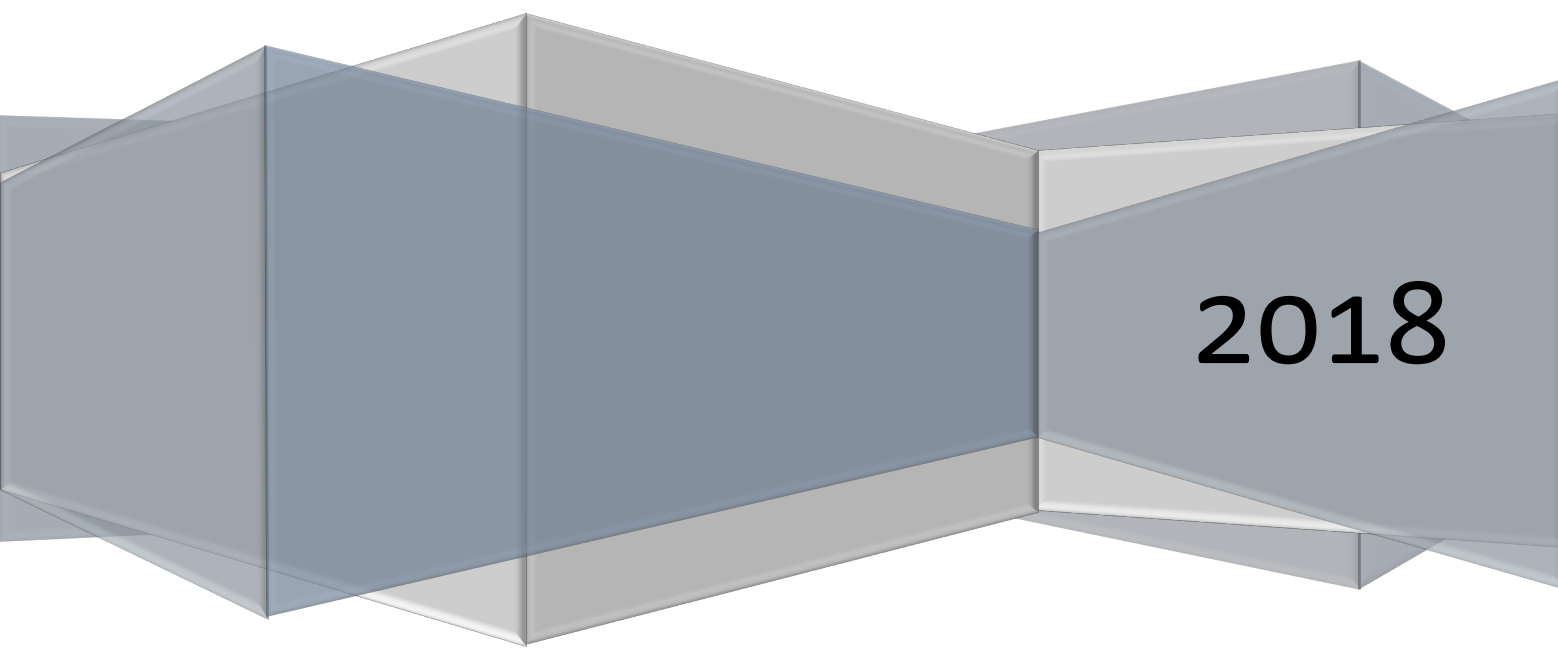


USIM Modifier

Tech Note

Ming

An abstract graphic composed of several overlapping, semi-transparent, light blue and grey geometric shapes, primarily hexagons and pentagons, arranged in a horizontal, somewhat symmetrical pattern. The shapes have a 3D effect with subtle shadows and highlights.

