CCNx-based Cloud-Native Function: Networking and Applications
～[2nd Part] Cefpyco: Python Compact Package for Developing Cefore Applications ～

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

2022/09/19

Contents

- 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;
}
```

33 lines→4lines

Python

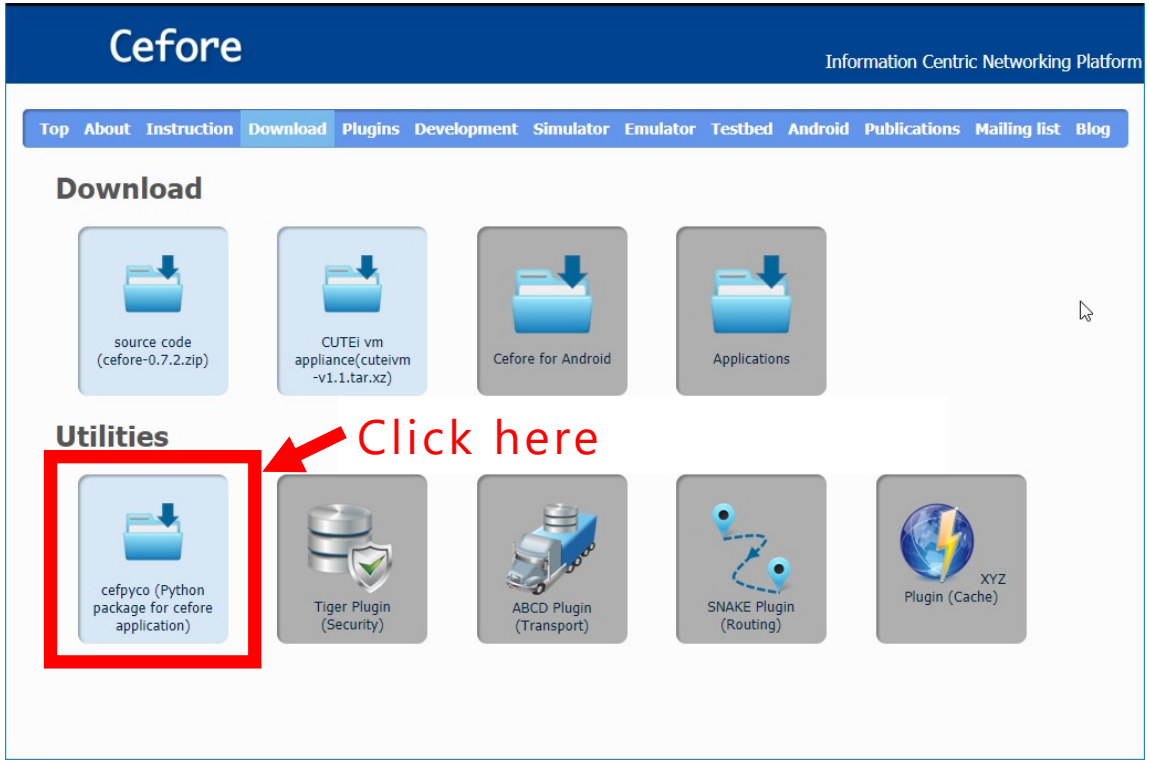
```python
import cefpyco

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 > 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)*

