Python main entry point
if __name__ == "__main__":
        main()
```

Note: All function information is available in section *All Functions*.

CHAPTER 4

All Functions

This section provides the prototypes and descriptions of all functions integrated into CPC library. These functions allow you to control CPC.

Warning: More or less functions are available according to the device type. The compatibility depends on the device part number recovered by the "CPC_getSystemVersion" function. Please see notes functions to check the compatibility with your device: -> version compatibility: PN_CPC_x_xx_xx_xx Refer to section *Code Examples*, to recover version in C++ or in Python.

4.1 Library information

4.1.1 CPC_getLibVersion

short **CPC_getLibVersion**(unsigned short *value)

Get the librarie version.

Return the version librarie in format 0x0000MMmm

with: MM=major version

mm=minor version

Parameters ***value** – return lib version by pointer

Format: 0xMMmm

with: MM: major version

mm: minor version

Returns

0 : Function success

-1 : Function failed

-2 : Parameter(s) error

4.2 Connection Functions

4.2.1 CPC_listDevices

short **CPC_listDevices**(char **devices, short *number)

List Aurea Technology devices connected.

List Aurea Technology devices connected

Note: Mandatory to do before any other action on the system device.

Parameters

- ****devices** – pointer to the table buffer which contain list of devices connected

  Output format: “deviceName - serialNumber”

  Example:

  devices[0]=”CPC - SN_xxxxx1xxxxx\r\n”

  devices[1]=”CPC - SN_xxxxx6xxxxx\r\n”

- ***number** – pointer to the number devices connected

Returns

0 : Function success

-1 : Function failed

-2 : Parameter(s) error

4.2.2 CPC_openDevice

short **CPC_openDevice**(short iDev)
Open and initialize CPC device.

Open USB connection and initialize internal configuration

Note: Mandatory to do before any other action on the system device.

Parameters **iDev** – Device index indicate by "CPC_listDevices" function

Returns

0 : Function success

-1 : Function failed

-2 : Parameter(s) error

4.2.3 CPC_closeDevice

short **CPC_closeDevice**(short iDev)
Close CPC device.

Close USB connection of previously CPC opened.

Note: Mandory to do after each end of system transfer.

Parameters **iDev** – Device index indicate by "CPC_listDevices" function

Returns

0 : Function success

-1 : Function failed

-2 : Parameter(s) error

4.3 Save and Reset Settings

4.3.1 CPC_saveAllSettings

short **CPC_saveAllSettings**(short iDev)
Save all parameters.

Save all parameters (deadtime, detection mode, ...).

Note: version compatibility : all

Parameters **iDev** – Device index indicate by “CPC_listDevices” function

Returns

0 : Function success

-1 : Function failed

-2 : Parameter(s) error

4.3.2 CPC_factorySettings

short **CPC_factorySettings**(short iDev)
System factory settings.

Set all parameters with factory settings.

Note: version compatibility : all

Parameters **iDev** – Device index indicate by “CPC_listDevices” function

Returns

0 : Function success

-1 : Function failed

-2 : Parameter(s) error

4.4 Reboot System

4.4.1 CPC_resetSystem

short **CPC_resetSystem**(short iDev)
Reset system.

Reset system

Note: version compatibility : all

Parameters **iDev** – Device index indicate by “CPC_listDevices” function

Returns

0 : Function success

-1 : Function failed

-2 : Parameter(s) error

4.5 Device Information

4.5.1 CPC_getSystemVersion

short **CPC_getSystemVersion**(short iDev, char *version)

Get system version.

Get system version: Serial number, product number and firmware version

---

Note: version compatibility : all

Parameters

- **iDev** – Device index indicate by "CPC_listDevices" function
- **version** – Pointer to the buffer which receive the system version.

  String format: SN_'serialNumber':PN_'ProductNumber':'FirmwareVersion'

  The receive buffer size must be of 64 octets min.

Returns

0 : Function success

-1 : Function failed

-2 : Parameter(s) error

4.5.2 CPC_getSystemFeature

short **CPC_getSystemFeature**(short iDev, short iFeature, short *value)

Get system feature.

Read EEPROM to recovery one feature of system