2018

Content

Content	1
History	2
Abstract	3
Diagram	3
File Structure	4
State machine	5
Logging mechanism	6
Extend the locale resource	7
Implement new plugin.....	7
Template	7
Concept	9
Connection Class	9
select	10
verify.....	10
read_binary	10
update_binary	11
read_record	11
update_record	11
API	12
Select USIM File	12
mf.....	12
adfusim	12
select_file_in_mf	12
select_file_in_adf	13
Convert	13
convert_arguments_to_dict	13
convert_bcd_to_string.....	13
convert_string_to_bcd.....	14
convert_alpha_to_string	14
convert_dialing_number_to_string	14
FCP	14
get_record_count	15
get_data_length.....	15
get_pin1_status	15
search_fcp_content	15

History

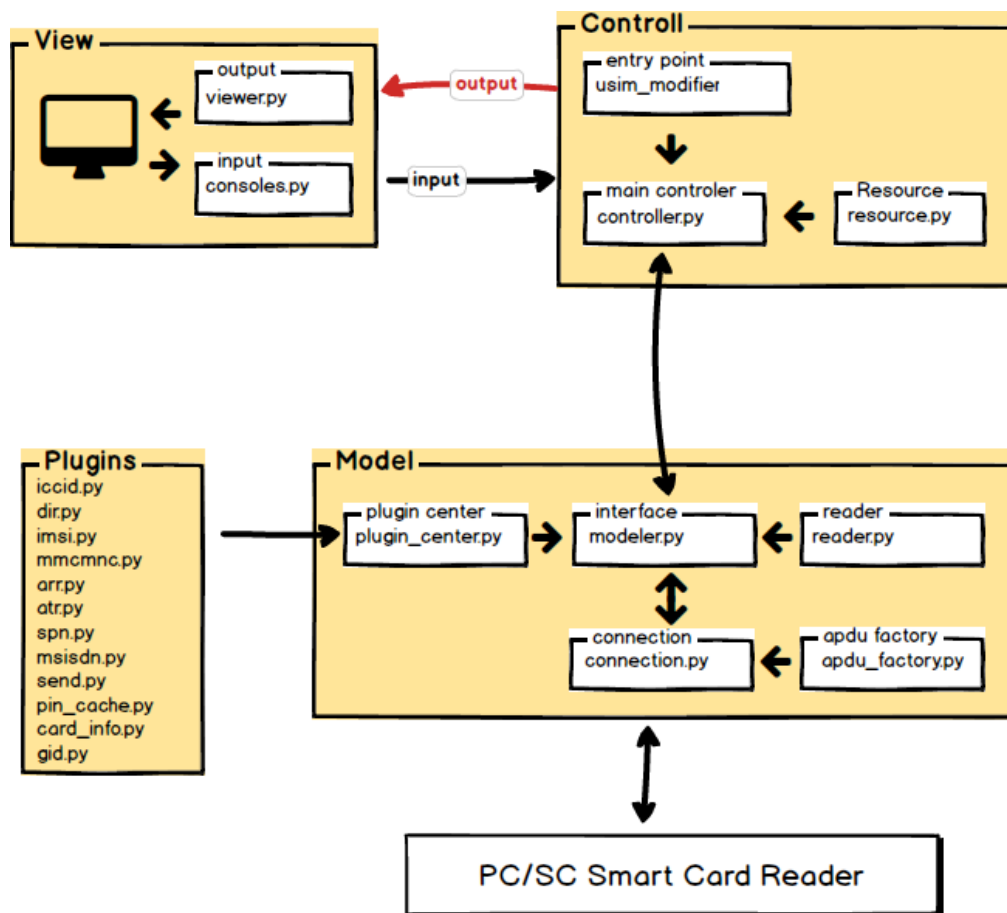
Revision	Common	Date
V1.0	Initial release	2019/01/25

