
osmopysim-usermanual

Sylvain Munaut, Harald Welte, Philipp Maier, Supreeth Herle, Merl

Jan 18, 2025

CONTENTS:

1	Introduction	1
1.1	pySim-shell	1
1.1.1	Video Presentation	2
1.1.2	Running pySim-shell	2
1.1.3	Usage Examples	4
1.1.4	Advanced Topics	8
1.1.5	cmd2 basics	10
1.1.6	pySim commands	10
1.1.7	ISO7816 commands	15
1.1.8	TS 102 221 commands	19
1.1.9	Linear Fixed EF commands	21
1.1.10	Transparent EF commands	23
1.1.11	BER-TLV EF commands	26
1.1.12	USIM commands	27
1.1.13	File-specific commands	27
1.1.14	UICC Administrative commands	29
1.1.15	ARA-M commands	31
1.1.16	GlobalPlatform commands	33
1.1.17	eUICC ISD-R commands	38
1.1.18	cmd2 settable parameters	45
1.2	pySim-trace	45
1.2.1	Demo	46
1.2.2	Running pySim-trace	46
1.2.3	pySim-trace command line reference	46
1.2.4	Constraints	46
1.3	Legacy tools	46
1.3.1	pySim-prog	46
1.3.2	pySim-read	50
1.4	pySim library	52
1.4.1	pySim filesystem abstraction	52
1.4.2	pySim commands abstraction	63
1.4.3	pySim Transport	68
1.4.4	pySim utility functions	72
1.4.5	pySim exceptions	76
1.4.6	pySim card_handler	76
1.4.7	pySim card_key_provider	77
1.5	osmo-smdpp	78
1.5.1	Running osmo-smdpp	79
1.6	sim-rest-server	80
1.6.1	REST API Calls	80

1.6.2	Concrete example using the included sysmolSIM-SJA2	82
1.7	suci-keytool	82
1.7.1	Generating keys	82
1.7.2	Dumping public keys	83
1.7.3	suci-keytool syntax	83
2	Indices and tables	85
	Python Module Index	87
	Index	89

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 [new] *pySim-trace APDU trace decoder*
- 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, including classic GSM SIM, GSM-R SIM, UICC, USIM, ISIM, HPSIM and recently even eUICC.

If you're familiar with Unix/Linux shells: Think of it like *the bash for SIM cards*.

The pySim-shell interactive shell provides commands 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 most files (where related file-specific encoder/decoder classes have been developed):

- decoded reading (display file data represented in human and machine readable JSON format)
- decoded writing (encode from JSON to binary format, then write)
- if your card supports it, and you have the related privileges: resizing, creating, enabling and disabling of files
- performing GlobalPlatform operations, including establishment of Secure Channel Protocol (SCP), Installing applications, installing key material, etc.
- listing/enabling/disabling/deleting eSIM profiles on Consumer eUICC

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

A typical interactive pySim workflow would look like this:

- starting the program, specifying which smart card interface to use to talk to the card
- verifying the PIN (if needed) or the ADM1 PIN in case you want to write/modify the card
- selecting on-card application dedicated files like ADF.USIM and navigating the tree of DFs
- reading and potentially modifying file contents, in raw binary (hex) or decoded JSON format

1.1.1 Video Presentation

There is a [video recording of the presentation back when pySim-shell was originally released](#). While it is slightly dated, it should still provide a good introduction.

1.1.2 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

```
usage: pySim-shell.py [-h] [-d DEV] [-b BAUD] [--pcsc-shared]
                    [-p PCSC | --pcsc-regex REGEX] [--modem-device DEV]
                    [--modem-baud BAUD] [--osmocon PATH] [--apdu-trace]
                    [--script PATH] [--csv FILE]
                    [--csv-column-key FIELD:AES_KEY_HEX]
                    [--card_handler FILE] [--noprompt] [--skip-card-init]
                    [-a PIN_ADM1 | -A PIN_ADM1_HEX] [-e EXECUTE_COMMAND]
                    [command] ...
```

Positional Arguments

command	A pySim-shell command that would optionally be executed at startup
command_args	Optional Arguments for command

Named Arguments

--apdu-trace	Trace the command/response APDUs exchanged with the card Default: False
-e, --execute-command	A pySim-shell command that will be executed at startup Default: []

Serial Reader

Use a simple/ultra-low-cost serial reader attached to a (physical or USB/virtual) RS232 port. This doesn't work with all RS232-attached smart card readers, only with the very primitive readers following the ancient *Phoenix* or *Smart Mouse* design.

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

PC/SC Reader

Use a PC/SC card reader to talk to the SIM card. PC/SC is a standard API for how applications access smart card readers, and is available on a variety of operating systems, such as Microsoft Windows, MacOS X and Linux. Most vendors of smart card readers provide drivers that offer a PC/SC interface, if not even a generic USB CCID driver is used. You can use a tool like `pcsc_scan -r` to obtain a list of readers available on your system.

--pcsc-shared	Open PC/SC reader in SHARED access (default: EXCLUSIVE) Default: False
-p, --pcsc-device	Number of PC/SC reader to use for SIM access
--pcsc-regex	Regex matching PC/SC reader to use for SIM access

AT Command Modem Reader

Talk to a SIM Card inside a mobile phone or cellular modem which is attached to this computer and offers an AT command interface including the AT+CSIM interface for Generic SIM access as specified in 3GPP TS 27.007.

--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

Use an OsmocomBB compatible phone to access the SIM inserted to the phone SIM slot. This will require you to run the OsmocomBB firmware inside the phone (can be ram-loaded). It also requires that you run the `osmocon` program, which provides a unix domain socket to which this reader driver can attach.

--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
--csv-column-key	per-CSV-column AES transport key Default: []
--card_handler	Use automatic card handling machine
--noprompt	Run in non interactive mode Default: False
--skip-card-init	Skip all card/profile initialization Default: False
-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)

1.1.3 Usage Examples

Guide: Enabling 5G SUCI

SUPI/SUCI Concealment is a feature of 5G-Standalone (SA) to encrypt the IMSI/SUPI with a network operator public key. 3GPP Specifies two different variants for this:

- SUCI calculation *in the UE*, using key data from the SIM
- SUCI calculation *on the card itself*

pySim supports writing the 5G-specific files for *SUCI calculation in the UE* on USIM cards, assuming that your cards contain the required files, and you have the privileges/credentials to write to them. This is the case using sysmocom sysmoISIM-SJA2 or any flavor of sysmoISIM-SJA5.

There is no 3GPP/ETSI standard method for configuring *SUCI calculation on the card*; pySim currently supports the vendor-specific method for the sysmoISIM-SJA5-S17).

This document describes both methods.

Technical References

This guide covers the basic workflow of provisioning SIM cards with the 5G SUCI feature. For detailed information on the SUCI feature and file contents, the following documents are helpful:

- USIM files and structure: [3GPP TS 31.102](#)
- USIM tests (incl. file content examples): [3GPP TS 31.121](#)
- Test keys for SUCI calculation: [3GPP TS 33.501](#)

For specific information on sysmocom SIM cards, refer to

- the [sysmoISIM-SJA5 User Manual](#) for the current sysmoISIM-SJA5 product
- the [sysmoISIM-SJA2 User Manual](#) for the older sysmoISIM-SJA2 product

Enabling 5G SUCI *calculated in the UE*

In short, you can enable *SUCI calculation in the UE* with these steps:

- activate **USIM Service 124**
- make sure **USIM Service 125** is disabled
- store the public keys in **EF.SUCI_Calc_Info**
- set the **Routing Indicator** (required)

If you want to disable the feature, you can just disable USIM Service 124 (and 125) in *EF.UST*.

Admin PIN

The usual way to authenticate yourself to the card as the cellular operator is to validate the so-called ADM1 (admin) PIN. This may differ from card model/vendor to card model/vendor.

Start pySIM-shell and enter the admin PIN for your card. If you bought the SIM card from your network operator and don't have the admin PIN, you cannot change SIM contents!

Launch pySIM:

```
$ ./pySim-shell.py -p 0

Using PC/SC reader interface
Autodetected card type: sysmoISIM-SJA2
Welcome to pySim-shell!
pySIM-shell (00:MF)>
```

Enter the ADM PIN:

```
pySIM-shell (00:MF)> verify_adm XXXXXXXX
```

Otherwise, write commands will fail with SW Mismatch: Expected 9000 and got 6982.

Key Provisioning

```
pySIM-shell (00:MF)> select MF
pySIM-shell (00:MF)> select ADF.USIM
pySIM-shell (00:MF/ADF.USIM)> select DF.5GS
pySIM-shell (00:MF/ADF.USIM/DF.5GS)> select EF.SUCI_Calc_Info
```

By default, the file is present but empty:

```
pySIM-shell (00:MF/ADF.USIM/DF.5GS/EF.SUCI_Calc_Info)> read_binary_decoded
missing Protection Scheme Identifier List data object tag
9000:
→ ffffffffffffffffffffffffffffffffffffffffffffffffffffffffffffffffffffffffffffffffffffffffffffffffffffffffffffffffff
→ -> {}
```

The following JSON config defines the testfile from 3GPP TS 31.121, Section 4.9.4 with test keys from 3GPP TS 33.501, Annex C.4. Highest priority (0) has a Profile-B (identifier: 2) key in key slot 1, which means the key with hnet_pubkey_identifier: 27.

```
{
  "prot_scheme_id_list": [
    {"priority": 0, "identifier": 2, "key_index": 1},
    {"priority": 1, "identifier": 1, "key_index": 2},
    {"priority": 2, "identifier": 0, "key_index": 0}],
  "hnet_pubkey_list": [
    {"hnet_pubkey_identifier": 27,
    "hnet_pubkey":
    ↪ "0472DA71976234CE833A6907425867B82E074D44EF907DFB4B3E21C1C2256EBCD15A7DED52FCBB097A4ED250E036C7B9C8C7",
    ↪ },
    {"hnet_pubkey_identifier": 30,
    "hnet_pubkey": "5A8D38864820197C3394B92613B20B91633CBD897119273BF8E4A6F4EEC0A650",
    ↪ }]]
}
```

Write the config to file (must be single-line input as for now):

```
pySIM-shell (00:MF/ADF.USIM/DF.5GS/EF.SUCI_Calc_Info)> update_binary_decoded '{ "prot_
↪ scheme_id_list": [ {"priority": 0, "identifier": 2, "key_index": 1}, {"priority": 1,
↪ "identifier": 1, "key_index": 2}, {"priority": 2, "identifier": 0, "key_index": 0}],
↪ "hnet_pubkey_list": [ {"hnet_pubkey_identifier": 27, "hnet_pubkey":
↪ "0472DA71976234CE833A6907425867B82E074D44EF907DFB4B3E21C1C2256EBCD15A7DED52FCBB097A4ED250E036C7B9C8C7",
↪ }, {"hnet_pubkey_identifier": 30, "hnet_pubkey":
↪ "5A8D38864820197C3394B92613B20B91633CBD897119273BF8E4A6F4EEC0A650"}]}'
```

WARNING: These are TEST KEYS with publicly known/specified private keys, and hence unsafe for live/secure deployments! For use in production networks, you need to generate your own set[s] of keys.

Routing Indicator

The Routing Indicator must be present for the SUCI feature. By default, the contents of the file is **invalid** (ffffff):

```
pySIM-shell (00:MF)> select MF
pySIM-shell (00:MF)> select ADF.USIM
pySIM-shell (00:MF/ADF.USIM)> select DF.5GS
pySIM-shell (00:MF/ADF.USIM/DF.5GS)> select EF.Routing_Indicator
pySIM-shell (00:MF/ADF.USIM/DF.5GS/EF.Routing_Indicator)> read_binary_decoded
9000: ffffffff -> {'raw': 'fffffff'}
```

The Routing Indicator is a four-byte file but the actual Routing Indicator goes into bytes 0 and 1 (the other bytes are reserved). To set the Routing Indicator to 0x71:

```
pySIM-shell (00:MF/ADF.USIM/DF.5GS/EF.Routing_Indicator)> update_binary 17ffffff
```

You can also set the routing indicator to **0x0**, which is *valid* and means “routing indicator not specified”, leaving it to the modem.

USIM Service Table

First, check out the USIM Service Table (UST):

```
pySIM-shell (00:MF)> select MF
pySIM-shell (00:MF)> select ADF.USIM
```

(continues on next page)

(continued from previous page)

```

pySIM-shell (00:MF/ADF.USIM)> select EF.UST
pySIM-shell (00:MF/ADF.USIM/EF.UST)> read_binary_decoded
9000: beff9f9de73e04084001707300000002e000000000 -> [2, 3, 4, 5, 6, 9, 10, 11, 12, 13, 14,
↳ 15, 17, 18, 19, 20, 21, 25, 27, 28, 29, 33, 34, 35, 38, 39, 42, 43, 44, 45, 46, 51, 60,
↳ 71, 73, 85, 86, 87, 89, 90, 93, 94, 95, 122, 123, 124, 126]

```

Table 1: From 3GPP TS 31.102

Service No.	Description
122	5GS Mobility Management Information
123	5G Security Parameters
124	Subscription identifier privacy support
125	SUCI calculation by the USIM
126	UAC Access Identities support
129	5GS Operator PLMN List

If you'd like to enable/disable any UST service:

```

pySIM-shell (00:MF/ADF.USIM/EF.UST)> ust_service_deactivate 124
pySIM-shell (00:MF/ADF.USIM/EF.UST)> ust_service_activate 124
pySIM-shell (00:MF/ADF.USIM/EF.UST)> ust_service_deactivate 125

```

In this case, UST Service 124 is already enabled and you're good to go. The sysmoISIM-SJA2 does not support on-SIM calculation, so service 125 must be disabled.

USIM Error with 5G and sysmoISIM

sysmoISIM-SJA2 come 5GS-enabled. By default however, the configuration stored in the card file-system is **not valid** for 5G networks: Service 124 is enabled, but EF.SUCI_Calc_Info and EF.Routing_Indicator are empty files (hence do not contain valid data).

At least for Qualcomm's X55 modem, this results in an USIM error and the whole modem shutting 5G down. If you don't need SUCI concealment but the smartphone refuses to connect to any 5G network, try to disable the UST service 124.

sysmoISIM-SJA5 are shipped with a more forgiving default, with valid EF.Routing_Indicator contents and disabled Service 124

SUCI calculation by the USIM

The SUCI calculation can also be performed by the USIM application on the UICC directly. The UE then uses the GET IDENTITY command (see also 3GPP TS 31.102, section 7.5) to retrieve a SUCI value.

The sysmoISIM-SJA5-S17 supports *SUCI calculation by the USIM*. The configuration is not much different to the above described configuration of *SUCI calculation in the UE*.

The main difference is how the key provisioning is done. When the SUCI calculation is done by the USIM, then the key material is not accessed by the UE. The specification (see also 3GPP TS 31.102, section 7.5.1.1), also does not specify any file or file format to store the key material. This means the exact way to perform the key provisioning is an implementation detail of the USIM card application.

In the case of sysmoISIM-SJA5-S17, the key material for *SUCI calculation by the USIM* is stored in *ADF.USIM/DF.SAIP/EF.SUCI_Calc_Info* (**not** in *ADF.USIM/DF.5GS/EF.SUCI_Calc_Info*!).

```
pySIM-shell (00:MF)> select MF
pySIM-shell (00:MF)> select ADF.USIM
pySIM-shell (00:MF/ADF.USIM)> select DF.SAIP
pySIM-shell (00:MF/ADF.USIM/DF.SAIP)> select EF.SUCI_Calc_Info
```

The file format is exactly the same as specified in 3GPP TS 31.102, section 4.4.11.8. This means the above described key provisioning procedure can be applied without any changes, except that the file location is different.