Parameters

- **iDev** – Device index indicate by "CPC_listDevices" function
- **iFeature** – 0: detection speed

  1: module type

2: module performance

3: reserved

4: reserved

5: Output format type

- ***value** – pointer to the feature value (format: short)

Returns

0 : Function success

-1 : Function failed

-2 : Parameter(s) error

-4 : iDev index Out Of Range

4.5.3 CPC_getSystemHardwareVersion

short **CPC_getSystemHardwareVersion**(short iDev, short card, unsigned short *version, unsigned short *model)

Get system hardware card version and model.

Parameters

- **iDev** – Device index indicate by "CPC_listDevices" function
- **card** – 0 : uMCE

  1 : MTM

  2 : MCT
- **version** – pointer on value of card's version
- **model** – pointer on value of card's model

Returns

0 : Function success

-1 : Function failed

-2 : Parameter(s) error

-4 : iDev index Out Of Range

4.6 Recover Parameters Range

4.6.1 CPC_getEfficiencyRange

short **CPC_getEfficiencyRange**(short iDev, char *range)
Get efficiency range.

Send all APD efficiency available values

Note: version compatibility : PN_CPC_A_xx_xx

Parameters

- **iDev** – Device index indicate by "CPC_listDevices" function
- **range** – Pointer to the string buffer of the efficiency range.

Returns

0 : Function success

-1 : Function failed

-2 : Parameter(s) error

4.6.2 CPC_getDeadTimeRange

short **CPC_getDeadTimeRange**(short iDev, double *MinVal, double *MaxVal)
Get deadtime range.

Get min and max values of the deadtime range

Note: version compatibility : PN_CPC_A_xx_xx

Parameters

- **iDev** – Device index indicate by "CPC_listDevices" function
- **MinVal** – pointer to the current deadtime min value (format: double)
- **MaxVal** – pointer to the current deadtime max value (format: double)

Returns

0 : Function success

-1 : Function failed

-2 : Parameter(s) error

4.7 Set and Get Parameters

4.7.1 CPC_setEfficiency

short **CPC_setEfficiency**(short iDev, short efficiency)
Set efficiency.

Set APD effciency value (in %).

Note: version compatibility : PN_CPC_A_xx_xx

Parameters

- **iDev** – Device index indicate by "CPC_listDevices" function
- **efficiency** – Efficiency value in % and in multiple of 5

Returns

0 : Function success

-1 : Function failed

-2 : Parameter(s) error

4.7.2 CPC_getEfficiency

short **CPC_getEfficiency**(short iDev, short *efficiency)
Get efficiency.

Get actual APD efficiency value (in %).

Note: version compatibility : PN_CPC_A_xx_xx

Parameters

- **iDev** – Device index indicate by "CPC_listDevices" function
- ***efficiency** – pointer to the efficiency value (format: double)

Returns

0 : Function success

-1 : Function failed

-2 : Parameter(s) error

4.7.3 CPC_setDeadTime

short **CPC_setDeadTime**(short iDev, double deadTime)
Set deadtime.

Set APD deadtime value (in us).

Note: version compatibility : PN_CPC_A_xx_xx

Parameters

- **iDev** – Device index indicate by "CPC_listDevices" function
- **deadTime** – DeadTime value in us (format: double)

Returns

0 : Function success

-1 : Function failed

-2 : Parameter(s) error

4.7.4 CPC_getDeadTime

short **CPC_getDeadTime**(short iDev, double *deadTime)
Get deadtime.

Get actual APD deadtime value (in us).

Note: version compatibility : PN_CPC_A_xx_xx

Parameters

- **iDev** – Device index indicate by "CPC_listDevices" function
- ***deadTime** – pointer to the deadTime value (format: double)

Returns

0 : Function success

-1 : Function failed

-2 : Parameter(s) error

4.7.5 CPC_setCountingRate

short **CPC_setCountingRate**(short iDev, double rate)
Set the rate of sending photons counted.

Set the rate of sending photons counted. Rate value between 0.1s to 10s.

Note: version compatibility : all

Parameters

- **iDev** – Device index indicate by "CPC_listDevices" function
- **rate** – Rate value in s (format: double)

  Value between 0.1 to 10.0s with step of 0.1s

Returns

0 : Function success

-1 : Function failed

-2 : Parameter(s) error

4.7.6 CPC_getCountingRate

short **CPC_getCountingRate**(short iDev, double *rate)
Get counting rate value.

Get the rate of sending photons counted. Rate value between 0.1s to 10s.

Note: version compatibility : all

Parameters