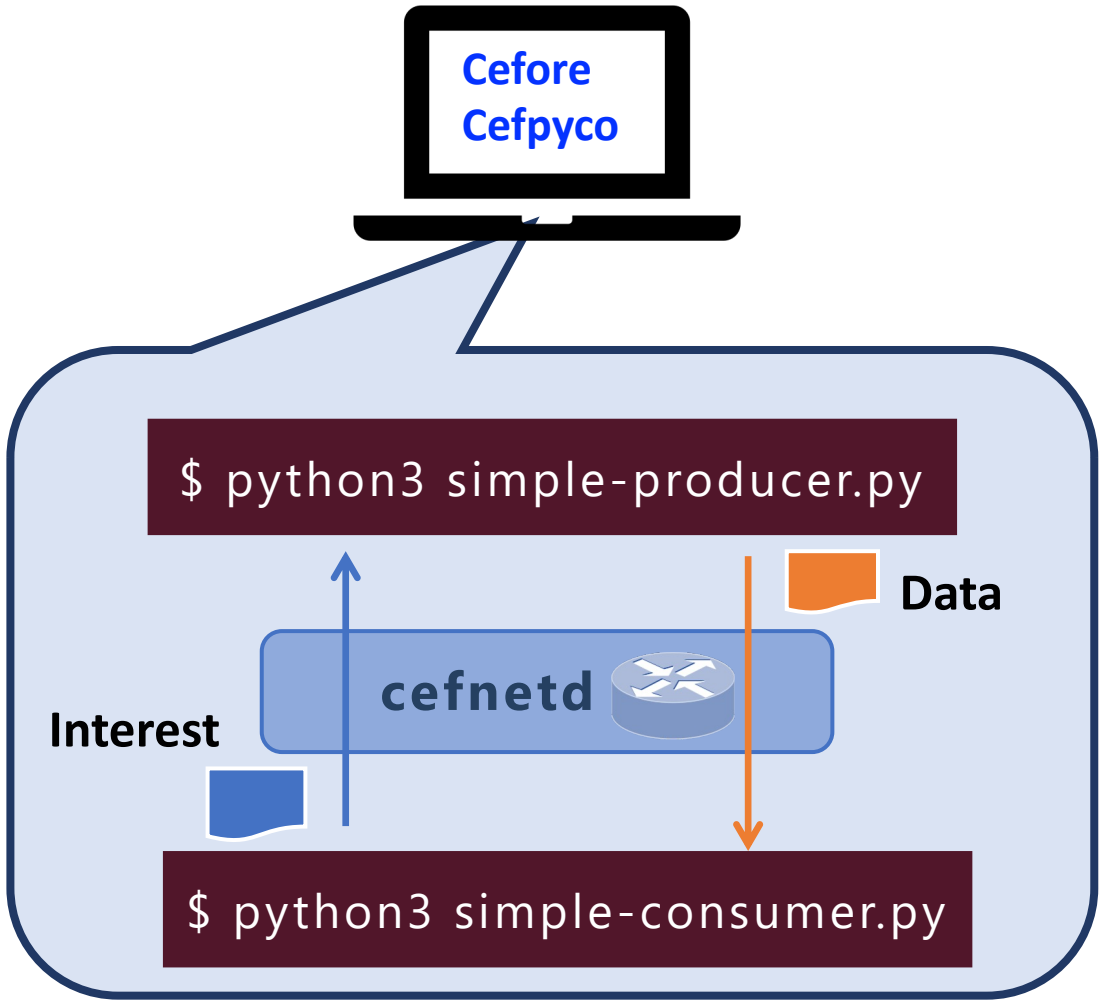
④ Receive a packet
  - *receive()*

- **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

test1.py

```
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/cefore
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

```
# 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

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

Tab文字

Error example
- 4-character Space and 2-character Space
- 4-character Space and 1-Tab

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

Without "with statement"

```
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"

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

② Send a data packet

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

test2.py

```
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=3,600,000)
```

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

```
cefore:~/cefpyco$ sudo csmgrdstart
cefore:~/cefpyco$ sudo cefnetdstart
cefore:~/cefpyco$ sudo python test2.py
…
cefore:~/cefpyco$ csmgrstatus ccnx:/
…
*****   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. conent store)

③ Send an interest packet

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

test3.py

```
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

```
# 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.

④ Receive a packet

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

test4.py

```
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
cefore:~/cefpyco$ sudo python test4.py
…
the content of info (received data packet) is displayed here
…
```

CcnPacketInfo Property

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

Implementing Simple Consumer and Producer

⑤ Simple consumer application

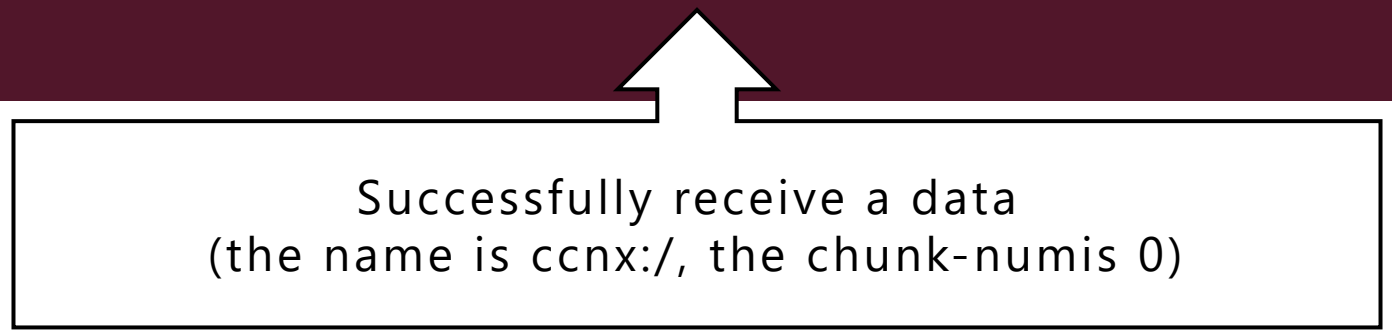
- Create a Python file simple-consumer.py with the following program
  - To receive a content of which the name ccnx:/test and the chunk-number is 0
  - Repeat the receive() method until the packet reception success
- **Let's complete simple-consumer.py below**

simple-consumer.py

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

from time import sleep
import cefpyco

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

Example of how simple-consumer works

```
cefore:~/cefpyco$ csmgrdstart
cefore:~/cefpyco$ cefnetdstart
cefore:~/cefpyco$ echo hello > test
cefore:~/cefpyco$ cefputfile ccnx:/test
[cefputfile] Start
...
[cefputfile] Throughput = 11819 bps
cefore:~/cefpyco$ python3 simple-consumer.py
[cefpyco] 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)
```