To signal to the UE that the USIM is setup up for SUCI calculation, service 125 must be enabled in addition to service 124 (see also 3GPP TS 31.102, section 5.3.48)

```
pySIM-shell (00:MF/ADF.USIM/EF.UST)> ust_service_activate 124
pySIM-shell (00:MF/ADF.USIM/EF.UST)> ust_service_activate 125
```

To verify that the SUCI calculation works as expected, it is possible to issue a GET IDENTITY command using pySim-shell:

```
select ADF.USIM
get_identity
```

The USIM should then return a SUCI TLV Data object that looks like this:

```
SUCI TLV Data Object:
0199f90717ff021b027a2c58ce1c6b89df088a9eb4d242596dd75746bb5f3503d2cf58a7461e4fd106e205c86f76544e9d732
```

1.1.4 Advanced Topics

Retrieving card-individual keys via CardKeyProvider

When working with a batch of cards, or more than one card in general, it is a lot of effort to manually retrieve the card-specific PIN (like ADM1) or key material (like SCP02/SCP03 keys).

To increase productivity in that regard, pySim has a concept called the *CardKeyProvider*. This is a generic mechanism by which different parts of the pySim[-shell] code can programmatically request card-specific key material from some data source (*provider*).

For example, when you want to verify the ADM1 PIN using the *verify_adm* command without providing an ADM1 value yourself, pySim-shell will request the ADM1 value for the ICCID of the card via the CardKeyProvider.

There can in theory be multiple different CardKeyProviders. You can for example develop your own CardKeyProvider that queries some kind of database for the key material, or that uses a key derivation function to derive card-specific key material from a global master key.

The only actual CardKeyProvider implementation included in pySim is the *CardKeyProviderCsv* which retrieves the key material from a [potentially encrypted] CSV file.

The CardKeyProviderCsv

The *CardKeyProviderCsv* allows you to retrieve card-individual key material from a CSV (comma separated value) file that is accessible to pySim.

The CSV file must have the expected column names, for example *ICCID* and *ADM1* in case you would like to use that CSV to obtain the card-specific ADM1 PIN when using the *verify_adm* command.

You can specify the CSV file to use via the *-csv* command-line option of pySim-shell. If you do not specify a CSV file, pySim will attempt to open a CSV file from the default location at *~/osmocomb/pySim/card_data.csv*, and use that, if it exists.

Column-Level CSV encryption

pySim supports column-level CSV encryption. This feature will make sure that your key material is not stored in plaintext in the CSV file.

The encryption mechanism uses AES in CBC mode. You can use any key length permitted by AES (128/192/256 bit).

Following GSMA FS.28, the encryption works on column level. This means different columns can be decrypted using different key material. This means that leakage of a column encryption key for one column or set of columns (like a specific security domain) does not compromise various other keys that might be stored in other columns.

You can specify column-level decryption keys using the `-csv-column-key` command line argument. The syntax is `FIELD:AES_KEY_HEX`, for example:

```
pySim-shell.py -csv-column-key SCP03_ENC_ISDR:000102030405060708090a0b0c0d0e0f
```

In order to avoid having to repeat the column key for each and every column of a group of keys within a keyset, there are pre-defined column group aliases, which will make sure that the specified key will be used by all columns of the set:

- `UICC_SCP02` is a group alias for `UICC_SCP02_KIC1`, `UICC_SCP02_KID1`, `UICC_SCP02_KIK1`
- `UICC_SCP03` is a group alias for `UICC_SCP03_KIC1`, `UICC_SCP03_KID1`, `UICC_SCP03_KIK1`
- `SCP03_ECASD` is a group alias for `SCP03_ENC_ECASD`, `SCP03_MAC_ECASD`, `SCP03_DEK_ECASD`
- `SCP03_ISDA` is a group alias for `SCP03_ENC_ISDA`, `SCP03_MAC_ISDA`, `SCP03_DEK_ISDA`
- `SCP03_ISDR` is a group alias for `SCP03_ENC_ISDR`, `SCP03_MAC_ISDR`, `SCP03_DEK_ISDR`

Field naming

- For look-up of UICC/SIM/USIM/ISIM or eSIM profile specific key material, pySim uses the `ICCID` field as lookup key.
- For look-up of eUICC specific key material (like SCP03 keys for the ISD-R, ECASD), pySim uses the `EID` field as lookup key.

As soon as the `CardKeyProviderCsv` finds a line (row) in your CSV where the ICCID or EID match, it looks for the column containing the requested data.

ADM PIN

The `verify_adm` command will attempt to look up the `ADM1` column indexed by the ICCID of the SIM/UICC.

SCP02 / SCP03

SCP02 and SCP03 each use key triplets consisting of ENC, MAC and DEK keys. For more details, see the applicable GlobalPlatform specifications.

If you do not want to manually enter the key material for each specific card as arguments to the `establish_scp02` or `establish_scp03` commands, you can make use of the `-key-provider-suffix` option. pySim uses this suffix to compose the column names for the `CardKeyProvider` as follows.

- `SCP02_ENC_` + suffix for the SCP02 ciphering key
- `SCP02_MAC_` + suffix for the SCP02 MAC key
- `SCP02_DEK_` + suffix for the SCP02 DEK key
- `SCP03_ENC_` + suffix for the SCP03 ciphering key
- `SCP03_MAC_` + suffix for the SCP03 MAC key

- `SCP03_DEK_` + suffix for the SCP03 DEK key

So for example, if you are using a command like `establish_scp03 -key-provider-suffix ISDR`, then the column names for the key material look-up are `SCP03_ENC_ISDR`, `SCP03_MAC_ISDR` and `SCP03_DEK_ISDR`, respectively.

The identifier used for look-up is determined by the definition of the Security Domain. For example, the eUICC ISD-R and ECASD will use the EID of the eUICC. On the other hand, the ISD-P of an eSIM or the ISD of an UICC will use the ICCID.

Remote access to an UICC/eUICC

To access a card with pySim-shell, it is not strictly necessary to have physical access to it. There are solutions that allow remote access to UICC/eUICC cards. In this section we will give a brief overview.

osmo-remsim

osmo-remsim is a suite of software programs enabling physical/geographic separation of a cellular phone (or modem) on the one hand side and the UICC/eUICC card on the other side.

Using osmo-remsim, you can operate an entire fleet of modems/phones, as well as banks of SIM cards and dynamically establish or remove the connections between modems/phones and cards.

To access remote cards with pySim-shell via osmo-remsim (RSPRO), the provided `libfd_remsim_client` would be used to provide a virtual PC/SC reader on the local machine. pySim-shell can then access this reader like any other PC/SC reader.

More information on osmo-remsim can be found under:

- <https://osmocom.org/projects/osmo-remsim/wiki>
- <https://ftp.osmocom.org/docs/osmo-remsim/master/osmo-remsim-usermanual.pdf>

Android APDU proxy

Android APDU proxy is an Android app that provides a bridge between a host computer and the UICC/eUICC slot of an Android smartphone.

The APDU proxy connects to VPCD server that runs on the remote host (in this case the local machine where pySim-shell is running). The VPCD server then provides a virtual PC/SC reader, that pySim-shell can access like any other PC/SC reader.

On the Android side the UICC/eUICC is accessed via OMAPI (Open Mobile API), which is available in Android since API level Android 8 (API level 29).

More information Android APDU proxy can be found under:

- <https://gitea.osmocom.org/sim-card/android-apdu-proxy>

1.1.5 cmd2 basics

As pySim-shell is built upon cmd2, some generic cmd2 commands/features are available. You may want to check out the [cmd2 Builtin commands](#) to learn about those.

1.1.6 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

```
usage: build.py [-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

Example:

```
pySIM-shell (00: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

```
usage: build.py [-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.

Example:

```
pySIM-shell (00:MF)> export --json > /tmp/export.json
EXCEPTION of type 'RuntimeError' occurred with message: 'unable to export 50 elementary_
↪file(s) and 2 dedicated file(s), also had to stop early due to exception:6e00: ARA-M -_'
```

(continues on next page)

(continued from previous page)

```

↪Invalid class'
To enable full traceback, run the following command: 'set debug true'
pySIM-shell (00:MF)>

```

The exception above is more or less expected. It just means that 50 files which are defined (most likely as optional files in some later 3GPP release) were not found on the card, or were invalidated/disabled when trying to SELECT them.

fsdump

```
usage: build.py [-h] [--filename FILENAME] [--json]
```

Named Arguments

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

Please note that *fsdump* works relative to the current working directory, so if you are in *MF*, then the dump 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 dump.

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.

One use case for this is to systematically analyze the differences between the contents of two cards. To do this, you can create fsdumps of the two cards, and then use some general-purpose JSON diffing tool like *jycm -show* (see <https://github.com/eggachecat/jycm>).

Example:

```

pySIM-shell (00:MF)> fsdump > /tmp/fsdump.json
pySIM-shell (00:MF)>

```

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 (00:MF)> tree
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

```
usage: build.py [-h] [--pin-is-hex]
               [--adm-type {ADM1,ADM2,ADM3,ADM4,ADM5,ADM6,ADM7,ADM8,ADM9,ADM10}]
               [ADM]
```

Positional Arguments

ADM ADM pin value. If none given, CSV file will be queried

Named Arguments

--pin-is-hex ADM pin value is specified as hex-string (not decimal)
Default: False

--adm-type Possible choices: ADM1, ADM2, ADM3, ADM4, ADM5, ADM6, ADM7, ADM8, ADM9, ADM10
Override ADM number. Default is card-model-specific, usually 1

Example (successful):

```
pySIM-shell (00:MF)> verify_adm 11111111
pySIM-shell (00:MF)>
```

In the above case, the ADM was successfully verified. Please make always sure to use the correct ADM1 for the specific card you have inserted! If you present a wrong ADM1 value several times consecutively, your card ADM1 will likely be permanently locked, meaning you will never be able to reach ADM1 privilege level. For sysmoUSIM/ISIM products, three consecutive wrong ADM1 values will lock the ADM1.

Example (erroneous):

```
pySIM-shell (00:MF)> verify_adm 1
EXCEPTION of type 'RuntimeError' occurred with message: 'Failed to verify chv_no 0x0A_
↳with code 0x31FFFFFFFFFFFFFF, 2 tries left.'
To enable full traceback, run the following command: 'set debug true'
```

If you frequently work with the same set of cards that you need to modify using their ADM1, you can put a CSV file with those cards ICCID + ADM1 values into a CSV (comma separated value) file at `~/osmocombuild/pysim/card_data.csv`. In this case, you can use the `verify_adm` command *without specifying an ADM1 value*.

Example (successful):

```
pySIM-shell (00:MF)> verify_adm
found ADM-PIN '11111111' for ICCID '8988211900000000512'
pySIM-shell (00:MF)>
```

In this case, the CSV file contained a record for the ICCID of the card (11111111) and that value was used to successfully verify ADM1.

Example (erroneous):

```
pySIM-shell (00:MF)> verify_adm
EXCEPTION of type 'ValueError' occurred with message: 'cannot find ADM-PIN for ICCID
↳'8988211900000000512''
To enable full traceback, run the following command: 'set debug true'
```

In this case there was no record for the ICCID of the card in the CSV file.

reset

Perform card reset and display the card ATR.

Example:

```
pySIM-shell (00:MF)> reset
Card ATR: 3b9f96801f878031e073fe211b674a357530350259c4
pySIM-shell (00:MF)> reset
```

intro

[Re-]Display the introductory banner

Example:

```
pySIM-shell (00:MF)> intro
Welcome to pySim-shell!
(C) 2021-2023 by Harald Welte, sysmocom - s.f.m.c. GmbH and contributors
Online manual available at https://downloads.osmocom.org/docs/pysim/master/html/shell.html
↪html
```

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

```
usage: build.py [-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

```
usage: build.py [-h] STRING [STRING ...]
```

Positional Arguments

STRING string to echo on the shell

apdu

```
usage: build.py [-h] [--expect-sw EXPECT_SW]
               [--expect-response-regex EXPECT_RESPONSE_REGEX] [--raw]
               APDU
```

Positional Arguments

APDU APDU as hex string

Named Arguments

--expect-sw expect a specified status word

--expect-response-regex match response against regex

--raw Bypass the logical channel (and secure channel)

 Default: False

Example:

```
pySIM-shell (00:MF)> apdu 00a40400023f00
SW: 6700
```

In the above case the raw APDU hex-string `00a40400023f00` was sent to the card, to which it responded with status word `6700`. Keep in mind that pySim-shell has no idea what kind of raw commands you are sending to the card, and it hence is unable to synchronize its internal state (such as the currently selected file) with the card. The use of this command should hence be constrained to commands that do not have any high-level support in pySim-shell yet.

1.1.7 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 (00:MF)> select
..                2fe2                a00000000871004      EF.ARR           MF
2f00              3f00                ADF.ISIM          EF.DIR
2f05              7f10                ADF.USIM          EF.ICCID
2f06              7f20                DF.GSM            EF.PL
2f08              a00000000871002      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 (00:MF)> select ADF.USIM
{
  "file_descriptor": {
    "file_descriptor_byte": {
      "shareable": true,
      "file_type": "df",
      "structure": "no_info_given"
    }
  },
  "df_name": "A00000000871002FFFFFFFFF8907090000",
  "proprietary_info": {
    "uicc_characteristics": "71",
    "available_memory": 101640
  },
  "life_cycle_status_int": "operational_activated",
  "security_attrib_compact": "00",
  "pin_status_template_do": {
    "ps_do": "70",
    "key_reference": 11
  }
}

pySIM-shell (00:MF/ADF.USIM)>
```

status

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

Example:

```
pySIM-shell (00: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": "a00000000871004fffffffff8907090000",
"proprietary_information": {
  "uicc_characteristics": "71",
  "available_memory": 101640
}
```

(continues on next page)

(continued from previous page)

```

    },
    "life_cycle_status_integer": "operational_activated",
    "security_attrib_compact": "00",
    "pin_status_template_do": {
        "ps_do": "70",
        "key_reference": 11
    }
}

```

change_chv

```
usage: build.py [-h] [--pin-nr PIN_NR] [NEWPIN] [PIN]
```

Positional Arguments

NEWPIN	PIN code value. If none given, CSV file will be queried
PIN	PIN code value. If none given, CSV file will be queried

Named Arguments

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

disable_chv

```
usage: build.py [-h] [--pin-nr PIN_NR] [PIN]
```

Positional Arguments

PIN	PIN code value. If none given, CSV file will be queried
------------	---

Named Arguments

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

enable_chv

```
usage: build.py [-h] [--pin-nr PIN_NR] [PIN]
```

Positional Arguments

PIN	PIN code value. If none given, CSV file will be queried
------------	---

Named Arguments

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

unlock_chv

```
usage: build.py [-h] [--pin-nr PIN_NR] [PUK] [NEWPIN]
```

Positional Arguments

PUK	PUK code value. If none given, CSV file will be queried
NEWPIN	PIN code value. If none given, CSV file will be queried

Named Arguments

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

verify_chv

```
usage: build.py [-h] [--pin-nr PIN_NR] [PIN]
```

Positional Arguments

PIN	PIN code value. If none given, CSV file will be queried
------------	---

Named Arguments

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

deactivate_file

Deactivate the currently selected file. A deactivated file can no longer be accessed for any further operation (such as selecting and subsequently reading or writing).

Any access to a file that is deactivated will trigger the error *SW 6283 'Selected file invalidated/disabled'*

In order to re-access a deactivated file, you need to activate it again, see the *activate_file* command below. Note that for *deactivation* the to-be-deactivated EF must be selected, but for *activation*, the DF above the to-be-activated EF must be selected!

This command sends a DEACTIVATE FILE APDU to the card (used to be called INVALIDATE in TS 11.11 for classic SIM).

activate_file

```
usage: build.py [-h] NAME
```

Positional Arguments

NAME	File name or FID of file to activate
-------------	--------------------------------------

open_channel

```
usage: build.py [-h] {1,2,3,4,5,6,7,8,9,10,11,12,13,14,15}
```

Positional Arguments

chan_nr Possible choices: 1, 2, 3, 4, 5, 6, 7, 8, 9, 10, 11, 12, 13, 14, 15
 Channel Number
 Default: 1

close_channel

```
usage: build.py [-h] {1,2,3,4,5,6,7,8,9,10,11,12,13,14,15}
```

Positional Arguments

chan_nr Possible choices: 1, 2, 3, 4, 5, 6, 7, 8, 9, 10, 11, 12, 13, 14, 15
 Channel Number
 Default: 1

switch_channel

```
usage: build.py [-h] {0,1,2,3,4,5,6,7,8,9,10,11,12,13,14,15}
```

Positional Arguments

chan_nr Possible choices: 0, 1, 2, 3, 4, 5, 6, 7, 8, 9, 10, 11, 12, 13, 14, 15
 Channel Number
 Default: 0

1.1.8 TS 102 221 commands

These are commands as specified in ETSI TS 102 221, the core UICC specification.

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.

