osmopysim-usermanual

Sylvain Munaut, Harald Welte, Philipp Maier, Supreeth Herle

Apr 11, 2021

CONTENTS:

CHAPTER

ONE

INTRODUCTION

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/ 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 cmd2 basics

FIXME

1.1.2 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": {
        "shareable": true,
        "file_type": "df",
        "structure": "no_info_given"
    },
    "df_name": "A0000000871002FFFFFFFF8907090000",
    "proprietary_info": {
        "uicc_characteristics": "71",
        "available_memory": 101640
    },
    "life_cycle_status_int": "operational_activated",
    "security_attrib_compact": "00",
```

(continues on next page)

(continued from previous page)

```
    "pin_status_template_do": "90017083010183018183010A83010B"
}
pySIM-shell (MF/ADF.USIM)>
```

change_chv

Change PIN code to a new PIN code

```
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

```
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

```
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

This command allows you to verify a CHV (PIN), which is how the specifications call it if you authenticate yourself with the said CHV/PIN.

Verify (authenticate) using specified PIN code

```
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 currently selected file. This used to be called REHABILITATE in TS 11.11.

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

1.1.3 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

```
usage: dir [-h] [--fids] [--names] [--apps] [--all]
```

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

export

Export files to script that can be imported back later

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

Named Arguments

--filename only export specific file

tree

FIXME

verify_adm

FIXME

1.1.4 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 multiple records from a record-oriented EF

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

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_record_decoded [-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_records_decoded [-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 Data bytes (hex 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.

1.1.5 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 tat field:

Th 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
```

(continues on next page)

(continued from previous page)

```
}
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.

1.1.6 USIM commands

authenticate

Perform Authentication and Key Agreement (AKA).

```
usage: authenticate [-h] rand autn
```

Positional Arguments

rand	Random challenge
autn	Authentication Nonce

1.1.7 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 (3f00/2fe2)> 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`.

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)

Specifically, pySim-read will dump the following:

- MF
- EF.ICCID
- 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

```
Usage: pySim-read.py [options]

Options:
  -h, --help            show this help message and exit
  -d DEV, --device=DEV  Serial Device for SIM access [default: /dev/ttyUSB0]
  -b BAUD, --baud=BAUD  Baudrate used for SIM access [default: 9600]
  -p PCSC, --pcsc-device=PCSC
                        Which PC/SC reader number for SIM access
  --modem-device=DEV    Serial port of modem for Generic SIM Access (3GPP TS
                        27.007)
  --modem-baud=BAUD     Baudrate used for modem's port [default: 115200]
  --osmocon=PATH        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.

class pySim.filesystem.**CardADF**(*aid: str*, ***kwargs*)

ADF (Application Dedicated File) in the smart card filesystem

Args: 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

class pySim.filesystem.**CardApplication**(*name*, *adf: Optional[*pySim.filesystem.CardADF*] = None*, *aid: Optional[str] = None*, *sw: Optional[dict] = None*)

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.

Args: 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

class ShellCommands

add_file (*child:* pySim.filesystem.CardFile, *ignore_existing: bool = False*)
Add a child (DF/EF) to this DF. :param child: The new DF/EF to be added :param ignore_existing: Ignore, if file with given FID already exists. Old one will be kept.

add_files (*children: Iterable[*pySim.filesystem.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=[]*) → 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*) → Optional[*pySim.filesystem.CardFile*]
Find a file with given file ID within current DF.

lookup_file_by_name (*name: Optional[str]*) → Optional[*pySim.filesystem.CardFile*]
Find a file with given name within current DF.

lookup_file_by_sfid (*sfid: Optional[str]*) → Optional[*pySim.filesystem.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

Args: 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

get_selectables (*flags=[]*) → 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: Optional[str] = None*, *sfid: Optional[str] = None*, *name: Optional[str] = None*, *desc: Optional[str] = None*, *parent: Optional[*pySim.filesystem.CardDF*] = 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

decode_select_response(*data_hex: str*)

Decode the response to a SELECT command.

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

get_mf() → Optional[*pySim.filesystem.CardMF*]

Return the MF (root) of the file system.

