SOSSE Reference Manual

Generated by Doxygen 1.2.14

Wed Jul 9 16:22:07 2003

Contents

1 SOSSE - Simple Operating System for Smartcard Education

1.1 Overview

SOSSE is a simple operating system for smart cards which is opensource and free. It has been developed to provide a smart card platform for learning, experimenting and implementing inovative concepts. The hardware platform choosen for this project are the Funcards based on commercial-of-the-shelf AVR microcontrollers, because the hardware documentation is available without non-disclosure agreement.

1.2 Hardware

All of the currently supported cards use the `AVR-Microcontrollers` from `Atmel`. The are e.g. called Funcard, Jupiter or Pink card. The used AVRs are the AT90S2323, AT90S2343, AT90S8515 and AT90S8535. There are even a few boards with the ATmega161 available. The data sheets for all Atmel AVR microcontrollers are available `on the Atmel website`.

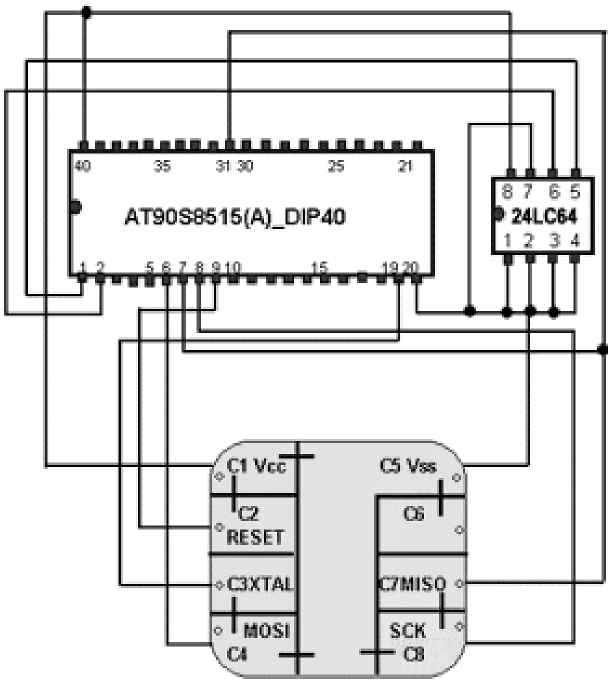

Figure 1: Connection scheme for the AVR cards.

The pinout of the AVR cards is as follows:

- C1 (VCC): Vcc
- C2 (RST): RESET
- C3 (CLK): XTAL
- C4: MOSI
- C5 (VSS): Vss
- C6: Not connected
- C7 (I/O): MISO
- C8: SCK

Additional to the smart cards, a programmer is needed. These are available at the same shops as the AVR cards.

1.3 Software

SOSSE has been developed using the GCC development tools. These are available at `http://combio.de/avr/`. There are the sources, binaries and RPMs for Linux and binaries for MS Windows.

1.4 Design Decisions

Programming Language

The C programming language has been choosen instead of assembler, because:

- It is easier to understand C source code.
- Modern smart card operating systems are probably all written mostly in C. Who does want to write and verify 128kB assembler code?
- When new smart card architectures are available for SOSSE, these can be utilized more easily.

1.5 Building Instructions

Standard Build

1. Go to the src directory and edit **config.h**. This file contains the configuration data for the build process. The meaning of the options is described in the doxygen documention of **config.h** itself. (It is not possible to switch on all features, because then the code size is above 8kB.)

2. You can edit eedata.S if you want to change the EEPROM data loaded onto the AVR card. You might want to change the card serial number, the start value for the random number generator, the keys, the file permissions or the created files.

3. There are two defines in the Makefile, which could need editing. The first is the -mmcu setting in CC and the second is -mtiny-stack in CFLAGS. Adjust these to reflect your used AVR card.

4. To build the HEX files start *make*.

5. The resulting HEX files sosse.hex and eedata.hex (or BIN files sosse.bin and eedata.bin) have then to be loaded on the card with the programmer and an apropriate software. The destination for sosse.hex is the Flash and for eedata.hex the internal EEPROM.