```
usage: build.py [-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

resume_uicc

This command allows you to perform the SUSPEND UICC command for the RESUME operation on the card.

Suspend/Resume 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 (RESUME) command and doesn't perform the full related sequence including the electrical power down.

```
usage: build.py [-h] TOKEN
```

Positional Arguments

TOKEN Token provided during SUSPEND

terminal_capability

This command allows you to perform the TERMINAL CAPABILITY command towards the card.

TS 102 221 specifies the TERMINAL CAPABILITY command using which the terminal (Software + hardware talking to the card) can expose their capabilities. This is also used in the eUICC universe to let the eUICC know which features are supported.

```
usage: build.py [-h] [--used-supply-voltage-class {a,b,c,d,e}]
               [--maximum-available-power-supply MAXIMUM_AVAILABLE_POWER_SUPPLY]
               [--actual-used-freq-100k ACTUAL_USED_FREQ_100K]
               [--extended-logical-channel] [--uicc-clf] [--lui-d] [--lpd-d]
               [--lds-d] [--lui-e-scws] [--metadata-update-alerting]
               [--enterprise-capable-device] [--lui-e-e4e] [--lpr]
```

Terminal Power Supply

--used-supply-voltage-class Possible choices: a, b, c, d, e

Actual used Supply voltage class

--maximum-available-power-supply Maximum available power supply of the terminal

--actual-used-freq-100k Actual used clock frequency (in units of 100kHz)

Extended logical channels terminal support

--extended-logical-channel Extended Logical Channel supported

Default: False

Additional interfaces support

--uicc-clf Local User Interface in the Device (LUId) supported
Default: False

Additional Terminal capability indications related to eUICC

--lui-d Local User Interface in the Device (LUId) supported
Default: False

--lpd-d Local Profile Download in the Device (LPDd) supported
Default: False

--lds-d Local Discovery Service in the Device (LPDd) supported
Default: False

--lui-e-scws LUIe based on SCWS supported
Default: False

--metadata-update-alerting Metadata update alerting supported
Default: False

--enterprise-capable-device Enterprise Capable Device
Default: False

--lui-e-e4e LUIe using E4E (ENVELOPE tag E4) supported
Default: False

--lpr LPR (LPA Proxy) supported
Default: False

1.1.9 Linear Fixed EF commands

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

read_record

```
usage: build.py [-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

```
usage: build.py [-h] [--online] RECORD_NR
```

Positional Arguments

RECORD_NR	Number of record to be read
------------------	-----------------------------

Named Arguments

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

If this command fails, it means that the record is not decodable, and you should use the *read_record* command and proceed with manual decoding of the contents.

read_records

```
usage: build.py [-h]
```

read_records_decoded

```
usage: build.py [-h] [--online]
```

Named Arguments

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

If this command fails, it means that the record[s] are not decodable, and you should use the *read_records* command and proceed with manual decoding of the contents.

update_record

```
usage: build.py [-h] RECORD_NR DATA
```

Positional Arguments

RECORD_NR	Number of record to be read
DATA	Data bytes (hex format) to write

update_record_decoded

```
usage: build.py [-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

If this command fails, it means that the record is not encodable; please check your input and/or use the raw *update_record* command.

edit_record_decoded

```
usage: build.py [-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.

If this command fails before the editor is spawned, it means that the current record contents is not decodable, and you should use the *update_record_decoded* or *update_record* command.

If this command fails after making your modifications in the editor, it means that the new file contents is not encodable; please check your input and/or use the raw *update_record* command.

decode_hex

```
usage: build.py [-h] [--online] HEXSTR
```

Positional Arguments

HEXSTR Hex-string of encoded data to decode

Named Arguments

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

1.1.10 Transparent EF commands

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

read_binary

```
usage: build.py [-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

```
usage: build.py [-h] [--online]
```

Named Arguments

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

If this command fails, it means that the file is not decodable, and you should use the [read_binary](#) command and proceed with manual decoding of the contents.

update_binary

```
usage: build.py [-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

```
usage: build.py [-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 ciphering indicator field within EF.AD:

```

pySIM-shell (00:MF/ADF.USIM/EF.AD)> read_binary_decoded

{
  "ms_operation_mode": "normal_and_specific_facilities",
  "additional_info": {
    "ciphering_indicator": false,
    "csg_display_control": false,
    "prose_services": false,
    "extended_drx": true
  },
  "rfu": 0,
  "mnc_len": 2,
  "extensions": "ff"
}
pySIM-shell (00:MF/ADF.USIM/EF.AD)> update_binary_decoded --json-path additional_info.
↪ciphering_indicator true
"01000902ff"
pySIM-shell (00:MF/ADF.USIM/EF.AD)> read_binary_decoded
{
  "ms_operation_mode": "normal_and_specific_facilities",
  "additional_info": {
    "ciphering_indicator": true,
    "csg_display_control": false,
    "prose_services": false,
    "extended_drx": true
  },
  "rfu": 0,
  "mnc_len": 2,
  "extensions": "ff"
}

```

If this command fails, it means that the file is not encodable; please check your input and/or use the raw *update_binary* command.

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.

If this command fails before the editor is spawned, it means that the current file contents is not decodable, and you should use the *update_binary_decoded* or *update_binary* command.

If this command fails after making your modifications in the editor, it means that the new file contents is not encodable; please check your input and/or use the raw *update_binary* command.

decode_hex

```
usage: build.py [-h] [--online] HEXSTR
```

Positional Arguments

HEXSTR	Hex-string of encoded data to decode
---------------	--------------------------------------

Named Arguments

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

1.1.11 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

```
usage: build.py [-h] TAG
```

Positional Arguments

TAG	BER-TLV Tag of value to retrieve
------------	----------------------------------

set_data

```
usage: build.py [-h] TAG data
```

Positional Arguments

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

del_data

```
usage: build.py [-h] TAG
```

Positional Arguments

TAG	BER-TLV Tag of value to set
------------	-----------------------------

1.1.12 USIM commands

These commands are available only while ADF.USIM (or ADF.ISIM, respectively) is selected.

authenticate

```
usage: build.py [-h] RAND AUTN
```

Positional Arguments

RAND	Random challenge
AUTN	Authentication Nonce

terminal_profile

```
usage: build.py [-h] PROFILE
```

Positional Arguments

PROFILE	Hexstring of encoded terminal profile
----------------	---------------------------------------

envelope

```
usage: build.py [-h] PAYLOAD
```

Positional Arguments

PAYLOAD	Hexstring of encoded payload to ENVELOPE
----------------	--

envelope_sms

```
usage: build.py [-h] TPDU
```

Positional Arguments

TPDU	Hexstring of encoded SMS TPDU
-------------	-------------------------------

get_identity

```
usage: build.py [-h] [--nsw-context]
```

Named Arguments

--nsw-context	use SUCI 5G Non-Seamless WLAN Offload context
	Default: False

1.1.13 File-specific commands

These commands are valid only if the respective file is currently selected. They perform some operation that's specific to this file only.

EF.ARR: read_arr_record

Read one EF.ARR record in flattened, human-friendly form.

EF.ARR: read_arr_records

Read + decode all EF.ARR records in flattened, human-friendly form.

DF.GSM/EF.SST: sst_service_allocate

Mark a given single service as allocated in EF.SST. Requires service number as argument.

DF.GSM/EF.SST: sst_service_activate

Mark a given single service as activated in EF.SST. Requires service number as argument.

DF.GSM/EF.SST: sst_service_deallocate

Mark a given single service as deallocated in EF.SST. Requires service number as argument.

DF.GSM/EF.SST: sst_service_deactivate

Mark a given single service as deactivated in EF.SST. Requires service number as argument.

ADF.USIM/EF.EST: est_service_enable

Enables a single service in EF.EST. Requires service number as argument.

ADF.USIM/EF.EST: est_service_disable

Disables a single service in EF.EST. Requires service number as argument.

EF.IMSI: update_imsi_plmn

Change the PLMN part (MCC+MNC) of the IMSI. Requires a single argument consisting of 5/6 digits of concatenated MCC+MNC.

ADF.USIM/EF.UST: ust_service_activate

Activates a single service in EF.UST. Requires service number as argument.

ADF.USIM/EF.UST: ust_service_deactivate

Deactivates a single service in EF.UST. Requires service number as argument.

ADF.USIM/EF.UST: ust_service_check

Check consistency between services of this file and files present/activated. Many services determine if one or multiple files shall be present/activated or if they shall be absent/deactivated. This performs a consistency check to ensure that no services are activated for files that are not - and vice-versa, no files are activated for services that are not. Error messages are printed for every inconsistency found.

ADF.ISIM/EF.IST: ist_service_activate

Activates a single service in EF.IST. Requires service number as argument.

ADF.ISIM/EF.IST: ist_service_deactivate

Deactivates a single service in EF.UST. Requires service number as argument.

ADF.ISIM/EF.IST: ist_service_check

Check consistency between services of this file and files present/activated. Many services determine if one or multiple files shall be present/activated or if they shall be absent/deactivated. This performs a consistency check to ensure that no services are activated for files that are not - and vice-versa, no files are activated for services that are not. Error messages are printed for every inconsistency found.

1.1.14 UICC Administrative commands

ETSI TS 102 222 specifies a set of *Administrative Commands*, which can be used by the card issuer / operator to modify the file system structure (delete files, create files) or even to terminate individual files or the entire card.

pySim-shell supports those commands, but **use extreme caution**. Unless you know exactly what you're doing, it's very easy to render your card unusable. You've been warned!

delete_file

```
usage: build.py [-h] [--force-delete] NAME
```

Positional Arguments

NAME	File name or FID to delete
-------------	----------------------------

Named Arguments

--force-delete	I really want to permanently delete the file. I know pySim cannot re-create it yet!
	Default: False

terminate_df

```
usage: build.py [-h] [--force] NAME
```

Positional Arguments

NAME	File name or FID
-------------	------------------

Named Arguments

--force	I really want to terminate the file. I know I can not recover from it!
	Default: False

terminate_ef

```
usage: build.py [-h] [--force] NAME
```

Positional Arguments

NAME	File name or FID
------	------------------

Named Arguments

--force	I really want to terminate the file. I know I can not recover from it!
	Default: False

terminate_card

```
usage: build.py [-h] [--force-terminate-card]
```

Named Arguments

--force-terminate-card	I really want to permanently terminate the card. It will not be usable afterwards!
	Default: False

create_ef

```
usage: build.py [-h] --ef-arr-file-id EF_ARR_FILE_ID --ef-arr-record-nr
EF_ARR_RECORD_NR --file-size FILE_SIZE --structure
{transparent,linear_fixed,ber_tlv}
[--short-file-id SHORT_FILE_ID] [--shareable]
[--record-length RECORD_LENGTH]
FILE_ID
```

Positional Arguments

FILE_ID	File Identifier as 4-character hex string
---------	---

required arguments

--ef-arr-file-id	Referenced Security: File Identifier of EF.ARR
--ef-arr-record-nr	Referenced Security: Record Number within EF.ARR
--file-size	Size of file in octets
--structure	Possible choices: transparent, linear_fixed, ber_tlv Structure of the to-be-created EF

optional arguments

--short-file-id	Short File Identifier as 2-digit hex string
--shareable	Should the file be shareable? Default: False
--record-length	Length of each record in octets

create_df

```
usage: build.py [-h] --ef-arr-file-id EF_ARR_FILE_ID --ef-arr-record-nr
                EF_ARR_RECORD_NR [--shareable] [--aid AID]
                [--total-file-size TOTAL_FILE_SIZE] [--permit-rfm-create]
                [--permit-rfm-delete-terminate] [--permit-other-applet-create]
                [--permit-other-applet-delete-terminate]
                FILE_ID
```

Positional Arguments

FILE_ID File Identifier as 4-character hex string

required arguments

--ef-arr-file-id Referenced Security: File Identifier of EF.ARR
--ef-arr-record-nr Referenced Security: Record Number within EF.ARR

optional arguments

--shareable Should the file be shareable?
 Default: False
--aid Application ID (creates an ADF, instead of a DF)
--total-file-size Physical memory allocated for DF/ADi in octets

sysmoSIM-SJA optional arguments

--permit-rfm-create Default: False
--permit-rfm-delete-terminate Default: False
--permit-other-applet-create Default: False
--permit-other-applet-delete-terminate Default: False

resize_ef

```
usage: build.py [-h] --file-size FILE_SIZE NAME
```

Positional Arguments

NAME Name or FID of file to be resized

required arguments

--file-size Size of file in octets

1.1.15 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

```
pySIM-shell (00:MF/ADF.ARA-M)> aram_get_all

[
  {
    "response_all_ref_ar_do": [
      {
        "ref_ar_do": [
          {
            "ref_do": [
              {
                "aid_ref_do": "ffffffffffff"
              },
              {
                "dev_app_id_ref_do":
↪ "e46872f28b350b7e1f140de535c2a8d5804f0be3"
              }
            ]
          },
          {
            "ar_do": [
              {
                "apdu_ar_do": {
                  "generic_access_rule": "always"
                }
              },
              {
                "perm_ar_do": {
                  "permissions": "000000000000000001"
                }
              }
            ]
          }
        ]
      }
    ]
  }
]
```

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

```
usage: build.py [-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 or 0 hex bytes)
--aid-empty	No specific SE application, applies to implicitly selected application (all channels) 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: multiple groups of 8 hex bytes (4 byte CLA/INS/P1/P2 followed by 4 byte mask)
--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 (00:MF/ADF.ARA-M)> aram_store_ref_ar_do --aid FFFFFFFFFF --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*

1.1.16 GlobalPlatform commands

pySim-shell has only the most rudimentary support for GlobalPlatform at this point. Please use dedicated projects like GlobalPlatformPro meanwhile.

get_data

```
usage: build.py [-h] data_object_name
```

Positional Arguments

data_object_name Name of the data object to be retrieved from the card

get_status

```
usage: build.py [-h] [--aid AID] {isd,applications,files,files_and_modules}
```

Positional Arguments

subset Possible choices: isd, applications, files, files_and_modules
Subset of statuses to be included in the response

Named Arguments

--aid AID Search Qualifier (search only for given AID)
Default: ----

set_status

```
usage: build.py [-h] [--aid AID]
           {isd,app_or_ssd,isd_and_assoc_apps}
           {loaded,installed,selectable,personalized,locked}
```

Positional Arguments

scope Possible choices: isd, app_or_ssd, isd_and_assoc_apps
Defines the scope of the requested status change

status Possible choices: loaded, installed, selectable, personalized, locked
Specify the new intended status

Named Arguments

--aid AID of the target Application or Security Domain

store_data

```
usage: build.py [-h] [--data-structure {none,dgi,ber_tlv,rfu}]
           [--encryption {none,application_dependent,rfu,encrypted}]
           [--response {not_expected,may_be_returned}]
           DATA
```

Positional Arguments

DATA

Named Arguments

--data-structure	Possible choices: none, dgi, ber_tlv, rfu Default: 'none'
--encryption	Possible choices: none, application_dependent, rfu, encrypted Default: 'none'
--response	Possible choices: not_expected, may_be_returned Default: 'not_expected'

put_key

