CCNx-based Cloud-Native Function: Networking and Applications

〜[2nd Part] Cefpyco: Python Compact Package for Developing Cefore Applications〜

Yusaku Hayamizu, Atushi Ooka, Kazuhisa Matsuzono, Hitoshi Asaeda
(NICT, Japan)

2022/09/19
We introduce the overview and features of Cefpyco, and explain how to install and use it.

### Contents

- Cefpyco overview
- Cefpyco install
- Cefpyco basics
  - ICN Communications using Cefpyco/Cefore
  - Implementing simple consumer and producer
  - Implementing sample applications
Cefpyco Overview
Cefpyco Overview

- cefpyco (CEFore PYthon COmpact package)
  - A python package enabling a user-friendly implementation as developers can call CCNx functions such as sending Interest and Data Packets
  - It’s easier to write than developing in C language

**C language**

```c
#include <stdio.h>
#include <stdlib.h>
#include <unistd.h>
#include <ctype.h>
#include <cefore/cef_define.h>
#include <cefore/cef_client.h>
#include <cefore/cef_frame.h>
#include <cefore/cef_log.h>

int main(int argc, char *argv[]) {
    CefT_Client_Handle fhdl;
    CefT_Interest_TLVs params_i;
    int res;
    cef_log_init("cefpyco");
    cef_frame_init();
    res = cef_client_init(port_num, conf_path);
    if (res < 0) return -1;
    fhdl = cef_client_connect();
    if (fhdl < 1) return -1;
    memset(&params_i, 0, sizeof(CefT_Interest_TLVs));
    res = cef_frame_conversion_uri_to_name("ccnx:/test", params_i.name);
    if (res < 0) return -1; // Failed to convert URI to name.
    params_i.name_len = res;
    params_i.hoplimit = 32;
    params_i.opt.lifetime_f = 1;
    params_i.opt.lifetime = 4000ull; /* 4 seconds */
    params_i.opt.symbolic_f = CefC_T_OPT_REGULAR;
    params_i.chunk_num_f = 1;
    params_i.chunk_num = 0;
    cef_client_interest_input(fhdl, &params_i);
    if (fhdl > 0) cef_client_close(fhdl);
    return 0;
}
```

**Python**

```python
import cefpyco

with cefpyco.create_handle() as h:
    h.send_interest("ccnx:/test", 0)
```

Example:

a Code for sending an interest packet
Cefpyco Install
Cefpyco Download

- [https://cefore.net/download](https://cefore.net/download) > Utilities > cefpyco
  - Including a Cefpyco manual (README.html)
Build and Install Cefpyco

(Assuming that **Cefore has been successfully installed.**)

1. **Install libraries (Ubuntu)**

   ```
   $ sudo apt-get install cmake python3-pip
   $ sudo pip3 install setuptools click numpy
   ```

2. **Build and Install Cefoyco (Ubuntu)**

   ```
   $ unzip cefpyco-0.6.0.zip
   $ cd cefpyco-0.6.0
   $ cmake .
   $ sudo make install
   $ sudo python3
   >> import cefpyco # if no errors, cefpyco install is OK!
   >> (Ctrl-D)        # exit
   ```

(In the case of Mac)

```
$ brew install cmake
$ pip3 install setuptools click numpy
```
Cefpyco Basics
Communications using Cefpyco/Cefore

- **Main methods**
  1. Connect to cefnetd
     - `create_handle()`
  2. Send a data packet
     - `send_data(name, chunk_num, payload)`
  3. Send an interest packet
     - `send_interest(name, chunk_num)`
  4. Receive a packet
     - `receive()`

- **Simple Consumer/Producer Applications**
  5. `simple-consumer.py`
  6. `simple-producer.py`
     - Use `register(name)` method

- **Sample Applications**
1) Connect to cefnetd

1.) Create a Python file `test1.py` with the following program

```python
import cefpyco

with cefpyco.create_handle() as handle:
    pass  # connect to cefnetd at block start and disconnect at block end.
```

2.) Start csgmrdstart and cefnetd, and Run `test1.py`

(it’s OK if no error message appears)

```
cefore:~/cefpyco$ sudo csmgrdstart  # if needed
cefore:~/cefpyco$ sudo cefnetdstart
cefore:~/cefpyco$ sudo python3 test1.py
[cefpyco] Configure directory is /usr/local/cefpre
cefore:~/cefpyco$
```
Additional Info. : python syntax

- Sentences and blocks are represented by indents instead of semicolons and parentheses as in C language.

  The range of blocks can be identified at a glance

  ```python
  # when a=1, b=1, b is displayed
  # when a=1, b≠1, a is displayed
  # when a≠1, nothing.
  if a == 1:
      if b == 1:
          print("b")
      else:
          print("a")
  ```

  Note: it's a bug if the indent width is not aligned

  ```python
  if a == 1:
      print("correct")
  else:
      print("error")
      print("error")
  ```

- `with` statement : used in exception handling to make the code cleaner and much more readable.
  - Representative example : file open/close

  Without "with statement"
  ```python
  print("Begin.")
  try:
      h = cefpyco.CefpycoHandle()
      h.begin()
      print("Do something.")
  except Exception as e:
      print(e)  # 例外処理
  finally:
      h.end()
  print("End.")
  ```

  With "with statement"
  ```python
  print("Begin.")
  with cefpyco.create_handle() as h:
      print("Do something.")
  print("End.")
  ```

Note: it's a bug if the indent width is not aligned

- Error example
  - 4-character Space and 2-character Space
  - 4-character Space and 1-Tab
② Send a data packet