Debugging

Test programms which run natively on the host operating system are built running:

make -f Makefile.emu

These can be used for comfortable debugging and testing using the normal GNU debugger. When the reset sequence is implemented, this can be used to simulate a SOSSE smart card via e.g. a Season interface.

CT-API driver

Building

A virtual card sitting in a virtual reader can be build with:

make -f Makefile.ctapi

This builds a CT-API driver, which can be used to test applications and to debug using a normal debugger.

Supported CT-BCS commands

- EJECT ICC: Supported with functional unit ICC1. P2 is ignored.
- GET STATUS: Supported tag is 80. Possible card states are 01 and 05.
- REQUEST ICC: Supported with functional unit ICC1 and response data none and complete ATR. Display settings are ignored.
- RESET CT: Supported with functional unit CT and ICC1 and response data none and complete ATR. This command is the same as REQUEST ICC, but does support reseting the CT, which stops the ICC.

Proprietary command: EEPROM FILE

Header

CLA	INS	P1	P2	P3
20	E0	Flags	RFU	Length

Command data

Byte(s)	Description	Length
1 - Length	File name	Length

Status words

SW	Description
90 00	Command completed successfully

Description

This command sets the parameter for EEPROM loading and saving for the virtual card.

If bit 1 of P1 is set, the EEPROM is saved at stopping of the ICC and if bit 2 is set, the EEPROM loaded at starting of the ICC. The file name for this operation is specified in the data field of the command.

The proprietary commands must be executed before a REQUEST ICC and RESET CT, because the parameters are held in RAM, which is separated by a fork at card reset.

A success of the command means only, that the variables have correctly been set. It does not verify, that the file is createable, readable or writeable.

1.6 More Information and Download

The web page with up to date information about SOSSE is at:

`http://www.mbsks.franken.de/sosse/`

The download location of the SOSSE sources is at:

`ftp://ftp.franken.de/pub/crypt/chipcards/sosse/`

The documentation was generated using DoxyGen:

`http://www.doxygen.org/`

1.7 Special Files

These File are only used, when filesystem support is compiled in. Otherwise special absolut locations in EEPROM are used to hold the data.

EF Ext Auth

Path: 3F00FF00

Content:

- 16 Bytes Key
- 1 Byte Retry Counter (resets to 10)

EF Int Auth

Path: 3F00FF01

Content:

- 16 Bytes Key
- 1 Byte Retry Counter (unused)

EF PIN

Path: 3F00FF02

Content:

- 8 Bytes PIN
- 1 Byte Retry Counter (resets to 3)
- 8 Bytes PUK
- 1 Byte Retry Counter (resets to 10)

1.8 Commands

Write EEPROM. (Test command)

Header

CLA	INS	P1	P2	P3
80	02	Address high	Address low	Length

Command data

Byte(s)	Description	Length
1 - Length	Data to write to *Address*	Length

Status words

SW	Description
90 00	Command completed successfully
65 00	Memory failure (unsuccessful writing)
67 00	Wrong Lc

Description

This command can be used to directly write into EEPROM. The start address is specified in P1P2 and the data length in P3. If there exists an internal and external EEPROM, the address convention of the HAL is used. Currently the internal EEPROM is in the address range 0 - sizeof internal EEPROM. The external EEPROM starts directly afterwards.

This command should be disabled by setting CONF_WITH_TESTCMDS to 0 for a production version.

Data updates are NOT protected with transactions.

Read EEPROM. (Test command)

Header

CLA	INS	P1	P2	P3
80	04	Address high	Address low	Length

Response data

Byte(s)	Description	Length
1 - Length	Data read from *Address*	Length

Status words

SW	Description
90 00	Command completed successfully

Description

This command can be used to directly read from EEPROM. The start address is specified in P1P2 and the data length in P3. If there exists an internal and external EEPROM, the address convention of the HAL is used. Currently the internal EEPROM is in the address range 0 - sizeof internal EEPROM. The external EEPROM starts directly afterwards.

This command should be disabled by setting CONF_WITH_TESTCMDS to 0 for a production version.

LED Effects (Fun command)

Header

CLA	INS	P1	P2	P3
80	06	Effect	Parameter	00

Status words

SW	Description
90 00	Command completed successfully

Description

This command does control the LEDs present on some PCB AVR cards and shows some nice effects.

