

# USIM Modifier

Tech Note

Ming



2018

# Content

Content.....	1
History .....	3
Abstract .....	4
Diagram .....	4
File Structure .....	5
State machine .....	6
Logging mechanism .....	7
Extend the locale resource.....	8
Implement new plugin .....	8
Template.....	8
Concept .....	10
Execute the implemented plugin .....	10
Connection Class .....	11
select .....	11
verify .....	11
read_binary.....	11
update_binary.....	12
read_record.....	12
update_record.....	12
API .....	13
Select USIM File .....	13
mf.....	13
adfusim.....	13
select_file_in_mf.....	13
select_file_in_adf.....	14
Convert.....	14
convert_arguments_to_dict .....	14
convert_bcd_to_string.....	14
convert_string_to_bcd.....	15
convert_alpha_to_string.....	15
convert_dialing_number_to_string .....	15
FCP.....	16
get_record_count.....	16
get_data_length.....	16
get_pin1_status.....	16

search_fcp_content .....	17
--------------------------	----

# History

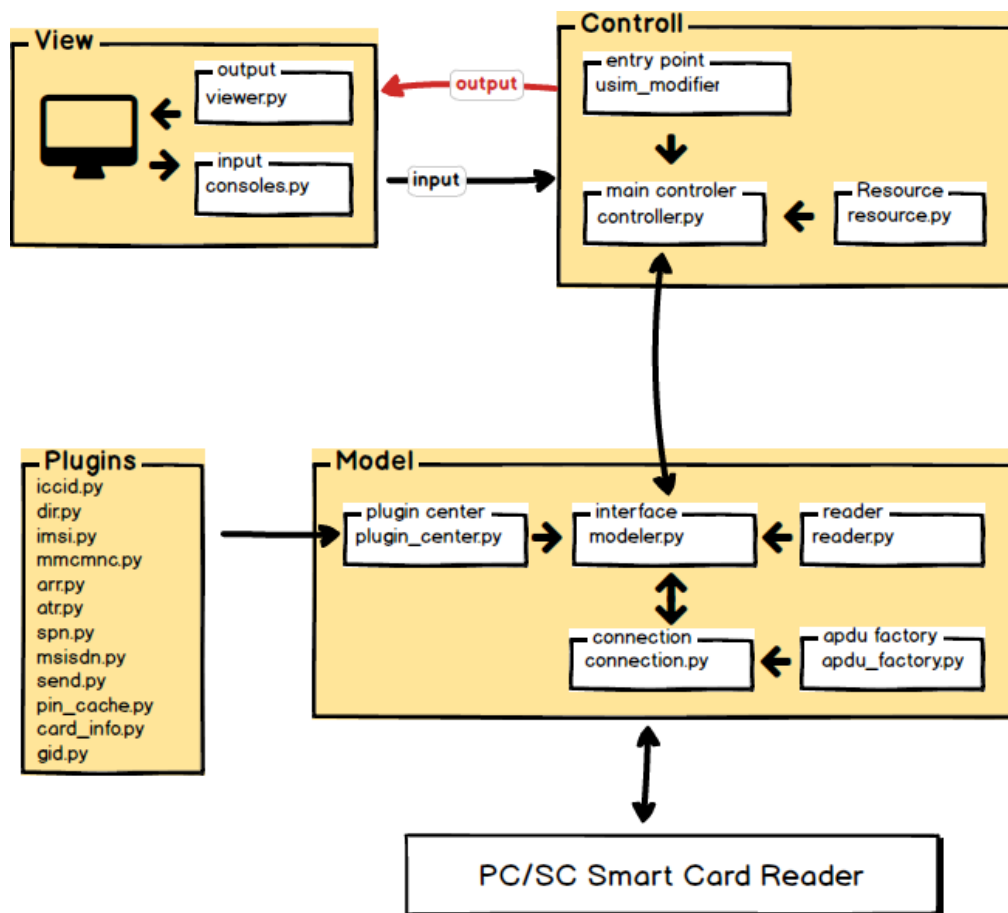
Revision	Common	Date
V1.0	Initial release	2019/01/25
V1.0.1	Fixed some minor typo	2019/01/27