Abstract

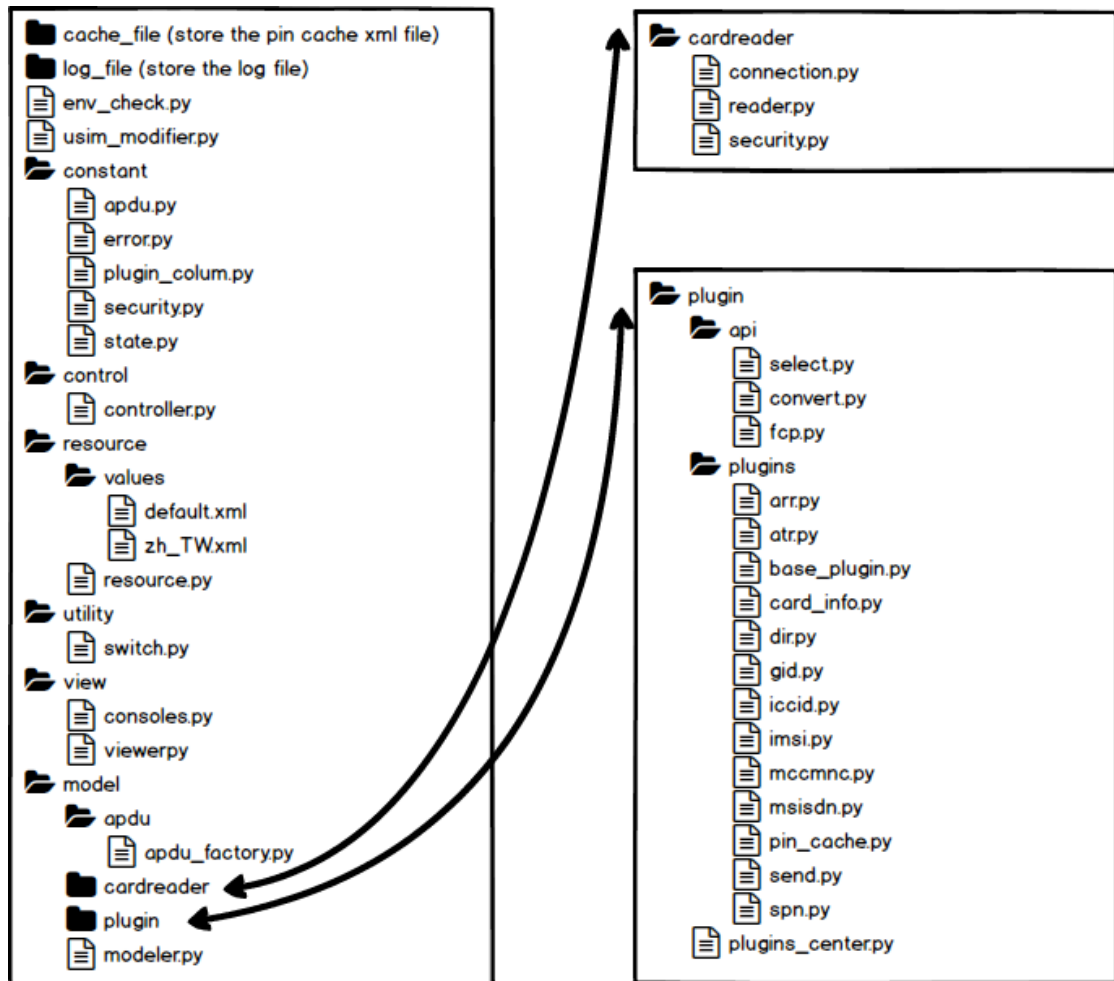
The document will describe the diagram, file structure, state machine, logging mechanism, extend the locale resource, how to implement new plugin and some public API for plugin develop.

Diagram

Just show the concept of the architecture, but my design should not a standard model. ><