1.) Create a Python file `test2.py` with the following program

```python
import cefpyco

with cefpyco.create_handle() as handle:
    # send a data packet (the name is "ccnx:/test", the content is "hello",
    # the chunk number is 0, the cache time is 3,600,000 ms.)
    handle.send_data("ccnx:/test", "hello", 0, cache_time=3600000)
```

2.) Start csmgrd and cefnetd, and Run test2.py

```
... cefore:~/cefpyco$ sudo python test2.py
...
```

```
***** Cache Status Report *****
Number of Cached Contents : 1
[0]
    Content Name : ccnx:/test/
    Content Size : 5 Bytes
    Access Count : 0
...
```

Data is stored in csmgr (i.e. content store)
③ Send an interest packet

1.) Create a Python file `test3.py` with the following program

```python
import cefpyco

with cefpyco.create_handle() as handle:
    # send an interest packet to get a data
    # named ccnx:/test with chunk-num 0
    handle.send_interest("ccnx:/test", 0)
```

2.) Start csmgrd and cefnetd, and then, Run test2.py and test3.py

```bash
# After run the scenario of test2.py
cefore:~/.cefpyco$ sudo python test3.py
...
cefore:~/.cefpyco$ csmgrstatus ccnx:/
...
***** Cache Status Report *****
Number of Cached Contents : 1
[0]
  Content Name : ccnx:/test/
  Content Size : 5 Bytes
  Access Count  : 1
...
```

Access Count is increased by 1.
Receive a packet

Create a Python file `test4.py` with the following program

```python
import cefpyco

with cefpyco.create_handle() as handle:
    handle.send_interest("ccnx:/test", 0)  # to receive the data packet
    # handle.register("ccnx:/test")  # to receive an interest packet
    info = handle.receive()
    print(info)
```

※ `receive()`メソッド
  ・waits for a packet for about 4 sec., after execution.
  (if you want to receive it until success, a while loop etc. is required.)
  ・returns CcnPacketInfo object (the list of properties is shown in the next page.)

Start csmgrd and cefnetd, Run test2.py, and then, Run test4.py

# After run the scenario of test2.py
```bash
$ cefore:~/cefpyco$ sudo python test4.py
...
```

the content of info (received data packet) is displayed here
...
# CcnPacketInfo Property

<table>
<thead>
<tr>
<th>プロパティ名</th>
<th>型</th>
<th>説明</th>
</tr>
</thead>
<tbody>
<tr>
<td>is_succeeded, is_failed</td>
<td>bool</td>
<td>Success or failure flag for receiving packets</td>
</tr>
<tr>
<td>is_interest, is_data</td>
<td>bool</td>
<td>Flag indicating whether the received packet is interest/data</td>
</tr>
<tr>
<td></td>
<td></td>
<td>(if reception fails, both returns false)</td>
</tr>
<tr>
<td>name</td>
<td>string</td>
<td>Name using URI format (ccnx:/~)</td>
</tr>
<tr>
<td>name_len</td>
<td>int</td>
<td>Name length for the URI</td>
</tr>
<tr>
<td>chunk_num</td>
<td>int</td>
<td>Chunk number</td>
</tr>
<tr>
<td>payload</td>
<td>bytes</td>
<td>(in the case of data) data content</td>
</tr>
<tr>
<td>payload_s</td>
<td>string</td>
<td>(in the case of data) data content as character strings</td>
</tr>
<tr>
<td></td>
<td></td>
<td>(invalid for binary data)</td>
</tr>
<tr>
<td>payload_len</td>
<td>int</td>
<td>(in the case of data) byte length of the content data</td>
</tr>
<tr>
<td>version</td>
<td>int</td>
<td>Version value of the received packet</td>
</tr>
<tr>
<td>type</td>
<td>int</td>
<td>Type value of the received packet</td>
</tr>
<tr>
<td>actual_data_len</td>
<td>int</td>
<td>Byte length of the received packet including the header</td>
</tr>
<tr>
<td>end_chunk_num</td>
<td>int</td>
<td>Last chunk number (valid only when specified)</td>
</tr>
</tbody>
</table>
Implementing Simple Consumer and Producer
Simple consumer application

- Create a Python file `simple-consumer.py` with the following program
  - To receive a content of which the name `ccnx:/test` and the chunk-number is 0
  - Repeat the `receive()` method until the packet reception success

- Let’s complete `simple-consumer.py` below

```python
#!/usr/bin/env python3

from time import sleep
import cefpyco

with cefpyco.create_handle() as handle:
    while True:
        handle.send_interest("ccnx:/test", 0)
        info = handle.receive()
        if info.is_data and (info.name == "ccnx:/test") and (info.chunk_num == 0):
            print("Reception Success!!")
            print(info)
            break
        sleep(1)
```

simple-consumer.py
Example of how simple-consumer works

```bash
cefore:/cefpypco$ csmgrdstart
cefore:/cefpypco$ cefnetdstart
cefore:/cefpypco$ echo hello > test
cefore:/cefpypco$ cefputfile ccnx:/test
[cefputfile] Start
...
[cefputfile] Throughput = 11819 bps

cefore:/cefpypco$ python3 simple-consumer.py
[cefpypco] Configure directory is /usr/local/cefore
Success
Info: Succeeded in receiving Data packet with name 'ccnx:/test' (#chunk: 0) and payload 'b'hello\n'' (6 Bytes)
```

Successfully receive a data
(the name is ccnx:/, the chunk-numis 0)
⑥ Simple producer application

- Create a Python file `simple-producer.py` with the following program
  - Use `register()` method to receive interest packets
    - `register(name-prefix)` method to register to send interest packets to the app, if the name of received interest is the specified `name-prefix`.

