## Contents:

1 Introduction

1.1 pySim-shell

1.1.1 Running pySim-shell

1.1.2 cmd2 basics

1.1.3 ISO7816 commands

1.1.4 pySim commands

1.1.5 Linear Fixed EF commands

1.1.6 Transparent EF commands

1.1.7 BER-TLV EF commands

1.1.8 USIM commands

1.1.9 ARA-M commands

1.1.10 cmd2 settable parameters

1.2 Legacy tools

1.2.1 pySim-prog

1.2.2 pySim-read

1.3 pySim library

1.3.1 pySim filesystem abstraction

1.3.2 pySim commands abstraction

1.3.3 pySim Transport

1.3.4 pySim construct utilities

1.3.5 pySim TLV utilities

1.3.6 pySim utility functions

1.3.7 pySim exceptions

1.3.8 pySim card_handler

1.3.9 pySim card_key_provider

2 Indices and tables

Python Module Index

Index
pySim is a python implementation of various software that helps you with managing subscriber identity cards for cellular networks, so-called SIM cards.

Many Osmocom (Open Source Mobile Communications) projects relate to operating private / custom cellular networks, and provisioning SIM cards for said networks is in many cases a requirement to operate such networks.

To make use of most of pySim’s features, you will need a **programmable** SIM card, i.e. a card where you are the owner/operator and have sufficient credentials (such as the ADM PIN) in order to write to many if not most of the files on the card.

Such cards are, for example, available from sysmocom, a major contributor to pySim. See [https://www.sysmocom.de/products/lab/sysmousim/](https://www.sysmocom.de/products/lab/sysmousim/) for more details.

pySim supports classic GSM SIM cards as well as ETSI UICC with 3GPP USIM and ISIM applications. It is easily extensible, so support for additional files, card applications, etc. can be added easily by any python developer. We do encourage you to submit your contributions to help this collaborative development project.

pySim consists of several parts:

- a python **library** containing plenty of objects and methods that can be used for writing custom programs interfacing with SIM cards.
- the [new] **interactive pySim-shell command line program**
- the [legacy] **pySim-prog and pySim-read tools**

### 1.1 pySim-shell

pySim-shell is an interactive command line shell for all kind of interactions with SIM cards.

The interactive shell provides command for

- navigating the on-card filesystem hierarchy
- authenticating with PINs such as ADM1
- CHV/PIN management (VERIFY, ENABLE, DISABLE, UNBLOCK)
- decoding of SELECT response (file control parameters)
- reading and writing of files and records in raw, hex-encoded binary format
- for some files where related support has been developed:
  - decoded reading (display file data in JSON format)
  - decoded writing (encode from JSON to binary format, then write)
By means of using the python `cmd2` module, various useful features improve usability:

- history of commands (persistent across restarts)
- output re-direction to files on your computer
- output piping through external tools like ‘grep’
- tab completion of commands and SELECT-able files/directories
- interactive help for all commands

### 1.1.1 Running pySim-shell

pySim-shell has a variety of command line arguments to control

- which transport to use (how to use a reader to talk to the SIM card)
- whether to automatically verify an ADM pin (and in which format)
- whether to execute a start-up script

**interactive SIM card shell**

```
                 [-m BAUD] [-o osmocon PATH] [-s script PATH] 
                 [-c csv FILE] [-t card_handler FILE] 
                 [-a PIN_ADM1 | -A PIN_ADM1_HEX]
```

**Serial Reader**

- `-d, --device` Serial Device for SIM access
  
  Default: “/dev/ttyUSB0”
  
- `-b, --baud` Baud rate used for SIM access
  
  Default: 9600

**PC/SC Reader**

- `-p, --pcsc-device` PC/SC reader number to use for SIM access

**AT Command Modem Reader**

- `--modem-device` Serial port of modem for Generic SIM Access (3GPP TS 27.007)
- `--modem-baud` Baud rate used for modem port
  
  Default: 115200
**OsmocomBB Reader**

--osmocon  Socket path for Calypso (e.g. Motorola C1XX) based reader (via OsmocomBB)

**General Options**

--script  script with pySim-shell commands to be executed automatically at start-up

--csv  Read card data from CSV file

--card_handler  Use automatic card handling machine

-a, --pin-adm  ADM PIN used for provisioning (overwrites default)

-A, --pin-adm-hex  ADM PIN used for provisioning, as hex string (16 characters long)

### 1.1.2 cmd2 basics

FIXME

### 1.1.3 ISO7816 commands

This category of commands relates to commands that originate in the ISO 7861-4 specifications, most of them have a 1:1 resemblance in the specification.

**select**

The `select` command is used to select a file, either by its FID, AID or by its symbolic name.

Try `select` with tab-completion to get a list of all current selectable items:

```
pySIM-shell (MF)> select
..  2fe2  a0000000871004  EF.ARR  MF
2f00  3f00  ADF.ISIM  EF.DIR
2f05  7f10  ADF.USIM  EF.ICCID
2f06  7f20  DF.GSM  EF.PL
2f08  a0000000871002  DF.TELECOM  EF.UMPC
```

Use `select` with a specific FID or name to select the new file.

This will

- output the [JSON decoded, if possible] select response
- change the prompt to the newly selected file
- enable any commands specific to the newly-selected file

```
pySIM-shell (MF)> select ADF.USIM
{
    "file_descriptor": {
        "file_descriptor_byte": {
            "shareable": true,
            "file_type": "df",
            "structure": "no_info_given"
        }
    }
```

(continues on next page)
status

The status command [re-]obtains the File Control Template of the currently-selected file and print its decoded output.

Example:

```json
pySIM-shell (MF/ADF.ISIM)> status
{
    "file_descriptor": {
        "file_descriptor_byte": {
            "shareable": true,
            "file_type": "df",
            "structure": "no_info_given"
        },
        "record_len": null,
        "num_of_rec": null
    },
    "file_identifier": "ff01",
    "df_name": "A0000000871004FFFFFFFF8907090000",
    "proprietary_information": {
        "uicc_characteristics": "71",
        "available_memory": 101640
    },
    "life_cycle_status_integer": "operational_activated",
    "security_attrib_compact": "00",
    "pin_status_template_do": {
        "ps_do": "70",
        "key_reference": 11
    }
}
```
**change_chv**

Change PIN code to a new PIN code

```bash
usage: change_chv [-h] [--pin-nr PIN_NR] pin_code new_pin_code
```

### Positional Arguments

- **pin_code**: PIN code digits “PIN1” or “PIN2” to get PIN code from external data source
- **new_pin_code**: PIN code digits “PIN1” or “PIN2” to get PIN code from external data source

### Named Arguments

- **--pin-nr**: PUK Number, 1=PIN1, 2=PIN2 or custom value (decimal)
  
  Default: 1

**disable_chv**

Disable PIN code using specified PIN code

```bash
usage: disable_chv [-h] [--pin-nr PIN_NR] pin_code
```

### Positional Arguments

- **pin_code**: PIN code digits, “PIN1” or “PIN2” to get PIN code from external data source

### Named Arguments

- **--pin-nr**: PIN Number, 1=PIN1, 2=PIN2 or custom value (decimal)
  
  Default: 1

**enable_chv**

Enable PIN code using specified PIN code

```bash
usage: enable_chv [-h] [--pin-nr PIN_NR] pin_code
```
Positional Arguments

**pin_code**
PIN code digits, “PIN1” or “PIN2” to get PIN code from external data source

Named Arguments

**--pin-nr**
PIN Number, 1=PIN1, 2=PIN2 or custom value (decimal)
Default: 1

**unblock_chv**

Unblock PIN code using specified PUK code

Usage: unblock_chv [-h] [--pin-nr PIN_NR] puk_code new_pin_code

Positional Arguments

**puk_code**
PUK code digits “PUK1” or “PUK2” to get PUK code from external data source

**new_pin_code**
PIN code digits “PIN1” or “PIN2” to get PIN code from external data source

Named Arguments

**--pin-nr**
PUK Number, 1=PIN1, 2=PIN2 or custom value (decimal)
Default: 1

**verify_chv**

Verify (authenticate) using specified CHV (PIN) code, which is how the specifications call it if you authenticate yourself using the specified PIN. There usually is at least PIN1 and PIN2.

Usage: verify_chv [-h] [--pin-nr PIN_NR] pin_code

Positional Arguments

**pin_code**
PIN code digits, “PIN1” or “PIN2” to get PIN code from external data source
Named Arguments

--pin-nr  PIN Number, 1=PIN1, 2=PIN2 or custom value (decimal)
  Default: 1

deactivate_file
Deactivate the currently selected file. This used to be called INVALIDATE in TS 11.11.

activate_file

Activate the specified EF. This used to be called REHABILITATE in TS 11.11 for classic SIM. You need to specify the name or FID of the file to activate.

usage: activate_file [-h] NAME

Positional Arguments

NAME  File name or FID of file to activate

open_channel

Open a logical channel.

usage: open_channel [-h] chan_nr

Positional Arguments

chan_nr  Channel Number
  Default: 0

close_channel

Close a logical channel.

usage: close_channel [-h] chan_nr
Positional Arguments

    chan_nr        Channel Number
        Default: 0

suspend_uicc

This command allows you to perform the SUSPEND UICC command on the card. This is a relatively recent power-saving addition to the UICC specifications, allowing for suspend/resume while maintaining state, as opposed to a full power-off (deactivate) and power-on (activate) of the card.

The pySim command just sends that SUSPEND UICC command and doesn’t perform the full related sequence including the electrical power down.

Perform the SUSPEND UICC command. Only supported on some UICC.

usage: suspend_uicc [-h] [--min-duration-secs MIN_DURATION_SECS]
                  [--max-duration-secs MAX_DURATION_SECS]

Named Arguments

    --min-duration-secs  Proposed minimum duration of suspension
        Default: 60

    --max-duration-secs  Proposed maximum duration of suspension
        Default: 86400

1.1.4 pySim commands

Commands in this category are pySim specific; they do not have a 1:1 correspondence to ISO 7816 or 3GPP commands. Mostly they will operate either only on local (in-memory) state, or execute a complex sequence of card-commands.

desc

Display human readable file description for the currently selected file.

dir

Show a listing of files available in currently selected DF or MF

Named Arguments

--fids  Show file identifiers
Default: False
--names  Show file names
Default: False
--apps  Show applications
Default: False
--all  Show all selectable identifiers and names
Default: False

Example:

```
pySIM-shell (MF)> dir
MF
3f00
  ..  ADF.USIM  DF.SYSTEM  EF.DIR  EF.UMPC
 ADF.ARA-M  DF.EIRENE  DF.TELECOM  EF.ICCID  MF
 ADF.ISIM  DF.GSM  EF.ARR  EF.PL
14 files
```

export

Export files to script that can be imported back later

```
usage: export [-h] [--filename FILENAME] [--json]
```

Named Arguments

--filename  only export specific file
--json  export as JSON (less reliable)
Default: False

Please note that export works relative to the current working directory, so if you are in MF, then the export will contain all known files on the card. However, if you are in ADF.ISIM, only files below that ADF will be part of the export.

Furthermore, it is strongly advised to first enter the ADM1 pin (verify_adm) to maximize the chance of having permission to read all/most files.
**tree**

Display a tree of the card filesystem. It is important to note that this displays a tree of files that might potentially exist (based on the card profile). In order to determine if a given file really exists on a given card, you have to try to select that file.