# Abstract

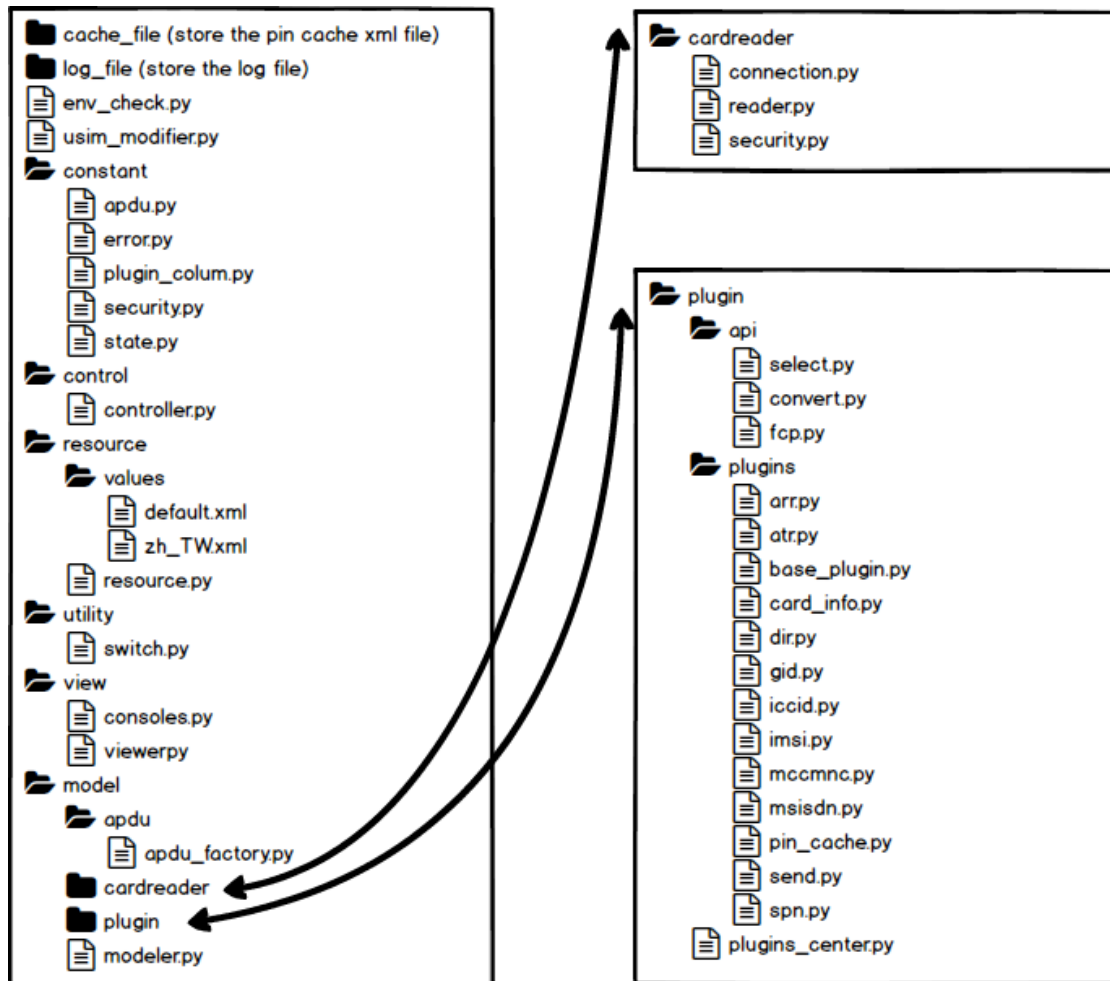
The document will describe the diagram, file structure, state machine, logging mechanism, extend the locale resource, how to implement new plugin and some public API for plugin develop.