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**

simple-producer.py

```
import cefpyco

with cefpyco.create_handle() as handle:
    handle.register("ccnx:/test")
    while True:
        info = [redacted]
        if info.is_[redacted] and (info.name [redacted]) and (info.[redacted]):
            handle.send_([redacted])
            # break # Uncomment if publisher provides content once
```

Example of how simple-producer works

- After disabling csmgrd, perform the following two steps

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

(producer)

Start content distribution

```
cefore:~/cefpyco$ 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)
```

(consumer)

Successful data reception without csmgr

Cefpyco API

Cefpyco Method Name	argment・return value ("key=value" indicates an optional argument with a default value)	explanation
begin	• ceforedir=None: directory pass to cefnetd.conf • portnum=9896: port number of cefnetd	Initiates a connection to cefnetd. It's called automatically when using the "with" statement.
end	Nothong	Complete the connection to cefnetd. It's called automatically when using the "with" statement.
send_interest	• name: content name (ccnx:/...) • chunk_num=0: chunk number • symbolic_f=INTEREST_TYPE_REGULAR: Interest type • hop_limit=32: max hop number • lifetime=4000: Interest lifetime （Milliseconds from the current time）	Sends an interest packet requesting content with the specified name.
send_data	• name: content name (ccnx:/...) • payload: payload of data packet • chunk_num=-1: chunk number • end_chunk_num=-1: last chunk number of content （omitted in the case of negative numbers） • hop_limit=32: max hop number • expirty=36000000: content expiration time （Milliseconds from the current time） • cache_time=-1: recommended cache time （omitted in the case of negative numbers）	Send a data packet with the specified name and payload
receive	• error_on_timeout=false: whether or not to return an error on timeout • timeout_ms=4000: time between the start of reception and the timeout （milliseconds） • return value: CcnPacketInfo （refere to another slide）	Waits for a specified time for an interest or data packet, and returns information about the received packet
register	• name: specify the name prefix of the interest you want to receive	Register the prefix name of interest you want to receive with the cefnetd, and enables the application to receive the interest by receive().
deregister	• name: specify the prefix name you want to unregister	Unrester the name prefix registered by register() method.

Sample Application using Cefpyco/Cefore

Sample app.: CefApp

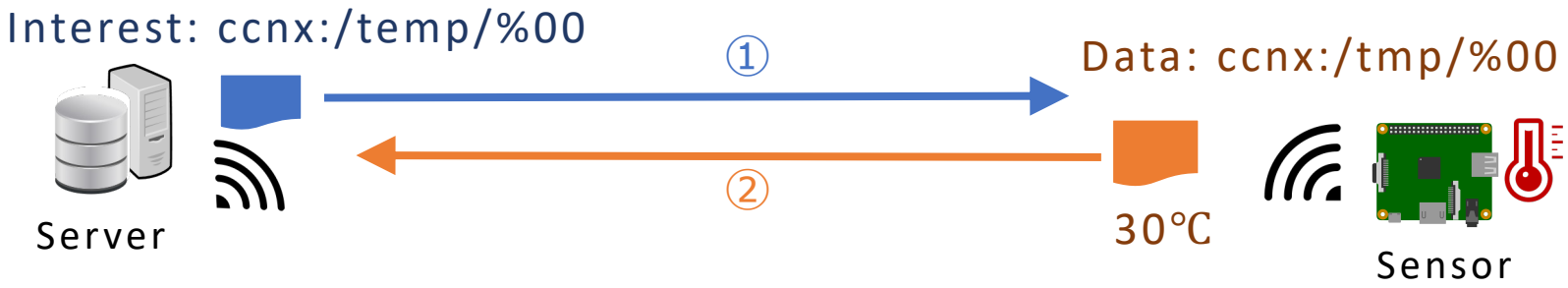
- Provide two appli. in cefpyco/cefapp directory
  - cefappconsumer.py: consumer appli
  - cefappproducer.py: producer appli