- Let's complete `simple-producer.py` below
  - After receiving interests named “ccnx:/test”, the data of “hello” is sent out

```python
import cefpyco

with cefpyco.create_handle() as handle:
    handle.register("ccnx:/test")
while True:
    info = 
    if info.is and (info.name ) and (info.):
        handle.send(
        # break # Uncomment if publisher provides content once
```
Example of how simple-producer works

- After disabling csmgrd, perform the following two steps

```bash
$ sudo cefnetdstart...
$ sudo python3 simple-producer.py
[cefpyco] Configure directory is /usr/local/cefore
```

Start content distribution

```bash
$ sudo python3 simple-consumer.py
[[cefpyco] Configure directory is /usr/local/cefore
------ [DEBUG] returncode: 0 (<class 'int'>)
Reception Success!!
Info: Succeeded in receiving Data packet with name 'ccnx:/test' (#chunk: 0) and payload 'b'hello'" (5 Bytes)
```

Successful data reception without csmgr
<table>
<thead>
<tr>
<th>Cefpyco Method Name</th>
<th>argument * return value</th>
<th>explanation</th>
</tr>
</thead>
<tbody>
<tr>
<td><strong>begin</strong></td>
<td>• ceforedir=None: directory pass to cefnetd.conf&lt;br&gt; • portnum=9896: port number of cefnetd</td>
<td>Initiates a connection to cefnetd. It’s called automatically when using the “with” statement.</td>
</tr>
<tr>
<td><strong>end</strong></td>
<td>Nothing</td>
<td>Complete the connection to cefnetd. It’s called automatically when using the “with” statement.</td>
</tr>
<tr>
<td><strong>send_interest</strong></td>
<td>• name: content name (ccnx:/…)&lt;br&gt; • chunk_num=0: chunk number&lt;br&gt; • symbolic_f=INTEREST_TYPE_REGULAR: Interest type&lt;br&gt; • hop_limit=32: max hop number&lt;br&gt; • lifetime=4000: Interest lifetime (Milliseconds from the current time)</td>
<td>Sends an interest packet requesting content with the specified name.</td>
</tr>
<tr>
<td><strong>send_data</strong></td>
<td>• name: content name (ccnx:/…)&lt;br&gt; • payload: payload of data packet&lt;br&gt; • chunk_num=1: chunk number&lt;br&gt; • end_chunk_num=-1: last chunk number of content (omitted in the case of negative numbers)&lt;br&gt; • hop_limit=32: max hop number&lt;br&gt; • expiry=3600000: content expiration time (Milliseconds from the current time)&lt;br&gt; • cache_time=-1: recommended cache time (omitted in the case of negative numbers)</td>
<td>Send a data packet with the specified name and payload</td>
</tr>
<tr>
<td><strong>receive</strong></td>
<td>• error_on_timeout=false: whether or not to return an error on timeout&lt;br&gt; • timeout_ms=4000: time between the start of reception and the timeout (milliseconds)&lt;br&gt; • return value: CcnPacketInfo (refer to another slide)</td>
<td>Waits for a specified time for an interest or data packet, and returns information about the received packet</td>
</tr>
<tr>
<td><strong>register</strong></td>
<td>• name: specify the name prefix of the interest you want to receive</td>
<td>Register the prefix name of interest you want to receive with the cefnetd, and enables the application to receive the interest by receive().</td>
</tr>
<tr>
<td><strong>deregister</strong></td>
<td>• name: specify the prefix name you want to unregister</td>
<td>Unregister the prefix name registered by register() method.</td>
</tr>
</tbody>
</table>
Sample Application using Cefpyco/Cefore
Sample app.: CefApp

- Provide two apply in cefpyco/cefapp directory
  - cefappconsumer.py: consumer apply
  - cefappproducer.py: producer apply

- Feature
  - Supports sending and receiving multiple chunks of
  - Supports 3 types of INPUT/OUTPUT: Inline • Standard I/O • File
  - Cefappconsumer supports pipeline processing for sending interests

- Refer to README for more details
  - https://github.com/cefore/cefpico/blob/master/README.md
Sample app.: Push communication

- Name-based communication in ICN
  - **Pull-based communication**:  
    1) Send an interest with the name of content to retrieve  
    2) Obtain the corresponding data from the network (Cache).

- E.g., IoT

- Push communication is often required:
Sample app. for push communication

**Method 1:**
- The sensor node puts information on the interest and sends it to the server.

**Feature**
- It’s so simple 😊
- but deviates from the ICN basic principle that one Interest retrieves at most one Data packet 😞
Sample App. for push communication

- **Method 2**:  
  - Pull communication is considered a network function that pulls data from a cache in a network.  
  - Define a network function that pushes some data to a network, and call the *function name*.

ладство:
Interest: ccnx:/function-name/parameter → 線路をつなぐ
Interest: ccnx:/temp/%00 → 線路をつなぐ
Data: ccnx:/temp/%00 → データの転送
30℃ → センサー

Let’s create it easily using Cefpyco/Cefore !!
Sample App. for Push communication

- **Goal**: create a program for push communication using Cefpyco/Cefore
  1. A sensor node (push-consumer) sends an interest that calls a network function for push communication
  2. A server (push-server) sends some specified interests for the sensor node to push data
  3. The sensor sends a data (string: 30 degree celsius) to the server

(push-server.py) → Cefnetd → push-consumer.py

Cache (csmgrd or local-cache)

(push-server.py) → Cefnetd → push-consumer.py

(※ for demo, I only use one laptop running one cefnetd (with local-cache mode))
Sample App. for Push communication

- Simple Naming example (used by push-server.py, push-consumer.py)
  - ccnx:/_SF_abcddef_/K/NAME/%00