Example:

```
pySIM-shell (MF)> tree --help
EF.DIR 2f00 Application Directory
EF.ICCID 2fe2 ICC Identification
EF.PL 2f05 Preferred Languages
EF.ARR 2f06 Access Rule Reference
EF.UMPC 2f08 UICC Maximum Power Consumption
DF.TELECOM 7f10 None
    EF.ADN 6f3a Abbreviated Dialing Numbers
...```

**verify_adm**

Verify the ADM (Administrator) PIN specified as argument. This is typically needed in order to get write/update permissions to most of the files on SIM cards.

Currently only ADM1 is supported.

**reset**

Perform card reset and display the card ATR.

**intro**

[Re-]Display the introductory banner

**equip**

Equip pySim-shell with a card; particularly useful if the program was started before a card was present, or after a card has been replaced by the user while pySim-shell was kept running.

**bulk_script**

Run script on multiple cards (bulk provisioning)

```
usage: bulk_script [-h] [--halt_on_error] [--tries TRIES]
                  [--on_stop_action ON_STOP_ACTION]
                  [--pre_card_action PRE_CARD_ACTION]
                  script_path
```
Positional Arguments

    script_path  path to the script file

Named Arguments

    --halt_on_error  stop card handling if an exception occurs
                     Default: False
    --tries  how many tries before trying the next card
                     Default: 2
    --on_stop_action  commandline to execute when card handling has stopped
    --pre_card_action  commandline to execute before actually talking to the card

echo

Echo (print) a string on the console

usage: echo [-h] string

Positional Arguments

    string  string to echo on the shell

apdu

1.1.5 Linear Fixed EF commands

These commands become enabled only when your currently selected file is of Linear Fixed EF type.

read_record

Read one or more records from a record-oriented EF

usage: read_record [-h] [--count COUNT] record_nr

1.1. pySim-shell
Positional Arguments

```
record_nr                Number of record to be read
```

Named Arguments

```
--count                  Number of records to be read, beginning at record_nr
                         Default: 1
```

`read_record_decoded`

Read + decode a record from a record-oriented EF

```
usage: read_arr_record [-h] [--oneline] record_nr
```

Positional Arguments

```
record_nr                Number of record to be read
```

Named Arguments

```
--oneline                No JSON pretty-printing, dump as a single line
                         Default: False
```

`read_records`

Read all records from a record-oriented EF

```
usage: read_records [-h]
```

`read_records_decoded`

Read + decode all records from a record-oriented EF

```
usage: read_arr_records [-h] [--oneline]
```
Named Arguments

--oneline No JSON pretty-printing, dump as a single line
Default: False

update_record

Update (write) data to a record-oriented EF

usage: update_record [-h] record_nr data

Positional Arguments

record_nr Number of record to be read
data Data bytes (hex format) to write

update_record_decoded

Encode + Update (write) data to a record-oriented EF

usage: update_record_decoded [-h] [--json-path JSON_PATH] record_nr data

Positional Arguments

record_nr Number of record to be read
data Abstract data (JSON format) to write

Named Arguments

--json-path JSON path to modify specific element of record only

edit_record_decoded

Edit the JSON representation of one record in an editor.

usage: edit_record_decoded [-h] record_nr
Positional Arguments

record_nr Number of record to be edited

This command will read the selected record, decode it to its JSON representation, save that JSON to a temporary file on your computer, and launch your configured text editor.

You may then perform whatever modifications to the JSON representation, save + leave your text editor.

Afterwards, the modified JSON will be re-encoded to the binary format, and the result written back to the record on the SIM card.

This allows for easy interactive modification of records.

decode_hex

Decode command-line provided hex-string as if it was read from the file.

usage: decode_hex [-h] [--oneline] HEXSTR

Positional Arguments

HEXSTR Hex-string of encoded data to decode

Named Arguments

--oneline No JSON pretty-printing, dump as a single line
Default: False

1.1.6 Transparent EF commands

These commands become enabled only when your currently selected file is of Transparent EF type.

read_binary

Read binary data from a transparent EF

usage: read_binary [-h] [--offset OFFSET] [--length LENGTH]
Named Arguments

--offset
Byte offset for start of read
Default: 0
--length
Number of bytes to read

read_binary_decoded

Read + decode data from a transparent EF

usage: read_binary_decoded [-h] [--oneline]

Named Arguments

--oneline
No JSON pretty-printing, dump as a single line
Default: False

update_binary

Update (Write) data of a transparent EF

usage: update_binary [-h] [--offset OFFSET] data

Positional Arguments

data
Data bytes (hex format) to write

Named Arguments

--offset
Byte offset for start of read
Default: 0

update_binary_decoded

Encode + Update (Write) data of a transparent EF

usage: update_binary_decoded [-h] [--json-path JSON_PATH] data
Positional Arguments

data Abstract data (JSON format) to write

Named Arguments

--json-path JSON path to modify specific element of file only

In normal operation, update_binary_decoded needs a JSON document representing the entire file contents as input. This can be inconvenient if you want to keep 99% of the content but just toggle one specific parameter. That’s where the JSONpath support comes in handy: You can specify a JSONpath to an element inside the document as well as a new value for that field:

The below example demonstrates this by modifying the ofm field within EF.AD:

```
pySIM-shell (MF/ADF.USIM/EF.AD)> read_binary_decoded
{
   "ms_operation_mode": "normal",
   "specific_facilities": {
       "ofm": true
   },
   "len_of_mnc_in_imsi": 2
}
pySIM-shell (MF/ADF.USIM/EF.AD)> update_binary_decoded --json-path specific_facilities.ofm false
pySIM-shell (MF/ADF.USIM/EF.AD)> read_binary_decoded
{
   "ms_operation_mode": "normal",
   "specific_facilities": {
       "ofm": false
   },
   "len_of_mnc_in_imsi": 2
}
```

edit_binary_decoded

This command will read the selected binary EF, decode it to its JSON representation, save that JSON to a temporary file on your computer, and launch your configured text editor.

You may then perform whatever modifications to the JSON representation, save + leave your text editor.

Afterwards, the modified JSON will be re-encoded to the binary format, and the result written to the SIM card.

This allows for easy interactive modification of file contents.
decode_hex

Decode command-line provided hex-string as if it was read from the file.

usage: decode_hex [-h] [--oneline] HEXSTR

Positional Arguments

   HEXSTR   Hex-string of encoded data to decode

Named Arguments

   --oneline   No JSON pretty-printing, dump as a single line
               Default: False

1.1.7 BER-TLV EF commands

BER-TLV EFs are files that contain BER-TLV structured data. Every file can contain any number of variable-length IEs (DOs). The tag within a BER-TLV EF must be unique within the file.

The commands below become enabled only when your currently selected file is of BER-TLV EF type.

retrieve_tags

Retrieve a list of all tags present in the currently selected file.

retrieve_data

Retrieve (Read) data from a BER-TLV EF

usage: retrieve_data [-h] tag

Positional Arguments

   tag   BER-TLV Tag of value to retrieve
osmopysim-usermanual

set_data

Set (Write) data for a given tag in a BER-TLV EF

usage: set_data [-h] tag data

Positional Arguments

  tag          BER-TLV Tag of value to set
  data         Data bytes (hex format) to write

del_data

Delete data for a given tag in a BER-TLV EF

usage: delete_data [-h] tag

Positional Arguments

  tag          BER-TLV Tag of value to set

1.1.8 USIM commands

authenticate

Perform Authentication and Key Agreement (AKA).

usage: authenticate [-h] rand autn

Positional Arguments

  rand         Random challenge
  autn         Authentication Nonce
terminal_profile

Send a TERMINAL PROFILE command to the card.
This is used to inform the card about which optional features the terminal (modem/phone) supports, particularly in the context of SIM Toolkit, Proactive SIM and OTA. You must specify a hex-string with the encoded terminal profile you want to send to the card.

usage: terminal_profile [-h] PROFILE

Positional Arguments

PROFILE Hexstring of encoded terminal profile

envelope

Send an ENVELOPE command to the card. This is how a variety of information is communicated from the terminal (modem/phone) to the card, particularly in the context of SIM Toolkit, Proactive SIM and OTA.

usage: envelope [-h] PAYLOAD

Positional Arguments

PAYLOAD Hexstring of encoded payload to ENVELOPE

tenvelope_sms

Send an ENVELOPE(SMS-PP-Download) command to the card.
This emulates a terminal (modem/phone) having received a SMS with a PID of ‘SMS for the SIM card’. You can use this command in the context of testing OTA related features without a modem/phone or a cellular netwokr.

usage: envelope_sms [-h] TPDU

Positional Arguments

TPDU Hexstring of encoded SMS TPDU
1.1.9 ARA-M commands

The ARA-M commands exist to manage the access rules stored in an ARA-M applet on the card.

ARA-M in the context of SIM cards is primarily used to enable Android UICC Carrier Privileges, please see https://source.android.com/devices/tech/config/uicc for more details on the background.

**aram_get_all**

Obtain and decode all access rules from the ARA-M applet on the card.

NOTE: if the total size of the access rules exceeds 255 bytes, this command will fail, as it doesn’t yet implement fragmentation/reassembly on rule retrieval. YMMV

```plaintext
pySIM-shell (MF/ADF.ARA-M)> aram_get_all
[
    {
        "ResponseAllRefArDO": [
            {
                "RefArDO": [
                    {
                        "RefDO": [
                            {
                                "AidRefDO": "ffffffffffff"
                            },
                            {
                                "DevAppIdRefDO": "e46872f28b350b7e1f140de535c2a8d5804f0be3"
                            }
                        }
                    }
                ]
            }
        ]
    },
    {
        "ArDO": [
            {
                "ApduArDO": {
                    "generic_access_rule": "always"
                },
                {
                    "PermArDO": {
                        "permissions": "0000000000000001"
                    }
                }
            }
        ]
    }
]
```
aram_get_config

Perform Config handshake with ARA-M applet: Tell it our version and retrieve its version.
NOTE: Not supported in all ARA-M implementations.

aram_store_ref_ar_do

Perform STORE DATA [Command-Store-REF-AR-DO] to store a (new) access rule.

usage: aram_store_ref_ar_do [-h] --device-app-id DEVICE_APP_ID
       [--aid AID | --aid-empty] [--pkg-ref PKG_REF]
       [--apdu-never | --apdu-always | --apdu-filter APDU_FILTER]
       [--nfc-always | --nfc-never]
       [--android-permissions ANDROID_PERMISSIONS]

Named Arguments

--device-app-id  Identifies the specific device application that the rule applies to. Hash of Certificate of Application Provider, or UUID. (20/32 hex bytes)
--aid             Identifies the specific SE application for which rules are to be stored. Can be a partial AID, containing for example only the RID. (5-16 hex bytes)
--aid-empty       No specific SE application, applies to all applications
Default: False
--pkg-ref         Full Android Java package name (up to 127 chars ASCII)
--apdu-never      APDU access is not allowed
Default: False
--apdu-always     APDU access is allowed
Default: False
--apdu-filter     APDU filter: 4 byte CLA/INS/P1/P2 followed by 4 byte mask (8 hex bytes)
--nfc-always      NFC event access is allowed
Default: False
--nfc-never       NFC event access is not allowed
Default: False
--android-permissions   Android UICC Carrier Privilege Permissions (8 hex bytes)

For example, to store an Android UICC carrier privilege rule for the SHA1 hash of the certificate used to sign the CoIMS android app of Supreeth Herle (https://github.com/herlesupreeth/CoIMS_Wiki) you can use the following command:

```
pySIM-shell (MF/ADF.ARA-M)> aram_store_ref_ar_do --aid FFFFFFFFFFFFFFFF --device-app-id E46872F28B350B7E1F140DE535C2A8D5804F0BE3 --android-permissions 0000000000000001 --apdu-always
```
aram_delete_all

This command will request deletion of all access rules stored within the ARA-M applet. Use it with caution, there is no undo. Any rules later intended must be manually inserted again using *aram_store_ref_ar_do*.

### 1.1.10 cmd2 settable parameters

cmd2 has the concept of *settable parameters* which act a bit like environment variables in an OS-level shell: They can be read and set, and they will influence the behavior somehow.

**conserve_write**

If enabled, pySim will (when asked to write to a card) always first read the respective file/record and verify if the to-be-written value differs from the current on-card value. If not, the write will be skipped. Writes will only be performed if the new value is different from the current on-card value.

If disabled, pySim will always write irrespective of the current/new value.

**json_pretty_print**

This parameter determines if generated JSON output should (by default) be pretty-printed (multi-line output with indent level of 4 spaces) or not.

The default value of this parameter is ‘true’.

**debug**

If enabled, full python back-traces will be displayed in case of exceptions.

**apdu_trace**

Boolean variable that determines if a hex-dump of the command + response APDU shall be printed.

**numeric_path**

Boolean variable that determines if path (e.g. in prompt) is displayed with numeric FIDs or string names.