- 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

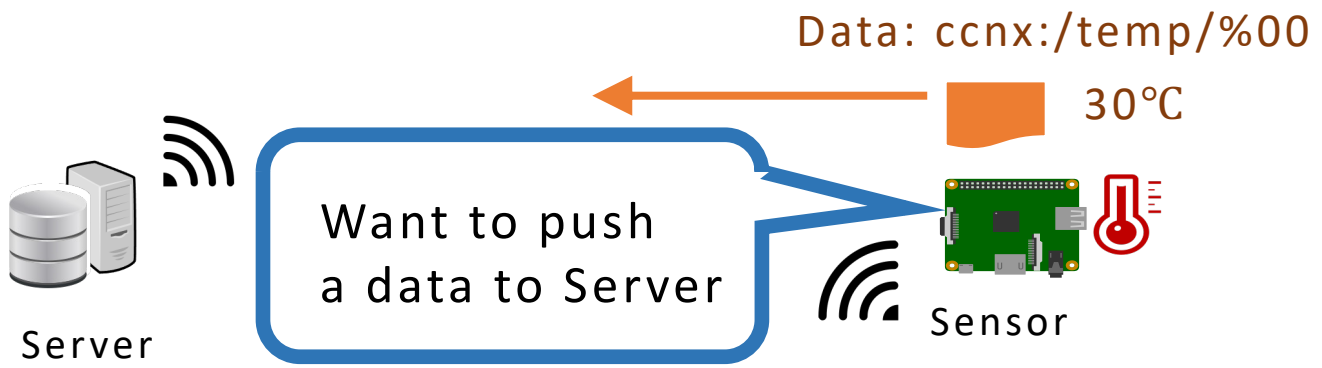
- Refer to README for more details
  - https://github.com/cefore/cefpyco/blob/master/README.md

Sample app.: Push communication

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



- Push communication is often required：



Sample app. for push communication

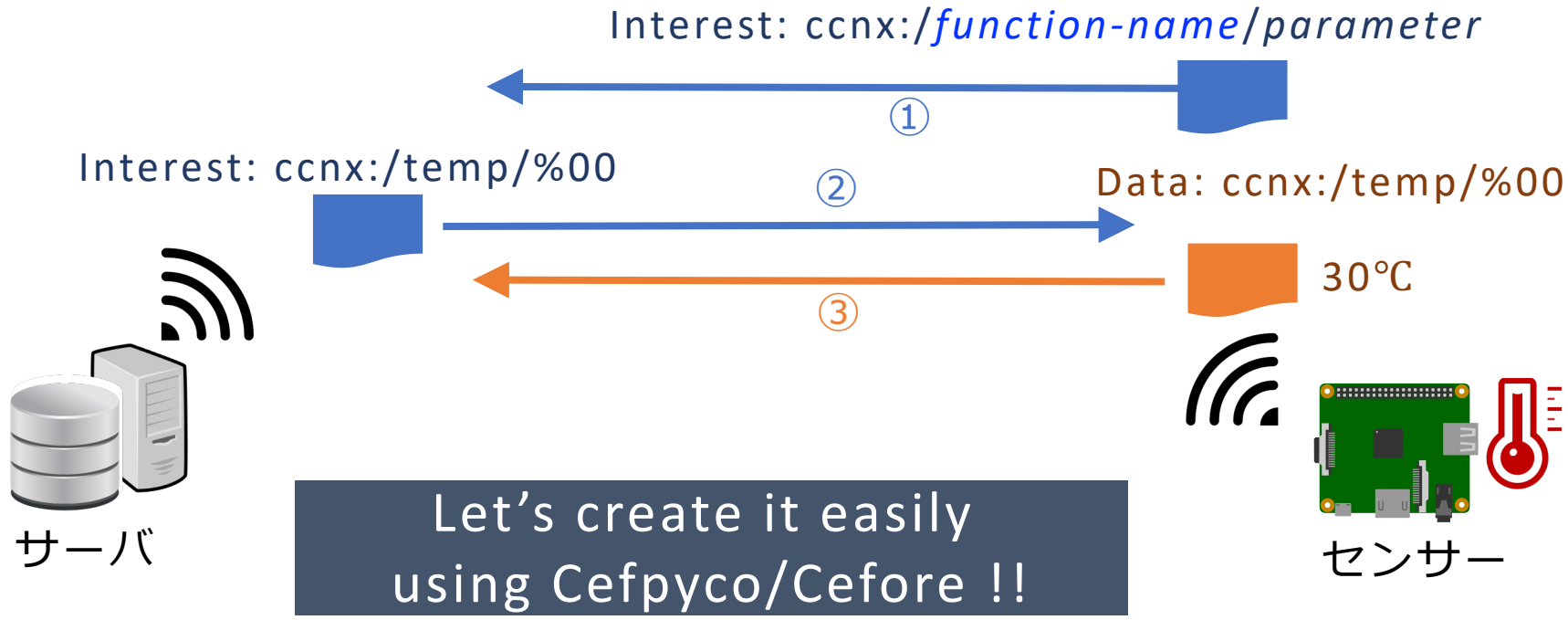
- **Method 1 ：**
  - The senser node puts information on the interest and sends it to the server.