① **Function Name(_SF_xxxx_)**

② **total number of chunks(K)**

③ **Name of the data to push (NAME)**

- Example of interest name to call a push function
  - ccnx:/_SF_abcddef_/5/Current-Temp/%00
Sample App. for Push communication

Example: simple-push-server.py

```python
import cefpyco
import re

FunctionName = "/_SF_abcddef_"

With cefpyco.create_handle() as handle:
    # Interest Name Prefix for Cache Function
    interestNamePrefix = "ccnx:" + FunctionName
    handle.register(interestNamePrefix)
    while True:
        info = handle.receive()
        if info.is_succeeded and (interestNamePrefix in info.name) and (info.chunk_num==0):
            chunkNum = re.findall(interestNamePrefix + "/(.*?)/", info.name)
            namePrefix = re.findall(interestNamePrefix + "/(.*?)/.*?(/.*)", info.name)

            interestName = "ccnx:" + namePrefix[0]
            for i in range( int(chunkNum[0]) ):
                handle.send_interest(interestName, i)

            dataRecvNum = 0
            while True:
                tmpinfo = handle.receive()
                if tmpinfo.is_succeeded and (tmpinfo.name == interestName):
                    dataRecvNum+=1
                    if(dataRecvNum == int(chunkNum[0])):
                        print("Receive all Data to be cached (" + str(dataRecvNum) + ") name: " + interestName)
                        break
```

Example: simple-push-server.py
Sample App. for Push communication

- **Example: simple-push-server.py**

```python
from time import sleep
import sys, cefpyco
FunctionName = "/_SF_abedef_"
pushDataNameSuffix = "/Current-Temp"
pushDataNum = 5

With cefpyco.create_handle() as handle:
    Name_PushData = "ccnx:" + pushDataNameSuffix
    handle.register(Name_PushData)

    interestName = "ccnx:" + FunctionName + "/" + str(pushDataNum) + pushDataNameSuffix
    handle.send_interest(interestName, 0)

recvInterestNum = 0
while True:
    info = handle.receive()
    if info.is_succeeded and ( info.name == ("ccnx:" + pushDataNameSuffix) ):
        msg = "Current-Temp: 30 degree celsius, chunk=" + str(info.chunk_num) + "¥n"
        recvInterestNum += 1
        handle.send_data(info.name, msg, info.chunk_num, expiry=3600000, cache_time=3600000)

    if recvInterestNum == pushDataNum:
        break
```
Sample App. for Push communication

- Preparation before execution
  - at sense node (push-consumer)
    - Execute the following command to set FIB
      ```bash
      cefore:~/cefpyco$ cefroute add ccnx:/_SF_abcd_def udp [PushServer-IP-Address]
      ```
  - サーバー (push-server)
    - Execute the following command to set FIB
      ```bash
      cefore:~/cefpyco$ cefroute add ccnx:/Current-Temp udp [PushConsumer-IP-Address]
      ```
    - To cache received data, make sure the cache mode is “Localcache” by using “cefstatus” command
      ```bash
      cefore:~/cefpyco$ cefstatus
      ```
  - To cache received data, it is required to run cefnetd with CS_MODE=1 (enable Localcache).
    (or, with CS_MODE=2 (use csmgrd) )
  - How to enable Local-Cache
    - 1) Edit /usr/local/cefore/cefnedt.conf to set CS_MODE as follows:
      - CS_MODE=1
    - 2) restart cefnetd
      ```bash
      cefore:~/cefpyco$ cefnetdstop
      cefore:~/cefpyco$ cefnetdstart
      ```
Sample App. for Push communication

- Make sure before execution
  - FIB at sensor node (**push-consumer**)
    - Check using “cefstatus” command
      ```
      ...
      faceid = XX : address = [PushServer-IP]:9896 (udp)
      FIB: 1
      ccnx:/_SF_abcddef_
      Faces: XX (-s-)
      ...
      ```
  - FIB at server node (**push-server**)
    - Check using “cefstatus” command
      ```
      ...
      faceid = YY : address = [PushConsumer-IP]:9896 (udp)
      FIB: 1
      ccnx:/Current-Temp
      Faces: YY (-s-)
      ...
      ```

- The cache mode at server node is set to Localcache (or ExCache)?
  - Check using “cefstatus” command
    ```
    ...
    Cache Mode : LocalCache
    ...
    ```
Sample App. for Push communication

Execute the push commun.

1) Run the program at server node (**push-server**)

```
$ sudo ./simple-push-server.py
```

2) Run the program at senser node (**push-consumer**)

```
$ sudo ./simple-push-consumer.py
```

3) Make sure that push data is cached properly at server node

```
$ sudo cefgetfile ccnx:/Current-Temp -f ./cache-data.txt
$ cat cache-data.txt
```

- (“30 degree celsius” is displayed five times!)

If restart this scenario again

- To erase the data cache, stop and restart cefnetd at server node

```
$ sudo cefnetdstop
$ sudo cefnetdstart
$ sudo ./simple-push-server.py
```
Sample Application using Sense HAT

- Raspberry Pi (RPi) + Sense HAT
  - Sense HAT: RPI expansion board designed for use on the International Space Station.
    - Temperature • humidity • pressure sensor
    - Acceleration • gyroscope • magnetic force sensor
    - 8x8 LED Display
    - Etc.
  - Python libraries are pre-installed on RPI

Interest for getting data or controlling

Data

センサー
（RPi + Sense HAT）
“Hello world” prog. for Sense HAT

- Hello world
  - Print “Hello world!” on the LED display.

```python
main.py
from sense_hat import SenseHat
sense = SenseHat()
sense.show_message("Hello world!")
```

When executed, “Hello world” is played on the LED display
Major methods for operation (Sense HAT)

- **clear()**: illuminates (turns off) the entire LED in a single color specified.