```
pySIM-shell (MF/EF.ICCID)> set numeric_path True
numeric_path - was: False
now: True
pySIM-shell (3f08/2fe2)> set numeric_path False
numeric_path - was: True
now: False
pySIM-shell (MF/EF.ICCID)> help set
```
1.2 Legacy tools

Legacy tools are the classic pySim-prog and pySim-read programs that existed long before pySim-shell. These days, you should primarily use pySim-shell instead of these legacy tools.

1.2.1 pySim-prog

pySim-prog was the first part of the pySim software suite. It started as a tool to write ICCID, IMSI, MSISDN and Ki to very simplistic SIM cards, and was later extended to a variety of other cards. As the number of features supported became no longer bearable to express with command-line arguments, pySim-shell was created.

Basic use cases can still use pySim-prog.

Program customizable SIMs

Two modes are possible:

• one where you specify every parameter manually:

  ./pySim-prog.py -n 26C3 -c 49 -x 262 -y 42 -i <IMSI> -s <ICCID>

• one where they are generated from some minimal set:

  ./pySim-prog.py -n 26C3 -c 49 -x 262 -y 42 -z <random_string_of_choice> -j <card_num>

With <random_string_of_choice> and <card_num>, the soft will generate ‘predictable’ IMSI and ICCID, so make sure you choose them so as not to conflict with anyone. (for eg. your name as <random_string_of_choice> and 0 1 2… for <card_num>).

You also need to enter some parameters to select the device:
- t TYPE: type of card (supersim, magicsim, fakemagicsim or try ‘auto’) - d DEV: Serial port device (default /dev/ttyUSB0) - b BAUD : Baudrate (default 9600)

1.2.2 pySim-read

pySim-read allows you to read some data from a SIM card. It will only some files of the card, and will only read files accessible to a normal user (without any special authentication)

These days, you should use the export command of pySim-shell instead. It performs a much more comprehensive export of all of the [standard] files that can be found on the card. To get a human-readable decode instead of the raw hex export, you can use export --json.

Specifically, pySim-read will dump the following:

• MF
• EF.IICCID
• DF.GSM
• EF.IMSI
• EF.GID1
• EF.GID2
• EF.SMSP
• EF.SPN
- EF.PLMNsel
- EF.PLMNwAcT
- EF.OPLMNwAcT
- EF.HPLMNAcT
- EF.ACC
- EF.MSISDN
- EF.AD
- EF.SST
- ADF.USIM
- EF.EHPLMN
- EF.UST
- EF.ePDGId
- EF.ePDGSelection
- ADF.ISIM
- EF.PCSCF
- EF.DOMAIN
- EF.IMPI
- EF.IMPU
- EF.UICCIARI
- EF.IST

**pySim-read usage**

Legacy tool for reading some parts of a SIM card

```
```

**Serial Reader**

- `-d, --device` Serial Device for SIM access
  Default: “/dev/ttyUSB0”

- `-b, --baud` Baud rate used for SIM access
  Default: 9600
PC/SC Reader

- `p, --pcsc-device` PC/SC reader number to use for SIM access

AT Command Modem Reader

- `--modem-device` Serial port of modem for Generic SIM Access (3GPP TS 27.007)
- `--modem-baud` Baud rate used for modem port
  Default: 115200

OsmocomBB Reader

- `--osmocon` Socket path for Calypso (e.g. Motorola C1XX) based reader (via OsmocomBB)

1.3 pySim library

1.3.1 pySim filesystem abstraction

Representation of the ISO7816-4 filesystem model.

The File (and its derived classes) represent the structure / hierarchy of the ISO7816-4 smart card file system with the MF, DF, EF and ADF entries, further sub-divided into the EF sub-types Transparent, Linear Fixed, etc.

The classes are intended to represent the specification of the filesystem, not the actual contents / runtime state of interacting with a given smart card.

```python
class pySim.filesystem.BerTlvEF(fid: str, sfid: Optional[str] = None, name: Optional[str] = None, desc: Optional[str] = None, parent: Optional[CardDF] = None, size={1, None}, **kwargs)
```

BER-TLV EF (Entry File) in the smart card filesystem. A BER-TLV EF is a binary file with a BER (Basic Encoding Rules) TLV structure

NOTE: We currently don’t really support those, this class is simply a wrapper around TransparentEF as a placeholder, so we can already define EFs of BER-TLV type without fully supporting them.

Parameters

- `fid` – File Identifier (4 hex digits)
- `sfid` – Short File Identifier (2 hex digits, optional)
- `name` – Brief name of the file, lik EF_ICCID
- `desc` – Description of the file
- `parent` – Parent CardFile object within filesystem hierarchy
- `size` – tuple of (minimum_size, recommended_size)

class ShellCommands

Shell commands specific for BER-TLV EFs.