# Diagram

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



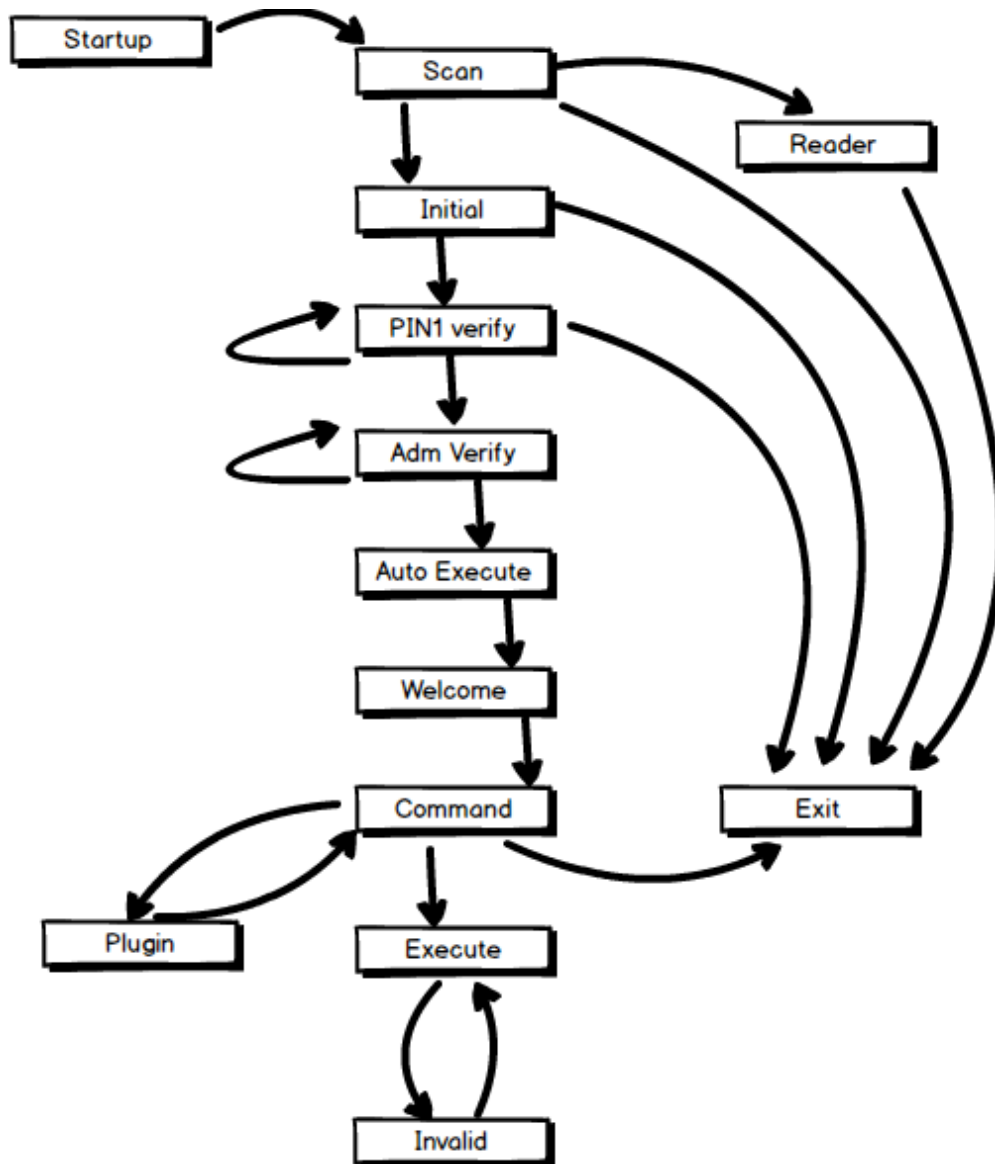
# File Structure



# 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, and “self.\_\_logging.debug(‘debug log’)” method to output the debug message.

```
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__()")
```