<table>
<thead>
<tr>
<th>パラメータ名</th>
<th>型</th>
<th>説明</th>
</tr>
</thead>
<tbody>
<tr>
<td>colour</td>
<td>(r, g, b) (Tuple of int)</td>
<td>RGB (red, green, blue)  (0 \leq R, G, B \leq 255) (integer value)。If no parameters are specified, ((R,G,B)) is set to ((0,0,0)), which turns off the LED。</td>
</tr>
</tbody>
</table>

- **show_message()**: characters are streamed on the LED like an electronic bulletin board.

<table>
<thead>
<tr>
<th>パラメータ名</th>
<th>型</th>
<th>説明</th>
</tr>
</thead>
<tbody>
<tr>
<td>text_string</td>
<td>string</td>
<td>The text shown in the LED。</td>
</tr>
<tr>
<td>scroll_speed</td>
<td>float</td>
<td>Specifies the amount of time (in seconds) to shift characters by one pixel。Note that the higher the value, the slower it becomes。The default value is 0.1。</td>
</tr>
<tr>
<td>text_colour</td>
<td>(r, g, b) (Tuple of int)</td>
<td>RGB value representing the text color。The default value is ((255,255,255)) , which means white color。</td>
</tr>
<tr>
<td>back_colour</td>
<td>(r, g, b) (Tuple of int)</td>
<td>RGB representing the background color。The default value is ((0,0,0)) , which means no light emission。</td>
</tr>
</tbody>
</table>

- **set_pixel(x, y, r, g, b)**: Sets the specified coordinates to the specified color。Refer to API for more details。
Major methods for getting sensor information (Sense HAT)

<table>
<thead>
<tr>
<th>メソッド名</th>
<th>説明</th>
</tr>
</thead>
<tbody>
<tr>
<td>get_temperature()</td>
<td>temperature (°C)</td>
</tr>
<tr>
<td>get_humidity()</td>
<td>humidity (%)</td>
</tr>
<tr>
<td>get_pressure()</td>
<td>pressure (hPa)</td>
</tr>
<tr>
<td>get_orientation()</td>
<td>Rotation angle (pitch, roll, yaw (dictionary), in degrees)</td>
</tr>
<tr>
<td>get_accelerometer()</td>
<td>Rotation angle (same as above), &quot;_raw&quot; is acceleration (x,y,z (dictionary), gravitational acceleration G)</td>
</tr>
<tr>
<td>get_gyroscope()</td>
<td>Rotation angle (same as above), &quot;_raw&quot; is rotation speed (x,y,z (dictionary), radian method (per sec))</td>
</tr>
<tr>
<td>get_compass()</td>
<td>Rotation angle (same as above), &quot;_raw&quot; is magnetic force (x,y,z (dictionary), μT)</td>
</tr>
</tbody>
</table>

Code Example

t = sense.get_temperature()
print("Temp: {0:5.2f}".format(t)) # > Temp: 31.24

d = sense.get_orientation()
print("Pitch: {0:5.2f}".format(d["pitch"])) # > Pitch: 3.11
print(" Roll: {0:5.2f}".format(d["roll"])) # > Roll: 1.43
print("  Yaw: {0:5.2f}".format(d["yaw"])) # > Yaw: 91.12

d = sense.get_accelerometer_raw()
print("X: {0:5.2f}".format(d["x"])) # > X: -0.07
print("Y: {0:5.2f}".format(d["y"])) # > Y: -0.02
print("Z: {0:5.2f}".format(d["z"])) # > Z: 0.99

Sample App. using Sense HAT

Sample programs using Cefpyco/Cefore communicate with a sense HAT node.

1. LED emission application
   - Send an interest named “ccnx:/sensorN/show/msg-<str>” to display the message on the LED of SenseHAT

2. Application for obtaining sensor information
   - Send an interest named “ccnx:/sensorN/getdata/type-<str>” to get the specified type of information
   - example: “type-temperature” can be used to get temperature information

3. Application for receiving data pushed by senser node (Sense HAT)
   - Send an interest named “ccnx:/sensorN/register/id-<str>/ip-<str>” to make a FI
   - After receiving an interest named “ccnx:/polling/sensorN/<id>”, run the procedure (2) to get the data.
Naming rule: “ccnx:/sensorN/command/param1-value1/…”

- N is the id assigned to each RPi used.
- The list of available commands is shown in the table below.

The program implementing these APIs is shown later on.