```python
do_delete_data(opts)
```

Delete data for a given tag in a BER-TLV EF
do_retrieve_data(opts)
Retrieve (Read) data from a BER-TLV EF

do_retrieve_tags(opts)
List tags available in a given BER-TLV EF

do_set_data(opts)
Set (Write) data for a given tag in a BER-TLV EF

class pySim.filesystem.CardADF(aid: str, **kwargs)
ADF (Application Dedicated File) in the smart card filesystem

Parameters
- **fid** – File Identifier (4 hex digits)
- **sfid** – Short File Identifier (2 hex digits, optional)
- **name** – Brief name of the file, lik EF_ICCID
- **desc** – Description of the file
- **parent** – Parent CardFile object within filesystem hierarchy
- **profile** – Card profile that this file should be part of
- **service** – Service (SST/UST/IST) associated with the file

A card application is represented by an ADF (with contained hierarchy) and optionally some SW definitions.

Parameters
- **adf** – ADF name
- **sw** – Dict of status word conversions

interpret_sw(sw)
Interpret a given status word within the application.

Parameters
- **sw** – Status word as string of 4 hex digits

Returns
Tuple of two strings

class pySim.filesystem.CardDF(**kwargs)
DF (Dedicated File) in the smart card filesystem. Those are basically sub-directories.

Parameters
- **fid** – File Identifier (4 hex digits)
- **sfid** – Short File Identifier (2 hex digits, optional)
- **name** – Brief name of the file, lik EF_ICCID
- **desc** – Description of the file
- **parent** – Parent CardFile object within filesystem hierarchy
- **profile** – Card profile that this file should be part of
- **service** – Service (SST/UST/IST) associated with the file
class ShellCommands

add_file(child: CardFile, ignore_existing: bool = False)
Add a child (DF/EF) to this DF.

Parameters
- **children** – List of new DF/EFs to be added
- **ignore_existing** – Ignore, if file[s] with given FID already exists. Old one[s] will be kept.

add_files(children: Iterable[CardFile], ignore_existing: bool = False)
Add a list of child (DF/EF) to this DF.

Parameters
- **children** – List of new DF/EFs to be added
- **ignore_existing** – Ignore, if file[s] with given FID already exists. Old one[s] will be kept.

get_selectables(flags=[]) \(\rightarrow\) dict
Return a dict of `{‘identifier’: File}` that is selectable from the current DF.

Parameters
- **flags** – Specify which selectables to return ‘FIDS’ and/or ‘NAMES’; If not specified, all selectables will be returned.

Returns
dict containing all selectable items. Key is identifier (string), value a reference to a CardFile (or derived class) instance.

lookup_file_by_fid(fid: str) \(\rightarrow\) Optional[CardFile]
Find a file with given file ID within current DF.

lookup_file_by_name(name: Optional[str]) \(\rightarrow\) Optional[CardFile]
Find a file with given name within current DF.

lookup_file_by_sfid(sfid: Optional[str]) \(\rightarrow\) Optional[CardFile]
Find a file with given short file ID within current DF.

class pySim.filesystem.CardEF(*, fid, **kwargs)
EF (Entry File) in the smart card filesystem

Parameters
- **fid** – File Identifier (4 hex digits)
- **sfid** – Short File Identifier (2 hex digits, optional)
- **name** – Brief name of the file, lik EF_ICCID
- **desc** – Description of the file
- **parent** – Parent CardFile object within filesystem hierarchy
- **profile** – Card profile that this file should be part of
- **service** – Service (SST/UST/IST) associated with the file

get_selectables(flags=[]) \(\rightarrow\) dict
Return a dict of `{‘identifier’: File}` that is selectable from the current DF.

Parameters
- **flags** – Specify which selectables to return ‘FIDS’ and/or ‘NAMES’; If not specified, all selectables will be returned.
Returns
dict containing all selectable items. Key is identifier (string), value a reference to a CardFile
(or derived class) instance.

class pySim.filesystem.CardFile(fid: str = None, sfid: str = None, name: str = None, desc: str = None,
parent: Optional[CardDF] = None, profile: Optional[CardProfile] = None, service: Optional[Union[int, List[int], Tuple[int, ...]]] = None)

Base class for all objects in the smart card filesystem. Serve as a common ancestor to all other file types; rarely
used directly.

Parameters
• fid – File Identifier (4 hex digits)
• sfid – Short File Identifier (2 hex digits, optional)
• name – Brief name of the file, lik EF_ICCID
• desc – Description of the file
• parent – Parent CardFile object within filesystem hierarchy
• profile – Card profile that this file should be part of
• service – Service (SST/UST/IST) associated with the file

build_select_path_to(target: CardFile) → Optional[List[CardFile]]
Build the relative sequence of files we need to traverse to get from us to ‘target’.

decode_select_response(data_hex: str)
Decode the response to a SELECT command.

Parameters
data_hex – Hex string of the select response

fully_qualified_path(prefer_name: bool = True) → List[str]
Return fully qualified path to file as list of FID or name strings.

Parameters
prefer_name – Preferably build path of names; fall-back to FIDs as required

fully_qualified_path_fobj() → List[CardFile]
Return fully qualified path to file as list of CardFile instance references.

get_mf() → Optional[CardMF]
Return the MF (root) of the file system.

get_profile()
Get the profile associated with this file. If this file does not have any profile assigned, try to find a file above
(usually the MF) in the filesystem hierarchy that has a profile assigned

get_selectable_names(flags=[]) → List[str]
Return a dict of {‘identifier’: File} that is selectable from the current file.

Parameters
flags – Specify which selectables to return ‘FIDS’ and/or ‘NAMES’; If not specified, all
selectables will be returned.

Returns
list containing all selectable names.
get_selectables(\texttt{flags=[]}) \rightarrow \text{dict[str, CardFile]}

Return a dict of \{'identifier': File\} that is selectable from the current file.

**Parameters**

\texttt{flags} – Specify which selectables to return ‘FIDS’ and/or ‘NAMES’; If not specified, all selectables will be returned.

**Returns**

dict containing all selectable items. Key is identifier (string), value a reference to a CardFile (or derived class) instance.

should_exist_for_services(\texttt{services: List[int]})

Assuming the provided list of activated services, should this file exist and be activated?.

class pySim.filesystem.CardMF(**kwargs)

MF (Master File) in the smart card filesystem

**Parameters**

• \texttt{fid} – File Identifier (4 hex digits)
• \texttt{sfid} – Short File Identifier (2 hex digits, optional)
• \texttt{name} – Brief name of the file, lik EF_ICCID
• \texttt{desc} – Description of the file
• \texttt{parent} – Parent CardFile object within filesystem hierarchy
• \texttt{profile} – Card profile that this file should be part of
• \texttt{service} – Service (SST/UST/IST) associated with the file

add_application_df(\texttt{app: CardADF})

Add an Application to the MF

decode_select_response(\texttt{data_hex: Optional[str]}) \rightarrow \text{object}

Decode the response to a SELECT command.

This is the fall-back method which automatically defers to the standard decoding method defined by the card profile. When no profile is set, then no decoding is performed. Specific derived classes (usually ADF) can overload this method to install specific decoding.

get_app_names()

Get list of completions (AID names)

get_app_selectables(\texttt{flags=[]}) \rightarrow \text{dict}

Get applications by AID + name

get_selectables(\texttt{flags=[]}) \rightarrow \text{dict}

Return a dict of \{'identifier': File\} that is selectable from the current DF.

**Parameters**

\texttt{flags} – Specify which selectables to return ‘FIDS’ and/or ‘NAMES’; If not specified, all selectables will be returned.

**Returns**

dict containing all selectable items. Key is identifier (string), value a reference to a CardFile (or derived class) instance.
class pySim.filesystem.CardModel
A specific card model, typically having some additional vendor-specific files. All you need to do is to define a
sub-class with a list of ATRs or an overridden match method.

abstract classmethod add_files(rs: RuntimeState)
Add model specific files to given RuntimeState.

static apply_matching_models(scc: SimCardCommands, rs: RuntimeState)
Check if any of the CardModel sub-classes ‘match’ the currently inserted card (by ATR or overriding the
‘match’ method). If so, call their ‘add_files’ method.

classmethod match(scc: SimCardCommands) → bool
Test if given card matches this model.

class pySim.filesystem.CyclicEF(fid: str, sfid: Optional[str] = None, name: Optional[str] = None, desc:
Optional[str] = None, parent: Optional[CardDF] = None, rec_len={1, None}, **kwargs)
Cyclic EF (Entry File) in the smart card filesystem

Parameters
• **fid** – File Identifier (4 hex digits)
• **sfid** – Short File Identifier (2 hex digits, optional)
• **name** – Brief name of the file, lik EF_ICCID
• **desc** – Description of the file
• **parent** – Parent CardFile object within filesystem hierarchy
• **rec_len** – set of {minimum_length, recommended_length}

class pySim.filesystem.FileData(fdesc)
Represent the runtime, on-card data.

class pySim.filesystem.LinFixedEF(fid: str, sfid: Optional[str] = None, name: Optional[str] = None, desc:
Optional[str] = None, parent: Optional[CardDF] = None, rec_len={1, None}, **kwargs)
Linear Fixed EF (Entry File) in the smart card filesystem.
Linear Fixed EFs are record oriented files. They consist of a number of fixed-size records. The records can be
individually read/updated.

Parameters
• **fid** – File Identifier (4 hex digits)
• **sfid** – Short File Identifier (2 hex digits, optional)
• **name** – Brief name of the file, lik EF_ICCID
• **desc** – Description of the file
• **parent** – Parent CardFile object within filesystem hierarchy
• **rec_len** – set of {minimum_length, recommended_length}

class ShellCommands(**kwargs)
Shell commands specific for Linear Fixed EFs.

do_decode_hex(opts)
Decode command-line provided hex-string as if it was read from the file.
do_edit_record_decoded(opts)
Edit the JSON representation of one record in an editor.

do_read_record(opts)
Read one or multiple records from a record-oriented EF

do_read_record_decoded(opts)
Read + decode a record from a record-oriented EF

do_read_records(opts)
Read all records from a record-oriented EF

do_read_records_decoded(opts)
Read + decode all records from a record-oriented EF

do_update_record(opts)
Update (write) data to a record-oriented EF

do_update_record_decoded(opts)
Encode + Update (write) data to a record-oriented EF

decode_record_bin(raw_bin_data: bytearray) → dict
Decode raw (binary) data into abstract representation.

A derived class would typically provide a _decode_record_bin() or _decode_record_hex() method for implementing this specifically for the given file. This function checks which of the method exists, add calls them (with conversion, as needed).

Parameters
raw_bin_data – binary encoded data

Returns
abstract_data; dict representing the decoded data

decode_record_hex(raw_hex_data: str) → dict
Decode raw (hex string) data into abstract representation.

A derived class would typically provide a _decode_record_bin() or _decode_record_hex() method for implementing this specifically for the given file. This function checks which of the method exists, add calls them (with conversion, as needed).

Parameters
raw_hex_data – hex-encoded data

Returns
abstract_data; dict representing the decoded data

encode_record_bin(abstract_data: dict) → bytearray
Encode abstract representation into raw (binary) data.

A derived class would typically provide an _encode_record_bin() or _encode_record_hex() method for implementing this specifically for the given file. This function checks which of the method exists, add calls them (with conversion, as needed).

Parameters
abstract_data – dict representing the decoded data

Returns
binary encoded data
**encode_record_hex**(*abstract_data*: dict) → str

Encode abstract representation into raw (hex string) data.

A derived class would typically provide an *_encode_record_bin()* or *_encode_record_hex()* method for implementing this specifically for the given file. This function checks which of the method exists, add calls them (with conversion, as needed).

**Parameters**

• *abstract_data* – dict representing the decoded data

**Returns**

hex string encoded data

class pySim.filesystem.RuntimeState(*card, profile*: CardProfile)

Represent the runtime state of a session with a card.

**Parameters**

• *card* – pysim.cards.Card instance
• *profile* – CardProfile instance

**activate_file**(*name*: str)

Request ACTIVATE FILE of specified file.

**get_application_df**() → Optional[CardADF]

Obtain the currently selected application DF (if any).

**Returns**

CardADF() instance or None

**get_cwd**() → CardDF

Obtain the current working directory.

**Returns**

CardDF instance

**interpret_sw**(*sw*: str)

Interpret a given status word relative to the currently selected application or the underlying card profile.

**Parameters**

• *sw* – Status word as string of 4 hex digits

**Returns**

Tuple of two strings

**probe_file**(*fid*: str, *cmd_app*=None)

Blindly try to select a file and automatically add a matching file object if the file actually exists.

**read_binary**(*length*: Optional[int] = None, *offset*: int = 0)

Read [part of] a transparent EF binary data.

**Parameters**

• *length* – Amount of data to read (None: as much as possible)
• *offset* – Offset into the file from which to read ‘length’ bytes

**Returns**

binary data read from the file
read_binary_dec() → Tuple[dict, str]
Read [part of] a transparent EF binary data and decode it.

Parameters

- **length** – Amount of data to read (None: as much as possible)
- **offset** – Offset into the file from which to read ‘length’ bytes

Returns
abstract decode data read from the file

read_record(rec_nr: int = 0)
Read a record as binary data.

Parameters
- **rec_nr** – Record number to read

Returns
hex string of binary data contained in record

read_record_dec(rec_nr: int = 0) → Tuple[dict, str]
Read a record and decode it to abstract data.

Parameters
- **rec_nr** – Record number to read

Returns
abstract data contained in record

reset(cmd_app=None) → str
Perform physical card reset and obtain ATR. :param cmd_app: Command Application State (for unregistering old file commands)

retrieve_data(tag: int = 0)
Read a DO/TLV as binary data.

Parameters
- **tag** – Tag of TLV/DO to read

Returns
hex string of full BER-TLV DO including Tag and Length

retrieve_tags()
Retrieve tags available on BER-TLV EF.

Returns
list of integer tags contained in EF

select(name: str, cmd_app=None)
Select a file (EF, DF, ADF, MF, …).

Parameters

- **name** – Name of file to select
- **cmd_app** – Command Application State (for unregistering old file commands)

select_file(file: CardFile, cmd_app=None)
Select a file (EF, DF, ADF, MF, …).

Parameters

- **file** – CardFile [or derived class] instance
• **cmd_app** – Command Application State (for unregistering old file commands)

**set_data**(tag: int, data_hex: str)
Update a TLV/DO with given binary data

**Parameters**
• **tag** – Tag of TLV/DO to be written
• **data_hex** – Hex string binary data to be written (value portion)

**status()**
Request STATUS (current selected file FCP) from card.

**unregister_cmds**(cmd_app=None)
Unregister all file specific commands.

**update_binary**(data_hex: str, offset: int = 0)
Update transparent EF binary data.

**Parameters**
• **data_hex** – hex string of data to be written
• **offset** – Offset into the file from which to write ‘data_hex’

**update_binary_dec**(data: dict)
Update transparent EF from abstract data. Encodes the data to binary and then updates the EF with it.

**Parameters**
• **data** – abstract data which is to be encoded and written

**update_record**(rec_nr: int, data_hex: str)
Update a record with given binary data

**Parameters**
• **rec_nr** – Record number to read
• **data_hex** – Hex string binary data to be written

**update_record_dec**(rec_nr: int, data: dict)
Update a record with given abstract data. Will encode abstract to binary data and then write it to the given record on the card.

**Parameters**
• **rec_nr** – Record number to read
• **data_hex** – Abstract data to be written

**class** pySim.filesystem.TransRecEF(fid: str, rec_len: int, sfid: Optional[str] = None, name: Optional[str] = None, desc: Optional[str] = None, parent: Optional[CardDF] = None, size={1, None}, **kwargs)
Transparent EF (Entry File) containing fixed-size records.

These are the real odd-balls and mostly look like mistakes in the specification: Specified as ‘transparent’ EF, but actually containing several fixed-length records inside. We add a special class for those, so the user only has to provide encoder/decoder functions for a record, while this class takes care of split / merge of records.

**Parameters**
• **fid** – File Identifier (4 hex digits)
• **sfid** – Short File Identifier (2 hex digits, optional)
• **name** – Brief name of the file, like EF_ICCID
• **desc** – Description of the file
• **parent** – Parent CardFile object within filesystem hierarchy
• **rec_len** – Length of the fixed-length records within transparent EF
• **size** – tuple of (minimum_size, recommended_size)

```python
decode_record_bin(raw_bin_data: bytearray) → dict
```
Decode raw (binary) data into abstract representation.
A derived class would typically provide a _decode_record_bin() or _decode_record_hex() method for implementing this specifically for the given file. This function checks which of the method exists, add calls them (with conversion, as needed).

**Parameters**
- `raw_bin_data` – binary encoded data

**Returns**
- abstract_data; dict representing the decoded data

```python
decode_record_hex(raw_hex_data: str) → dict
```
Decode raw (hex string) data into abstract representation.
A derived class would typically provide a _decode_record_bin() or _decode_record_hex() method for implementing this specifically for the given file. This function checks which of the method exists, add calls them (with conversion, as needed).

**Parameters**
- `raw_hex_data` – hex-encoded data

**Returns**
- abstract_data; dict representing the decoded data

```python
encode_record_bin(abstract_data: dict) → bytearray
```
Encode abstract representation into raw (binary) data.
A derived class would typically provide an _encode_record_bin() or _encode_record_hex() method for implementing this specifically for the given file. This function checks which of the method exists, add calls them (with conversion, as needed).

**Parameters**
- `abstract_data` – dict representing the decoded data

**Returns**
- binary encoded data