```
usage: build.py [-h] [--old-key-version-nr OLD_KEY_VERSION_NR]
               --key-version-nr KEY_VERSION_NR --key-id KEY_ID --key-type
               {des,tls_psk,aes,hmac_sha1,hmac_sha1_160,rsa_public_exponent_e_cleartext,
               ↪rsa_modulus_n_cleartext,rsa_modulus_n,rsa_private_exponent_d,rsa_chines_remainder_p,
               ↪rsa_chines_remainder_q,rsa_chines_remainder_pq,rsa_chines_remainder_dpi,rsa_chines_
               ↪remainder_dqi,ecc_public_key,ecc_private_key,ecc_field_parameter_p,ecc_field_parameter_
               ↪a,ecc_field_parameter_b,ecc_field_parameter_g,ecc_field_parameter_n,ecc_field_
               ↪parameter_k,ecc_key_parameters_reference,not_available}
               --key-data KEY_DATA [--key-check KEY_CHECK]
               [--suppress-key-check]
```

Named Arguments

--old-key-version-nr	Old Key Version Number Default: 0
--key-version-nr	Key Version Number
--key-id	Key Identifier (base)
--key-type	Possible choices: des, tls_psk, aes, hmac_sha1, hmac_sha1_160, rsa_public_exponent_e_cleartext, rsa_modulus_n_cleartext, rsa_modulus_n, rsa_private_exponent_d, rsa_chines_remainder_p, rsa_chines_remainder_q, rsa_chines_remainder_pq, rsa_chines_remainder_dpi, rsa_chines_remainder_dqi, ecc_public_key, ecc_private_key, ecc_field_parameter_p, ecc_field_parameter_a, ecc_field_parameter_b, ecc_field_parameter_g, ecc_field_parameter_n, ecc_field_parameter_k, ecc_key_parameters_reference, not_available Key Type
--key-data	Key Data Block
--key-check	Key Check Value
--suppress-key-check	Suppress generation of Key Check Values Default: False

delete_key

```
usage: build.py [-h] [--key-id KEY_ID] [--key-ver KEY_VER]
               [--delete-related-objects]
```

Named Arguments

--key-id	Key Identifier (KID)
--key-ver	Key Version Number (KVN)
--delete-related-objects	Delete not only the object but also its related objects Default: False

install_for_personalization

```
usage: build.py [-h] application_aid
```

Positional Arguments

application_aid	Application AID
------------------------	-----------------

install_for_install

```
usage: build.py [-h] [--load-file-aid LOAD_FILE_AID] [--module-aid MODULE_AID]
               --application-aid APPLICATION_AID
               [--install-parameters INSTALL_PARAMETERS]
               [--privilege {security_domain,dap_verification,delegated_management,card_
↳lock,card_terminate,card_reset,cvm_management,mandated_dap_verification,trusted_path,
↳authorized_management,token_management,global_delete,global_lock,global_registry,final_
↳application,global_service,receipt_generation,ciphored_load_file_data_block,
↳contactless_activation,contactless_self_activation}]
               [--install-token INSTALL_TOKEN] [--make-selectable]
```

Named Arguments

--load-file-aid	Executable Load File AID Default: ****
--module-aid	Executable Module AID Default: ****
--application-aid	Application AID
--install-parameters	Install Parameters (GPC_SPE_034, section 11.5.2.3.7, table 11-49) Default: ****
--privilege	Possible choices: security_domain, dap_verification, delegated_management, card_lock, card_terminate, card_reset, cvm_management, mandated_dap_verification, trusted_path, authorized_management, token_management, global_delete, global_lock, global_registry, final_application, global_service, receipt_generation, ciphored_load_file_data_block, contactless_activation, contactless_self_activation

	Privilege granted to newly installed Application
	Default: []
--install-token	Install Token (Section GPCS C.4.2/C.4.7)
	Default: ****
--make-selectable	Install and make selectable
	Default: False

delete_card_content

```
usage: build.py [-h] [--delete-related-objects] aid
```

Positional Arguments

aid	Executable Load File or Application AID
------------	---

Named Arguments

--delete-related-objects	Delete not only the object but also its related objects
	Default: False

establish_scp02

```
usage: build.py [-h] --key-ver KEY_VER [--host-challenge HOST_CHALLENGE]
               [--security-level SECURITY_LEVEL] [--key-enc KEY_ENC]
               [--key-mac KEY_MAC] [--key-dek KEY_DEK]
               [--key-provider-suffix KEY_PROVIDER_SUFFIX]
```

Named Arguments

--key-ver	Key Version Number (KVN)
--host-challenge	Hard-code the host challenge; default: random
--security-level	Security Level. Default: 0x01 (C-MAC only)
	Default: 1

Manual key specification

--key-enc	Secure Channel Encryption Key
--key-mac	Secure Channel MAC Key
--key-dek	Data Encryption Key

Obtain keys from CardKeyProvider (e.g. CSV

--key-provider-suffix	Suffix for key names in CardKeyProvider
------------------------------	---

establish_scp03

```
usage: build.py [-h] --key-ver KEY_VER [--host-challenge HOST_CHALLENGE]
               [--security-level SECURITY_LEVEL] [--key-enc KEY_ENC]
               [--key-mac KEY_MAC] [--key-dek KEY_DEK]
               [--key-provider-suffix KEY_PROVIDER_SUFFIX] [--s16-mode]
```

Named Arguments

--key-ver	Key Version Number (KVN)
--host-challenge	Hard-code the host challenge; default: random
--security-level	Security Level. Default: 0x01 (C-MAC only) Default: 1
--s16-mode	S16 mode (S8 is default) Default: False

Manual key specification

--key-enc	Secure Channel Encryption Key
--key-mac	Secure Channel MAC Key
--key-dek	Data Encryption Key

Obtain keys from CardKeyProvider (e.g. CSV

--key-provider-suffix Suffix for key names in CardKeyProvider

release_scp

Release any previously established SCP (Secure Channel Protocol)

1.1.17 eUICC ISD-R commands

These commands are to perform a variety of operations against eUICC for GSMA consumer eSIM. They implement the so-called ES10a, ES10b and ES10c interface. Basically they perform the tasks that usually would be done by the LPA in the UE.

In order to use those commands, you need to go through the specified steps as documented in GSMA SGP.22:

- open a new logical channel (and start to use it)
- select the ISD-R application

Example:

```
pySIM-shell (00:MF)> open_channel 2
pySIM-shell (00:MF)> switch_channel 2
pySIM-shell (02:MF)> select ADF.ISD-R
{
  "application_id": "a0000005591010ffffffff8900000100",
  "proprietary_data": {
    "maximum_length_of_data_field_in_command_message": 255
  },
}
```

(continues on next page)

(continued from previous page)

```

    "isdr_proprietary_application_template": {
        "supported_version_number": "020200"
    }
}
pySIM-shell (02:ADF.ISD-R)>

```

Once you are at this stage, you can issue the various eUICC related commands against the ISD-R application

es10x_store_data

```
usage: build.py [-h] TX_DO
```

Positional Arguments

TX_DO Hexstring of encoded to-be-transmitted DO

get_euicc_configured_addresses

Obtain the configured SM-DP+ and/or SM-DS addresses using the ES10a GetEuiccConfiguredAddresses() function.

Example:

```

pySIM-shell (00:MF/ADF.ISD-R)> get_euicc_configured_addresses
{
    "root_ds_address": "testrootsmds.gsma.com"
}

```

set_default_dp_address

```
usage: build.py [-h] DP_ADDRESS
```

Positional Arguments

DP_ADDRESS Default SM-DP+ address as UTF-8 string

get_euicc_challenge

Obtain an authentication challenge from the eUICC using the ES10b GetEUICCChallenge() function.

Example:

```

pySIM-shell (00:MF/ADF.ISD-R)> get_euicc_challenge
{
    "euicc_challenge": "3668f20d4e6c8e85609bbca8c14873fd"
}

```

get_euicc_info1

Obtain EUICC Information (1) from the eUICC using the ES10b GetEUICCCInfo() function.

Example:

```

pySIM-shell (00:MF/ADF.ISD-R)> get_euicc_info1
{
  "svn": "2.2.0",
  "euicc_ci_pki_list_for_verification": [
    {
      "subject_key_identifier_seq": {
        "unknown_ber_tlv_ie_c0": null
      }
    },
    {
      "subject_key_identifier_seq": {
        "unknown_ber_tlv_ie_f5": {
          "raw": "72bdf98a95d65cbeb88a38a1c11d800a85c3"
        }
      }
    }
  ],
  "euicc_ci_pki_list_for_signing": [
    {
      "subject_key_identifier_seq": {
        "unknown_ber_tlv_ie_c0": null
      }
    },
    {
      "subject_key_identifier_seq": {
        "unknown_ber_tlv_ie_f5": {
          "raw": "72bdf98a95d65cbeb88a38a1c11d800a85c3"
        }
      }
    }
  ]
}

```

get_euicc_info2

Obtain EUICC Information (2) from the eUICC using the ES10b GetEUICCCInfo() function.

Example:

```

pySIM-shell (00:MF/ADF.ISD-R)> get_euicc_info2
{
  "profile_version": "2.1.0",
  "svn": "2.2.0",
  "euicc_firmware_ver": "4.4.0",
  "ext_card_resource": "81010082040006ddc68304000016e0",
  "uicc_capability": "067f36c0",
  "ts102241_version": "9.2.0",
  "global_platform_version": "2.3.0",
  "rsp_capability": "0490",
  "euicc_ci_pki_list_for_verification": [
    {
      "subject_key_identifier_seq": {
        "unknown_ber_tlv_ie_c0": null
      }
    }
  ]
}

```

(continues on next page)

(continued from previous page)

```

    }
  },
  {
    "subject_key_identifier_seq": {
      "unknown_ber_tlv_ie_f5": {
        "raw": "72bdf98a95d65cbeb88a38a1c11d800a85c3"
      }
    }
  }
],
"euicc_ci_pki_list_for_signing": [
  {
    "subject_key_identifier_seq": {
      "unknown_ber_tlv_ie_c0": null
    }
  },
  {
    "subject_key_identifier_seq": {
      "unknown_ber_tlv_ie_f5": {
        "raw": "72bdf98a95d65cbeb88a38a1c11d800a85c3"
      }
    }
  }
],
"unknown_ber_tlv_ie_99": {
  "raw": "06c0"
},
"pp_version": "0.0.1",
"ss_accreditation_number": "G&DAccreditationNbr",
"unknown_ber_tlv_ie_ac": {
  "raw":
→ "801f312e322e3834302e313233343536372f6d79506c6174666f726d4c6162656c812568747470733a2f2f6d79636f6d7061
→ "
}
}

```

list_notification

Obtain the list of notifications from the eUICC using the ES10b ListNotification() function.

Example:

```

pySIM-shell (00:MF/ADF.ISD-R)> list_notification
{
  "notification_metadata_list": {
    "notification_metadata": {
      "seq_number": 61,
      "profile_mgmt_operation": {
        "pmo": {
          "install": true,
          "enable": false,
          "disable": false,

```

(continues on next page)

(continued from previous page)

```

        "delete": false
    },
    "notification_address": "testsmdppplus1.example.com",
    "iccid": "89000123456789012358"
}
}
}

```

remove_notification_from_list

```
usage: build.py [-h] SEQ_NR
```

Positional Arguments

SEQ_NR Sequence Number of the to-be-removed notification

Example:

```

pySIM-shell (00:MF/ADF.ISD-R)> remove_notification_from_list 60
{
    "delete_notification_status": "ok"
}

```

get_profiles_info

Obtain information about the profiles present on the eUICC using the ES10c GetProfilesInfo() function.

Example:

```

pySIM-shell (00:MF/ADF.ISD-R)> get_profiles_info
{
    "profile_info_seq": [
        {
            "profile_info": {
                "iccid": "89000123456789012341",
                "isdpa_id": "a0000005591010ffffffff8900001100",
                "profile_state": "disabled",
                "service_provider_name": "GSMA Test 1A",
                "profile_name": "GSMA Generic eUICC Test Profile 1A",
                "profile_class": "operational"
            }
        },
        {
            "profile_info": {
                "iccid": "89000123456789012358",
                "isdpa_id": "a0000005591010ffffffff8900001200",
                "profile_state": "disabled",
                "service_provider_name": "OsmocomSPN",
                "profile_name": "OsmocomProfile",
                "profile_class": "operational"
            }
        }
    ]
}

```

(continues on next page)

(continued from previous page)

```

    }
  ]
}

```

enable_profile

```
usage: build.py [-h] [--isd-p-aid ISDP_AID | --iccid ICCID]
               [--refresh-required]
```

Named Arguments

--isd-p-aid	Profile identified by its ISD-P AID
--iccid	Profile identified by its ICCID
--refresh-required	whether a REFRESH is required
	Default: False

Example (successful):

```
pySIM-shell (00:MF/ADF.ISD-R)> enable_profile --iccid 89000123456789012358
{
  "enable_result": "ok"
}
```

Example (failed attempt enabling a profile that's already enabled):

```
pySIM-shell (00:MF/ADF.ISD-R)> enable_profile --iccid 89000123456789012358
{
  "enable_result": "profileNotInDisabledState"
}
```

disable_profile

```
usage: build.py [-h] [--isd-p-aid ISDP_AID | --iccid ICCID]
               [--refresh-required]
```

Named Arguments

--isd-p-aid	Profile identified by its ISD-P AID
--iccid	Profile identified by its ICCID
--refresh-required	whether a REFRESH is required
	Default: False

Example (successful):

```
pySIM-shell (00:MF/ADF.ISD-R)> disable_profile --iccid 89000123456789012358
{
  "disable_result": "ok"
}
```

delete_profile

```
usage: build.py [-h] [--isd-p-aid ISDP_AID | --iccid ICCID]
```

Named Arguments

--isd-p-aid	Profile identified by its ISD-P AID
--iccid	Profile identified by its ICCID

Example:

```
pySIM-shell (00:MF/ADF.ISD-R)> delete_profile --iccid 89000123456789012358
{
  "delete_result": "ok"
}
```

get_eid

Obtain the EID of the eUICC using the ES10c GetEID() function.

Example:

```
pySIM-shell (00:MF/ADF.ISD-R)> get_eid
{
  "eid_value": "89049032123451234512345678901235"
}
```

set_nickname

```
usage: build.py [-h] [--profile-nickname PROFILE_NICKNAME] ICCID
```

Positional Arguments

ICCID	ICCID of the profile whose nickname to set
--------------	--

Named Arguments

--profile-nickname	Nickname of the profile
---------------------------	-------------------------

Example:

```
pySIM-shell (00:MF/ADF.ISD-R)> set_nickname --profile-nickname asdf 89000123456789012358
{
  "set_nickname_result": "ok"
}
```

get_certs

Obtain the certificates from an IoT eUICC using the ES10c GetCerts() function.

get_eim_configuration_data

Obtain the eIM configuration data from an IoT eUICC using the ES10b GetEimConfigurationData() function.

1.1.18 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 (00:MF/EF.ICCID)> set numeric_path True
numeric_path - was: False
now: True
pySIM-shell (00:3f00/2fe2)> set numeric_path False
numeric_path - was: True
now: False
pySIM-shell (00:MF/EF.ICCID)> help set

```

1.2 pySim-trace

pySim-trace is a utility for high-level decode of APDU protocol traces such as those obtained with [Osmocom SIMtrace2](#) or [osmo-qcdiag](#).

pySim-trace leverages the existing knowledge of pySim-shell on anything related to SIM cards, including the structure/encoding of the various files on SIM/USIM/ISIM/HPSIM cards, and applies this to decoding protocol traces. This means that it shows not only the name of the command (like READ BINARY), but actually understands what the currently selected file is, and how to decode the contents of that file.

pySim-trace also understands the parameters passed to commands and how to decode them, for example of the AUTHENTICATE command within the USIM/ISIM/HPSIM application.

1.2.1 Demo

To get an idea how pySim-trace usage looks like, you can watch the relevant part of the 11/2022 SIMtrace2 tutorial whose [recording is freely accessible](#).

