Sixdof Falcon
Python Bindings

Version 2

Contents

1 Introduction

In this document we will describe how to generate python bindings for the SdsFalcon Sdk, these bindings will allow running the C++ SDK from Python code.

There are multiple tools that enable exposing C++ code to Python, each has its own advantages and disadvantages. In this tutorial we will demonstrate how to expose the C++ SDK by using Pybind11. Pybind11 has many advantages, such as; it is header only, simple to use and supports modern C++ constructs. However, one of the disadvantages of Pybind11 is that it is sensitive to the Python version. If the bindings are generated with a specific version of Python, they will not work on runtime with another version. Due to this limitation, users of the SdsFalcon SDK are required to build the Python bindings themselves, with the Python environment they will be using at runtime.

2 Build Instructions

The following steps describe how to build the Python bindings for your specific platform:

1. **Download Pybind11** - download Pybind11 from Github, un-zip and place the folder in the "python_bindings" directory. The downloaded pybind11 will likely have some version information in the folder name, so rename the folder to just "pybind11". Pybind11 release 2.11.1 was used in the following instructions, however higher versions should work as well.

2. **Install CMake** - install the CMake build tool. You can check if you already have CMake by typing `cmake --version` in a command terminal. The following steps were done with CMake version 3.17.5.

3. **Install VS Code** - download VS Code. Open the "python_bindings" folder. From the "Extensions" menu in the left hand toolbar install the "CMake" and "CMake Tools" extensions.

4. **Edit CMake** - edit the "CMakeLists.txt" file in the "python_bindings" folder. Specifically, un-comment the line between lines 8 and 13 that matches the platform you are building on. Saving the edits to CMakeLists.txt will trigger a CMake re-configure action, in this step many print outs will be shown in the "Output" console, make sure there are no errors. Additionally, it will send a message to the "Output" console once it finds a Python environment to use, it will say "Found PythonInterp" with the path of the python interpreter it is going to use, make sure this is the one you want it to use. If it is not the one you expect, you can point it to the correct Python installation by putting it higher up on your system PATH. (If you have not yet selected a compiler kit, these print outs will only show after one is selected).

5. **Build** - in VS Code navigate to the left-hand menu bar and select the "CMake" extension. Under the "Configure" section, select the compiler you wish to use, and set it to a "Release" build. Then under the "Build" section select "pySdsFalcon". Now you should see a "build" folder within the "python_bindings" directory. After doing the build, inside the "build" folder there should be a file called pySdsFalcon.pyd, or pySdsFalcon.so on Linux platforms.

3 Running from Python

After the Python bindings have been built, an example can be run by the following steps:

1. **Manage dependencies** - move the "falcon_test_bindings.py" example into the "build" folder. (If you have a "Release" folder in the "build" folder, put the file there.)

2. **Edit the example** - edit lines 25 and 26 in the file "falcon_test_bindings.py" to use your sensor id and network name.

3. **Check Python version** - In the next step you will run the Python example code. Before running the command please run `python --version` to check that the Python version matches the one that was used to build the bindings in the previous steps. On some systems the `python3` command should be used instead. Additionally, for Linux users, the first time the example script is run it will require administrator permissions. This can be achieved by running the command with the `sudo` command in front, please run `sudo python --version` to ensure it will still be pointing to the intended Python environment.

4. **Run the example** - connect the sensor and set up the beacons as described in the "Build_Examples.pdf". Then from within the "build" directory (or "Release" directory), run the following command:

   ```
   python falcon_test_bindings.py
   ```

   Now the example script should be running and you should see output in the python console.

5. **Handling Vectors in Python Bindings** - In the C++ SDK, fields that are `std::vector` types should be treated as Python lists when using the Python bindings. For instance, consider the struct `Config6Dof`, which contains a field `map` that is of type `std::vector<Beacon>`. In Python, you should handle this field as a list of `Beacon` objects. The file `falcon_test_bindings_2.py` contains an example to illustrate how to set this field. It is important to overwrite the list and not add objects to it using `append` This is a limitation of Pybind11, as documented here.