```python
encode_record_hex(abstract_data: dict) → str
```
Encode abstract representation into raw (hex string) data.
A derived class would typically provide an _encode_record_bin() or _encode_record_hex() method for implementing this specifically for the given file. This function checks which of the method exists, add calls them (with conversion, as needed).

**Parameters**
- `abstract_data` – dict representing the decoded data

**Returns**
- hex string encoded data
class pySim.filesystem.TransparentEF(fid: str, sfid: Optional[str] = None, name: Optional[str] = None, desc: Optional[str] = None, parent: Optional[CardDF] = None, size={1, None}, **kwargs)

Transparent EF (Entry File) in the smart card filesystem.

A Transparent EF is a binary file with no formal structure. This is contrary to Record based EFs which have [fixed size] records that can be individually read/updated.

Parameters

• **fid** – File Identifier (4 hex digits)
• **sfid** – Short File Identifier (2 hex digits, optional)
• **name** – Brief name of the file, lik EF_ICCID
• **desc** – Description of the file
• **parent** – Parent CardFile object within filesystem hierarchy
• **size** – tuple of (minimum_size, recommended_size)

class ShellCommands

Shell commands specific for transparent EFs.

**do_decode_hex**(opts)

Decode command-line provided hex-string as if it was read from the file.

**do_edit_binary_decoded**(opts)

Edit the JSON representation of the EF contents in an editor.

**do_read_binary**(opts)

Read binary data from a transparent EF

**do_read_binary_decoded**(opts)

Read + decode data from a transparent EF

**do_update_binary**(opts)

Update (Write) data of a transparent EF

**do_update_binary_decoded**(opts)

Encode + Update (Write) data of a transparent EF

decode_bin**(raw_bin_data: bytearray) → dict

Decode raw (binary) data into abstract representation.

A derived class would typically provide a _decode_bin() or _decode_hex() method for implementing this specifically for the given file. This function checks which of the method exists, add calls them (with conversion, as needed).

Parameters

**raw_bin_data** – binary encoded data

Returns

abstract_data; dict representing the decoded data

decode_hex**(raw_hex_data: str) → dict

Decode raw (hex string) data into abstract representation.

A derived class would typically provide a _decode_bin() or _decode_hex() method for implementing this specifically for the given file. This function checks which of the method exists, add calls them (with conversion, as needed).
Parameters

raw_hex_data – hex-encoded data

Returns

abstract_data; dict representing the decoded data

code_bin(abstract_data: dict) → bytearray

Encode abstract representation into raw (binary) data.

A derived class would typically provide an _encode_bin() or _encode_hex() method for implementing this specifically for the given file. This function checks which of the method exists, add calls them (with conversion, as needed).

Parameters

abstract_data – dict representing the decoded data

Returns

binary encoded data

code_hex(abstract_data: dict) → str

Encode abstract representation into raw (hex string) data.

A derived class would typically provide an _encode_bin() or _encode_hex() method for implementing this specifically for the given file. This function checks which of the method exists, add calls them (with conversion, as needed).

Parameters

abstract_data – dict representing the decoded data

Returns

hex string encoded data

pySim.filesystem.interpret_sw(sw_data: dict, sw: str)

Interpret a given status word.

Parameters

• sw_data – Hierarchical dict of status word matches
• sw – status word to match (string of 4 hex digits)

Returns

tuple of two strings (class_string, description)

1.3.2 pySim commands abstraction

pySim: SIM Card commands according to ISO 7816-4 and TS 11.11

1.3.3 pySim Transport

The pySim.transport classes implement specific ways how to communicate with a SIM card. A “transport” provides ways to transceive APDUs with the card.

The most commonly used transport uses the PC/SC interface to utilize a variety of smart card interfaces (“readers”).

1.3. pySim library
Transport base class

pySim: PCSC reader transport link base

class pySim.transport.LinkBase(sw_interpreter=None, apdu_tracer=None):
    Base class for link/transport to card.

    abstract connect()  
    Connect to a card immediately

    abstract disconnect()  
    Disconnect from card

    abstract reset_card()  
    Resets the card (power down/up)

    send_apdu(pdu)
    Sends an APDU and auto fetch response data

    Parameters
    pdu – string of hexadecimal characters (ex. “A0A40000023F00”)

    Returns

    tuple(data, sw), where
    data : string (in hex) of returned data (ex. “074F4EFFFF”)  
    sw : string (in hex) of status word (ex. “9000”)

    send_apdu_checksw(pdu, sw='9000')
    Sends an APDU and check returned SW

    Parameters

    • pdu – string of hexadecimal characters (ex. “A0A40000023F00”)

    • sw – string of 4 hexadecimal characters (ex. “9000”). The user may mask out certain digits using a ‘?’ to add some ambiguity if needed.

    Returns

    tuple(data, sw), where
    data : string (in hex) of returned data (ex. “074F4EFFFF”)  
    sw : string (in hex) of status word (ex. “9000”)

    send_apdu_constr(cla, ins, p1, p2, cmd_constr, cmd_data, resp_constr)
    Build and sends an APDU using a ‘construct’ definition; parses response.

    Parameters

    • clia – string (in hex) ISO 7816 class byte

    • ins – string (in hex) ISO 7816 instruction byte

    • p1 – string (in hex) ISO 7116 Parameter 1 byte

    • p2 – string (in hex) ISO 7116 Parameter 2 byte

    • cmd_constr – defining how to generate binary APDU command data

    • cmd_data – command data passed to cmd_constr

    • resp_constr – defining how to decode binary APDU response data
Returns
Tupel of (decoded_data, sw)

send_apdu_constr_checksw(cla, ins, p1, p2, cmd_constr, cmd_data, resp_constr, sw_exp='9000')
Build and sends an APDU using a ‘construct’ definition; parses response.

Parameters
- **cla** – string (in hex) ISO 7816 class byte
- **ins** – string (in hex) ISO 7816 instruction byte
- **p1** – string (in hex) ISO 7116 Parameter 1 byte
- **p2** – string (in hex) ISO 7116 Parameter 2 byte
- **cmd_constr** – defining how to generate binary APDU command data
- **cmd_data** – command data passed to cmd_constr
- **resp_constr** – defining how to decode binary APDU response data
- **exp_sw** – string (in hex) of status word (ex. “9000”)

Returns
Tupel of (decoded_data, sw)

send_apdu_raw(pdu: str)
Sends an APDU with minimal processing

Parameters
- **pdu** – string of hexadecimal characters (ex. “A0A4000023F00”)

Returns
tuple(data, sw), where
- data : string (in hex) of returned data (ex. “074F4EFFFF”) sw : string (in hex) of status word (ex. “9000”)

set_sw_interpreter(interp)
Set an (optional) status word interpreter.

abstract wait_for_card(timeout: Optional[int] = None, newcardonly: bool = False)
Wait for a card and connect to it

Parameters
- **timeout** – Maximum wait time in seconds (None=no timeout)
- **newcardonly** – Should we wait for a new card, or an already inserted one?

pySim.transport.argparse_add_reader_args(arg_parser)
Add all reader related arguments to the given argparse.ArgumentParser instance.

pySim.transport.init_reader(opts, **kwargs) → Optional[LinkBase]
Init card reader driver
**calypso / OsmocomBB transport**

This allows the use of the SIM slot of an OsmocomBB compatible phone with the TI Calypso chipset, using the L1CTL interface to talk to the layer1.bin firmware on the phone.

```python
class pySim.transport.calypso.CalypsoSimLink(sock_path: str = '/tmp/osmocom_l2', **kwargs)

Transport Link for Calypso based phones.

    connect()
    Connect to a card immediately

    disconnect()
    Disconnect from card

    reset_card()
    Resets the card (power down/up)

    wait_for_card(timeout=None, newcardonly=False)
    Wait for a card and connect to it

    Parameters
    • timeout – Maximum wait time in seconds (None=no timeout)
    • newcardonly – Should we wait for a new card, or an already inserted one ?
```

**AT-command Modem transport**

This transport uses AT commands of a cellular modem in order to get access to the SIM card inserted in such a modem.

```python
class pySim.transport.modem_atcmd.ModemATCommandLink(device: str = '/dev/ttyUSB0', baudrate: int = 115200, **kwargs)

Transport Link for 3GPP TS 27.007 compliant modems.

    connect()
    Connect to a card immediately

    disconnect()
    Disconnect from card

    reset_card()
    Resets the card (power down/up)

    wait_for_card(timeout=None, newcardonly=False)
    Wait for a card and connect to it

    Parameters
    • timeout – Maximum wait time in seconds (None=no timeout)
    • newcardonly – Should we wait for a new card, or an already inserted one ?
```
**PC/SC transport**

PC/SC is the standard API for accessing smart card interfaces on all major operating systems, including the MS Windows Family, OS X as well as Linux / Unix OSs.

```python
class pySim.transport.pcsc.PcscSimLink(reader_number: int = 0, **kwargs):
    pySim: PCSC reader transport link.
    connect()  # Connect to a card immediately
    disconnect()  # Disconnect from card
    reset_card()  # Resets the card (power down/up)
    wait_for_card(timeout: Optional[int] = None, newcardonly: bool = False)
        Wait for a card and connect to it
        Parameters
        • timeout – Maximum wait time in seconds (None=no timeout)
        • newcardonly – Should we wait for a new card, or an already inserted one ?
```

**Serial/UART transport**

This transport implements interfacing smart cards via very simplistic UART readers. These readers basically wire together the Rx+Tx pins of a RS232 UART, provide a fixed crystal oscillator for clock, and operate the UART at 9600 bps. These readers are sometimes called *Phoenix*.

```python
class pySim.transport.serial.SerialSimLink(device: str = '/dev/ttyUSB0', baudrate: int = 9600, rst: str = '-rts', debug: bool = False, **kwargs):
    pySim: Transport Link for serial (RS232) based readers included with simcard
    connect()  # Connect to a card immediately
    disconnect()  # Disconnect from card
    reset_card()  # Resets the card (power down/up)
    wait_for_card(timeout=None, newcardonly=False)
        Wait for a card and connect to it
        Parameters
        • timeout – Maximum wait time in seconds (None=no timeout)
        • newcardonly – Should we wait for a new card, or an already inserted one ?
```
1.3.4 pySim construct utilities

```python
class pySim.construct.BcdAdapter(subcon)
    convert a bytes() type to a string of BCD nibbles.
```

```python
pySim.construct.BitsRFU(n=1)
    Field that packs Reserved for Future Use (RFU) bit(s) as defined in TS 31.101 Sec. “3.4 Coding Conventions”
    Use this for (currently) unused/reserved bits whose contents should be initialized automatically but should not
    be cleared in the future or when restoring read data (unlike padding).
    Parameters
        n (Integer) – Number of bits (default: 1)
```

```python
pySim.construct.BitsRFU(n=1)
    Field that packs Reserved for Future Use (RFU) byte(s) as defined in TS 31.101 Sec. “3.4 Coding Conventions”
    Use this for (currently) unused/reserved bytes whose contents should be initialized automatically but should not
    be cleared in the future or when restoring read data (unlike padding).
    Parameters
        n (Integer) – Number of bytes (default: 1)
```

```python
class pySim.construct.GreedyInteger(signed=False, swapped=False, minlen=0)
    A variable-length integer implementation, think of combining GreedyBytes with BytesInteger.
```

```python
pySim.construct.GsmString(n)
    GSM 03.38 encoded byte string of fixed length n. Encoder appends padding bytes (b‘xff’) to maintain length.
    Decoder removes those trailing bytes.
    Exceptions are raised for invalid characters and length excess.
    Parameters
        n (Integer) – Fixed length of the encoded byte string
```

```python
class pySim.construct.GsmStringAdapter(subcon, codec='gsm03.38', err='strict')
    Convert GSM 03.38 encoded bytes to a string.
```