1.2.2 Running pySim-trace

Running pySim-trace requires you to specify the *source* of the to-be-decoded APDUs. There are several supported options, each with their own respective parameters (like a file name for PCAP decoding).

See the detailed command line reference below for details.

A typical execution of pySim-trace for doing live decodes of *GSMTAP (SIM APDU)* e.g. from SIMtrace2 or osmo-qcdiag would look like this:

```
./pySim-trace.py gsmtap-udp
```

This binds to the default UDP port 4729 (GSMTAP) on localhost (127.0.0.1), and decodes any APDUs received there.

1.2.3 pySim-trace command line reference

1.2.4 Constraints

- In order to properly track the current location in the filesystem tree and other state, it is important that the trace you're decoding includes all of the communication with the SIM, ideally from the very start (power up).
- pySim-trace currently only supports ETSI UICC (USIM/ISIM/HPSIM) and doesn't yet support legacy GSM SIM. This is not a fundamental technical constraint, it's just simply that nobody got around developing and testing that part. Contributions are most welcome.

1.3 Legacy tools

legacy tools are the classic pySim-prog and pySim-read programs that existed long before pySim-shell.

These days, it is highly recommended to use pySim-shell instead of these legacy tools.

1.3.1 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 the user specifies every parameter manually:

This is the most common way to use pySim-prog. The user will specify all relevant parameters directly via the commandline. A typical commandline would look like this:

```
pySim-prog.py -p <pcsc_reader> --ki <ki_value> --opc <opc_value> --mcc <mcc_value>
--mnc <mnc_value> --country <country_code> --imsi <imsi_value> --iccid <iccid_value>
--pin-adm <adm_pin>
```

Please note, that this already lengthy commandline still only contains the most common card parameters. For a full list of all possible parameters, use the `--help` option of pySim-prog. It is also important to mention that

not all parameters are supported by all card types. In particular, very simple programmable SIM cards will only support a very basic set of parameters, such as MCC, MNC, IMSI and KI values.

- one where the parameters are generated from a minimal set:

It is also possible to leave the generation of certain parameters to `pySim-prog`. This is in particular helpful when a large number of cards should be initialized with randomly generated key material.

```
pySim-prog.py -p <pcsc_reader> --mcc <mcc_value> --mnc <mnc_value> --secret  
<random_secret> --num <card_number> --pin-adm <adm_pin>
```

The parameter `--secret` specifies a random seed that is used to generate the card individual parameters. (IMSI). The secret should contain enough randomness to avoid conflicts. It is also recommended to store the secret safely, in case cards have to be re-generated or the current card batch has to be extended later. For security reasons, the key material, which is also card individual, will not be derived from the random seed. Instead a new random set of Ki and OPc will be generated during each programming cycle. This means fresh keys are generated, even when the `--num` remains unchanged.

The parameter `--num` specifies a card individual number. This number will be manged into the random seed so that it serves as an identifier for a particular set of randomly generated parameters.

In the example above the parameters `--mcc`, and `--mnc` are specified as well, since they identify the GSM network where the cards should operate in, it is absolutely required to keep them static. `pySim-prog` will use those parameters to generate a valid IMSI that has the specified MCC/MNC at the beginning and a random tail.

Specifying the card type:

`pySim-prog` usually autodetects the card type. In case auto detection does not work, it is possible to specify the parameter `--type`. The following card types are supported:

- Fairwaves-SIM
- fakemagicsim
- gialersim
- gcardsim
- magicsim
- OpenCells-SIM
- supersim
- sysmoISIM-SJA2
- sysmoISIM-SJA5
- sysmosim-gr1
- sysmoSIM-GR2
- sysmoUSIM-SJS1
- Wavemobile-SIM

Specifying the card reader:

It is most common to use `pySim-prog` together with a PCSC reader. The PCSC reader number is specified via the `--pcsc-device` or `-p` option. However, other reader types (such as serial readers and modems) are supported. Use the `--help` option of `pySim-prog` for more information.

Card programming using CSV files

To simplify the card programming process, `pySim-prog` also allows to read the card parameters from a CSV file. When a CSV file is used as input, the user does not have to craft an individual commandline for each card. Instead all card related parameters are automatically drawn from the CSV file.

A CSV files may hold rows for multiple (hundreds or even thousands) of cards. `pySim-prog` is able to identify the rows either by ICCID (recommended as ICCIDs are normally not changed) or IMSI.

The CSV file format is a flexible format with mandatory and optional columns, here the same rules as for the commandline parameters apply. The column names match the command line options. The CSV file may also contain columns that are unknown to `pySim-prog`, such as inventory numbers, nicknames or parameters that are unrelated to the card programming process. `pySim-prog` will silently ignore all unknown columns.

A CSV file may contain the following columns:

- name
- iccid (typically used as key)
- mcc
- mnc
- imsi (may be used as key, but not recommended)
- smsp
- ki
- opc
- acc
- pin_adm, adm1 or pin_adm_hex (must be present)
- msisdn
- epdgid
- epdgSelection
- pcscf
- ims_hdomain
- impi
- impu
- opmode
- fplmn

Due to historical reasons, and to maintain the compatibility between multiple different CSV file formats, the ADM pin may be stored in three different columns. Only one of the three columns must be available.

- adm1: This column contains the ADM pin in numeric ASCII digit format. This format is the most common.
- pin_adm: Same as adm1, only the column name is different
- pin_adm_hex: If the ADM pin consists of raw HEX digits, rather than of numerical ASCII digits, then the ADM pin can also be provided as HEX string using this column.

The following example shows a typical minimal example

```
"imsi","iccid","acc","ki","opc","adm1"
"999700000053010","8988211000000530108","0001","51ACE8BD6313C230F0BFE1A458928DF0",
↪ "E5A00E8DE427E21B206526B5D1B902DF","65942330"
"999700000053011","8988211000000530116","0002","746AAFD7F13CFED3AE626B770E53E860",
↪ "38F7CE8322D2A7417E0BBBD1D7B1190EC","13445792"
"999700123053012","8988211000000530124","0004","D0DA4B7B150026ADC966DC637B26429C",
↪ "144FD3AEAC208DFFF4E2140859BAE8EC","53540383"
"999700000053013","8988211000000530132","0008","52E59240ABAC6F53FF5778715C5CE70E",
↪ "D9C988550DC70B95F40342298EB84C5E","26151368"
"999700000053014","8988211000000530140","0010","3B4B83CB9C5F3A0B41EBD17E7D96F324",
↪ "D61DCC160E3B91F284979552CC5B4D9F","64088605"
"999700000053015","8988211000000530157","0020","D673DAB320D81039B025263610C2BBB3",
↪ "4BCE1458936B338067989A06E5327139","94108841"
"999700000053016","8988211000000530165","0040","89DE5ACB76E06D14B0F5D5CD3594E2B1",
↪ "411C4B8273FD7607E1885E59F0831906","55184287"
"999700000053017","8988211000000530173","0080","977852F7CEE83233F02E69E211626DE1",
↪ "2EC35D48DBF2A99C07D4361F19EF338F","70284674"
```

The following commandline will instruct `pySim-prog` to use the provided CSV file as parameter source and the ICCID (read from the card before programming) as a key to identify the card. To use the IMSI as a key, the parameter `--read-imsi` can be used instead of `--read-iccid`. However, this option is only recommended to be used in very specific corner cases.

```
pySim-prog.py -p <pcsc_reader> --read-csv <path_to_csv_file> --source csv --read-iccid
```

It is also possible to pick a row from the CSV file by manually providing an ICCID (option `--iccid`) or an IMSI (option `--imsi`) that is then used as a key to find the matching row in the CSV file.

```
pySim-prog.py -p <pcsc_reader> --read-csv <path_to_csv_file> --source csv --iccid
<iccid_value>
```

Writing CSV files

`pySim-prog` is also able to generate CSV files that contain a subset of the parameters it has generated or received from some other source (commandline, CSV-File). The generated file will be header-less and contain the following columns:

- name
- iccid
- mcc
- mnc
- imsi
- smsp
- ki
- opc

A commandline that makes use of the CSV write feature would look like this:

```
pySim-prog.py -p <pcsc_reader> --read-csv <path_to_input_csv_file> --read-iccid --source
csv --write-csv <path_to_output_csv_file>
```

Batch programming

In case larger card batches need to be programmed, it is possible to use the `--batch` parameter to run `pySim-prog` in batch mode.

The batch mode will prompt the user to insert a card. Once a card is detected in the reader, the programming is carried out. The user may then remove the card again and the process starts over. This allows for a quick and efficient card programming without permanent commandline interaction.

1.3.2 pySim-read

`pySim-read` allows to read some of the most important data items from a SIM card. This means it will only read some files of the card, and will only read files accessible to a normal user (without any special authentication)

These days, it is recommended to 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.ICCID
- DF.GSM
- EF.IMSI
- EF.GID1
- EF.GID2
- EF.SMSP
- EF.SPN
- EF.PLMNsel
- EF.PLMNwAcT
- EF.OPLMNwAcT
- EF.HPLMNwAcT
- 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.IMPUP
- EF.UICCIARI
- EF.IST

pySim-read usage

Legacy tool for reading some parts of a SIM card

```
usage: pySim-read.py [-h] [-d DEV] [-b BAUD] [--pcsc-shared]
                    [-p PCSC | --pcsc-regex REGEX] [--modem-device DEV]
                    [--modem-baud BAUD] [--osmocon PATH] [--apdu-trace]
```

Named Arguments

--apdu-trace Trace the command/response APDUs exchanged with the card
Default: False

Serial Reader

Use a simple/ultra-low-cost serial reader attached to a (physical or USB/virtual) RS232 port. This doesn't work with all RS232-attached smart card readers, only with the very primitive readers following the ancient *Phoenix* or *Smart Mouse* design.

-d, --device Serial Device for SIM access
Default: '/dev/ttyUSB0'

-b, --baud Baud rate used for SIM access
Default: 9600

PC/SC Reader

Use a PC/SC card reader to talk to the SIM card. PC/SC is a standard API for how applications access smart card readers, and is available on a variety of operating systems, such as Microsoft Windows, MacOS X and Linux. Most vendors of smart card readers provide drivers that offer a PC/SC interface, if not even a generic USB CCID driver is used. You can use a tool like `pcsc_scan -r` to obtain a list of readers available on your system.

--pcsc-shared Open PC/SC reader in SHARED access (default: EXCLUSIVE)
Default: False

-p, --pcsc-device Number of PC/SC reader to use for SIM access

--pcsc-regex Regex matching PC/SC reader to use for SIM access

AT Command Modem Reader

Talk to a SIM Card inside a mobile phone or cellular modem which is attached to this computer and offers an AT command interface including the AT+CSIM interface for Generic SIM access as specified in 3GPP TS 27.007.

--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

Use an OsmocomBB compatible phone to access the SIM inserted to the phone SIM slot. This will require you to run the OsmocomBB firmware inside the phone (can be ram-loaded). It also requires that you run the `osmocon` program, which provides a unix domain socket to which this reader driver can attach.

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

1.4 pySim library

1.4.1 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.BerTlvEF(fid: str, sfid: str = None, name: str = None, desc: str = None, parent:
                                CardDF = None, size: Tuple[int, int | None] = (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.

do_delete_all(opts)

Delete all data from a BER-TLV EF

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

static export(*as_json: bool, lchan*)

Export the file contents of a BerTlvEF. This method returns a shell command string (See also ShellCommand definition in this class) that can be used to write the file contents back.

class pySim.filesystem.**CardADF**(*aid: str, has_fs: bool = False, **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, like 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

static export(*as_json: bool, lchan*)

Export application specific parameters that are not part of the UICC filesystem.

class pySim.filesystem.**CardApplication**(*name, adf: CardADF | None = None, aid: str = None, sw: 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

static export(*as_json: bool, lchan*)

Export application specific parameters, in the form of commandline script. (see also comment in the export method of class “CardFile”)

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, like 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. :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[[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) → [CardFile](#) | None

Find a file with given file ID within current DF.

lookup_file_by_name(name: str | None) → [CardFile](#) | None

Find a file with given name within current DF.

lookup_file_by_sfid(sfid: str | None) → [CardFile](#) | None

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, like 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=[]*) → 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: CardDF | None = None, profile: CardProfile | None = None, service: int | List[int] | Tuple[int, ...] | None = 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, like 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*) → List[CardFile] | None

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

static export(*as_json: bool, lchan*)

Export file contents in the form of commandline script. This method is meant to be overloaded by a subclass in case any exportable contents are present. The generated script may contain multiple command lines separated by line breaks (“

“), where the last commandline shall have no line break at the end
(e.g. “update_record 1 112233

update_record 1 445566”). Naturally this export method will always refer to the currently selected file of the presented lchan.

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.

fully_qualified_path_str(*prefer_name: bool = True*) → str

Return fully qualified path to file as string.

Parameters

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

get_mf() → CardMF | None

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(*flags=[]*) → Dict[str, 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.

should_exist_for_services(*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

- **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
- **profile** – Card profile that this file should be part of
- **service** – Service (SST/UST/IST) associated with the file

add_application_df(*app: CardADF*)

Add an Application to the MF

decode_select_response(*data_hex: str | None*) → 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(*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.**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.

abstractmethod **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: str = None, name: str = None, desc: str = None, parent: CardDF = None, rec_len: Tuple[int, int | None] = (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** – Tuple of (minimum_length, recommended_length)
- **leftpad** – On write, data must be padded from the left to fit physical record length

class pySim.filesystem.**LinFixedEF**(*fid: str, sfid: str = None, name: str = None, desc: str = None, parent: CardDF | None = None, rec_len: Tuple[int, int | None] = (1, None), leftpad: bool = False, **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** – Tuple of (minimum_length, recommended_length)
- **leftpad** – On write, data must be padded from the left to fit physical record length

class ShellCommands

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, record_nr: int*) → 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
- **record_nr** – record number (1 for first record, ...)

Returns

abstract_data; dict representing the decoded data

decode_record_hex(*raw_hex_data*: str, *record_nr*: int = 1) → 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
- **record_nr** – record number (1 for first record, ...)