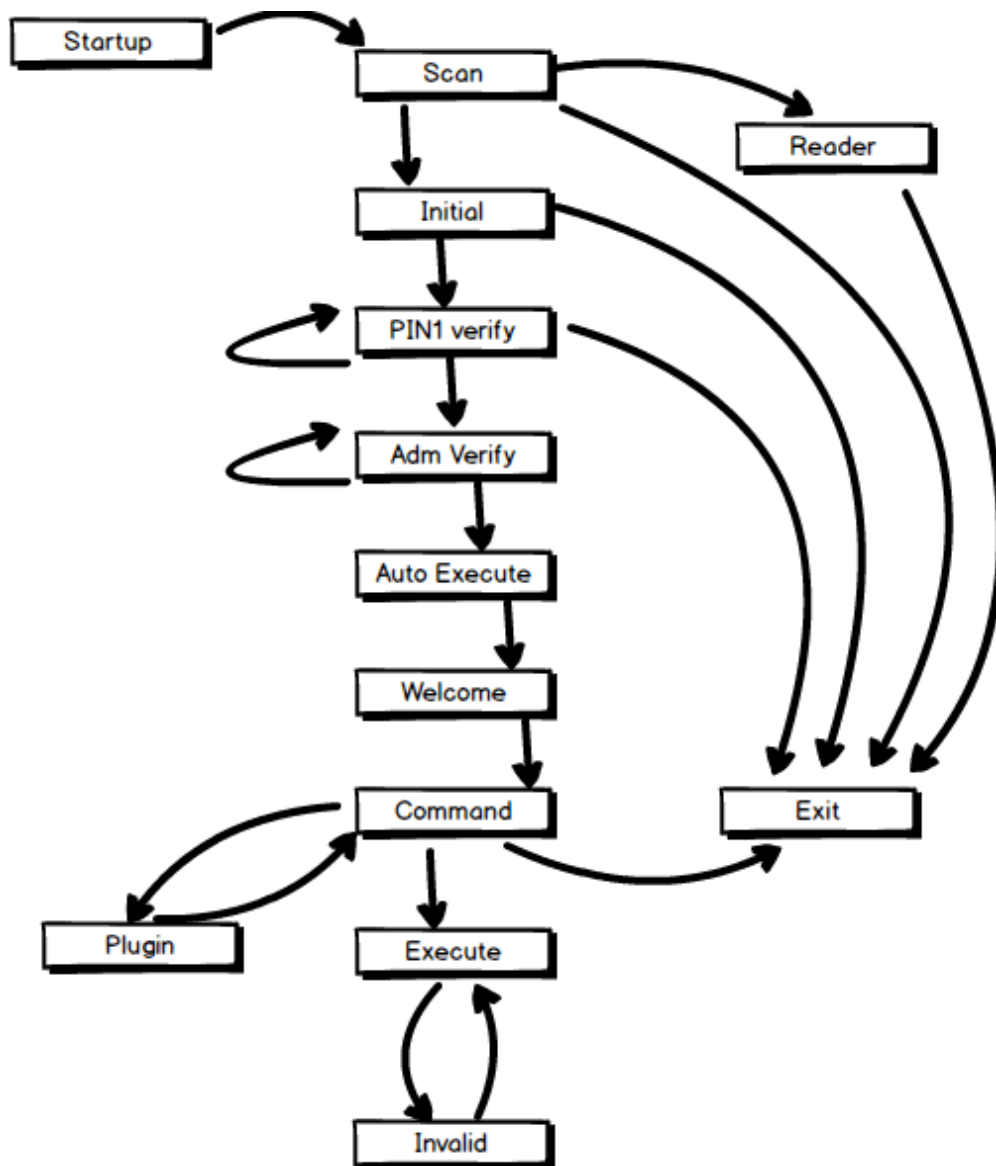
File Structure



State machine

It's a simple state machine of "usim_modifier", and you may saw a strange statue and called "Reader".

The state was not implemented and my aim is support multiple card reader and let use can select which reader will be to use, but ... I missed that. XDD



Logging mechanism

The “usim_modifier” was supported logging mechanism, the logs will store to “log_file” folder and follow “YYYYMMDD-HHMMSS.log” format as the filename, you can see the below example.

12-24 14:30:57 controller.py	DEBUG	STATE.STARTUP
12-24 14:30:57 controller.py	DEBUG	STATE.SCAN
12-24 14:30:57 controller.py	DEBUG	STATE.INITIAL
12-24 14:30:57 connection.py	DEBUG	open() > Gemalto PC Twin Reader
12-24 14:30:57 security.py	DEBUG	__init__
12-24 14:30:57 security.py	DEBUG	__query_pin1_status
12-24 14:30:57 connection.py	DEBUG	select() > File ID: 3F00, P1: 00, P2: 04
12-24 14:30:57 apdu_factory.py	DEBUG	SELECT
12-24 14:30:57 connection.py	DEBUG	>>> 00 A4 00 04 02 3F 00
12-24 14:30:57 connection.py	DEBUG	<<< , 61 1F
12-24 14:30:57 apdu_factory.py	DEBUG	GET RESPONSE
12-24 14:30:57 connection.py	DEBUG	>>> 00 C0 00 00 1F

If you want to implement the plugin by self, I suggest you can follow below **yellow line** to enable the “logging” mechanism.

```
import logging
import os

class your_class_name(base_plugin):
    def __init__(self):
        # configured the “name”
        self.__logging = logging.getLogger(os.path.basename(__file__))
        # write the message by “DEBUG” level
        self.__logging.debug("__init__()")
```