```python
class pySim.construct.HexAdapter(subcon)
    convert a bytes() type to a string of hex nibbles.
```

```python
class pySim.construct.InvertAdapter(subcon)
    inverse logic (false->true, true->false).
```

```python
class pySim.construct.Rpad(subcon, pattern=b‘xff’)
    Encoder appends padding bytes (b’xff’) up to target size. Decoder removes trailing padding bytes.
    Parameters
        • subcon – Subconstruct as defined by construct library
        • pattern – set padding pattern (default: b’xff’)
```

```python
pySim.construct.filter_dict(d, exclude_prefix='_')
    filter the input dict to ensure no keys starting with ‘exclude_prefix’ remain.
```

```python
pySim.construct.normalize_construct(c)
    Convert a construct specific type to a related base type, mostly useful so we can serialize it.
```
parse_construct(c, raw_bin_data: bytes, length: Optional[int] = None, exclude_prefix: str = '.')

Helper function to wrap around normalize_construct() and filter_dict().

1.3.5 pySim TLV utilities

Object-oriented TLV parser/encoder library.

class pySim.tlv.BER_TLV_IE(**kwargs)
   TLV_IE formatted as ASN.1 BER described in ITU-T X.690 8.1.2.

class pySim.tlv.COMPR_TLV_IE(**kwargs)
   TLV_IE formatted as COMPREHENSION-TLV as described in ETSI TS 101 220.

class pySim.tlv.IE(**kwargs)
   Base class for various Information Elements. We understand the notion of a hierarchy of IEs on top of the Transcodable class.

   from_bytes(do: bytes)
      Parse _the value part_ from binary bytes to internal representation.

   from_dict(decoded: dict)
      Set the IE internal decoded representation to data from the argument. If this is a nested IE, the child IE instance list is re-created.

   is_constructed()
      Is this IE constructed by further nested IEs?

   to_bytes() → bytes
      Convert the internal representation _of the value part_ to binary bytes.

   to_dict()
      Return a JSON-serializable dict representing the [nested] IE data.

   abstract to_ie() → bytes
      Convert the internal representation to entire IE including IE header.

class pySim.tlv.TLV_IE(**kwargs)
   Abstract base class for various TLV type Information Elements.

   to_ie()
      Convert the internal representation to entire IE including IE header.

   to_tlv()
      Convert the internal representation to binary TLV bytes.

class pySim.tlv.TLV_IE_Collection(desc=None, **kwargs)
   A TLV_IE_Collection consists of multiple TLV_IE classes identified by their tags. A given encoded DO may contain any of them in any order, and may contain multiple instances of each DO.

   from_bytes(binary: bytes) → List[TLV_IE]
      Create a list of TLV_IEs from the collection based on binary input data. :param binary: binary bytes of encoded data

      Returns
         list of instances of TLV_IE sub-classes containing parsed data
from_dict(decoded: List[dict]) → List[TLV_IE]

Create a list of TLV_IE instances from the collection based on an array of dicts, where they key indicates
the name of the TLV_IE subclass to use.

class pySim.tlv.TlvCollectionMeta(name, bases, namespace, **kwargs)

Metaclass which we use to set some class variables at the time of defining a subclass. This allows us to create
subclasses for each Collection type, where the class represents fixed parameters like the nested IE classes and
instances of it represent the actual TLV data.

class pySim.tlv.TlvMeta(name, bases, namespace, **kwargs)

Metaclass which we use to set some class variables at the time of defining a subclass. This allows us to create
subclasses for each TLV/IE type, where the class represents fixed parameters like the tag/type and instances of it
represent the actual TLV data.

class pySim.tlv.Transcodable

from_bytes(do: bytes)

Convert from binary bytes to internal representation. Store the decoded result in the internal state and return
it.

to_bytes() → bytes

Convert from internal representation to binary bytes. Store the binary result in the internal state and return
it.

pySim.tlv.flatten_dict_lists(inp)

Hierarchically flatten each list-of-dicts into a single dict. This is useful to make the output of hierarchical TLV
decoder structures flatter and more easy to read.

1.3.6 pySim utility functions

pySim: various utilities

class pySim.utils.CardCommand(name, ins, cla_list=None, desc=None)

A single card command / instruction.

match_cla(cla)

Does the given CLA match the CLA list of the command?.

class pySim.utils.CardCommandSet(name, cmds=[])

A set of card instructions, typically specified within one spec.

lookup(ins, cla=None)

look-up the command within the CommandSet.

class pySim.utils.DataObject(name: str, desc: Optional[str] = None, tag: Optional[int] = None)

A DataObject (DO) in the sense of ISO 7816-4. Contrary to ‘normal’ TLVs where one simply has any number
of different TLVs that may occur in any order at any point, ISO 7816 has the habit of specifying TLV data but
with very specific ordering, or specific choices of tags at specific points in a stream. This class tries to represent
this.

Parameters

• name – A brief, all-lowercase, underscore separated string identifier
• desc – A human-readable description of what this DO represents
• tag – The tag associated with this DO
decode(binary: bytes) → Tuple[dict, bytes]
Decode a single DOs from the input data. :param binary: binary bytes of encoded data

Returns
tuple of (decoded_result, binary_remainder)

abstract from_bytes(do: bytes)
Parse the value part of the DO into the internal state of this instance. :param do: binary encoded bytes

from_tlv(do: bytes) → bytes
Parse binary TLV representation into internal state. The resulting decoded representation is _not_ returned, but just internalized in the object instance! :param do: input bytes containing TLV-encoded representation

Returns
bytes remaining at end of ‘do’ after parsing one TLV/DO.

abstract to_bytes() → bytes
Encode the internal state of this instance into the TLV value part. :returns: binary bytes encoding the internal state

to_dict() → dict
Return a dict in form “name: decoded_value”

to_tlv() → bytes
Encode internal representation to binary TLV. :returns: bytes encoded in TLV format.

class pySim.utils.DataObjectChoice(name: str, desc: Optional[str] = None, members=None)
One Data Object from within a choice, identified by its tag. This means that exactly one member of the choice must occur, and which one occurs depends on the tag.

decode(binary: bytes) → Tuple[dict, bytes]
Decode a single DOs from the choice based on the tag. :param binary: binary bytes of encoded data

Returns
tuple of (decoded_result, binary_remainder)

class pySim.utils.DataObjectCollection(name: str, desc: Optional[str] = None, members=None)
A DataObjectCollection consists of multiple Data Objects identified by their tags. A given encoded DO may contain any of them in any order, and may contain multiple instances of each DO.

decode(binary: bytes) → Tuple[List, bytes]
Decode any number of DOs from the collection until the end of the input data, or uninitialized memory (0xFF) is found. :param binary: binary bytes of encoded data

Returns
tuple of (decoded_result, binary_remainder)

class pySim.utils.DataObjectSequence(name: str, desc: Optional[str] = None, sequence=None)
A sequence of DataObjects or DataObjectChoices. This allows us to express a certain ordered sequence of DOs or choices of DOs that have to appear as per the specification. By wrapping them into this formal DataObject-Sequence, we can offer convenience methods for encoding or decoding an entire sequence.

decode(binary: bytes) → Tuple[list, bytes]
Decode a sequence by calling the decoder of each element in the sequence. :param binary: binary bytes of encoded data

Returns
tuple of (decoded_result, binary_remainder)
decode_multi(do: bytes) → Tuple[list, bytes]

Decode multiple occurrences of the sequence from the binary input data. :param do: binary input data to be decoded

Returns

list of results of the decoder of this sequences

encode(decoded) → bytes

Encode a sequence by calling the encoder of each element in the sequence.

encode_multi(decoded) → bytes

Encode multiple occurrences of the sequence from the decoded input data. :param decoded: list of json-serializable input data; one sequence per list item

Returns

binary encoded output data

class pySim.utils.JsonEncoder(*, skipkeys=False, ensure_ascii=True, check_circular=True, allow_nan=True, sort_keys=False, indent=None, separators=None, default=None)

Extend the standard library JSONEncoder with support for more types.

Constructor for JSONEncoder, with sensible defaults.

If skipkeys is false, then it is a TypeError to attempt encoding of keys that are not str, int, float or None. If skipkeys is True, such items are simply skipped.

If ensure_ascii is true, the output is guaranteed to be str objects with all incoming non-ASCII characters escaped. If ensure_ascii is false, the output can contain non-ASCII characters.

If check_circular is true, then lists, dicts, and custom encoded objects will be checked for circular references during encoding to prevent an infinite recursion (which would cause an OverflowError). Otherwise, no such check takes place.

If allow_nan is true, then NaN, Infinity, and -Infinity will be encoded as such. This behavior is not JSON specification compliant, but is consistent with most JavaScript based encoders and decoders. Otherwise, it will be a ValueError to encode such floats.

If sort_keys is true, then the output of dictionaries will be sorted by key; this is useful for regression tests to ensure that JSON serializations can be compared on a day-to-day basis.

If indent is a non-negative integer, then JSON array elements and object members will be pretty-printed with that indent level. An indent level of 0 will only insert newlines. None is the most compact representation.

If specified, separators should be an (item_separator, key_separator) tuple. The default is (`,`, `:`) if indent is None and (`,`, `:`) otherwise. To get the most compact JSON representation, you should specify (``, `:`) to eliminate whitespace.

If specified, default is a function that gets called for objects that can’t otherwise be serialized. It should return a JSON encodable version of the object or raise a TypeError.

default(o)

Implement this method in a subclass such that it returns a serializable object for o, or calls the base implementation (to raise a TypeError).

For example, to support arbitrary iterators, you could implement default like this:

```python
def default(self, o):
    try:
        iterable = iter(o)
```
except TypeError:
    pass
else:
    return list(iterable)
# Let the base class default method raise the TypeError
return JSONEncoder.default(self, o)

class pySim.utils.TL0_DataObject(name: str, desc: str, tag: int, val=None):
    Data Object that has Tag, Len=0 and no Value part.

    Parameters
    • name – A brief, all-lowercase, underscore separated string identifier
    • desc – A human-readable description of what this DO represents
    • tag – The tag associated with this DO

    from_bytes(binary: bytes)
    Parse the value part of the DO into the internal state of this instance. :param do: binary encoded bytes

to_bytes() → bytes
    Encode the internal state of this instance into the TLV value part. :returns: binary bytes encoding the
    internal state

pySim.utils.TLV_parser([0xAA, ... ,0xFF]) → [T, L, [V], T, L, [V], ...]
    loops on the input list of bytes with the “first_TLV_parser()” function returns a list of 3-Tuples

pySim.utils.all_subclasses(cls) → set
    Recursively get all subclasses of a specified class

pySim.utils.auto_int(x)
    Helper function for argparse to accept hexadecimal integers.

pySim.utils.b2h(b: bytearray) → str
    convert from a sequence of bytes to a string of hex nibbles

pySim.utils.bertlv_encode_len(length: int) → bytes
    Encode a single Length value according to ITU-T X.690 8.1.3; only the definite form is supported here. :param
    length: length value to be encoded
    Returns
    binary output data of BER-TLV length field

pySim.utils.bertlv_encode_tag(t) → bytes
    Encode a single Tag value according to ITU-T X.690 8.1.2

pySim.utils.bertlv_parse_len(binary: bytes) → Tuple[int, bytes]
    Parse a single Length value according to ITU-T X.690 8.1.3; only the definite form is supported here. :param
    binary: binary input data of BER-TLV length field
    Returns
    Tuple of (length, remainder)

pySim.utils.bertlv_parse_one(binary: bytes) → Tuple[dict, int, bytes, bytes]
    Parse a single TLV IE at the start of the given binary data. :param binary: binary input data of BER-TLV length field

1.3. pySim library
Returns
dict, len: int, remainder: bytes)

Return type
tuple of (tag

pySim.utils.bertlv_parse_tag(binary: bytes) \rightarrow Tuple[dict, bytes]
Parse a single Tag value according to ITU-T X.690 8.1.2 :param binary: binary input data of BER-TLV length field

Returns
int, constructed: bool, tag: int], remainder: bytes)

Return type
tuple of (class

pySim.utils.bertlv_parse_tag_raw(binary: bytes) \rightarrow Tuple[int, bytes]
Get a single raw Tag from start of input according to ITU-T X.690 8.1.2 :param binary: binary input data of BER-TLV length field

Returns: Tuple of (tag: int, remainder: bytes)

pySim.utils.boxed_heading_str(heading, width=80)
Generate a string that contains a boxed heading.

pySim.utils.calculate_luhn(cc) \rightarrow int
Calculate Luhn checksum used in e.g. ICCID and IMEI

pySim.utils.comprehensiontlv_encode_tag(tag) \rightarrow bytes
Encode a single Tag according to ETSI TS 101 220 Section 7.1.1

pySim.utils.comprehensiontlv_parse_one(binary: bytes) \rightarrow Tuple[dict, int, bytes, bytes]
Parse a single TLV IE at the start of the given binary data. :param binary: binary input data of BER-TLV length field

Returns
dict, len: int, remainder: bytes)

Return type
tuple of (tag

pySim.utils.comprehensiontlv_parse_tag(binary: bytes) \rightarrow Tuple[dict, bytes]
Parse a single Tag according to ETSI TS 101 220 Section 7.1.1

pySim.utils.comprehensiontlv_parse_tag_raw(binary: bytes) \rightarrow Tuple[int, bytes]
Parse a single Tag according to ETSI TS 101 220 Section 7.1.1

pySim.utils.dec_addr_tlv(hexstr)
Decode hex string to get EF.P-CSCF Address or EF.ePDGId or EF.ePDGIdEm. See 3GPP TS 31.102 version 13.4.0 Release 13, section 4.2.8, 4.2.102 and 4.2.104.

pySim.utils.dec_ePDGSelection(sixhexbytes)
Decode ePDGSelection to get EF.ePDGSelection or EF.ePDGSelectionEm. See 3GPP TS 31.102 version 15.2.0 Release 15, section 4.2.104 and 4.2.106.

pySim.utils.dec_imsi(ef: str) \rightarrow Optional[str]
Converts an EF value to the IMSI string representation
pySim.utils.dec_msisdn(ef_msisdn: str) → Optional[Tuple[int, int, Optional[str]]]
    Decode MSISDN from EF.MSISDN or EF.ADN (same structure). See 3GPP TS 31.102, section 4.2.26 and 4.4.2.3.

pySim.utils.dec_spn(ef)
    Obsolete, kept for API compatibility

pySim.utils.dec_st(st, table='sim') → str
    Parses the EF S/U/IST and prints the list of available services in EF S/U/IST

pySim.utils.derive_mcc(digit1: int, digit2: int, digit3: int) → int
    Derive decimal representation of the MCC (Mobile Country Code) from three given digits.

pySim.utils.derive_milenage_opc(ki_hex: str, op_hex: str) → str
    Run the milenage algorithm to calculate OPC from Ki and OP

pySim.utils.derive_mnc(digit1: int, digit2: int, digit3: int = 15) → int
    Derive decimal representation of the MNC (Mobile Network Code) from two or (optionally) three given digits.

pySim.utils.enc_addr_tlv(addr, addr_type='00')
    Encode address TLV object used in EF.P-CSCF Address, EF.ePDGId and EF.ePDGIdEm. See 3GPP TS 31.102 version 13.4.0 Release 13, section 4.2.8, 4.2.102 and 4.2.104.

    Default values:
    • addr_type: 00 - FQDN format of Address

pySim.utils.enc_ePDGSelection(hexstr, mcc, mnc, epdg_priority='0001', epdg_fqdn_format='00')
    Encode ePDGSelection so it can be stored at EF.ePDGSelection or EF.ePDGSelectionEm. See 3GPP TS 31.102 version 15.2.0 Release 15, section 4.2.104 and 4.2.106.

    Default values:
    • epdg_priority: ‘0001’ - 1st Priority
    • epdg_fqdn_format: ‘00’ - Operator Identifier FQDN

pySim.utils.enc_imsi(imsi: str)
    Converts a string IMSI into the encoded value of the EF

pySim.utils.enc_msisdn(msisdn: str, npi: int = 1, ton: int = 3) → str
    Encode MSISDN as LHV so it can be stored to EF.MSISDN. See 3GPP TS 31.102, section 4.2.26 and 4.4.2.3. (The result will not contain the optional Alpha Identifier at the beginning.)

    Default NPI / ToN values:
    • NPI: ISDN / telephony numbering plan (E.164 / E.163),
    • ToN: network specific or international number (if starts with ‘+’).

pySim.utils.enc_plmn(mcc: str, mnc: str) → str
    Converts integer MCC/MNC into 3 bytes for EF

pySim.utils.enc_spn(name: str, show_in_hplmn=False, hide_in_oplnm=False)
    Obsolete, kept for API compatibility

pySim.utils.enc_st(st, service, state=1)
    Encodes the EF S/U/IST/EST and returns the updated Service Table

    Parameters
    • Table(st - Current value of SIM/USIM/ISIM Service) –
• **activated/de-activated** *(service - Service Number to encode as)* –
  • **activate** *(state - 1 mean)* –
  • **de-activate** *(0 means)* –

**Returns**

s - Modified value of SIM/USIM/ISIM Service Table

**Default values:**

• state: 1 - Sets the particular Service bit to 1

**pySim.utils.expand_hex**(hexstring, length)

Expand a given hexstring to a specified length by replacing “.” or “..”
with a filler that is derived from the neighboring nibbles respective bytes. Usually this will be the nibble respective byte before “.” or “..”, except when the string begins with “.” or “..”, then the nibble respective byte after “.” or “..” is used.”. In case the string cannot be expanded for some reason, the input string is returned unmodified.

**Parameters**

• **hexstring** – hexstring to expand

• **length** – desired length of the resulting hexstring.

**Returns**

expanded hexstring

**pySim.utils.first_TLV_parser**([0xAA, 0x02, 0xAB, 0xCD, 0xFF, 0x00]) -> (170, 2, [171, 205])

parses first TLV format record in a list of bytelist returns a 3-Tuple: Tag, Length, Value Value is a list of bytes parsing of length is ETSI'style 101.220

**pySim.utils.get_addr_type**(addr)

Validates the given address and returns it’s type (FQDN or IPv4 or IPv6) Return: 0x00 (FQDN), 0x01 (IPv4), 0x02 (IPv6), None (Bad address argument given)

TODO: Handle IPv6

**pySim.utils.h2b**(s: str) → bytearray

convert from a string of hex nibbles to a sequence of bytes

**pySim.utils.h2i**(s: str) → List[int]

convert from a string of hex nibbles to a list of integers

**pySim.utils.h2s**(s: str) → str

convert from a string of hex nibbles to an ASCII string

**pySim.utils.i2h**(s: List[int]) → str

convert from a list of integers to a string of hex nibbles

**pySim.utils.i2s**(s: List[int]) → str

convert from a list of integers to an ASCII string

**pySim.utils.is_hex**(string: str, minlen: int = 2, maxlen: Optional[int] = None) → bool

Check if a string is a valid hexstring
pySim.utils.lpad(s: str, l: int, c='f') → str
    pad string on the left side. :param s: string to pad :param l: total length to pad to :param c: padding character

    Returns
    String ‘s’ padded with as many ‘c’ as needed to reach total length of ‘l’

pySim.utils.mcc_from_imsi(imsi: str) → Optional[str]
    Derive the MCC (Mobile Country Code) from the first three digits of an IMSI

pySim.utils.mnc_from_imsi(imsi: str, long: bool = False) → Optional[str]
    Derive the MNC (Mobile Country Code) from the 4th to 6th digit of an IMSI

pySim.utils.rpad(s: str, l: int, c='f') → str
    pad string on the right side. :param s: string to pad :param l: total length to pad to :param c: padding character

    Returns
    String ‘s’ padded with as many ‘c’ as needed to reach total length of ‘l’

pySim.utils.s2h(s: str) → str
    convert from an ASCII string to a string of hex nibbles

pySim.utils.sanitize_pin_adm(pin_adm, pin_adm_hex=None) → str
    The ADM pin can be supplied either in its hexadecimal form or as ascii string. This function checks the supplied
    opts parameter and returns the pin_adm as hex encoded string, regardless in which form it was originally supplied
    by the user

pySim.utils.str_sanitize(s: str) → str
    replace all non printable chars, line breaks and whitespaces, with ‘ ‘, make sure that there are no whitespaces at
    the end and at the beginning of the string.

    Parameters
    s – string to sanitize

    Returns
    filtered result of string ‘s’

pySim.utils.sw_match(sw: str, pattern: str) → bool
    Match given SW against given pattern.

pySim.utils.swap_nibbles(s: str) → str
    swap the nibbles in a hex string

pySim.utils.tabulate_str_list(str_list, width: int = 79, hspace: int = 2, lspace: int = 1, align_left: bool = True) → str
    Pretty print a list of strings into a tabulated form.

    Parameters
    • width – total width in characters per line
    • space – horizontal space between cells
    • lspace – number of spaces before row
    • align_left – Align text to the left side

    Returns
    multi-line string containing formatted table
1.3.7 pySim exceptions

pySim: Exceptions

exception pySim.exceptions.NoCardError
    No card was found in the reader.

exception pySim.exceptions.ProtocolError
    Some kind of protocol level error interfacing with the card.

exception pySim.exceptions.ReaderError
    Some kind of general error with the card reader.

exception pySim.exceptions.SwMatchError(sw_actual: str, sw_expected: str, rs=None)
    Raised when an operation specifies an expected SW but the actual SW from the card doesn't match.

    Parameters
    - **sw_actual** – the SW we actually received from the card (4 hex digits)
    - **sw_expected** – the SW we expected to receive from the card (4 hex digits)
    - **rs** – interpreter class to convert SW to string

1.3.8 pySim card_handler

pySim: card handler utilities. A ‘card handler’ is some method by which cards can be inserted/removed into the card reader. For normal smart card readers, this has to be done manually. However, there are also automatic card feeders.

class pySim.card_handler.CardHandler(sl: LinkBase)
    Manual card handler: User is prompted to insert/remove card from the reader.