- 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

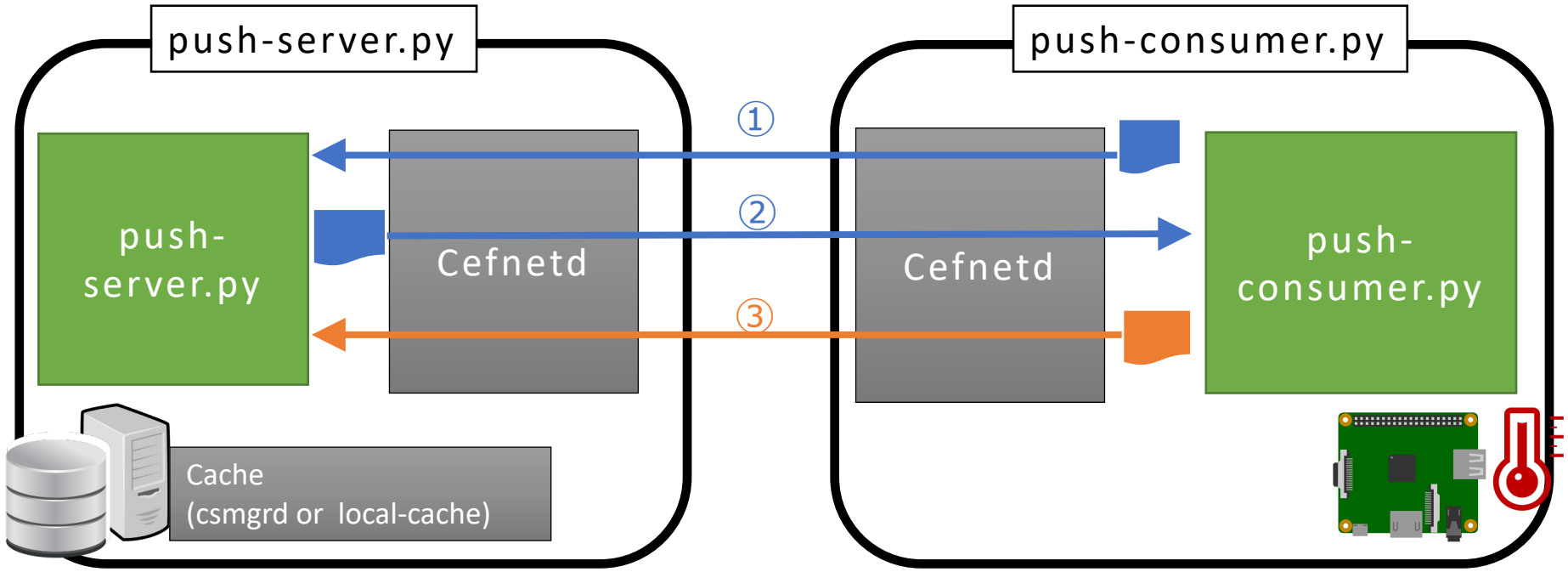
- **Method 2：**
  - Pull communication is considered a network function that pulls data from a cache in a network

  - Define a network function that pushs some data to a network, and call the function name.



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 senser node to push data
  - ③ The senser sends a data (string: 30 degree celsius) to the server



(※ 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_abcdef_/K/NAME/%00

① Function Name(_SF_xxxx_)

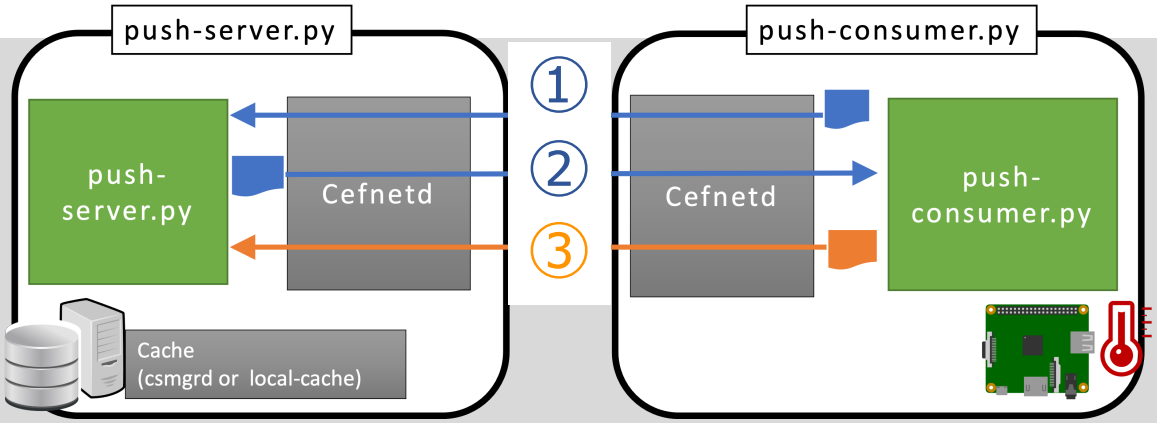
② total number of chunks(K)

③ Name of the data to push (NAME)

- Example of interest name to call a push function
  - ccnx:/_SF_abcdef_/5/Current-Temp/%00

Sample App. for Push communication

- Example: simple-push-server.py




```
import cefpyco
import re
FunctionName = "/_SF_abcdef_”

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 info.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
```