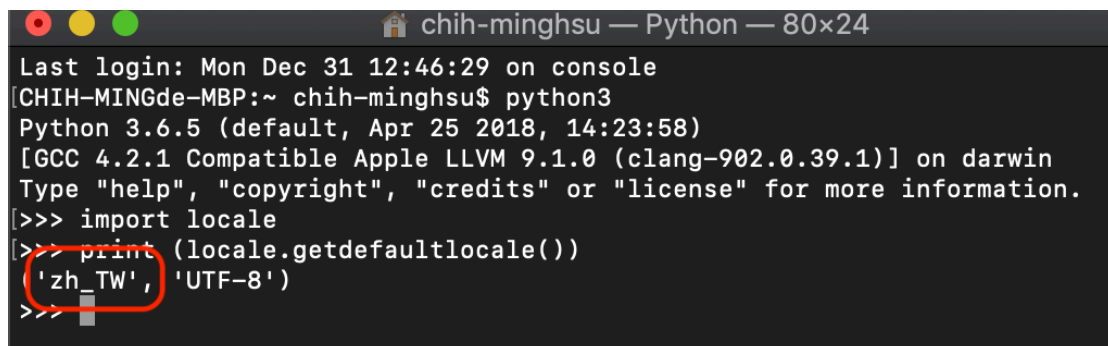
Extend the locale resource

This project can support multiple language (English & zh_TW) to core system, but can't support "plugin" mechanism.

The locale resource is based on ".xml" file, using "English" as the default language, and you can found the "default.xml" on "resource/values" folder.

If system can't figure out the resource by current locale resource, will try to search the resource of English, if still not exist, will return the error message.

To identity the system locale, please follow below steps to check the locale of your system, by the example, the "zh_TW" is locale configuration of my system.

A terminal window titled "chih-minghsu — Python — 80x24" showing a series of commands and their outputs. The commands are: "python3", "import locale", and "print(locale.getdefaultlocale())". The output of the last command is "['zh_TW', 'UTF-8']", which is highlighted with a red circle.

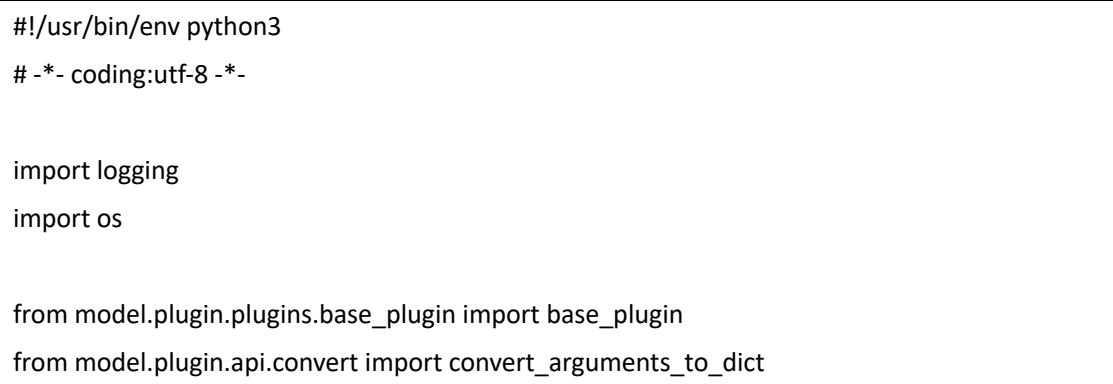
```
Last login: Mon Dec 31 12:46:29 on console
[CHIH-MINGde-MBP:~ chih-minghsu$ python3
Python 3.6.5 (default, Apr 25 2018, 14:23:58)
[GCC 4.2.1 Compatible Apple LLVM 9.1.0 (clang-902.0.39.1)] on darwin
Type "help", "copyright", "credits" or "license" for more information.
[>>> import locale
[>>> print(locale.getdefaultlocale())
['zh_TW', 'UTF-8']
[>>>
```

After confirmed the locale configuration, you can copy the "default.xml" and rename to "[locale configuration].xml", then translate all the English by self.

Implement new plugin

Template

Please copy the "plugin_template.py" on "docs" folder, we will base on this template to implement a new plugin and called "hello".

