Brief

Name: nanoLambda NSP32 Java API for Android and desktop

Type: API

Version: 1.0.0

Language: Java

Platform: Android / desktop

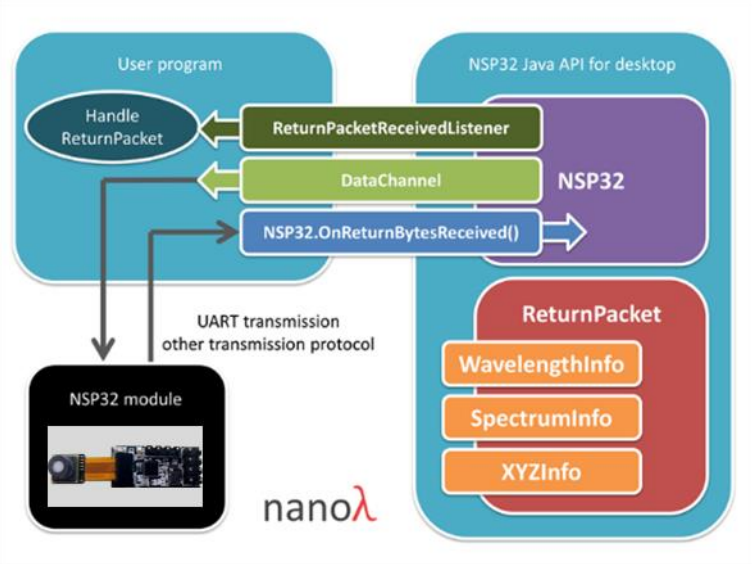
Introduction

The API is designed for use on Android or desktop applications coded in Java. 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

- DataChannel interface

  NSP32 sends out command packets through DataChannel interface, 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.

- ReturnPacketReceivedListener interface

  When a return packet is successfully parsed by NSP32, NSP32 will notify the user program by ReturnPacketReceivedListener.

iii) Data

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

Example:

```
// OnReturnPacketReceived() is an OnReturnPacketReceivedDelegate
private void OnReturnPacketReceived(ReturnPacket pkt)
{
    switch (pkt.CmdCode)
    {
        // we'll get this type of return packet after calling NSP32.AcqSpectrum()
        case CmdCodeEnum.GetSpectrum:
            SpectrumInfo info = pkt.ExtractSpectrumInfo();
            float[] data = info.Spectrum;
            break;
    }
}
```

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. And 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 "case GetSpectrum:".

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

Development Tool Recommendation

JDK 8

Pre-built jar

The pre-built jar is located at [/src/NanoLambdaNSP32.jar].

If you do want to rebuild it, you can follow these steps:

1) Windows users can run [/src/build.bat] batch file, and you will get the jar at [/src/NanoLambdaNSP32.jar].
2) Linux or macOS users can run [/src/build.sh] by terminal commands:

```
$ chmod +x build.sh
$ sudo ./build.sh
```

And you will get the jar at [/src/NanoLambdaNSP32.jar].

How to Use

1) Compile and run your project along with "NanoLambdaNSP32.jar" (the detailed steps depend on your development environment).
2) You can also view [/examples/desktop/Beginner/run.bat] to see how we do it by JDK commands.

API Features

1) Packet error detection: The API monitors and validates each return packet. In ReturnPacketReceivedListener.OnReturnPacketReceived(), you can use ReturnPacket.IsPacketValid() 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() 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 just 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 Android / desktop, there is NSP32.Standby() but no NSP32.Wakeup(), due to the lack of GPIO pins on Android / 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]