## Brief

Name: nanoLambda NSP32 Python API for desktop

Type: API

Version: 1.0.0

Language: Python

Platform: desktop

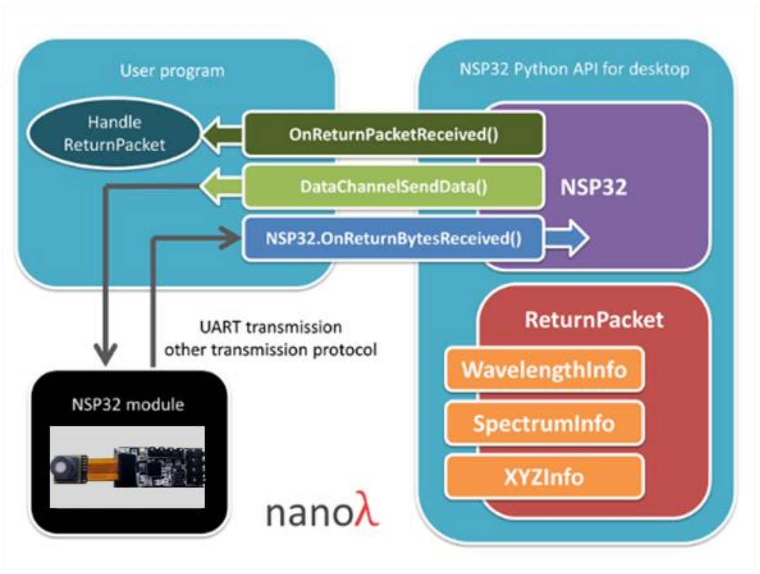
## Introduction

The API is designed for use on desktop applications coded in Python. By using this API, users can easily control NSP32 module by high level function calls, without dealing with the raw packet bytes and packet queueing / parsing things.

Note: General concepts are illustrated in NSP32 datasheet. Please see datasheet in advance.

## Architecture & Concepts

1) API architecture



Note: different classes/structs and their corresponding source files are listed in different colored blocks.

The architecture contains three major parts.

i) Controller

- NSP32 class is the main controller that deals with commands, packets, error detections, packet queueing / parsing, and other flow logic.

ii) Interface and notification functions

- DataChannelSendData()

  This is a regular function implemented in user program, and is passed as a parameter when creating NSP32 object. NSP32 sends out command packets through DataChannelSendData(), and user program sends out the bytes via available transmission protocol.

- NSP32.OnReturnBytesReceived()

  User program calls NSP32.OnReturnBytesReceived() and feeds any bytes received from NSP32 module.

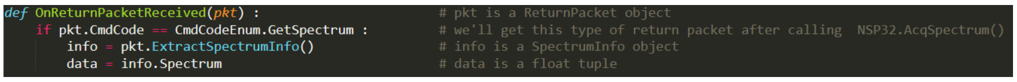
- OnReturnPacketReceived()

This is a regular function implemented in user program, and is passed as a parameter when creating NSP32 object. When a return packet is successfully parsed by NSP32, NSP32 will notify the user program by OnReturnPacketReceived().

iii) Data

ReturnPacket class encapsulates the return packet received from NSP32 module. We also define three classes (which are placed in orange blocks in the picture) to encapsulate wavelength, spectrum, and XYZ data. Users can extract these class objects from ReturnPacket object, and retrieve their interested information.

Example:



```
def OnReturnPacketReceived(pkt) :                        # pkt is a ReturnPacket object
    if pkt.CmdCode == CmdCodeEnum.GetSpectrum :          # we'll get this type of return packet after calling  NSP32.AcqSpectrum()
        info = pkt.ExtractSpectrumInfo()                 # info is a SpectrumInfo object
        data = info.Spectrum                             # data is a float tuple
```

Note:

For commands like CMD_ACQ_SPECTRUM and CMD_ACQ_XYZ, API handles the complete cycle (from starting acquisition to fetch results) inside. So you will see only NSP32.AcqSpectrum() function, but no NSP32.GetSpectrum() function. Due to this reason, if you investigate the FUNCTION_CODE byte in the return packet of NSP32.AcqSpectrum(), you'll find the byte is CMD_GET_SPECTRUM instead of CMD_ACQ_SPECTRUM. That's why in the above example, we use "if pkt.CmdCode == CmdCodeEnum.GetSpectrum :".

Please refer [/examples/] examples for complete demonstration.

## Development Tool Recommendation

Python 3.5 or above (Python 2 doesn't work)

## How to Use

1) Copy [/src/NanoLambdaNSP32.py] to your project folder.
2) Import "NanoLambdaNSP32" module and write codes (please refer our examples).

## API Features

1) Packet error detection: The API monitors and validates each return packet. In OnReturnPacketReceived(), you can use ReturnPacket.IsPacketValid property to check if any packet error occurred.
2) Both return packet raw bytes and easy-to-use data objects are available: In most cases, users extract their desired information (data object) from the return packet

by calling ReturnPacket.Extract...() functions. However users can also use ReturnPacket.PacketBytes property to access the return packet raw bytes if needed.

3) Command queue management: Normally, users should send new command to NSP32 module only after the previous command returns. To make the flow control easier in user's program, our API hosts an internal command queue. Users can call API to send multiple commands at once, and the API will make sure each command goes to NSP32 module at the right time.

## Note

In APIs for desktop, there is NSP32.Standby() but no NSP32.Wakeup(), due to the lack of GPIO pins on desktop to wakeup / reset NSP32 module.

Note: Users should call NSP32.Standby() only if there is an extra designed mechanism to wakeup / reset NSP32 module (e.g. a hardware push button).

## API Reference

1. html version: [/doc/reference_html/index.html]
2. pdf version: [/doc/reference.pdf]