Effect type 0x01 does switch the LEDs specified in P2 on. Each bit controlls one LED. Which LED is controlled may be card dependent. Currently bits 0 - 6 control the leds beginning from the left, when looking down at them with the contacts to the top.

Effect type 0x02 does implement a runlight with the left 4 LED block. It may need adjustment on other cards to select the correct LEDs.

This command is only enabled when setting CONF_WITH_FUNNY to 1.

Change PIN.

Header

CLA	INS	P1	P2	P3
80	24	00	00	10

Command data

Byte(s)	Description	Length
1 - 8	Current PIN	8
9 - 16	New PIN	8

Status words

SW	Description
90 00	Command completed successfully
63 CX	Verification failed, retry counter specified
67 00	Wrong Lc
69 83	Authentication method blocked
6A 86	Incorrect parameters P1-P2
6A 88	Referenced data not found

Description

This command changes the PIN of the card. The first four bytes of the data is the current PIN, which is checked for correctness. The second four bytes of the data is the new PIN, which will be written. On success the AUTH_FLAG_PIN bit in authstate is set, on fauilure it is cleared.

Warning: Currently the retry counter is not handled in a secure way. For more information see e.g. Rankl/Effing.

PIN update is protected with transactions.

Create File.

Header

CLA	INS	P1	P2	P3
80	E0	00	00	06

Command data

Byte(s)	Description	Length
1 - 2	Size	2
3 - 4	FID	2
5	File type (DF: 38, EF: 00)	1
6	Access conditions (DF: Create/Delete, EF: Update/Read)	1

Status words

SW	Description
90 00	Command completed successfully
67 00	Wrong Lc
69 82	Security status not satisfied
69 86	Command not allowed (no DF selected)
6A 80	Incorrect arameters in the data field
6A 84	Not enough free space in DF
6A 86	Incorrect parameters P1-P2
6A 89	File exists already

Description

This command creates an elementary or dedicated file. To succeed the current file must be a dedicated file or the master file and the access conditions for file creation must be fulfilled.

The data of the command is the file header. It consists of a 2-byte size, 2-byte FID, 1-byte type and 1-byte access conditions. Size can't be 0000. The FID can't be 3F00 or FFFF. Valid file types are 00 for EF and 38 for DF. The access conditions are described in the next tables.

File type	High Nibble	Low Nibble
DF	Create File	Delete File
EF	Update	Read

Nibble Value	Access condition
0	Always
1	PIN verified
2	External Authentication key verified
3	PIN or External Authentication Key verified
4	PIN and External Authentication Key verified
F	Never

File header updates are protected with transactions.

Delete File.

Header

CLA	INS	P1	P2	P3
80	E4	00	00	02

Command data

Byte(s)	Description	Length
1 - 2	FID	2

Status words

SW	Description
90 00	Command completed successfully
67 00	Wrong Lc
69 82	Security status not satisfied
69 86	Command not allowed (no DF selected)
6A 82	File not found
6A 86	Incorrect parameters P1-P2

Description

The file with the specified FID is deleted from the current file, which must be a DF or the MF. Also the access conditions for delete must be satisfied and the specified file must be the last one in the DF/MF. Deletion of a DF, which contains EFs, is possible.

File header updates are protected with transactions.

External Authentication.

Header

CLA	INS	P1	P2	P3
80	82	00	00	08

Command data

Byte(s)	Description	Length
1 - 8	Encrypted Challenge	8

Status words

SW	Description
90 00	Command completed successfully
63 CX	Verification failed, retry counter specified
67 00	Wrong Lc
69 83	Authentication method blocked
69 85	Conditions of use not satisfied
6A 86	Incorrect parameters P1-P2
6A 88	Referenced data not found

Description

Directly before this command a 8 byte challenge must be fetched from the card with the Get Challenge command. This challenge must then be encrypted with the TEA algorithm and the External Authentication key. For authentication the resulting data must then be sent back via this command.

Warning: Currently the retry counter is not handled in a secure way. For more information see e.g. Rankl/Effing.

Get Challenge

Header

CLA	INS	P1	P2	P3
80	84	00	00	08

Response data

Byte(s)	Description	Length
1 - 8	Random Challenge	8

Status words

SW	Description
90 00	Command completed successfully
67 00	Wrong Le
6A 86	Incorrect parameters P1-P2

Description

Returns a challenge usable for the External Authentication command, which must then be the next issued command.