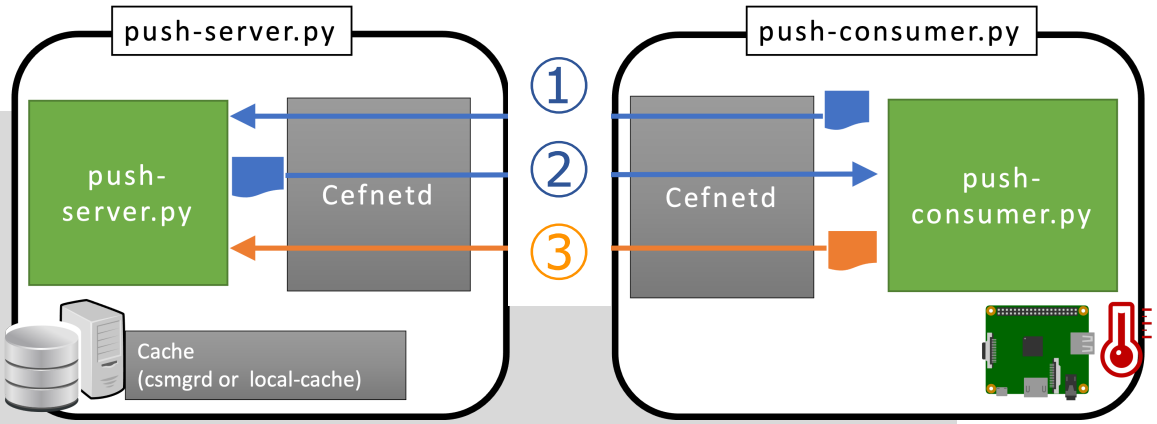
①

②

③

Sample App. for Push communication

- Example: simple-push-server.py




```
from time import sleep
import sys, cefpyco
FunctionName = "/_SF_abcdef_"
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
```

②

①

②

③

Sample App. for Push communication

- Preparation before execution
  - at sense node (**push-consumer**)
    - Execute the following command to set FIB

```
cefore:~/cefpyco$ cefroute add ccnx:/_SF_abcdef udp [PushServer-IP-Address]
```

  - サーバー(**push-server**)
    - Execute the following command to set FIB

```
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

```
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/cefnetd.conf to set CS_MODE as follows:
        - CS_MODE=1
      - 2) restart cefnetd

```
cefore:~/cefpyco$ cefnetdstop
cefore:~/cefpyco$ cefnetdstart
```

Sample App. for Push communication

- Make sure before execution
  - FIB at sensor node (**push-consumer**)
    - Check using ”cefstatus” comand

```
…
faceid = XX : address = [PushServer-IP]:9896 (udp)
FIB: 1
        ccnx:/_SF_abcdef_
                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**)

```
cefore:~/cefpyco$ sudo ./simple-push-server.py
```

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

```
cefore:~/cefpyco$ sudo ./simple-push-consumer.py
```

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

```
cefore:~/cefpyco$ sudo cefgetfile ccnx:/Curremt-Temp –f ./cache-data.txt
cefore:~/cefpyco$ cat cache-data.txt
```

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

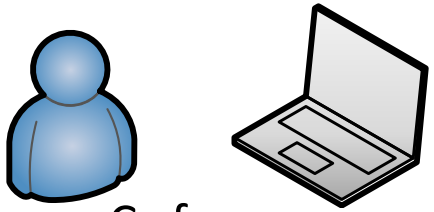
If restart this scenario again

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