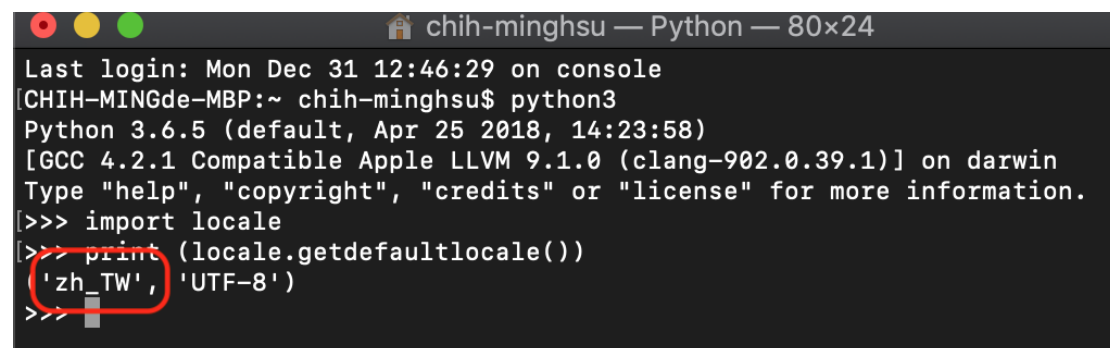
# Extend the locale resource

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 find 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 Python 3.6.5 shell. The user enters 'import locale' and 'print(locale.getdefaultlocale())', which outputs ('zh\_TW', 'UTF-8'). The output is circled in red.

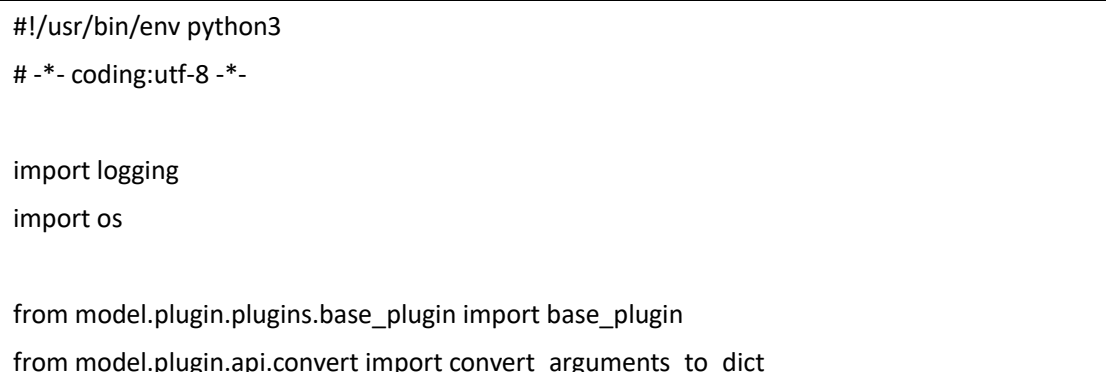
```
Last login: Mon Dec 31 12:46:29 on console
[CHIH-MINGde-MBP:~ chih-minghsu$ python3
Python 3.6.5 (default, Apr 25 2018, 14:23:58)
[GCC 4.2.1 Compatible Apple LLVM 9.1.0 (clang-902.0.39.1)] on darwin
Type "help", "copyright", "credits" or "license" for more information.
[>>> import locale
[>>> print(locale.getdefaultlocale())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

```
#!/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 ot mandtory 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.

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

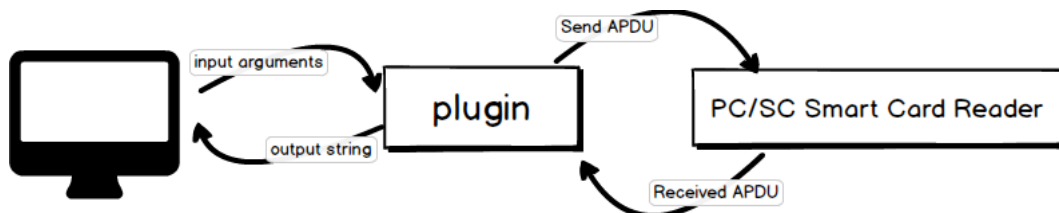
The final step is testing the “hello” plugin, see ... it’s very easy to implement new plugin for “usim\_modifier”.

```
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.



## Execute the implemented plugin

Follow below example to execute another implemented plugin.