- **iDev** – Device index indicate by "CPC_listDevices" function
- **rate** – Pointer to the current rate value (format: double).

Returns

0 : Function success

-1 : Function failed

-2 : Parameter(s) error

4.7.7 CPC_setOutputState

short **CPC_setOutputState**(short iDev, short state)
Set output state.

Set output state

Note: version compatibility : all

Parameters

- **iDev** – Device index indicate by "CPC_listDevices" function
- **state** – output state: 1=enabled; 0=disabled

Returns

0 : Function success

-1 : Function failed

-2 : Parameter(s) error

4.7.8 CPC_setIntegTime

short **CPC_setIntegTime**(short iDev, double timeInMs)
Set integration time.

Set integration time of detections counted for analog output format

Note: version compatibility : all

Parameters

- **iDev** – Device index indicate by "CPC_listDevices" function
- **timeInMs** – integration time between 0.1 to 10000.0 ms

Returns

0 : Function success

-1 : Function failed

-2 : Parameter(s) error

4.7.9 CPC_getIntegTime

short **CPC_getIntegTime**(short iDev, double *timeInMs)
Get integration time.

Get current integration time (in ms) of analog output format

Note: version compatibility : all

Parameters

- **iDev** – Device index indicate by "CPC_listDevices" function
- ***timeInMs** – pointer to the time value in ms (format: double)

Returns

0 : Function success

-1 : Function failed

-2 : Parameter(s) error

4.7.10 CPC_setAnalogOutGain

short **CPC_setAnalogOutGain**(short iDev, double gain)
Set analog gain

Set gain of analogic format output

Note: version compatibility : all

Parameters

- **iDev** – Device index indicate by "CPC_listDevices" function
- **gain** – Analogic gain between 0.1 to 100.0

Returns

0 : Function success

-1 : Function failed

-2 : Parameter(s) error

4.7.11 CPC_getAnalogOutGain

short **CPC_getAnalogOutGain**(short iDev, double *gain)

Get analog gain.

Get gain of analogic format output

Note: version compatibility : all

Parameters

- **iDev** – Device index indicate by "CPC_listDevices" function
- ***gain** – pointer to the gain value (format: double)

Returns

0 : Function success

-1 : Function failed

-2 : Parameter(s) error

4.7.12 CPC_setDetectionMode

short **CPC_setDetectionMode**(short iDev, short mode)

Set detection mode.

Set detection mode in continuous or gated mode

Note:

version compatibility :

PN_CPC_x_Cx_xx

PN_CPC_x_Qx_xx,

PN_CPC_x_Ix_xx,

PN_CPC_x_Jx_xx

Parameters

- **iDev** – Device index indicate by "CPC_listDevices" function
- **mode** – 0: continuous

  1: gated

Returns

0 : Function success

-1 : Function failed

-2 : Parameter(s) error

4.7.13 CPC_getDetectionMode

short **CPC_getDetectionMode**(short iDev, short *mode)

Get detection mode.

Get detection mode in continuous or gated mode

Note:

version compatibility :

PN_CPC_x_Cx_xx

PN_CPC_x_Qx_xx,

PN_CPC_x_Ix_xx,

PN_CPC_x_Jx_xx

Parameters

- **iDev** – Device index indicate by "CPC_listDevices" function
- ***mode** – pointer to the detection mode (format: short)

Returns

0 : Function success

-1 : Function failed

-2 : Parameter(s) error

4.7.14 CPC_setInputVoltageThreshold

short **CPC_setInputVoltageThreshold**(short iDev, double voltage)

Set input voltage threshold.

Set voltage threshold for system pulses input

Note: version compatibility : PN_CPC_V_xx_xx

Parameters

- **iDev** – Device index indicate by "CPC_listDevices" function
- **voltage** – Voltage threshold between 0.2 to 4.0 V

Returns

0 : Function success

-1 : Function failed

-2 : Parameter(s) error

4.7.15 CPC_getInputVoltageThreshold

short **CPC_getInputVoltageThreshold**(short iDev, double *voltage)

Get input voltage threshold.

Get voltage threshold of system pulses input

Note: version compatibility : PN_CPC_V_xx_xx

Parameters

- **iDev** – Device index indicate by "CPC_listDevices" function
- ***voltage** – pointer to the voltage value (format: double)

Returns

0 : Function success

-1 : Function failed

-2 : Parameter(s) error

4.7.16 CPC_getDetLimitThreshold

short **CPC_getDetLimitThreshold**(short iDev, double *detections)

Get detections limit threshold.

Get detections limit threshold after which the system power off the detector to protect it.