class pySim.card_handler.CardHandlerAuto(sl: LinkBase, config_file: str)
    Automatic card handler: A machine is used to handle the cards.

class pySim.card_handler.CardHandlerBase(sl: LinkBase)
    Abstract base class representing a mechanism for card insertion/removal.

    done()
        Method called when pySim failed to program a card. Move card to ‘good’ batch.

    error()
        Method called when pySim failed to program a card. Move card to ‘bad’ batch.

    get(first: bool = False)
        Method called when pySim needs a new card to be inserted.

        Parameters
        - **first** – set to true when the get method is called the first time. This is required to prevent blocking when a card is already inserted into the reader. The reader API would not recognize that card as “new card” until it would be removed and re-inserted again.
### 1.3.9 pySim card_key_provider

Obtaining card parameters (mostly key data) from external source.

This module contains a base class and a concrete implementation of obtaining card key material (or other card-individual parameters) from an external data source.

This is used e.g. to keep PIN/PUK data in some file on disk, avoiding the need of manually entering the related card-individual data on every operation with pySim-shell.

class pySim.card_key_provider.CardKeyProvider
    Base class, not containing any concrete implementation.

    abstract get(fields: List[str], key: str, value: str) -> Dict[str, str]
    Get multiple card-individual fields for identified card.

    Parameters
    • fields – list of valid field names such as ‘ADM1’, ‘PIN1’, … which are to be obtained
    • key – look-up key to identify card data, such as ‘ICCID’
    • value – value for look-up key to identify card data

    Returns
dictionary of {field, value} strings for each requested field from ‘fields’

get_field(field: str, key: str = 'ICCID', value: str = '') -> Optional[str]
get a single field from CSV file using a specified key/value pair

class pySim.card_key_provider.CardKeyProviderCsv(filename: str)
    Card key provider implementation that allows to query against a specified CSV file

    Parameters
    • filename – file name (path) of CSV file containing card-individual key/data

    get(fields: List[str], key: str, value: str) -> Dict[str, str]
    Get multiple card-individual fields for identified card.

    Parameters
    • fields – list of valid field names such as ‘ADM1’, ‘PIN1’, … which are to be obtained
    • key – look-up key to identify card data, such as ‘ICCID’
    • value – value for look-up key to identify card data

    Returns
dictionary of {field, value} strings for each requested field from ‘fields’

pySim.card_key_provider.card_key_provider_get(fields, key: str, value: str, provider_list=[]) -> Dict[str, str]
Query all registered card data providers for card-individual [key] data.

    Parameters
    • fields – list of valid field names such as ‘ADM1’, ‘PIN1’, … which are to be obtained
    • key – look-up key to identify card data, such as ‘ICCID’
    • value – value for look-up key to identify card data
    • provider_list – override the list of providers from the global default
Returns
dictionary of {field, value} strings for each requested field from ‘fields’

`pySim.card_key_provider.card_key_provider_get_field(field: str, key: str, value: str, provider_list=[])` → Optional[str]

Query all registered card data providers for a single field.

Parameters
- **field** – name valid field such as ‘ADM1’, ‘PIN1’, … which is to be obtained
- **key** – look-up key to identify card data, such as ‘ICCID’
- **value** – value for look-up key to identify card data
- **provider_list** – override the list of providers from the global default

Returns
dictionary of {field, value} strings for the requested field

`pySim.card_key_provider.card_key_provider_register(provider: CardKeyProvider, provider_list=[])`

Register a new card key provider.

Parameters
- **provider** – the to-be-registered provider
- **provider_list** – override the list of providers from the global default
CHAPTER
TWO

INDICES AND TABLES

- genindex
- modindex
- search
pySim.card_handler, 52  
pySim.card_key_provider, 53  
pySim.commands, 37  
pySim.construct, 42  
pySim.exceptions, 52  
pySim.filesystem, 25  
pySim.tlv, 43  
pySim.transport, 38  
pySim.transport.calypso, 40  
pySim.transport.modem_atcmd, 40  
pySim.transport.pcsc, 41  
pySim.transport.serial, 41  
pySim.utils, 44
## INDEX

| A | activate_file()  | (pySim.filesystem.RuntimeState method), 32 |
|   | add_application_df() | (pySim.filesystem.CardMF method), 29 |
|   | add_file()  | (pySim.filesystem.CardDF method), 27 |
|   | add_files() | (pySim.filesystem.CardModel class method), 30 |
|   | all_subclasses() | (in module pySim.utils), 47 |
|   | apply_matching_models() | (pySim.filesystem.CardModel static method), 30 |
|   | argparse_add_reader_args() | (in module pySim.transport), 39 |
|   | auto_int()  | (in module pySim.utils), 47 |
| B | b2h() | (in module pySim.utils), 47 |
|   | BcdAdapter | (class in pySim.construct), 42 |
|   | BER_TLV_IE | (class in pySim.tlv), 43 |
|   | bertlv_encode_len() | (in module pySim.utils), 47 |
|   | bertlv_encode_tag()  | (in module pySim.utils), 47 |
|   | bertlv_parse_len() | (in module pySim.utils), 47 |
|   | bertlv_parse_one()  | (in module pySim.utils), 47 |
|   | bertlv_parse_tag() | (in module pySim.utils), 48 |
|   | bertlv_parse_tag_raw()  | (in module pySim.utils), 48 |
|   | BerTlvEF | (class in pySim.filesystem), 25 |
|   | BerTlvEF.ShellCommands | (class in pySim.filesystem), 25 |
|   | BitsRFU()  | (in module pySim.construct), 42 |
|   | boxed_heading_str() | (in module pySim.utils), 48 |
|   | build_select_path_to() | (pySim.filesystem.CardFile method), 28 |
|   | BytesRFU() | (in module pySim.construct), 42 |
| C | calculate_luhn() | (in module pySim.utils), 48 |
|   | CalypsoSimLink | (class in pySim.transport.calypso), 40 |
|   | card_key_provider_get() | (in module pySim.card_key_provider), 53 |
|   | card_key_provider_get_field() | (in module pySim.card_key_provider), 54 |
|   | card_key_provider_register() | (in module pySim.card_key_provider), 54 |
|   | CardADDF | (class in pySim.filesystem), 26 |
|   | CardApplication | (class in pySim.filesystem), 26 |
|   | CardCommand | (class in pySim.utils), 44 |
|   | CardCommandSet | (class in pySim.utils), 44 |
|   | CardDF | (class in pySim.filesystem), 26 |
|   | CardDF.ShellCommands | (class in pySim.filesystem), 26 |
|   | CardEF | (class in pySim.filesystem), 27 |
|   | CardFile | (class in pySim.filesystem), 28 |
|   | CardHandler | (class in pySim.card_handler), 52 |
|   | CardHandlerAuto | (class in pySim.card_handler), 52 |
|   | CardHandlerBase | (class in pySim.card_handler), 52 |
|   | CardKeyProvider | (class in pySim.card_key_provider), 53 |
|   | CardKeyProviderCsv | (class in pySim.card_key_provider), 53 |
|   | CardMF | (class in pySim.filesystem), 29 |
|   | CardModel | (class in pySim.filesystem), 29 |
|   | COMPR_TLV_IE | (class in pySim.tlv), 43 |
|   | comprehensiotlv_encode_tag() | (in module pySim.utils), 48 |
|   | comprehensiotlv_parse_one() | (in module pySim.utils), 48 |
|   | comprehensiotlv_parse_tag() | (in module pySim.utils), 48 |
|   | comprehensiotlv_parse_tag_raw() | (in module pySim.utils), 48 |
|   | connect() | (pySim.transport.calypso.CalypsoSimLink method), 40 |
|   | connect() | (pySim.transport.LinkBase method), 38 |
|   | connect() | (pySim.transport.modem_atcmd.ModemATCommandLink method), 40 |
|   | connect() | (pySim.transport.psc.PscSimLink method), 41 |
|   | CyclicEF | (class in pySim.filesystem), 30 |
D

DataObject (class in pySim.utils), 44
DataObjectChoice (class in pySim.utils), 45
DataObjectCollection (class in pySim.utils), 45
DataObjectSequence (class in pySim.utils), 45

dec_addr_tlv() (in module pySim.utils), 48
dec_ePDGSelection() (in module pySim.utils), 48
dec_imsi() (in module pySim.utils), 48
dec_msisdn() (in module pySim.utils), 48
dec_spn() (in module pySim.utils), 49
dec_st() (in module pySim.utils), 49
decode() (pySim.utils.DataObject method), 44
decode() (pySim.utils.DataObjectChoice method), 45
decode() (pySim.utils.DataObjectCollection method), 45
decode() (pySim.utils.DataObjectSequence method), 45
decode_bin() (pySim.filesystem.TransparentEF method), 36
decode_hex() (pySim.filesystem.TransparentEF method), 36
decode_multi() (pySim.utils.DataObjectSequence method), 45
decode_record_bin() (pySim.filesystem.LinFixedEF method), 31
decode_record_bin() (pySim.filesystem.TransRecEF method), 35
decode_record_hex() (pySim.filesystem.LinFixedEF method), 31
decode_record_hex() (pySim.filesystem.TransRecEF method), 35
decode_select_response() (pySim.filesystem.CardFile method), 28
decode_select_response() (pySim.filesystem.CardMF method), 29
default() (pySim.utils.JsonEncoder method), 29
derive_mcc() (in module pySim.utils), 49
derive_milenage_opc() (in module pySim.utils), 49
derive_mmc() (in module pySim.utils), 49
disconnect() (pySim.transport.calypso.CalypsoSimLink method), 40
disconnect() (pySim.transport.LinkBase method), 38
disconnect() (pySim.transport.modem_atcmd.ModemATCommands method), 40
disconnect() (pySim.transport.psc.PscSimLink method), 41
disconnect() (pySim.transport.serial.SerialSimLink method), 41
do_decode_hex() (pySim.filesystem.LinFixedEF.ShellCommands method), 30
do_decode_hex() (pySim.filesystem.TransparentEF.ShellCommands method), 37
do_decode_hex() (pySim.filesystem.BerTlvEF.ShellCommands method), 46
do_decode_hex() (pySim.filesystem.TransparentEF.ShellCommands method), 36
do_delete_data() (pySim.filesystem.BerTlvEF.ShellCommands method), 25
do_edit_binary_decoded()

E

enc_addr_tlv() (in module pySim.utils), 49enc_ePDGSelection() (in module pySim.utils), 49enc_imsi() (in module pySim.utils), 49enc_msisdn() (in module pySim.utils), 49enc_plmn() (in module pySim.utils), 49enc_spn() (in module pySim.utils), 49enc_st() (in module pySim.utils), 49
encode() (pySim.utils.DataObjectSequence method), 46
encode_bin() (pySim.filesystem.TransparentEF method), 37
encode_hex() (pySim.filesystem.TransparentEF method), 37
encode_multi() (pySim.utils.DataObjectSequence method), 46
Index
swap_nibbles() (in module pySim.utils), 51
SwMatchError, 52

T

tabulate_str_list() (in module pySim.utils), 51
TL0_DataObject (class in pySim.utils), 47
TLV_IE (class in pySim.tlv), 43
TLV_IE_Collection (class in pySim.tlv), 43
TLV_parser() (in module pySim.utils), 47
TlvCollectionMeta (class in pySim.tlv), 44
TlvMeta (class in pySim.tlv), 44
to_bytes() (pySim.tlv.IE method), 43
to_bytes() (pySim.tlv.Transcodable method), 44
to_bytes() (pySim.utils.DataObject method), 45
to_bytes() (pySim.utils.TL0_DataObject method), 47
to_dict() (pySim.tlv.IE method), 43
to_dict() (pySim.utils.DataObject method), 45
to_ie() (pySim.tlv.IE method), 43
to_ie() (pySim.tlv.TLV_IE method), 43
to_tlv() (pySim.tlv.TLV_IE method), 43
to_tlv() (pySim.utils.DataObject method), 45
Transcodable (class in pySim.tlv), 44
TransparentEF (class in pySim.filesystem), 35
TransparentEF.ShellCommands (class in pySim.filesystem), 36
TransRecEF (class in pySim.filesystem), 34

U

unregister_cmds() (pySim.filesystem.RuntimeState method), 34
update_binary() (pySim.filesystem.RuntimeState method), 34
update_binary_dec() (pySim.filesystem.RuntimeState method), 34
update_record() (pySim.filesystem.RuntimeState method), 34
update_record_dec() (pySim.filesystem.RuntimeState method), 34

W

wait_for_card() (pySim.transport.calypso.CalypsoSimLink method), 40
wait_for_card() (pySim.transport.LinkBase method), 39
wait_for_card() (pySim.transport.modem_atcmd.ModemATCommandLink method), 40
wait_for_card() (pySim.transport.pcsc.PcscSimLink method), 41
wait_for_card() (pySim.transport.serial.SerialSimLink method), 41