```
ret_content = super([your_plugin_name], self).execute_plugin(
    [exec_plugin_name], arg_connection)
```

# Connection Class

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

## select

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

## verify

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

## read\_binary

<b>Function</b>	read_binary(arg_length)
<b>Comment</b>	Read binary data from USIM file
<b>Note</b>	<b>Arguments</b>

	<ul style="list-style-type: none"> <li>- arg_length: Number of bytes to be read</li> </ul> <p><b>Response</b></p> <ul style="list-style-type: none"> <li>- response: Response APDU from USIM</li> <li>- sw1: 1<sup>st</sup> byte of status word.</li> <li>- sw2: 2<sup>nd</sup> byte of status word.</li> </ul>
--	---

## update\_binary

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

## read\_record

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

## update\_record

<b>Function</b>	update_record(arg_idx, arg_update_record):
<b>Comment</b>	Update data to assigned record number
<b>Note</b>	<p><b>Arguments</b></p> <ul style="list-style-type: none"> <li>- arg_idx: Record number</li> <li>- arg_update_record: Bytes of the data to be update</li> </ul> <p><b>Response</b></p> <ul style="list-style-type: none"> <li>- response: Response APDU from USIM</li> </ul>

	<ul style="list-style-type: none"> <li>- sw1: 1<sup>st</sup> byte of status word.</li> <li>- sw2: 2<sup>nd</sup> byte of status word.</li> </ul>
--	--

# 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

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

### adfusim

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

### select\_file\_in\_mf

<b>Function</b>	select_file_in_mf(arg_connection, arg_file_id)
<b>Comment</b>	Selected assign file under MF.

<b>Usage</b>	from model.plugin.api.select import select_file_in_mf
<b>Example</b>	response, sw1, sw2 = select_file_in_adf(arg_connection, "2FE2")
<b>Note</b>	<ul style="list-style-type: none"> <li>- The "2FE2" mean the EF_ICCID symbol.</li> <li>- response: Response APDU from USIM</li> <li>- sw1: 1<sup>st</sup> byte of status word.</li> <li>- sw2: 2<sup>nd</sup> byte of status word.</li> </ul>

## select\_file\_in\_adf

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

## 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

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

## convert\_bcd\_to\_string

<b>Function</b>	convert_bcd_to_string(bytes=[])
<b>Comment</b>	Convert the BCD bytes array to string
<b>Usage</b>	from model.plugin.api.convert import convert_bcd_to_string

<b>Example</b>	<pre> bcd_vals = [0x98, 0x68, 0x00, 0x90, 0x91, 0x11, 0x09, 0x00, 0x10, 0x80] bcd_string = convert_bcd_to_string(vals) print (bcd_string) → '89860009191190000108' </pre>
----------------	---

## convert\_string\_to\_bcd

<b>Function</b>	convert_string_to_bcd(string="")
<b>Comment</b>	Convert the string to BCD array
<b>Usage</b>	from model.plugin.api.convert import convert_string_to_bcd
<b>Example</b>	<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

<b>Function</b>	convert_alpha_to_string(bytes=[])
<b>Comment</b>	Convert the bytes array of Alpha Identifier to string (Ex: EF_ADN)
<b>Usage</b>	from model.plugin.api.convert import convert_alpha_to_string
<b>Example</b>	<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><b>PS. The function only support ASCII code.</b></p>

## convert\_dialing\_number\_to\_string

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

<b>Function</b>	get_record_count(arg_bytes=[])
<b>Comment</b>	Returns the record size of LINER/CYCLIC type
<b>Usage</b>	from model.plugin.api.fcp import get_record_count
<b>Example</b>	<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

<b>Function</b>	get_data_length(arg_bytes=[])
<b>Comment</b>	Returns the data length by EF file
<b>Usage</b>	from model.plugin.api.fcp import get_data_length
<b>Example</b>	<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

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

	<pre>if sw1 == 0x90:     pin1_enabled = get_pin1_status(response)</pre>
--	---

# search\_fcp\_content

<b>Function</b>	search_fcp_content(arg_bytes=[], arg_tag=None):
<b>Comment</b>	Returns FCP content by TLV tag
<b>Usage</b>	from model.plugin.api.fcp import search_fcp_content, TLV_TAG
<b>Example</b>	<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>