Note: version compatibility : all

Parameters

- **iDev** – Device index indicate by "CPC_listDevices" function
- ***detections** – pointer to the detection value (format: double)

Returns

0 : Function success

-1 : Function failed

-2 : Parameter(s) error

4.8 Monitoring Functions

4.8.1 CPC_getCLKCountData

short **CPC_getCLKCountData**(short iDev, unsigned long *CLK, unsigned long *Count)

Get clock and count data.

Get both clock value and photons count.

Note: version compatibility : all

Parameters

- **iDev** – Device index indicate by "CPC_listDevices" function
- **CLK** – Pointer to the APD clock value (format: decimal)
- **Count** – Pointer to the APD count value (format: decimal)

Returns

0 : Function success

-1 : Function failed

-2 : Parameter(s) error

4.8.2 CPC_getBodySocketTemp

short **CPC_getBodySocketTemp**(short iDev, double *bodyTemp)

Get body socket system temperature.

Get body socket system temperature

Note: version compatibility : PN_CPC_A_xx_xx

Parameters

- **iDev** – Device index indicate by "CPC_listDevices" function
- ***bodyTemp** – pointer to the body temperature value (format: double)

Returns

0 : Function success

-1 : Function failed

-2 : Parameter(s) error

4.8.3 CPC_getOutputVoltage

short **CPC_getOutputVoltage**(short iDev, double *voltage)

Get output voltage.

Get the current output voltage in analog format output

Note: version compatibility : all

Parameters

- **iDev** – Device index indicate by "CPC_listDevices" function
- ***voltage** – Pointer to the voltage value (format: double)

Returns

0 : Function success

-1 : Function failed

-2 : Parameter(s) error

4.8.4 CPC_getSystemAlarms

short **CPC_getSystemAlarms**(short iDev, char *alarm)

Get System Alarms.

Get active alarm detected by system

Note: version compatibility : all

Parameters

- **iDev** – Device index indicate by "CPC_listDevices" function
- **alarm** – Pointer to the buffer which receive the alarm description.

  The receive buffer size must be of 64 octets min.

Returns

1 : Alarm detected

0 : Function success

-1 : Function failed

-2 : Parameter(s) error

4.9 Hardware Control

4.9.1 CPC_setFanState

short **CPC_setFanState**(short iDev, short state)

Set Fan state.

Set Fan On or Off

Parameters

- **iDev** – Device index indicate by "CPC_listDevices" function
- **state** – 0 : OFF, 1 : ON

Returns

0 : Function success

-1 : Function failed

-2 : Parameter(s) error

-4 : iDev index Out Of Range

4.9.2 CPC_getFanState

short **CPC_getFanState**(short iDev, short *state)

Get Fan state.

Get Fan On or Off

Parameters

- **iDev** – Device index indicate by "CPC_listDevices" function
- ***state** – pointer to the fan state (format: short)

Returns

0 : Function success

-1 : Function failed

-2 : Parameter(s) error

-4 : iDev index Out Of Range

4.9.3 CPC_setLedState

short **CPC_setLedState**(short iDev, short state)

Set Led state.

Set Led On or Off

Parameters

- **iDev** – Device index indicate by "CPC_listDevices" function
- **state** – 0 : OFF, 1 : ON

Returns

0 : Function success

-1 : Function failed

-2 : Parameter(s) error

-4 : iDev index Out Of Range

4.9.4 CPC_getLedState

short **CPC_getLedState**(short iDev, short *state)

Get Led state.

Get Led On or Off

Parameters

- **iDev** – Device index indicate by "CPC_listDevices" function
- ***state** – pointer to the Led state (format: short)

Returns

0 : Function success

-1 : Function failed

-2 : Parameter(s) error

-4 : iDev index Out Of Range

CHAPTER 5

C++ Wrapper

5.1 Wrapper Advantage

A C++ wrapper has been created for several reasons.

- To make CPC functions easy to use.
- To allow multiple CPC device control in the same application and at the same time.
- To link Dynamic Library inside C++ code and not in the project configuration.

Note: Except OpenDevice function, you do not need to specify iDev when using CPC wrapper function.

For example function `CPC_getCLKCountData(short iDev, unsigned long *CLK, unsigned long *Count)` can be replace by `ObjectName.GetCLKCountData(unsigned long *CLK, unsigned long *Count)`

5.2 C++ code

Here is an example of how to use this wrapper to recover data from 2 CPC :