Returns

abstract_data; dict representing the decoded data

encode_record_bin(*abstract_data*: dict, *record_nr*: int, *total_len*: int | None = None) → 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
- **record_nr** – record number (1 for first record, ...)
- **total_len** – expected total length of the encoded data (record length)

Returns

binary encoded data

encode_record_hex(*abstract_data*: dict, *record_nr*: int, *total_len*: int | None = None) → 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
- **record_nr** – record number (1 for first record, ...)
- **total_len** – expected total length of the encoded data (record length)

Returns

hex string encoded data

static export(*as_json*: bool, *lchan*)

Export the file contents of a LinFixedEF (or a CyclicEF). This method returns a shell command string (See also ShellCommand definition in this class) that can be used to write the file contents back.

class pySim.filesystem.**Path**(*p*: str | List[str] | List[int])

Representation of a file-system path.

is_parent(*other*: Path) → bool

Is this instance a parent of the given other instance?

relative_to_mf() → *Path*

Return a path relative to MF, i.e. without initial explicit MF.

class pySim.filesystem.**TransRecEF**(*fid: str, rec_len: int, sfid: str = None, name: str = None, desc: str = None, parent: CardDF | None = None, size: Tuple[int, int | None] = (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)

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, total_len: int | None = None*) → 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

- **total_len** – expected total length of the encoded data (record length)

Returns

binary encoded data

encode_record_hex(*abstract_data: dict, total_len: int | None = None*) → 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
- **total_len** – expected total length of the encoded data (record length)

Returns

hex string encoded data

```
class pySim.filesystem.TransparentEF(fid: str, sfid: str = None, name: str = None, desc: str = None,  
                                     parent: CardDF = None, size: Tuple[int, int | None] = (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

encode_bin(*abstract_data: dict, total_len: int | None = None*) → 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
- **total_len** – expected total length of the encoded data (file size)

Returns

binary encoded data

encode_hex(*abstract_data: dict, total_len: int | None = None*) → 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
- **total_len** – expected total length of the encoded data (file size)

Returns

hex string encoded data

static export(*as_json: bool, lchan*)

Export the file contents of a TransparentEF. This method returns a shell command string (See also Shell-Command definition in this class) that can be used to write the file contents back.

`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)

1.4.2 pySim commands abstraction

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

class `pySim.commands.SimCardCommands`(*transport*: `LinkBase`, *lchan_nr*: `int = 0`)

Class providing methods for various card-specific commands such as SELECT, READ BINARY, etc. Historically one instance exists below CardBase, but with the introduction of multiple logical channels there can be multiple instances. The lchan number will then be patched into the CLA byte by the respective instance.

activate_file(*fid*: `Hexstr`) → `Tuple[Hexstr, SwHexstr]`

Execute ACTIVATE FILE command as per TS 102 221 Section 11.1.15.

Parameters

fid – file identifier as hex string

authenticate(*rand*: `Hexstr`, *autn*: `Hexstr`, *context*: `str = '3g'`) → `Tuple[Hexstr, SwHexstr]`

Execute AUTHENTICATE (USIM/ISIM).

Parameters

- **rand** – 16 byte random data as hex string (RAND)
- **autn** – 8 byte Authentication Token (AUTN)
- **context** – 16 byte random data ('3g' or 'gsm')

binary_size(*ef*: `Hexstr` | `List[Hexstr]`) → `int`

Determine the size of given transparent file.

Parameters

ef – string or list of strings indicating name or path of transparent EF

change_chv(*chv_no*: `int`, *pin_code*: `Hexstr`, *new_pin_code*: `Hexstr`) → `Tuple[Hexstr, SwHexstr]`

Change a given CHV (Card Holder Verification == PIN)

Parameters

- **chv_no** – chv number (1=CHV1, 2=CHV2, ...)
- **pin_code** – current chv code as hex string
- **new_pin_code** – new chv code as hex string

create_file(*payload*: `Hexstr`) → `Tuple[Hexstr, SwHexstr]`

Execute CREATE FILE command as per TS 102 222 Section 6.3

deactivate_file() → `Tuple[Hexstr, SwHexstr]`

Execute DECATIVATE FILE command as per TS 102 221 Section 11.1.14.

delete_file(*fid: Hexstr*) → Tuple[Hexstr, SwHexstr]

Execute DELETE FILE command as per TS 102 222 Section 6.4

disable_chv(*chv_no: int, pin_code: Hexstr*) → Tuple[Hexstr, SwHexstr]

Disable a given CHV (Card Holder Verification == PIN)

Parameters

- **chv_no** – chv number (1=CHV1, 2=CHV2, ...)
- **pin_code** – current chv code as hex string
- **new_pin_code** – new chv code as hex string

enable_chv(*chv_no: int, pin_code: Hexstr*) → Tuple[Hexstr, SwHexstr]

Enable a given CHV (Card Holder Verification == PIN)

Parameters

- **chv_no** – chv number (1=CHV1, 2=CHV2, ...)
- **pin_code** – chv code as hex string

envelope(*payload: Hexstr*) → Tuple[Hexstr, SwHexstr]

Send one ENVELOPE command to the SIM

Parameters

payload – payload as hex string

fork_lchan(*lchan_nr: int*) → *SimCardCommands*

Fork a per-lchan specific SimCardCommands instance off the current instance.

get_atr() → Hexstr

Return the ATR of the currently inserted card.

manage_channel(*mode: str = 'open', lchan_nr: int = 0*) → Tuple[Hexstr, SwHexstr]

Execute MANAGE CHANNEL command as per TS 102 221 Section 11.1.17.

Parameters

- **mode** – logical channel operation code ('open' or 'close')
- **lchan_nr** – logical channel number (1-19, 0=assigned by UICC)

property max_cmd_len: int

Maximum length of the command apdu data section. Depends on secure channel protocol used.

read_binary(*ef: Hexstr | List[Hexstr], length: int = None, offset: int = 0*) → Tuple[Hexstr, SwHexstr]

Execute READ BINARY.

Parameters

- **ef** – string or list of strings indicating name or path of transparent EF
- **length** – number of bytes to read
- **offset** – byte offset in file from which to start reading

read_record(*ef: Hexstr | List[Hexstr], rec_no: int*) → Tuple[Hexstr, SwHexstr]

Execute READ RECORD.

Parameters

- **ef** – string or list of strings indicating name or path of linear fixed EF

- **rec_no** – record number to read

record_count(*ef: Hexstr | List[Hexstr]*) → int

Determine the number of records in given file.

Parameters

ef – string or list of strings indicating name or path of linear fixed EF

record_size(*ef: Hexstr | List[Hexstr]*) → int

Determine the record size of given file.

Parameters

ef – string or list of strings indicating name or path of linear fixed EF

reset_card() → Hexstr

Physically reset the card

resize_file(*payload: Hexstr*) → Tuple[Hexstr, SwHexstr]

Execute RESIZE FILE command as per TS 102 222 Section 6.10

resume_uicc(*token: Hexstr*) → Tuple[Hexstr, SwHexstr]

Send SUSPEND UICC (resume) to the card.

retrieve_data(*ef: Hexstr | List[Hexstr], tag: int*) → Tuple[Hexstr, SwHexstr]

Execute RETRIEVE DATA, see also TS 102 221 Section 11.3.1.

Args

ef : string or list of strings indicating name or path of transparent EF tag : BER-TLV Tag of value to be retrieved

run_gsm(*rand: Hexstr*) → Tuple[Hexstr, SwHexstr]

Execute RUN GSM ALGORITHM.

Parameters

rand – 16 byte random data as hex string (RAND)

select_adf(*aid: Hexstr*) → Tuple[Hexstr, SwHexstr]

Execute SELECT a given Application ADF.

Parameters

aid – application identifier as hex string

select_file(*fid: Hexstr*) → Tuple[Hexstr, SwHexstr]

Execute SELECT a given file by FID.

Parameters

fid – file identifier as hex string

select_parent_df() → Tuple[Hexstr, SwHexstr]

Execute SELECT to switch to the parent DF

select_path(*dir_list: Hexstr | List[Hexstr]*) → List[Hexstr]

Execute SELECT for an entire list/path of FIDs.

Parameters

dir_list – list of FIDs representing the path to select

Returns

list of return values (FCP in hex encoding) for each element of the path

send_apdu(*pdu: Hexstr, apply_lchan: bool = True*) → Tuple[Hexstr, SwHexstr]

Sends an APDU and auto fetch response data

Parameters

- **pdu** – string of hexadecimal characters (ex. “A0A40000023F00”)
- **apply_lchan** – apply the currently selected lchan to the CLA byte before sending

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: Hexstr, sw: SwMatchstr = '9000', apply_lchan: bool = True*) → Tuple[Hexstr, SwHexstr]

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.
- **apply_lchan** – apply the currently selected lchan to the CLA byte before sending

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: Hexstr, ins: Hexstr, p1: Hexstr, p2: Hexstr, cmd_constr: Construct, cmd_data: Hexstr, resp_constr: Construct, apply_lchan: bool = True*) → Tuple[dict, SwHexstr]

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
- **apply_lchan** – apply the currently selected lchan to the CLA byte before sending

Returns

Tuple of (decoded_data, sw)

send_apdu_constr_checksw(*cla: Hexstr, ins: Hexstr, p1: Hexstr, p2: Hexstr, cmd_constr: Construct, cmd_data: Hexstr, resp_constr: Construct, sw_exp: SwMatchstr = '9000', apply_lchan: bool = True*) → Tuple[dict, SwHexstr]

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)

set_data(*ef*, *tag*: int, *value*: str, *verify*: bool = False, *conserve*: bool = False) → Tuple[Hexstr, SwHexstr]

Execute SET DATA.

Args

ef : string or list of strings indicating name or path of transparent EF tag : BER-TLV Tag of value to be stored value : BER-TLV value to be stored

status() → Tuple[Hexstr, SwHexstr]

Execute a STATUS command as per TS 102 221 Section 11.1.2.

suspend_uicc(*min_len_secs*: int = 60, *max_len_secs*: int = 43200) → Tuple[int, Hexstr, SwHexstr]

Send SUSPEND UICC to the card.

Parameters

- **min_len_secs** – minimum suspend time seconds
- **max_len_secs** – maximum suspend time seconds

terminal_profile(*payload*: Hexstr) → Tuple[Hexstr, SwHexstr]

Send TERMINAL PROFILE to card

Parameters

payload – payload as hex string

terminate_card_usage() → Tuple[Hexstr, SwHexstr]

Execute TERMINATE CARD USAGE command as per TS 102 222 Section 6.9

terminate_df(*fid*: Hexstr) → Tuple[Hexstr, SwHexstr]

Execute TERMINATE DF command as per TS 102 222 Section 6.7

terminate_ef(*fid*: Hexstr) → Tuple[Hexstr, SwHexstr]

Execute TERMINATE EF command as per TS 102 222 Section 6.8

try_select_path(*dir_list*: List[Hexstr]) → List[Tuple[Hexstr, SwHexstr]]

Try to select a specified path

Parameters

dir_list – list of hex-string FIDs

unlock_chv(*chv_no*: int, *puk_code*: str, *pin_code*: str)

Unlock a given CHV (Card Holder Verification == PIN)

Parameters

- **chv_no** – chv number (1=CHV1, 2=CHV2, ...)
- **puk_code** – puk code as hex string
- **pin_code** – new chv code as hex string

update_binary(*ef: Hexstr | List[Hexstr], data: Hexstr, offset: int = 0, verify: bool = False, conserve: bool = False*) → Tuple[Hexstr, SwHexstr]

Execute UPDATE BINARY.

Parameters

- **ef** – string or list of strings indicating name or path of transparent EF
- **data** – hex string of data to be written
- **offset** – byte offset in file from which to start writing
- **verify** – Whether or not to verify data after write

update_record(*ef: Hexstr | List[Hexstr], rec_no: int, data: Hexstr, force_len: bool = False, verify: bool = False, conserve: bool = False, leftpad: bool = False*) → Tuple[Hexstr, SwHexstr]

Execute UPDATE RECORD.

Parameters

- **ef** – string or list of strings indicating name or path of linear fixed EF
- **rec_no** – record number to read
- **data** – hex string of data to be written
- **force_len** – enforce record length by using the actual data length
- **verify** – verify data by re-reading the record
- **conserve** – read record and compare it with data, skip write on match
- **leftpad** – apply 0xff padding from the left instead from the right side.

verify_chv(*chv_no: int, code: Hexstr*) → Tuple[Hexstr, SwHexstr]

Verify a given CHV (Card Holder Verification == PIN)

Parameters

- **chv_no** – chv number (1=CHV1, 2=CHV2, ...)
- **code** – chv code as hex string

`pySim.commands.cla_with_lchan`(*cla_byte: Hexstr, lchan_nr: int*) → Hexstr

Embed a logical channel number into the hex-string encoded CLA value.

`pySim.commands.lchan_nr_to_cla`(*cla: int, lchan_nr: int*) → int

Embed a logical channel number into the CLA byte.

1.4.3 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: ApduTracer | None = None,
                                proactive_handler: ProactiveHandler | None = None)
```

Base class for link/transport to card.

abstract connect()

Connect to a card immediately

abstract disconnect()

Disconnect from card

reset_card()

Resets the card (power down/up)

send_apdu(apdu: *Hexstr*) → Tuple[Hexstr, SwHexstr]

Sends an APDU with minimal processing

Parameters

apdu – string of hexadecimal characters (ex. “A0A40000023F00”, must comply to ISO/IEC 7816-3, section 12.1)

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(apdu: *Hexstr*, sw: *SwMatchstr* = '9000') → Tuple[Hexstr, SwHexstr]

Sends an APDU and check returned SW

Parameters

- **apdu** – string of hexadecimal characters (ex. “A0A40000023F00”, must comply to ISO/IEC 7816-3, section 12.1)
- **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”)

set_sw_interpreter(interp)

Set an (optional) status word interpreter.

abstract wait_for_card(timeout: *int* | None = 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 ?

```
class pySim.transport.LinkBaseTpdu(sw_interpreter=None, apdu_tracer: ApduTracer | None = None,
                                    proactive_handler: ProactiveHandler | None = None)
```

abstract send_tpd*u*(*tpdu: Hexstr*) → Tuple[Hexstr, SwHexstr]

Implementation specific method for sending the resulting TPDU. This method must accept TPDU's as defined in ETSI TS 102 221, section 7.3.1 and 7.3.2, depending on the protocol selected.

set_tpd*u***_format**(*protocol: int*)

Set TPDU format. Each transport protocol has its specific TPDU format. This method allows the concrete transport layer implementation to set the TPDU format it expects. (This method must not be called by higher layers. Switching the TPDU format does not switch the transport protocol that the reader uses on the wire)

Parameters

protocol – number of the transport protocol used. (0 => T=0, 1 => T=1)

class pySim.transport.**ProactiveHandler**

Abstract base class representing the interface of some code that handles the proactive commands, as returned by the card in responses to the FETCH command.

receive_fetch(*pcmd: ProactiveCommand*)

Default handler for not otherwise handled proactive commands.

class pySim.transport.**StdoutAduTracer**

Minimalistic APDU tracer, printing commands to stdout.

pySim.transport.**argparse_add_reader_args**(*arg_parser: ArgumentParser*)

Add all reader related arguments to the given argparse.ArgumentParser instance.

pySim.transport.**init_reader**(*opts, **kwargs*) → *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**(*opts: Namespace = Namespace(osmocon_sock='/tmp/osmocom_l2'), **kwargs*)

Transport Link for Calypso based phones.

connect()

Connect to a card immediately

disconnect()

Disconnect from card

send_tpd*u*(*tpdu: Hexstr*) → Tuple[Hexstr, SwHexstr]

Implementation specific method for sending the resulting TPDU. This method must accept TPDU's as defined in ETSI TS 102 221, section 7.3.1 and 7.3.2, depending on the protocol selected.

wait_for_card(*timeout: int | None = 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 ?

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(opts: Namespace =  

Namespace(modem_dev='/dev/ttyUSB0',  

modem_baud=115200), **kwargs)
```

Transport Link for 3GPP TS 27.007 compliant modems.

connect()

Connect to a card immediately

disconnect()

Disconnect from card

send_tpdu(*tpdu: Hexstr*) → Tuple[Hexstr, SwHexstr]

Implementation specific method for sending the resulting TPDU. This method must accept TPDU as defined in ETSI TS 102 221, section 7.3.1 and 7.3.2, depending on the protocol selected.

wait_for_card(*timeout: int | None = 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 ?

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(opts: Namespace = Namespace(pcsc_dev=0), **kwargs)
```

pySim: PCSC reader transport link.

connect()

Connect to a card immediately

disconnect()

Disconnect from card

send_tpdu(*tpdu: Hexstr*) → Tuple[Hexstr, SwHexstr]

Implementation specific method for sending the resulting TPDU. This method must accept TPDU as defined in ETSI TS 102 221, section 7.3.1 and 7.3.2, depending on the protocol selected.

wait_for_card(*timeout: int | None = 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(opts=Namespace(device='/dev/ttyUSB0', baudrate=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

send_tpdu(tpdu: Hexstr) → Tuple[Hexstr, SwHexstr]

Implementation specific method for sending the resulting TPDU. This method must accept TPDUs as defined in ETSI TS 102 221, section 7.3.1 and 7.3.2, depending on the protocol selected.

wait_for_card(timeout: int | None = 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 ?