```
cefore:~/cefpyco$ sudo cefnetdstop
cefore:~/cefpyco$ sudo cefnetdstart
cefore:~/cefpyco$ sudo ./simple-push-server.py
```

Sample Application using Sense HAT

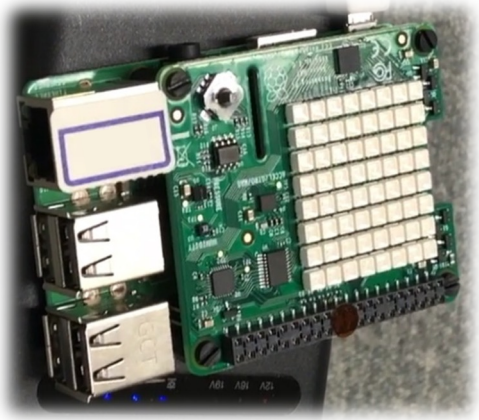
- Raspberry Pi (RPi) + Sense HAT
  - Sense HAT: RPI expansion board designed for use on the International Space Station.
    - Temperature・humidity・pressure sensor
    - Acceleration・gyroscope・magnetic force sensor
    - 8x8 LED Display
    - Etc.
  - Python libraries are pre-installed on RPI



Cefore・
cefpyco

Interest for getting data or controlling

Data



センサー
（RPi + Sense HAT）

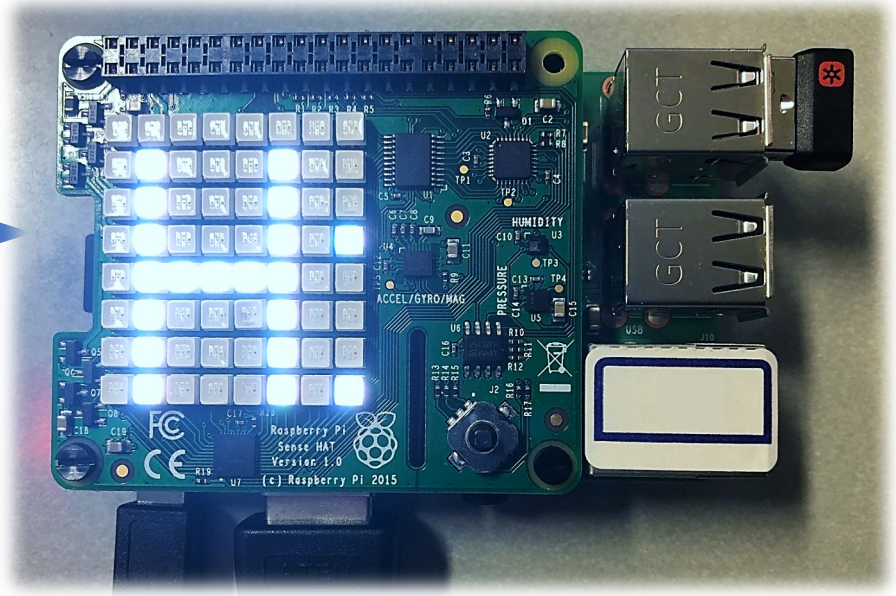
“Hello world” prog. for Sense HAT

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

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 (truns off) the entire LED in a single color specifiled.

パラメータ名	型	説明
colour	**(r, g, b)** **(Tuple of int)**	RGB (red, green, blue) 、 0<= R,G,B < =255 (integer value)。 If no parameters are specified, (R,G,B) is set to (0,0,0), which turns of the LED。

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

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

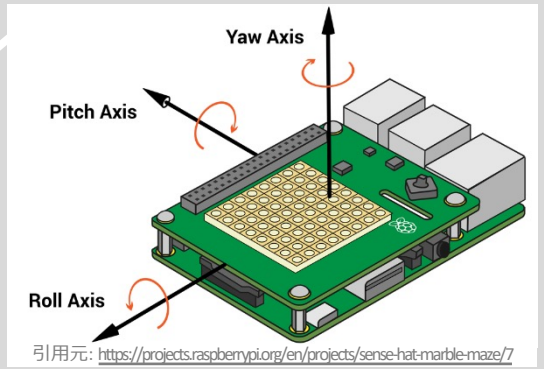
- **set_pixel(x, y, r, g, b)**：Sets the specified coordinates to the specified color. (refer to API for more details)

Major methods for getting senser information (Sense HAT)

メソッド名	説明
get_temperature()	temperature（℃）
get_humidity()	humidity（%）
get_pressure()	pressure（hPa）
get_orientation()	Rotation angle（pitch, roll, yaw (dictionary)、in degrees）
get_accelerometer()	Rotation angle（same as above）、”_raw” is acceleration（x,y,z (dictionary)、gravitational acceleration G）
get_gyroscope()	Rotation angle（same as above）、”_raw” is rotation speed（x,y,z (dictionary)、radian method (per sec)
get_compass()	Rotation angle（same as above）、”_raw” is magnetic force（x,y,z (dictionary)、μT)

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
```



引用元: https://projects.raspberrypi.org/en/projects/sense-hat-marble-maze/7

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.

API of Sample Senser App.

- 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.

command	explanation	argment
show	Displays the specified string on the LED	• msg-<str>: strings to display 例：ccnx:/sensor3/show/msg-helloworld
lighton	Illuminates the entire LED with the specified color	• r-<int>: Red value（0~255） • g-<int>: Green value（0~255） • b-<int>: Blue value（0~255） 例：ccnx:/sensor5/lighton/r-255/g-255/b-255
lightoff	Turns off the entire LED	• none

getdata	Returns data with the specified type of senser data as a string
	The string can be restored to the original data by literal_eval function of ast module.
	example：
	from ast import literal_eval
	info = h.receive()
	literal_eval(info.payload_s)
		• type-<str>：available data type
		  • temperature
		  • humidity
		  • pressure
		  • orientation: Rpi's rotation angle（pitch, roll, yaw）
		  • accelerometer: accceleration (x, y, z)
		  • gyroscope: rotation speed (x, y, z)
		  • compass: magnetic force (x, y, z)

register	Registers itself with the senser as a node to get push requests from the sensor.	• id-<str>：identifier of the node itself. • ip-<str>：IP address example：ccnx:/sensorN/register/id-ooka/ip-192.168.0.2
deregister	Clears the above register information	Same as register method

LED Emission

- Send an interest packet to execute show command

```
#!/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

```
#!/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

```
#!/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.")
        h.send_interest("ccnx:/sensor3/deregister/id-ooka/ip-10.0.1.2")
    else:
        print("Falied 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.

Sample app. at senser side (Sense HAT)

app.py