<table>
<thead>
<tr>
<th>command</th>
<th>explanation</th>
<th>argument</th>
</tr>
</thead>
<tbody>
<tr>
<td>show</td>
<td>Displays the specified string on the LED</td>
<td>• msg-&lt;str&gt;: strings to display</td>
</tr>
<tr>
<td></td>
<td></td>
<td>Example: ccnx:/sensor3/show/msg-helloworld</td>
</tr>
<tr>
<td>lighton</td>
<td>Illuminates the entire LED with the specified color</td>
<td>• r-&lt;int&gt;: Red value (0~255)</td>
</tr>
<tr>
<td></td>
<td></td>
<td>• g-&lt;int&gt;: Green value (0~255)</td>
</tr>
<tr>
<td></td>
<td></td>
<td>• b-&lt;int&gt;: Blue value (0~255)</td>
</tr>
<tr>
<td></td>
<td></td>
<td>Example: ccnx:/sensor5/lighton/r-255/g-255/b-255</td>
</tr>
<tr>
<td>lightoff</td>
<td>Turns off the entire LED</td>
<td>• none</td>
</tr>
<tr>
<td>getdata</td>
<td>Returns data with the specified type of senser data as</td>
<td>• type-&lt;str&gt;: available data type</td>
</tr>
<tr>
<td></td>
<td>a string</td>
<td>• temperature</td>
</tr>
<tr>
<td></td>
<td>The string can be restored to the original</td>
<td>• humidity</td>
</tr>
<tr>
<td></td>
<td>data by literal_eval function of ast module.</td>
<td>• pressure</td>
</tr>
<tr>
<td></td>
<td>example:</td>
<td>• orientation: Rpi’s rotation angle (pitch, roll, yaw)</td>
</tr>
<tr>
<td></td>
<td>from ast import literal_eval</td>
<td>• accelerometer: acceleration (x, y, z)</td>
</tr>
<tr>
<td></td>
<td>info = h.receive()</td>
<td>• gyroscope: rotation speed (x, y, z)</td>
</tr>
<tr>
<td></td>
<td>literal_eval(info.payload_s)</td>
<td>• compass: magnetic force (x, y, z)</td>
</tr>
<tr>
<td>register</td>
<td>Registers itself with the senser as a node to get</td>
<td>• id-&lt;str&gt;: identifier of the node itself.</td>
</tr>
<tr>
<td></td>
<td>push requests from the sensor.</td>
<td>• ip-&lt;str&gt;: IP address</td>
</tr>
<tr>
<td></td>
<td></td>
<td>example: ccnx:/sensorN/register/id-ooka/ip-192.168.0.2</td>
</tr>
<tr>
<td>deregister</td>
<td>Clears the above register information</td>
<td>Same as register method</td>
</tr>
</tbody>
</table>
LED Emission

- Send an interest packet to execute show command

```python
#!/usr/bin/env python

import cefpyco

with cefpyco.create_handle() as h:
    h.send_interest("ccnx:/sensor3/show/msg-hello")
```
Application for obtaining sensor information

- Issues “getdata” command
- Obtains temperature information from the received data

```python
#!/usr/bin/env python

import cefpyco
from time import sleep
from ast import literal_eval

with cefpyco.create_handle() as h:
    h.send_interest("ccnx:/sensor3/getdata/type-temperature")
    info = h.receive()
    if info.is_data:
        print(info)
        print("Temperature: {0}".format(literal_eval(info.payload_s)))
    else:
        print("Failed to receive data.")
```
Application for receiving data pushed by senser node
(Sense HAT)

- After getting "ccnx:/polling/sensorN/<id>" from the senser node, issues “getdata” command

```python
#!/usr/bin/env python

import cefpyco
from time import sleep
from ast import literal_eval

with cefpyco.create_handle() as h:
    h.register("ccnx:/polling")
    h.send_interest("ccnx:/sensor3/register/id-ooka/ip-10.0.1.2")
    info = h.receive(timeout_ms=10000)
    if info.is_interest and info.name == "ccnx:/polling/sensor3/ooka":
        h.send_interest("ccnx:/sensor3/getdata/type-temperature")
        info = h.receive()
        if info.is_data:
            print("Temperature: {0}".format(literal_eval(info.payload_s)))
        else:
            print("Failed to receive data.")
    else:
        print("Failed to receive polling")
```
Sample app. at senser side (Sense HAT)

- app.py
  - Main function

- command.py
  - Define a parent class for sensor commands

- ledcommand.py
  - Define commands related to LED display.

- sensorcommand.py
  - Define commands related to sensor information retrieval.
app.py

#!/usr/bin/env python3

from time import sleep
from sense_emu import SenseHat
from sense_hat import SenseHat
from command import *
from ledcommand import *
from sensorcommand import *
import cefpyco

class Sensor(object):
    def __init__(self, id):
        self.id = id
        self.prefix = "ccn:/sensor%d" % self.id
        self.polling_prefix = "ccn:/polling/sensor%d" % self.id
        self.sensehat = SenseHat()
        self.polling_targets = []

    def run(self):
        with cefpyco.create_handle() as h:
            self.setup_commands(h)
            self.start_to_run()
            self.start_command.action(""")

            count = 0
            while self.is_running:
                sleep(0.1)
                count += 1
                try:
                    self.action()
                    if count > 50:
                        count = 0
                        self.polling()
                except Exception as e:
                    self.sensehat.clear()
                    self.sensehat.show_message(" ERROR ",
                       text_colour=(255, 0, 0),
                       back_colour=(0, 0, 0),
                       scroll_speed=0.05)
                    self.sensehat.clear()
                    print("Error: [Xs]" % e)

                if count > 50:
                    count = 0
                    self.polling()
            except KeyboardInterrupt as e:
                self.finish_command.action(""")

            print("Force to finish.")

    def action(self):
        info = self.cef.receive(timeout_ms=1)
        if not info.is_succeeded:
            return
        if not info.name.startswith(self.prefix):
            return
        for command in self.commands:
            command.match_prefix(info.name)
            command.action(info.name)

    def polling(self):
        print("Polling to: %s" % self.polling_targets)
        for id in self.polling_targets:
            name = "ccn:/polling/sensor%d" % id
            self.cef.send_interest(name,
                           chunk_num=-1,
                           lifetime=1)

    def setup_commands(self, cefore_handler):
        self.cef = cefore_handler
        self.start_command = ShowMessageOnStart(self)
        self.finish_command = FinishOnFinish(self)
        self.commands = []
        self.commands.append(self.finish_command)
        self.commands.append(TurnOnLightForSpecifiedTime(self))
        self.commands.append(TurnOnLight(self))
        self.commands.append(TurnOffLight(self))
        self.commands.append(ShowMessage(self))
        self.commands.append(GetSensorData(self))
        self.commands.append(RegisterToReceivePollingInterest(self))
        self.commands.append(DeregisterToReceivePollingInterest(self))