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(*flags=[]*) → Dict[str, *pySim.filesystem.CardFile*]

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

class pySim.filesystem.**CardMF**(***kwargs*)

MF (Master File) in the smart card filesystem

Args: 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

add_application_df(*app:* pySim.filesystem.CardADF)

Add an Application to the MF

decode_select_response(*data_hex: str*) → Any

Decode the response to a SELECT command.

This is the fall-back method which doesn't perform any decoding. It mostly exists so specific derived classes can overload it for actual decoding.

get_app_names()

Get list of completions (AID names)

get_app_selectables(*flags=[]*) → dict

Get applications by AID + name

get_selectables (*flags=[]*) → 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.**CardProfile** (*name*, ***kw*)

A Card Profile describes a card, it's filesystem hierarchy, an [initial] list of applications as well as profile-specific SW and shell commands. Every card has one card profile, but there may be multiple applications within that profile.

Parameters

- **desc** (*str*) – Description
- **files_in_mf** – List of CardEF instances present in MF
- **applications** – List of CardApplications present on card
- **sw** – List of status word definitions
- **shell_cmdsets** – List of cmd2 shell command sets of profile-specific commands

add_application (*app: pySim.filesystem.CardApplication*)

Add an application to a card profile.

Parameters app – CardApplication instance to be added to profile

interpret_sw (*sw: str*)

Interpret a given status word within the profile.

Parameters sw – Status word as string of 4 hex digits

Returns Tuple of two strings

class pySim.filesystem.**CyclicEF** (*fid: str*, *sfid: Optional[str] = None*, *name: Optional[str] = None*, *desc: Optional[str] = None*, *parent: Optional[pySim.filesystem.CardDF] = None*, *rec_len={1, None}*)

Cyclic EF (Entry File) in the smart card filesystem

Args: 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 : tuple 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[pySim.filesystem.CardDF] = None*, *rec_len={1, None}*)

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** – tuple of (minimum_length, recommended_length)

class ShellCommands
Shell commands specific for Linear Fixed EFs.

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: pySim.filesystem.CardProfile*)

Represent the runtime state of a session with a card.

Parameters

- **`card`** – pysim.cards.Card instance
- **`profile`** – CardProfile instance

`get_application_df`() → Optional[*pySim.filesystem.CardADF*]

Obtain the currently selected application DF (if any).

Returns CardADF() instance or None

`get_cwd`() → *pySim.filesystem.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

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)

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[pySim.filesystem.CardDF] = None*, *size={1, None}*)
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)

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

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_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

encode_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

encode_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").

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.

`connect`()
Connect to a card immediately

`disconnect`()
Disconnect from card

`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

- **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_cosntr** – defining how to generate binary APDU command data
- **cmd_data** – command data passed to cmd_constr
- **resp_cosntr** – defining how to decode binary APDU response data

Returns Tuple 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_cosntr** – defining how to generate binary APDU command data
- **cmd_data** – command data passed to cmd_constr
- **resp_cosntr** – defining how to decode binary APDU response data
- **exp_sw** – string (in hex) of status word (ex. "9000")

Returns Tuple of (decoded_data, sw)

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

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")

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

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.`**`init_reader`**`(`*opts*, ***kwargs*`)` → Optional[*pySim.transport.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.

`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.

`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.

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*.

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

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

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

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

1.3.5 pySim utility functions

pySim: various utilities

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:

```
def default(self, o):
    try:
        iterable = iter(o)
    except TypeError:
        pass
```

(continues on next page)

(continued from previous page)

```
    else:
        return list(iterable)
    # Let the base class default method raise the TypeError
    return JSONEncoder.default(self, o)
```

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

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

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*) → 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_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.

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*, *mnc*)
Converts integer MCC/MNC into 3 bytes for EF

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

Parameters

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

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

Default values:

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

`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.**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_lef** – Align text to the left side

Returns multi-line string containing formatted table

1.3.6 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

1.3.7 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: pySim.transport.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** – FIXME

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

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

1.3.8 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.

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: pySim.card_key_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

PYTHON MODULE INDEX

p

INDEX

E

F

G

H

I

J

L

M

N

P

R

S

T

U

W