1.4.4 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: str | None = None, tag: int | None = 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: str | None = 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: str | None = 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: str | None = 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 DataObjectSequence, 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.TLV_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.**boxed_heading_str**(*heading, width=80*)

Generate a string that contains a boxed heading.

pySim.utils.**calculate_luhn**(*cc*) → int

Calculate Luhn checksum used in e.g. ICCID and IMEI

pySim.utils.**dec_imsi**(*ef: Hexstr*) → str | None

Converts an EF value to the IMSI string representation

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: Hexstr, op_hex: Hexstr*) → Hexstr

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_imsi**(*imsi: str*)

Converts a string IMSI into the encoded value of the EF

pySim.utils.**enc_plmn**(*mcc: Hexstr, mnc: Hexstr*) → Hexstr

Converts integer MCC/MNC into 3 bytes for EF

`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.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.mcc_from_imsi(imsi: str) → str | None`

Derive the MCC (Mobile Country Code) from the first three digits of an IMSI

`pySim.utils.mnc_from_imsi(imsi: str, long: bool = False) → str | None`

Derive the MNC (Mobile Country Code) from the 4th to 6th digit of an IMSI

`pySim.utils.parse_command_apdu(apdu: bytes) → int`

Parse a given command APDU and return case (see also ISO/IEC 7816-3, Table 12 and Figure 26), lc, le and the data field.

Parameters

apdu – hexstring that contains the command APDU

Returns

tuple containing case, lc and le values of the APDU (case, lc, le, data)

`pySim.utils.sanitize_pin_adm(pin_adm, pin_adm_hex=None) → Hexstr`

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.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

`pySim.utils.verify_luhn(digits: str)`

Verify the Luhn check digit; raises ValueError if it is incorrect.

1.4.5 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

1.4.6 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.

1.4.7 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 = ‘’) → str | None

get a single field from CSV file using a specified key/value pair

class pySim.card_key_provider.CardKeyProviderCsv(filename: str, transport_keys: dict)

Card key provider implementation that allows to query against a specified CSV file. Supports column-based encryption as it is generally a bad idea to store cryptographic key material in plaintext. Instead, the key material should be encrypted by a “key-encryption key”, occasionally also known as “transport key” (see GSMA FS.28).

Parameters

- **filename** – file name (path) of CSV file containing card-individual key/data
- **transport_keys** – a dict indexed by field name, whose values are hex-encoded AES keys for the respective field (column) of the CSV. This is done so that different fields (columns) can use different transport keys, which is strongly recommended by GSMA FS.28

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’

static `process_transport_keys(transport_keys: dict)`

Apply a single transport key to multiple fields/columns, if the name is a group.

`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=[])  
→ str | None
```

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

1.5 osmo-smdpp

osmo-smdpp is a proof-of-concept implementation of a minimal **SM-DP+** as specified for the *GSMA Consumer eSIM Remote SIM provisioning*.

At least at this point, it is intended to be used for research and development, and not as a production SM-DP+.

Unless you are a GSMA SAS-SM accredited SM-DP+ operator and have related DPtls, DPauth and DPpb certificates signed by the GSMA CI, you **can not use osmo-smdpp with regular production eUICC**. This is due to how the GSMA eSIM security architecture works. You can, however, use *osmo-smdpp* with so-called *test-eUICC*, which contain certificates/keys signed by GSMA test certificates as laid out in GSMA SGP.26.

At this point, *osmo-smdpp* does not support anything beyond the bare minimum required to download eSIM profiles to an eUICC. Specifically, there is no ES2+ interface, and there is no built-in support for profile personalization yet.

osmo-smdpp currently

- [by default] uses test certificates copied from GSMA SGP.26 into */smdpp-data/certs*, assuming that your *osmo-smdpp* would be running at the host name *testsmdppplus1.example.com*. You can of course replace those certificates with your own, whether SGP.26 derived or part of a *private root CA* setup with matheing eUICCs.
- doesn't understand profile state. Any profile can always be downloaded any number of times, irrespective of the EID or whether it was downloaded before. This is actually very useful for R&D and testing, as it doesn't require you to generate new profiles all the time. This logic of course is unsuitable for production usage.

- doesn't perform any personalization, so the IMSI/ICCID etc. are always identical (the ones that are stored in the respective UPP *.der* files)
- **is absolutely insecure**, as it
- does not perform all of the mandatory certificate verification (it checks the certificate chain, but not the expiration dates nor any CRL)
- does not evaluate/consider any *Confirmation Code*
- stores the sessions in an unencrypted *python shelf* and is hence leaking one-time key materials used for profile encryption and signing.

1.5.1 Running osmo-smdpp

osmo-smdpp does not have built-in TLS support as the used *twisted* framework appears to have problems when using the example elliptic curve certificates (both NIST and Brainpool) from GSMA.

So in order to use it, you have to put it behind a TLS reverse proxy, which terminates the ES9+ HTTPS from the LPA, and then forwards it as plain HTTP to osmo-smdpp.

nginx as TLS proxy

If you use *nginx* as web server, you can use the following configuration snippet:

```
upstream smdpp {
    server localhost:8000;
}

server {
    listen 443 ssl;
    server_name testsmdppplus1.example.com;

    ssl_certificate /my/path/to/pysim/smdpp-data/certs/DPtls/CERT_S_SM_DP_TLS_NIST.
↪pem;
    ssl_certificate_key /my/path/to/pysim/smdpp-data/certs/DPtls/SK_S_SM_DP_TLS_NIST.
↪pem;

    location / {
        proxy_read_timeout 600s;

        proxy_hide_header X-Powered-By;
        proxy_set_header X-Real-IP $remote_addr;
        proxy_set_header X-Forwarded-For $proxy_add_x_forwarded_for;
        proxy_set_header X-Forwarded-Proto https;
        proxy_set_header X-Forwarded-Port $proxy_port;
        proxy_set_header Host $host;

        proxy_pass http://smdpp;
    }
}
```

You can of course achieve a similar functionality with apache, lighttpd or many other web server software.

supplementary files

The *smdpp-data/certs* directory contains the DPtIs, DPauth and DPpb as well as CI certificates used; they are copied from GSMA SGP.26 v2. You can of course replace them with custom certificates if you're operating eSIM with a *private root CA*.

The *smdpp-data/upp* directory contains the UPP (Unprotected Profile Package) used. The file names (without .der suffix) are looked up by the *matchingID* parameter from the activation code presented by the LPA.

commandline options

osmo-smdpp currently doesn't have any configuration file or command line options. You just run it, and it will bind its plain-HTTP ES9+ interface to local TCP port 8000.

DNS setup for your LPA

The LPA must resolve *testsmdppplus1.example.com* to the IP address of your TLS proxy.

It must also accept the TLS certificates used by your TLS proxy.

Supported eUICC

If you run osmo-smdpp with the included SGP.26 certificates, you must use an eUICC with matching SGP.26 certificates, i.e. the EUM certificate must be signed by a SGP.26 test root CA and the eUICC certificate in turn must be signed by that SGP.26 EUM certificate.

sysmocom (sponsoring development and maintenance of pySim and osmo-smdpp) is selling SGP.26 test eUICC as *sysmoEUICC1-C2T*. They are publicly sold in the [sysmocom webshop](#).

In general you can use osmo-smdpp also with certificates signed by any other certificate authority. You just always must ensure that the certificates of the SM-DP+ are signed by the same root CA as those of your eUICCs.

Hypothetically, osmo-smdpp could also be operated with GSMA production certificates, but it would require that somebody brings the code in-line with all the GSMA security requirements (HSM support, ...) and operate it in a GSMA SAS-SM accredited environment and pays for the related audits.

1.6 sim-rest-server

Sometimes there are use cases where a [remote] application will need access to a USIM for authentication purposes. This is, for example, in case an IMS test client needs to perform USIM based authentication against an IMS core.

The pysim repository contains two programs: *sim-rest-server.py* and *sim-rest-client.py* that implement a simple approach to achieve the above:

sim-rest-server.py speaks to a [usually local] USIM via the PC/SC API and provides a high-level REST API towards [local or remote] applications that wish to perform UMTS AKA using the USIM.

sim-rest-client.py implements a small example client program to illustrate how the REST API provided by *sim-rest-server.py* can be used.

1.6.1 REST API Calls

POST /sim-auth-api/v1/slot/SLOT_NR

where SLOT_NR is the integer-encoded slot number (corresponds to PC/SC reader number). When using a single sysmoOCTSIM board, this is in the range of 0..7

Example: */sim-auth-api/v1/slot/0* for the first slot.

Request Body

The request body is a JSON document, comprising of

1. the RAND and AUTN parameters as hex-encoded string
2. the application against which to authenticate (USIM, ISIM)

Example:

```
{
  "rand": "bb685a4b2fc4d697b9d6a129dd09a091",
  "autn": "eea7906f8210000004faf4a7df279b56"
}
```

HTTP Status Codes

HTTP status codes are used to represent errors within the REST server and the SIM reader hardware. They are not used to communicate protocol level errors reported by the SIM Card. An unsuccessful authentication will hence have a 200 OK HTTP Status code and then encode the SIM specific error information in the Response Body.

Status	Code	Description
200	OK	Successful execution
400	Bad Request	Request body is malformed
404	Not Found	Specified SIM Slot doesn't exist
410	Gone	No SIM card inserted in slot

Response Body

The response body is a JSON document, either

1. a successful outcome; encoding RES, CK, IK as hex-encoded string
2. a sync failure; encoding AUTS as hex-encoded string
3. errors #. authentication error (incorrect MAC) #. authentication error (security context not supported) #. key freshness failure #. unspecified card error

Example (success):

```
{
  "successful_3g_authentication": {
    "res": "b15379540ec93985",
    "ck": "713fde72c28cbd282a4cd4565f3d6381",
    "ik": "2e641727c95781f1020d319a0594f31a",
    "kc": "771a2c995172ac42"
  }
}
```

Example (re-sync case):

```
{
  "synchronisation_failure": {
    "auts": "dc2a591fe072c92d7c46ecfe97e5"
  }
}
```

1.6.2 Concrete example using the included sysmolSIM-SJA2

This was tested using SIMs ending in IMSI numbers 45890...45899

The following command were executed successfully:

Slot 0

```
$ /usr/local/src/pysim/contrib/sim-rest-client.py -c 1 -n 0 -k_
↪ 841EAD87BC9D974ECA1C167409357601 -o 3211CACDD64F51C3FD3013ECD9A582A0
-> {'rand': 'fb195c7873b20affa278887920b9dd57', 'autn': 'd420895a6aa2000089cd016f8d8ae67c
↪ '}
<- {'successful_3g_authentication': {'res': '131004db2ff1ce8e', 'ck':
↪ 'd42eb5aa085307903271b2422b698bad', 'ik': '485f81e6fd957fe3cad374adf12fe1ca', 'kc':
↪ '64d3f2a32f801214'}}
```

Slot 1

```
$ /usr/local/src/pysim/contrib/sim-rest-client.py -c 1 -n 1 -k_
↪ 5C2CE9633FF9B502B519A4EACD16D9DF -o 9834D619E71A02CD76F00CC7AA34FB32
-> {'rand': '433dc5553db95588f1d8b93870930b66', 'autn': '126bafdcbe9e00000026a208da61075d
↪ '}
<- {'successful_3g_authentication': {'res': '026d7ac42d379207', 'ck':
↪ '83a90ba331f47a95c27a550b174c4a1f', 'ik': '31e1d10329ffaf0ca1684a1bf0b0a14a', 'kc':
↪ 'd15ac5b0fff73ecc'}}
```

1.7 suci-keytool

Subscriber concealment is an important feature of the 5G SA architecture: It avoids the many privacy issues associated with having a permanent identifier (SUPI, traditionally the IMSI) transmitted in plain text over the air interface. Using SUCI solves this issue not just for the air interface; it even ensures the SUPI/IMSI is not known to the visited network (VPLMN) at all.

In principle, the SUCI mechanism works by encrypting the SUPI by asymmetric (public key) cryptography: Only the HPLMN is in possession of the private key and hence can decrypt the SUCI to the SUPI, while each subscriber has the public key in order to encrypt their SUPI into the SUCI. In reality, the details are more complex, as there are ephemeral keys and cryptographic MAC involved.

In any case, in order to operate a SUCI-enabled 5G SA network, you will have to

1. generate a ECC key pair of public + private key
2. deploy the public key on your USIMs
3. deploy the private key on your 5GC, specifically the UDM function

pysim contains (int its *contrib* directory) a small utility program that can make it easy to generate such keys: *suci-keytool.py*

1.7.1 Generating keys

Example: Generating a *secp256r1* ECC public key pair and storing it to */tmp/suci.key*:

```
$ ./contrib/suci-keytool.py --key-file /tmp/suci.key generate-key --curve secp256r1
```

1.7.2 Dumping public keys

In order to store the key to SIM cards as part of *ADF.USIM/DF.5GS/EF.SUCI_Calc_Info*, you will need a hexadecimal representation of the public key. You can achieve that using the *dump-pub-key* operation of *suci-keytool*:

Example: Dumping the public key part from a previously generated key file:

```
$ ./contrib/suci-keytool.py --key-file /tmp/suci.key dump-pub-key
0473152f32523725f5175d255da2bd909de97b1d06449a9277bc629fe42112f8643e6b69aa6dce6c86714ccbe6f2e0f4f4898d1
```

If you want the point-compressed representation, you can use the *--compressed* option:

```
$ ./contrib/suci-keytool.py --key-file /tmp/suci.key dump-pub-key --compressed
0373152f32523725f5175d255da2bd909de97b1d06449a9277bc629fe42112f864
```

1.7.3 suci-keytool syntax

Generate or export SUCI keys for 5G SA networks

```
usage: contrib/suci-keytool.py [-h] --key-file KEY_FILE
                               {generate-key,dump-pub-key} ...
```

Positional Arguments

command	Possible choices: generate-key, dump-pub-key The command to perform
----------------	--

Named Arguments

--key-file	The key file to use
-------------------	---------------------

Sub-commands

generate-key

Generate a new key pair

```
contrib/suci-keytool.py generate-key [-h] --curve {secp256r1,curve25519}
```

Named Arguments

--curve	Possible choices: secp256r1, curve25519 The ECC curve to use
----------------	---

dump-pub-key

Dump the public key