```
#include <iostream>
using namespace std;

#include "CPC_wrapper.h"
#include "CPC.h"

// Select shared library compatible to current operating system
#ifdef _WIN32
#define DLL_PATH L"CPC.dll"
#elif __unix
#define DLL_PATH "CPC.so"
#else
#define DLL_PATH "CPC.dylib"
#endif

int main(int argc, const char* argv[]) {
        short iDev = 0;
        short ret;
        char* devicesList[10];
        short numberDevices;
        unsigned long CLK = 0, Count = 0;
        unsigned long CLK2 = 0, Count2 = 0;

        // Instancie the device from wrapper
        CPC_wrapper CPC0(DLL_PATH);
        CPC_wrapper CPC1(DLL_PATH);

    /*    ListDevices function    */
    // List Aurea Technology devices: MANDATORY BEFORE EACH␣
↪OTHER ACTION ON THE SYSTEM
    if (CPC0.ListDevices(devicesList, &numberDevices) == 0) {
        if (numberDevices == 0){
            cout << endl << "    Please connect AT device !" <<␣
↪endl << endl;
            do {
                delay(500);
                CPC0.ListDevices(devicesList, &numberDevices);
            } while (numberDevices == 0);
        }
    }

        // Open communication with device 0
        printf(" -%u: %s\n", 0, devicesList[0]);
        CPC0.OpenDevice(0);
```

(continues on next page)

(continued from previous page)

```
        printf("\n CPC %d-> Communication Open\n\n", 0);

        // Open communication with device 1
        printf(" -%u: %s\n", 1, devicesList[1]);
        CPC1.OpenDevice(1);
        printf("\n CPC %d-> Communication Open\n\n", 1);

        // Recover Clock and Photons count for both devices
        CPC0.GetCLKCountData(&CLK, &Count);
        printf("\n\nCPC 0 -> Clock: %7lu        Hz      Counts :
↪%7lu", CLK, Count);
        CPC1.GetCLKCountData(&CLK2, &Count2);
        printf("\nCPC 1 -> Clock: %7lu  Hz      Counts : %7lu",␣
↪CLK2, Count2);

        // Wait some time
        delay(2000);

        /*    CloseDevice function    */
        // Close initial device opened: MANDATORY AFTER EACH END␣
↪OF SYSTEM COMMUNICATION.
        if (CPC0.CloseDevice() == 0) cout << "    ->␣
↪Communication closed" << endl;
        else cout << "  -> Failed to close communication" <<␣
↪endl;
        if (CPC1.CloseDevice() == 0) cout << "    ->␣
↪Communication closed" << endl;
        else cout << "  -> Failed to close communication" <<␣
↪endl;

        // Call class destructor
        CPC0.~CPC_wrapper();
        CPC1.~CPC_wrapper();

        return 0;
}
```

CHAPTER 6

Revision History

6.1 v4.0 (14/06/22)

- New API architecutre
- Add Application example
- Add Device index in all functions
- Modify all get function to return value by pointer
- Add `CPC_` at the beginning of all functions
- **Replace functions :**
  - CPC_listATdevices() to CPC_listDevices()
  - CPC_openATdevice() to CPC_openDevice()
  - CPC_closeATdevice() to CPC_closeDevice()
- Add Wrapper
- Handle multiple device application

6.2 v3.7 (22/06/21)

- **Modification functions :**
  - SetOutputFormat() : add NIM format
  - GetOutputFormat() : add NIM format
  - Modifications of function compatibilities and comments

6.3 v3.6 (27/02/20)

- **Rename functions :**
  - SetVisibleModuleDetection() to SetDetectionMode()
  - GetVisibleModuleDetection() to GetDetectionMode()

6.4 v3.5 (26/02/19)

- **Improvement of functions :**
  - setDeadtime() (accuracy to the thousandth)

6.5 v3.4 (23/03/18)

- **Improvement of functions :**
  - setDeadtime() (accuracy to the thousandth)

6.6 v3.3 (12/01/17)

- Improvement of internals functions to close the DLL cleaner way

6.7 v3.2 (11/01/17)

- Internal improvement

6.8 v3.1 (23/12/16)

- Improvement GetCLKCountData() function
- **Add functions :**
  - GetOutputVoltage()

6.9 v3.0 (19/03/15)

- **Replace functions :**
  - OpenSystemTransfer() to OpenATdevice()
  - CloseSystemTransfer() to CloseATdevice()

6.10 v2.0 (22/09/14)

- **Add functions :**
  - SetOutputFormat()
  - GetOutputFormat()
  - SetIntegTime()
  - GetIntegTime()
  - SetAnalogOutGain()
  - GetAnalogOutGain()

6.11 v1.0 (21/05/14)

- First release