A code block containing the template for a new plugin. The code starts with a shebang line, a coding declaration, and imports for logging, os, and the base plugin class and convert function.

```
#!/usr/bin/env python3
# -*- coding:utf-8 -*-

import logging
import os

from model.plugin.plugins.base_plugin import base_plugin
from model.plugin.api.convert import convert_arguments_to_dict
```



```
class plugin_template(base_plugin): → Step 1
```

```
def __init__(self):  
    self.__logging = logging.getLogger(os.path.basename(__file__))
```

```
def summary(self):
```

```
    return "Please fill the summary of this plugin" → Step 2
```

```
def version(self):
```

```
    return "1.00"
```

```
def help(self):
```

```
    return ("Please fill the help message of this plugin\n"
```

```
        "\n"
```

```
        "The method is not mandatory function.\n"
```

```
        "If not provide 'help()' function,\n"
```

```
        "Will show 'summary()' directly."). → Step 3
```

```
@property
```

```
def auto_execute(self):
```

```
    # To decide this plugin will auto execute on system startup or not.
```

```
    return False
```

```
@property
```

```
def sort_index(self):
```

```
    # The sort index can let you decide the priority of auto execution.
```

```
    return 0xFFFF
```

```
def execute(self, arg_connection, arg_parameter=""):
```

```
    self.__logging.debug("execute()")
```

```
    ret_content = "Please fill the return content as default." → Step 4
```

```
    # Converted all parameters to dict type
```

```
    dict_args = convert_arguments_to_dict(arg_parameter)
```

```
    # figure out the key & value of parameters
```

```
    for key, value in dict_args.items():
```

```
        pass
```

```
return ret_content
```

First step, copy “docs/plugin_template.py” file to “model/plugin/plugins” folder, and rename to “hello.py”, for other steps, please follow below table to modify that.

STEP 1	To modify the class name from “plugin_template” to “hello”
STEP 2	To modify the return message to “Summary of hello plugin.”
STEP 3	To modify the return message to “Help of hello plugin.”
STEP 4	To modify the “ret_conent” variable to “Hello Plugin.”

The final step is test the “hello” plugin, see ... it’s very easy to implement new plugin for “usim_modifier”.

```
USIM modifier$ plugin

hello 1.00 > Summary of hello plugin.
iccid 1.00 > Display or modify the value of ICCID.
spn 1.00 > Display or modify the value of SPN.
dir 1.00 > Displayed all contents of EF_DIR file.
card_info 1.00 > Displayed the current status of USIM.
send 1.00 > Send the APDU command to USIM directly
mccmnc 1.00 > Display or modify the value of MCC/MNC.
atr 1.00 > Displayed the value of Answer To Reset (ATR).
pin_cache 1.00 > Cache the PIN1/ADM code to xml file for future verify automatically.
arr 1.00 > Displayed all contents of EF_ARR file.
gid 1.00 > Display or modify the value of GID1/GID2.
msisdn 1.00 > Display or modify the value of MSISDN.
imsi 1.00 > Display or modify the value of IMSI.

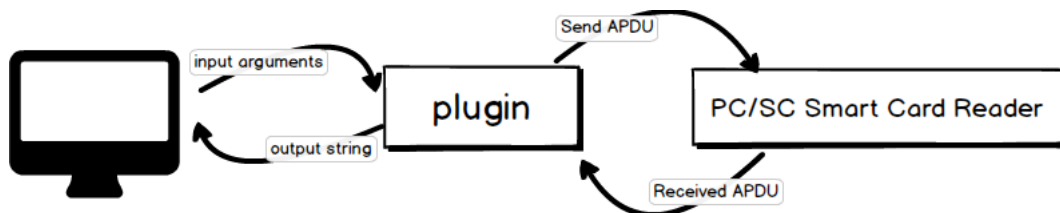
輸入 '[plugin name] help' 取得 plugin 的更多資訊
USIM modifier$ hello

Hello Plugin.
USIM modifier$ hello help

Help message of hello plugin
USIM modifier$
```

Concept

The concept is very simple, we just pass the arguments and get the feedback from plugin by “**STRING**” type.