```python
#!/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: [%s]" % e)
```

```python
    def action(self):
        info = self.cef.receive(timeout_ms=1)
        if not info.is_succeeded:
            return
        # print(info)
        if not info.name.startswith(self.prefix):
            return
        for command in self.commands:
            if 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 = "{0}/{1}".format(self.polling_prefix, 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)
        finish_command = ShowMessageOnFinish(self)
        self.finish_command = finish_command
        self.commands = []
        self.commands.append(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.")
```

Sample app. at senser side (Sense HAT)

command.py

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

from threading import Thread
from time import sleep

def generate_thread(func):
    def run_thread(self, *args):
        t = Thread(target=func, args=(self, *args))
        t.start()
    return run_thread

class Command(object):
    def __init__(self, id, sensor):
        self.prefix = sensor.prefix + "/" + id
        self.sensor = sensor
        self.sensehat = sensor.sensehat
        self.cef = sensor.cef

    def action(self, name):
        raise NotImplementedError()

    def match_prefix(self, name):
        return name.startswith(self.prefix)

    def parse_params(self, name):
        params = {}
        param_part = name[len(self.prefix)+1:]
        if len(param_part) == 0:
            return params
        for param in param_part.split("/"):
            keyval = param.split("-", maxsplit=1)
            print("[keyval]:", keyval)
            if len(keyval) <= 1:
                continue
            params[keyval[0]] = keyval[1]
        return params
```

```
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()

# [name] ccn:/sensorN/finish
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 senser side (Sense HAT)

ledcommand.py

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

from command import Command, generate_thread
from time import sleep

# [name] ccn:/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] ccn:/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] ccn:/sensorN/lightoff
class TurnOffLight(Command):
    def __init__(self, sensor):
        super().__init__("lightoff", sensor)

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


# [name] ccn:/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)

sensorcommand.py

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

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

# [name] ccn:/sensorN/getdata/type-<str>
# * type: one of following data types:
#     + temperature (float)
#     + humidity (float)
#     + pressure (float)
#     + orientation (tuple of (pitch(float), roll(float), yaw(float))
#     + accelerometer (tuple of (x(float), y(float), z(float))
#     + gyroscope (tuple of (x(float), y(float), z(float))
#     + compass (tuple of (x(float), y(float), z(float))
class GetSensorData(Command):
    def __init__(self, sensor):
        super().__init__("getdata", sensor)
        funcs = {}
        funcs["temperature"] = self.sensehat.get_temperature
        funcs["humidity"] = self.sensehat.get_humidity
        funcs["pressure"] = self.sensehat.get_pressure
        funcs["orientation"] = self.sensehat.get_orientation
        funcs["accelerometer"] = self.sensehat.get_accelerometer_raw
        funcs["gyroscope"] = self.sensehat.get_gyroscope_raw
        funcs["compass"] = self.sensehat.get_compass_raw
        self.get_data_funcs = funcs
        self.get_none = lambda: None

    def action(self, name):
        params = self.parse_params(name)
        datatype = params["type"]
        get_data = self.get_data_funcs.get(datatype, self.get_none)
        data = get_data()
        self.cef.send_data(name, str(data), chunk_num=0)

# [name] ccn:/sensorN/register/id-<str>/ip-<ipaddr>
# * id: An identifier to be included in polling Interest's name
# * ipaddr: An IPv4 address to which a polling Interest is sent.
class RegisterToReceivePollingInterest(Command):
    def __init__(self, sensor):
        super().__init__("register", sensor)

    def action(self, name):
        params = self.parse_params(name)
        id = params["id"]
        ip = params["ip"]
        if not id.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 = "cefroute 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}' is registerred.".format(id))
        else:
            print("Error in executing: {0}".format(cmd))
            print("  with return code: {0}".format(ret))


# [name] ccn:/sensorN/deregister/id-<str>/ip-<ipaddr>
# * id: An identifier to be included in polling Interest's name
#       (e.g., ccn:/polling/sensor3/aooka).
#       MUST use only alphabets ([A-Za-z]+).
# * ipaddr: An IPv4 address to which a polling Interest is sent.
class DeregisterToReceivePollingInterest(Command):
    def __init__(self, sensor):
        super().__init__("deregister", sensor)

    def action(self, name):
        params = self.parse_params(name)
        id = params["id"]
        ip = params["ip"]
        if not id.isalpha():
            raise Exception("Invalid ID: {0}".format(id))
        if not ip.replace(".", "").isdecimal():
            raise Exception("Invalid IP addr: {0}".format(ip))
        if id not in self.sensor.polling_targets:
            print("ID '{0}' is not registerred.".format(id))
            return
        name = "{0}/{1}".format(self.sensor.polling_prefix, id)
        cmd = "cefroute 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}' is deregisterred.".format(id))
        else:
            print("Error in executing: {0}".format(cmd))
            print("  with return code: {0}".format(ret))
```