    def start_to_run(self):
        self.is_running = True

    def stop_to_run(self):
        self.is_running = False

if __name__ == "__main__":
    s = Sensor(3)
    try:
        s.run()
    except KeyboardInterrupt as e:
        s.finish_command.action(""")
        print("Force to finish.")
class ShowMessageOnStart(Command):
    def __init__(self, sensor):
        super().__init__("start", sensor)

    def action(self, name):
        self.cef.register(self.sensor.prefix)
        self.sensehat.show_message(" Start sensor {0}. ".format(self.sensor.id),
                                   text_colour=(255, 255, 255),
                                   back_colour=(63, 63, 255),
                                   scroll_speed=0.03)
        self.sensehat.clear()
        self.sensor.stop_to_run()

class ShowMessageOnFinish(Command):
    def __init__(self, sensor):
        super().__init__("finish", sensor)

    def action(self, name):
        self.sensehat.show_message(" Finish. ",
                                   text_colour=(255, 255, 255),
                                   back_colour=(63, 63, 255),
                                   scroll_speed=0.03)
        self.sensehat.clear()
        self.sensor.stop_to_run()

Sample app. at sensor side (Sense HAT)
Sample app. at sensor side (Sense HAT)

```python
# Sample app. at sensor side (Sense HAT)

ledcommand.py

#!/usr/bin/env python3

from command import Command, generate_thread
from time import sleep

# [name] ccs::sensorN/LightFor/r=0-255/g=0-255/b=0-255/time=1~10
# * r, g, b: RGB (Red, Green, Blue) color value.
# * time: How long the light stays on in seconds.
class TurnOnLightForSpecifiedTime(Command):
    def __init__(self, sensor):
        super().__init__("lightfor", sensor)

        @generate_thread
def action(self, name):
            params = self.parse_params(name)
            color = (int(params["r"]), int(params["g"]), int(params["b"]))
            time = min(10, int(params["time"]))
            self.sensehat.clear(color)
            sleep(time)
            self.sensehat.clear()

# [name] ccs::sensorN/LightOn/r=0-255/g=0-255/b=0-255
# * r, g, b: RGB (Red, Green, Blue) color value.
class TurnOnLight(Command):
    def __init__(self, sensor):
        super().__init__("lighton", sensor)

        def action(self, name):
            params = self.parse_params(name)
            color = (int(params["r"]), int(params["g"]), int(params["b"]))
            self.sensehat.clear(color)
```

```python
# [name] ccs::sensorN/LightOff

class TurnOffLight(Command):
    def __init__(self, sensor):
        super().__init__("lightoff", sensor)

        def action(self, name):
            self.sensehat.clear((0,0,0))

# [name] ccs::sensorN/Show/Msg<str>
# * msg: String value to be shown on a LED display.
class ShowMessage(Command):
    def __init__(self, sensor):
        super().__init__("show", sensor)

        @generate_thread
def action(self, name):
            params = self.parse_params(name)
            self.sensehat.clear()
            self.sensehat.show_message(params["msg"], text_colour=(255, 255, 255),
                                      back_colour=( 0, 0, 0),
                                      scroll_speed=0.06)
            self.sensehat.clear()
```
Sample app. at senser side (Sense HAT)

```python
#!/usr/bin/env python3

import subprocess
from command import Command, generate_thread
from time import sleep

# [name] ccn:<sensor>/getData/type=<str>
# * id: An identifier to be included in polling Interest's name
#     + compass (tuple of (x(float), y(float), z(float))
#     + gyroscope (tuple of (x(float), y(float), z(float))
#     + accelerometer (tuple of (x(float), y(float), z(float))
#     + pressure (float)
#     + humidity (float)
#     + temperature (float)
# * type: one of following data types:

class SensorDataExchange:
    __init__(self, name):
        super().__init__(name)
        self.sensor = self.sensehat
        self.params = params
        self.data = self.get_data_func.getData(type, self.get_none)
        self.cef.send_data(name, str(self.data), chunk_num=0)

    def action(self, id, ipaddr):
        # * ipaddr: An IPv4 address to which a polling Interest is sent.
        #   ipaddr: An IPv4 address to which a polling Interest is sent.
        #     + compass (tuple of (x(float), y(float), z(float))
        #     + gyroscope (tuple of (x(float), y(float), z(float))
        #     + accelerometer (tuple of (x(float), y(float), z(float))
        #     + pressure (float)
        #     + humidity (float)
        #     + temperature (float)
        # * type: one of following data types:

        self.sensor.polling_targets.remove(id)
        print("ID '{0}' has deregisterred.").format(id))

        return

    def action(self, id, ipaddr):
        # * ipaddr: An IPv4 address to which a polling Interest is sent.
        #   ipaddr: An IPv4 address to which a polling Interest is sent.
        #     + compass (tuple of (x(float), y(float), z(float))
        #     + gyroscope (tuple of (x(float), y(float), z(float))
        #     + accelerometer (tuple of (x(float), y(float), z(float))
        #     + pressure (float)
        #     + humidity (float)
        #     + temperature (float)
        # * type: one of following data types:

        self.sensor.polling_targets.append(id)
        print("ID '{0}' has registerred.").format(id))

        return

    def action(self, id, ipaddr):
        # * ipaddr: An IPv4 address to which a polling Interest is sent.
        #   ipaddr: An IPv4 address to which a polling Interest is sent.
        #     + compass (tuple of (x(float), y(float), z(float))
        #     + gyroscope (tuple of (x(float), y(float), z(float))
        #     + accelerometer (tuple of (x(float), y(float), z(float))
        #     + pressure (float)
        #     + humidity (float)
        #     + temperature (float)
        # * type: one of following data types:

        if id not.isalpha():
            raise Exception("Invalid ID: {0}".format(id))

        if not ip.replace("","").isdecimal():
            raise Exception("Invalid IP addr: {0}".format(ip))

        if id in self.sensor.polling_targets:
            print("ID '{0}' has already registerred.").format(id))

            return

        name = "{0}/{1}".format(self.sensor.polling_prefix, id)
        cmd = "cfroute add {0} udp {1}".format(name, ip)
        ret = subprocess.run(cmd, shell=True)

        if ret.returncode == 0:
            self.sensor.polling_targets.append(id)
            print("ID '{0}' has registerred.").format(id))

        return

        print("ID '{0}' has deregisterred.").format(id))

        return

    def action(self, id, ipaddr):
        # * ipaddr: An IPv4 address to which a polling Interest is sent.
        #   ipaddr: An IPv4 address to which a polling Interest is sent.
        #     + compass (tuple of (x(float), y(float), z(float))
        #     + gyroscope (tuple of (x(float), y(float), z(float))
        #     + accelerometer (tuple of (x(float), y(float), z(float))
        #     + pressure (float)
        #     + humidity (float)
        #     + temperature (float)
        # * type: one of following data types:

        self.sensor.polling_targets.remove(id)
        print("ID '{0}' has deregisterred.").format(id))

        return

        print("Error in executing: {0}".format(cmd)))

        print(" with return code: {0}".format(ret))
```

```python
from command import Command
from time import sleep

# [name] ccn:<sensor>/register/id=<str>/ip=<ipaddr>
# * id: An identifier to be included in polling Interest's name
#     + compass (tuple of (x(float), y(float), z(float))
#     + gyroscope (tuple of (x(float), y(float), z(float))
#     + accelerometer (tuple of (x(float), y(float), z(float))
#     + pressure (float)
#     + humidity (float)
#     + temperature (float)
# * type: one of following data types:

class RegisterToReceivePollingInterest(Command):
    __init__(self, sensor):
        super().__init__("deregister", sensor)

        def action(self, name):
            if id not.isalpha():
                raise Exception("Invalid ID: {0}".format(id))

            if not ip.replace("","").isdecimal():
                raise Exception("Invalid IP addr: {0}".format(ip))

            if id in self.sensor.polling_targets:
                print("ID '{0}' has already registerred.").format(id))

                return

            name = "{0}/{1}".format(self.sensor.polling_prefix, id)
            cmd = "cfroute add {0} udp {1}".format(name, ip)
            ret = subprocess.run(cmd, shell=True)

            if ret.returncode == 0:
                self.sensor.polling_targets.append(id)
                print("ID '{0}' has registerred.").format(id))

            return

            print("Error in executing: {0}".format(cmd)))

            print(" with return code: {0}".format(ret))
```

```python
from command import Command
from time import sleep

# [name] ccn:<sensor>/register/id=<str>/ip=<ipaddr>
# * id: An identifier to be included in polling Interest's name
#     + compass (tuple of (x(float), y(float), z(float))
#     + gyroscope (tuple of (x(float), y(float), z(float))
#     + accelerometer (tuple of (x(float), y(float), z(float))
#     + pressure (float)
#     + humidity (float)
#     + temperature (float)
# * type: one of following data types:

class RegisterToReceivePollingInterest(Command):
    __init__(self, sensor):
        super().__init__("deregister", sensor)

        def action(self, name):
            if id not.isalpha():
                raise Exception("Invalid ID: {0}".format(id))

            if not ip.replace("","").isdecimal():
                raise Exception("Invalid IP addr: {0}".format(ip))

            if id in self.sensor.polling_targets:
                print("ID '{0}' has already registerred.").format(id))

                return

            name = "{0}/{1}".format(self.sensor.polling_prefix, id)
            cmd = "cfroute del {0} udp {1}".format(name, ip)
            ret = subprocess.run(cmd, shell=True)

            if ret.returncode == 0:
                self.sensor.polling_targets.remove(id)
                print("ID '{0}' has deregisterred.").format(id))

            return

            print("Error in executing: {0}".format(cmd)))

            print(" with return code: {0}".format(ret))
```

```python
from command import Command
from time import sleep

# [name] ccn:<sensor>/register/id=<str>/ip=<ipaddr>
# * id: An identifier to be included in polling Interest's name
#     + compass (tuple of (x(float), y(float), z(float))
#     + gyroscope (tuple of (x(float), y(float), z(float))
#     + accelerometer (tuple of (x(float), y(float), z(float))
#     + pressure (float)
#     + humidity (float)
#     + temperature (float)
# * type: one of following data types:

class RegisterToReceivePollingInterest(Command):
    __init__(self, sensor):
        super().__init__("deregister", sensor)

        def action(self, name):
            if id not.isalpha():
                raise Exception("Invalid ID: {0}".format(id))

            if not ip.replace("","").isdecimal():
                raise Exception("Invalid IP addr: {0}".format(ip))

            if id in self.sensor.polling_targets:
                print("ID '{0}' has already registerred.").format(id))

                return

            name = "{0}/{1}".format(self.sensor.polling_prefix, id)
            cmd = "cfroute add {0} udp {1}".format(name, ip)
            ret = subprocess.run(cmd, shell=True)

            if ret.returncode == 0:
                self.sensor.polling_targets.append(id)
                print("ID '{0}' has registerred.").format(id))

            return

            print("Error in executing: {0}".format(cmd)))

            print(" with return code: {0}".format(ret))
```