Connection Class

All communication with UISM will through “connection” class to access.

select

Function	select(arg_field, arg_p1_coding=CODING_P1_SELECT.SEL_BY_FILE_ID.value, arg_p2_coding=CODING_P2_SELECT.SEL_RETURN_FCP.value)
Comment	Selects a file according to the methods described
Note	Arguments <ul style="list-style-type: none">- arg_field: File ID, DF name, or path to file, according to P1- arg_p1: Selection control, refer "CODING_P1_SELECT"- arg_p2: Selection control, refer "CODING_P2_SELECT" Response <ul style="list-style-type: none">- response: Response APDU from USIM- sw1: 1st byte of status word.- sw2: 2nd byte of status word.

verify

Function	verify(arg_verify_type, arg_verify_key):
Comment	Verify PIN1/ADM key
Note	Arguments <ul style="list-style-type: none">- arg_verify_type: PIN1 (0x01) or ADM (0x0A)- arg_arg_verify_key: A list of bytes as verify code Response <ul style="list-style-type: none">- response: Response APDU from USIM- sw1: 1st byte of status word.- sw2: 2nd byte of status word.

read_binary

Function	read_binary(arg_length)
Comment	Read binary data from USIM file
Note	Arguments <ul style="list-style-type: none">- arg_length: Number of bytes to be read Response <ul style="list-style-type: none">- response: Response APDU from USIM- sw1: 1st byte of status word.- sw2: 2nd byte of status word.

update_binary

Function	update_binary(arg_update_content)
Comment	Update binary data to USIM file
Note	Arguments <ul style="list-style-type: none">- arg_update_content: Bytes of data to be update Response <ul style="list-style-type: none">- response: Response APDU from USIM- sw1: 1st byte of status word.- sw2: 2nd byte of status word.

read_record

Function	read_record(arg_idx, arg_length):
Comment	Read data from assigned record number
Note	Arguments <ul style="list-style-type: none">- arg_idx: Record number- arg_length: Number of bytes to be read Response <ul style="list-style-type: none">- response: Response APDU from USIM- sw1: 1st byte of status word.- sw2: 2nd byte of status word.

update_record

Function	update_record(arg_idx, arg_update_record):
Comment	Update data to assigned record number
Note	Arguments <ul style="list-style-type: none">- arg_idx: Record number- arg_update_record: Bytes of the data to be update Response <ul style="list-style-type: none">- response: Response APDU from USIM- sw1: 1st byte of status word.- sw2: 2nd byte of status word.

API

From previous chapter, it just shows “how to add new plugin”, but didn’t provide any relate operation of the card reader, so this chapter will describe supported API of “usim_modifier”, let you can easy and quickly to implement new plugin to access USIM.

Select USIM File

The method let you can select USIM file, but only support those files under MF or ADF USIM file.

mf

Function	mf(arg_connection)
Comment	Selected to MF (Master File).
Usage	from model.plugin.api.select import mf
Example	response, sw1, sw2 = mf(arg_connection)
Note	<ul style="list-style-type: none">- response: Response APDU from USIM- sw1: 1st byte of status word.- sw2: 2nd byte of status word.

adfusim

Function	adfusim(arg_connection)
Comment	Selected to ADF (Application Dedicated File)
Usage	from model.plugin.api.select import adfusim
Example	response, sw1, sw2 = adfusim(arg_connection)
Note	<ul style="list-style-type: none">- response: Response APDU from USIM- sw1: 1st byte of status word.- sw2: 2nd byte of status word.

select_file_in_mf

Function	select_file_in_mf(arg_connection, arg_file_id)
Comment	Selected assign file under MF.
Usage	from model.plugin.api.select import select_file_in_mf
Example	response, sw1, sw2 = select_file_in_adf(arg_connection, “2FE2”)
Note	<ul style="list-style-type: none">- The “2FE2” mean the EF_ICCID symbol.

	<ul style="list-style-type: none"> - response: Response APDU from USIM - sw1: 1st byte of status word. - sw2: 2nd byte of status word.
--	---

select_file_in_adf

Function	select_file_in_adf(arg_connection, arg_file_id)
Comment	Selected assign file under ADF.
Usage	from model.plugin.api.select import select_file_in_adf
Example	response, sw1, sw2 = select_file_in_adf(arg_connection, "6F07")
Note	<ul style="list-style-type: none"> - The "6F07" mean the EF_IMSI symbol. - response: The response APDU from USIM - sw1: 1st byte of status word. - sw2: 2nd byte of status word.