```
contrib/suci-keytool.py dump-pub-key [-h] [--compressed]
```

Named Arguments

--compressed Use point compression
Default: False

INDICES AND TABLES

- `genindex`
- `modindex`
- `search`

PYTHON MODULE INDEX

p

- `pySim.card_handler`, [76](#)
- `pySim.card_key_provider`, [77](#)
- `pySim.commands`, [63](#)
- `pySim.exceptions`, [76](#)
- `pySim.filesystem`, [52](#)
- `pySim.transport`, [69](#)
- `pySim.transport.calypso`, [70](#)
- `pySim.transport.modem_atcmd`, [71](#)
- `pySim.transport.pcsc`, [71](#)
- `pySim.transport.serial`, [72](#)
- `pySim.utils`, [72](#)

A

`activate_file()` (*pySim.commands.SimCardCommands* method), 63
`add_application_df()` (*pySim.filesystem.CardMF* method), 56
`add_file()` (*pySim.filesystem.CardDF* method), 54
`add_files()` (*pySim.filesystem.CardDF* method), 54
`add_files()` (*pySim.filesystem.CardModel* class method), 57
`apply_matching_models()` (*pySim.filesystem.CardModel* static method), 57
`argparse_add_reader_args()` (in module *pySim.transport*), 70
`authenticate()` (*pySim.commands.SimCardCommands* method), 63

B

`BerTlvEF` (class in *pySim.filesystem*), 52
`BerTlvEF.ShellCommands` (class in *pySim.filesystem*), 52
`binary_size()` (*pySim.commands.SimCardCommands* method), 63
`boxed_heading_str()` (in module *pySim.utils*), 74
`build_select_path_to()` (*pySim.filesystem.CardFile* method), 55

C

`calculate_luhn()` (in module *pySim.utils*), 74
`CalypsoSimLink` (class in *pySim.transport.calypso*), 70
`card_key_provider_get()` (in module *pySim.card_key_provider*), 77
`card_key_provider_get_field()` (in module *pySim.card_key_provider*), 78
`card_key_provider_register()` (in module *pySim.card_key_provider*), 78
`CardADF` (class in *pySim.filesystem*), 53
`CardApplication` (class in *pySim.filesystem*), 53
`CardCommand` (class in *pySim.utils*), 72
`CardCommandSet` (class in *pySim.utils*), 72
`CardDF` (class in *pySim.filesystem*), 53
`CardDF.ShellCommands` (class in *pySim.filesystem*), 54

`CardEF` (class in *pySim.filesystem*), 54
`CardFile` (class in *pySim.filesystem*), 55
`CardHandler` (class in *pySim.card_handler*), 76
`CardHandlerAuto` (class in *pySim.card_handler*), 76
`CardHandlerBase` (class in *pySim.card_handler*), 76
`CardKeyProvider` (class in *pySim.card_key_provider*), 77
`CardKeyProviderCsv` (class in *pySim.card_key_provider*), 77
`CardMF` (class in *pySim.filesystem*), 56
`CardModel` (class in *pySim.filesystem*), 57
`change_chv()` (*pySim.commands.SimCardCommands* method), 63
`cla_with_lchan()` (in module *pySim.commands*), 68
`connect()` (*pySim.transport.calypso.CalypsoSimLink* method), 70
`connect()` (*pySim.transport.LinkBase* method), 69
`connect()` (*pySim.transport.modem_atcmd.ModemATCommandLink* method), 71
`connect()` (*pySim.transport.pcsc.PcscSimLink* method), 71
`connect()` (*pySim.transport.serial.SerialSimLink* method), 72
`create_file()` (*pySim.commands.SimCardCommands* method), 63
`CyclicEF` (class in *pySim.filesystem*), 57

D

`DataObject` (class in *pySim.utils*), 72
`DataObjectChoice` (class in *pySim.utils*), 73
`DataObjectCollection` (class in *pySim.utils*), 73
`DataObjectSequence` (class in *pySim.utils*), 73
`deactivate_file()` (*pySim.commands.SimCardCommands* method), 63
`dec_imsi()` (in module *pySim.utils*), 74
`decode()` (*pySim.utils.DataObject* method), 72
`decode()` (*pySim.utils.DataObjectChoice* method), 73
`decode()` (*pySim.utils.DataObjectCollection* method), 73
`decode()` (*pySim.utils.DataObjectSequence* method), 73
`decode_bin()` (*pySim.filesystem.TransparentEF* method), 61

`decode_hex()` (*pySim.filesystem.TransparentEF method*), 62

`decode_multi()` (*pySim.utils.DataObjectSequence method*), 73

`decode_record_bin()` (*pySim.filesystem.LinFixedEF method*), 58

`decode_record_bin()` (*pySim.filesystem.TransRecEF method*), 60

`decode_record_hex()` (*pySim.filesystem.LinFixedEF method*), 59

`decode_record_hex()` (*pySim.filesystem.TransRecEF method*), 60

`decode_select_response()` (*pySim.filesystem.CardFile method*), 55

`decode_select_response()` (*pySim.filesystem.CardMF method*), 56

`delete_file()` (*pySim.commands.SimCardCommands method*), 63

`derive_mcc()` (*in module pySim.utils*), 74

`derive_milenage_opc()` (*in module pySim.utils*), 74

`derive_mnc()` (*in module pySim.utils*), 74

`disable_chv()` (*pySim.commands.SimCardCommands method*), 64

`disconnect()` (*pySim.transport.calypso.CalypsoSimLink method*), 70

`disconnect()` (*pySim.transport.LinkBase method*), 69

`disconnect()` (*pySim.transport.modem_atcmd.ModemATCommandLink method*), 71

`disconnect()` (*pySim.transport.pcsc.PcscSimLink method*), 71

`disconnect()` (*pySim.transport.serial.SerialSimLink method*), 72

`do_decode_hex()` (*pySim.filesystem.LinFixedEF.ShellCommands method*), 58

`do_decode_hex()` (*pySim.filesystem.TransparentEF.ShellCommands method*), 61

`do_delete_all()` (*pySim.filesystem.BerTlvEF.ShellCommands method*), 52

`do_delete_data()` (*pySim.filesystem.BerTlvEF.ShellCommands method*), 52

`do_edit_binary_decoded()` (*pySim.filesystem.TransparentEF.ShellCommands method*), 61

`do_edit_record_decoded()` (*pySim.filesystem.LinFixedEF.ShellCommands method*), 58

`do_read_binary()` (*pySim.filesystem.TransparentEF.ShellCommands method*), 61

`do_read_binary_decoded()` (*pySim.filesystem.TransparentEF.ShellCommands method*), 61

`do_read_record()` (*pySim.filesystem.LinFixedEF.ShellCommands method*), 58

`do_read_record_decoded()` (*pySim.filesystem.LinFixedEF.ShellCommands method*), 58

`do_read_records()` (*pySim.filesystem.LinFixedEF.ShellCommands method*), 58

`do_read_records_decoded()` (*pySim.filesystem.LinFixedEF.ShellCommands method*), 58

`do_retrieve_data()` (*pySim.filesystem.BerTlvEF.ShellCommands method*), 52

`do_retrieve_tags()` (*pySim.filesystem.BerTlvEF.ShellCommands method*), 52

`do_set_data()` (*pySim.filesystem.BerTlvEF.ShellCommands method*), 52

`do_update_binary()` (*pySim.filesystem.TransparentEF.ShellCommands method*), 61

`do_update_binary_decoded()` (*pySim.filesystem.TransparentEF.ShellCommands method*), 61

`do_update_record()` (*pySim.filesystem.LinFixedEF.ShellCommands method*), 58

`do_update_record_decoded()` (*pySim.filesystem.LinFixedEF.ShellCommands method*), 58

`done()` (*pySim.card_handler.CardHandlerBase method*), 76

E

`enable_chv()` (*pySim.commands.SimCardCommands method*), 64

`enc_imsi()` (*in module pySim.utils*), 74

`enc_plmn()` (*in module pySim.utils*), 74

`encode()` (*pySim.utils.DataObjectSequence method*), 74

`encode_bin()` (*pySim.filesystem.TransparentEF method*), 62

`encode_hex()` (*pySim.filesystem.TransparentEF method*), 62

`encode_multi()` (*pySim.utils.DataObjectSequence method*), 74

`encode_record_bin()` (*pySim.filesystem.LinFixedEF method*), 59

`encode_record_bin()` (*pySim.filesystem.TransRecEF method*), 60

`encode_record_hex()` (*pySim.filesystem.LinFixedEF method*), 59

`encode_record_hex()` (*pySim.filesystem.TransRecEF method*), 61

`envelopes()` (*pySim.commands.SimCardCommands method*), 64

`error()` (*pySim.card_handler.CardHandlerBase method*), 76

`expand_hex()` (*in module pySim.utils*), 74

`export()` (*pySim.filesystem.BerTlvEF static method*), 53

`export()` (*pySim.filesystem.CardADF static method*), 53

- `export()` (*pySim.filesystem.CardApplication static method*), 53
- `export()` (*pySim.filesystem.CardFile static method*), 55
- `export()` (*pySim.filesystem.LinFixedEF static method*), 59
- `export()` (*pySim.filesystem.TransparentEF static method*), 62
- ## F
- `fork_lchan()` (*pySim.commands.SimCardCommands method*), 64
- `from_bytes()` (*pySim.utils.DataObject method*), 73
- `from_bytes()` (*pySim.utils.TL0_DataObject method*), 74
- `from_tlv()` (*pySim.utils.DataObject method*), 73
- `fully_qualified_path()` (*pySim.filesystem.CardFile method*), 55
- `fully_qualified_path_fobj()` (*pySim.filesystem.CardFile method*), 55
- `fully_qualified_path_str()` (*pySim.filesystem.CardFile method*), 56
- ## G
- `get()` (*pySim.card_handler.CardHandlerBase method*), 76
- `get()` (*pySim.card_key_provider.CardKeyProvider method*), 77
- `get()` (*pySim.card_key_provider.CardKeyProviderCsv method*), 77
- `get_addr_type()` (*in module pySim.utils*), 75
- `get_app_names()` (*pySim.filesystem.CardMF method*), 57
- `get_app_selectables()` (*pySim.filesystem.CardMF method*), 57
- `get_atr()` (*pySim.commands.SimCardCommands method*), 64
- `get_field()` (*pySim.card_key_provider.CardKeyProvider method*), 77
- `get_mf()` (*pySim.filesystem.CardFile method*), 56
- `get_profile()` (*pySim.filesystem.CardFile method*), 56
- `get_selectable_names()` (*pySim.filesystem.CardFile method*), 56
- `get_selectables()` (*pySim.filesystem.CardDF method*), 54
- `get_selectables()` (*pySim.filesystem.CardEF method*), 54
- `get_selectables()` (*pySim.filesystem.CardFile method*), 56
- `get_selectables()` (*pySim.filesystem.CardMF method*), 57
- ## I
- `init_reader()` (*in module pySim.transport*), 70
- `interpret_sw()` (*in module pySim.filesystem*), 62
- `interpret_sw()` (*pySim.filesystem.CardApplication method*), 53
- `is_parent()` (*pySim.filesystem.Path method*), 59
- ## L
- `lchan_nr_to_cla()` (*in module pySim.commands*), 68
- `LinFixedEF` (*class in pySim.filesystem*), 57
- `LinFixedEF.ShellCommands` (*class in pySim.filesystem*), 58
- `LinkBase` (*class in pySim.transport*), 69
- `LinkBaseTpdu` (*class in pySim.transport*), 69
- `lookup()` (*pySim.utils.CardCommandSet method*), 72
- `lookup_file_by_fid()` (*pySim.filesystem.CardDF method*), 54
- `lookup_file_by_name()` (*pySim.filesystem.CardDF method*), 54
- `lookup_file_by_sfid()` (*pySim.filesystem.CardDF method*), 54
- ## M
- `manage_channel()` (*pySim.commands.SimCardCommands method*), 64
- `match()` (*pySim.filesystem.CardModel class method*), 57
- `match_cla()` (*pySim.utils.CardCommand method*), 72
- `max_cmd_len` (*pySim.commands.SimCardCommands property*), 64
- `mcc_from_imsi()` (*in module pySim.utils*), 75
- `mnc_from_imsi()` (*in module pySim.utils*), 75
- `ModemATCommandLink` (*class in pySim.transport.modem_atcmd*), 71
- module
- pySim.card_handler*, 76
 - pySim.card_key_provider*, 77
 - pySim.commands*, 63
 - pySim.exceptions*, 76
 - pySim.filesystem*, 52
 - pySim.transport*, 69
 - pySim.transport.calypso*, 70
 - pySim.transport.modem_atcmd*, 71
 - pySim.transport.pcsc*, 71
 - pySim.transport.serial*, 72
 - pySim.utils*, 72
- ## N
- `NoCardError`, 76
- ## P
- `parse_command_apdu()` (*in module pySim.utils*), 75
- `Path` (*class in pySim.filesystem*), 59
- `PcscSimLink` (*class in pySim.transport.pcsc*), 71
- `ProactiveHandler` (*class in pySim.transport*), 70
- `process_transport_keys()` (*pySim.card_key_provider.CardKeyProviderCsv static method*), 77

ProtocolError, 76
pySim.card_handler
 module, 76
pySim.card_key_provider
 module, 77
pySim.commands
 module, 63
pySim.exceptions
 module, 76
pySim.filesystem
 module, 52
pySim.transport
 module, 69
pySim.transport.calypso
 module, 70
pySim.transport.modem_atcmd
 module, 71
pySim.transport.pcsc
 module, 71
pySim.transport.serial
 module, 72
pySim.utils
 module, 72

R

read_binary() (pySim.commands.SimCardCommands
 method), 64
read_record() (pySim.commands.SimCardCommands
 method), 64
ReaderError, 76
receive_fetch() (pySim.transport.ProactiveHandler
 method), 70
record_count() (pySim.commands.SimCardCommands
 method), 65
record_size() (pySim.commands.SimCardCommands
 method), 65
relative_to_mf() (pySim.filesystem.Path method), 59
reset_card() (pySim.commands.SimCardCommands
 method), 65
reset_card() (pySim.transport.LinkBase method), 69
resize_file() (pySim.commands.SimCardCommands
 method), 65
resume_uicc() (pySim.commands.SimCardCommands
 method), 65
retrieve_data() (pySim.commands.SimCardCommands
 method), 65
run_gsm() (pySim.commands.SimCardCommands
 method), 65
select_file() (pySim.commands.SimCardCommands
 method), 65
select_parent_df() (pySim.commands.SimCardCommands
 method), 65
select_path() (pySim.commands.SimCardCommands
 method), 65
send_apdu() (pySim.commands.SimCardCommands
 method), 65
send_apdu() (pySim.transport.LinkBase method), 69
send_apdu_checks() (pySim.commands.SimCardCommands
 method), 66
send_apdu_checks() (pySim.transport.LinkBase
 method), 69
send_apdu_constr() (pySim.commands.SimCardCommands
 method), 66
send_apdu_constr_checks() (pySim.commands.SimCardCommands
 method), 66
send_tpdu() (pySim.transport.calypso.CalypsoSimLink
 method), 70
send_tpdu() (pySim.transport.LinkBaseTpdu method),
 69
send_tpdu() (pySim.transport.modem_atcmd.ModemATCommandLink
 method), 71
send_tpdu() (pySim.transport.pcsc.PcscSimLink
 method), 71
send_tpdu() (pySim.transport.serial.SerialSimLink
 method), 72
SerialSimLink (class in pySim.transport.serial), 72
set_data() (pySim.commands.SimCardCommands
 method), 67
set_sw_interpreter() (pySim.transport.LinkBase
 method), 69
set_tpdu_format() (pySim.transport.LinkBaseTpdu
 method), 70
should_exist_for_services() (pySim.filesystem.CardFile method), 56
SimCardCommands (class in pySim.commands), 63
status() (pySim.commands.SimCardCommands
 method), 67
StdoutApuTracer (class in pySim.transport), 70
suspend_uicc() (pySim.commands.SimCardCommands
 method), 67
sw_match() (in module pySim.utils), 75
SwMatchError, 76

T

sanitize_pin_adm() (in module pySim.utils), 75
select_adf() (pySim.commands.SimCardCommands
 method), 65
tabulate_str_list() (in module pySim.utils), 75
terminal_profile() (pySim.commands.SimCardCommands
 method), 67
terminate_card_usage() (pySim.commands.SimCardCommands
 method), 67

`terminate_df()` (*pySim.commands.SimCardCommands*
method), 67
`terminate_ef()` (*pySim.commands.SimCardCommands*
method), 67
`TL0_DataObject` (*class in pySim.utils*), 74
`to_bytes()` (*pySim.utils.DataObject method*), 73
`to_bytes()` (*pySim.utils.TL0_DataObject method*), 74
`to_dict()` (*pySim.utils.DataObject method*), 73
`to_tlv()` (*pySim.utils.DataObject method*), 73
`TransparentEF` (*class in pySim.filesystem*), 61
`TransparentEF.ShellCommands` (*class in*
pySim.filesystem), 61
`TransRecEF` (*class in pySim.filesystem*), 60
`try_select_path()` (*pySim.commands.SimCardCommands*
method), 67

U

`unlock_chv()` (*pySim.commands.SimCardCommands*
method), 67
`update_binary()` (*pySim.commands.SimCardCommands*
method), 68
`update_record()` (*pySim.commands.SimCardCommands*
method), 68

V

`verify_chv()` (*pySim.commands.SimCardCommands*
method), 68
`verify_luhn()` (*in module pySim.utils*), 76

W

`wait_for_card()` (*pySim.transport.calypso.CalypsoSimLink*
method), 70
`wait_for_card()` (*pySim.transport.LinkBase method*),
69
`wait_for_card()` (*pySim.transport.modem_atcmd.ModemATCommandLink*
method), 71
`wait_for_card()` (*pySim.transport.psc.PcscSimLink*
method), 71
`wait_for_card()` (*pySim.transport.serial.SerialSimLink*
method), 72