Convert

The "pyscard" package already supported some convert method but still not enough for us, we provide some convert function to meet more requirement.

convert_arguments_to_dict

Function	convert_arguments_to_dict(arguments="")
Comment	Convert the argument to dict type
Usage	from model.plugin.api.convert import convert_arguments_to_dict
Example	<pre>argument_string = ('name="super star" num="+12345678") dict = convert_arguments_to_dicts(argument_string) print (dict)</pre> <p>➔ { 'name': "super star", 'num': "+12345678" }</p>

convert_bcd_to_string

Function	convert_bcd_to_string(bytes=[])
Comment	Convert the BCD bytes array to string
Usage	from model.plugin.api.convert import convert_bcd_to_string
Example	<pre>bcd_vals = [0x98, 0x68, 0x00, 0x90, 0x91, 0x11, 0x09, 0x00, 0x10, 0x80] bcd_string = convert_bcd_to_string(vals) print (bcd_string)</pre>

	→ '89860009191190000108'
--	--------------------------

convert_string_to_bcd

Function	convert_string_to_bcd(string="")
Comment	Convert the string to BCD array
Usage	from model.plugin.api.convert import convert_string_to_bcd
Example	<pre> bcd_string = "89860009191190000108" bcd_vals = convert_string_to_bcd(bcd_string) print (bcd_vals) → [0x98, 0x68, 0x00, 0x90, 0x91, 0x11, 0x09, 0x00, 0x10, 0x80] </pre>

convert_alpha_to_string

Function	convert_alpha_to_string(bytes=[])
Comment	Convert the bytes array of Alpha Identifier to string (Ex: EF_ADN)
Usage	from model.plugin.api.convert import convert_alpha_to_string
Example	<pre> alpha_vals = [0x4D, 0x41, 0x49, 0x20, 0x54, 0x45, 0x53, 0x54, 0xFF, 0xFF] alpha_string = convert_alpha_to_string(alpha_vals) print (alpha_string) → 'MAI TEST' </pre> <p>PS. The function only support ASCII code.</p>

convert_dialing_number_to_string

Function	convert_dialing_number_to_string(bytes=[])
Comment	Convert the bytes array of dialing number to string (Include TON & NPI)
Usage	from model.plugin.api.convert import convert_dialing_number_to_string
Example	<pre> num_vals = [0x81, 0x90, 0x82, 0x00, 0x00, 0x00, 0xFF, 0xFF, 0xFF, 0xFF] num_string = convert_dialing_number_to_string(num_vals) print (num_string) → '0928000000' </pre>

FCP

Most operations of USIM APDU are depend on FCP (File Control Parameters), we designed some API for reduce the effort on plugin implement.

get_record_count

Function	get_record_count(arg_bytes=[])
Comment	Returns the record size of LINER/CYCLIC type
Usage	from model.plugin.api.fcp import get_record_count
Example	<pre>response, sw1, sw2 = select_file_in_adf(arg_connection, USIM_FILE_ID.MSISDN.value) if sw1 == 0x90: record_count = get_record_count(response) data_length = get_data_length(response)</pre>

get_data_length

Function	get_data_length(arg_bytes=[])
Comment	Returns the data length by EF file
Usage	from model.plugin.api.fcp import get_data_length
Example	<pre>response, sw1, sw2 = select_file_in_adf(arg_connection, USIM_FILE_ID.MSISDN.value) if sw1 == 0x90: record_count = get_record_count(response) data_length = get_data_length(response)</pre>

get_pin1_status

Function	get_pin1_status(arg_bytes=[])
Comment	Returns the PIN1 status
Usage	from model.plugin.api.fcp import get_pin1_status
Example	<pre>response, sw1, sw2 = mf(arg_connection) if sw1 == 0x90: pin1_enabled = get_pin1_status(response)</pre>

search_fcp_content

Function	search_fcp_content(arg_bytes=[], arg_tag=None):
Comment	Returns FCP content by TLV tag

Usage	from model.plugin.api.fcp import search_fcp_content, TLV_TAG
Example	<pre> fcp_response = [0x62, 0x1D, 0x82, 0x02, 0x78, 0x21, 0x83, 0x02, 0x3F, 0x00, 0xA5, 0x03, 0x80, 0x01, 0x71, 0x8A, 0x01, 0x05, 0x8B, 0x03, 0x2F, 0x06, 0x01, 0xC6, 0x06, 0x90, 0x01, 0x00, 0x83, 0x01, 0x01] return_fcp = search_fcp_content(fcp_response, TLV_TAG. FILE_DESCRIPTOR.value) print (return_fcp) ➔ [0x82, 0x02, 0x78, 0x21]</pre>