Warning: On the current hardware this challenge is produced with a PRNG based on TEA. It is not evaluated, if the current mechanism is appropriate for this task.

Get Response.

Header

CLA	INS	P1	P2	P3
80	C0	00	00	Length

Response data after Internal Authenticate

Byte(s)	Description	Length
1 - 8	Internal Authentication key encrypted challenge	8

Response data after Select

Byte(s)	Description	Length
1 - 2	Size	2
3 - 4	FID	2
5	File type (DF: 38, EF: 00)	1
6	Access conditions	1

Status words

SW	Description
90 00	Command completed successfully
69 85	Wrong condition (No data available)
6A 86	Incorrect parameters P1-P2
6C XX	Wrong Le

Description

This command fetches data from Internal Authentication and Select. These commands signal the availability of data with the status word 61XX. For more details about the data returned after a Select, see at the Create command.

Internal Authentication.

Header

CLA	INS	P1	P2	P3
80	88	00	Decrypt	08

This command decrypts the data if P2 is 01. This functionality is only enabled when CONF_WITH_DECRYPT is set to 1.

Command data

Byte(s)	Description	Length
1 - 8	Random Challenge	8

Status words

SW	Description
61 08	Command completed successfully (Data available)
67 00	Wrong Lc
69 85	Conditions of use not satisfied
6A 86	Incorrect parameters P1-P2
6A 88	Referenced data not found

Description

Provieds a random challenge to the card. It encryptes the challenge with TEA and the Internal Authentication key and makes it available via Get Response. For the successful execution either the PIN or the External Authentication key must have been successfully verified previously.

Because of possibility to decrypt data, this command can easily be used to implement some form of Remotely Keyed Encryption.

Read Binary.

Header

CLA	INS	P1	P2	P3
80	B0	Offset high	Offest low	Length

Response data

Byte(s)	Description	Length
1 - Length	Data	Length

Status words

SW	Description
90 00	Command completed successfully
69 82	Security status not satisfied
69 86	Command not allowed (no EF selected)
6A 82	File not found
6A 84	File to short
6B 00	Wrong parameters (offset outside EF)

Description

Reads *Lenght* bytes from *Offset* of the currently selected EF.

Select File.

Header

CLA	INS	P1	P2	P3
80	A4	00	00	02

Command data

Byte(s)	Description	Length
1 - 2	FID	2

Status words

SW	Description
61 06	Command completed successfully
67 00	Wrong Lc
6A 82	File not found
6A 86	Incorrect parameters P1-P2

Description

It tries to select a file with the specified FID and searches for it in the following order:

1. EF in the current MF/DF
2. DF in MF

If FID is 3F00, the MF is always selected.

Unblock PIN.

Header

CLA	INS	P1	P2	P3
80	2C	00	00	10

Command data

Byte(s)	Description	Length
1 - 8	PUK	8
9 - 16	New PIN	8

Status words

SW	Description
90 00	Command completed successfully
63 CX	Verification failed, retry counter specified
67 00	Wrong Lc
69 83	Authentication method blocked
6A 86	Incorrect parameters P1-P2
6A 88	Referenced data not found

Description

This command changes the PIN of the card and resets the retry counter of this PIN. The first eight bytes of the data is the PUK, which is checked for correctness. The second four bytes of the data is the new PIN, which will be written. On success the AUTH_FLAG_PIN bit in authstate is set, on fauilure it is cleared.

Warning: Currently the retry counter is not handled in a secure way. For more information see e.g. Rankl/Effing.

PIN update is protected with transactions.

Update Binary.

Header

CLA	INS	P1	P2	P3
80	D6	Offset high	Offset low	Length

Command data

Byte(s)	Description	Length
1 - Length	Data	Length

Status words

SW	Description
90 00	Command completed successfully
65 00	Memory failure (unsuccessful writing)
67 00	Wrong Le
69 82	Security status not satisfied
69 86	Command not allowed (no EF selected)
6A 82	File not found
6A 84	File to short
6B 00	Wrong parameters (offset outside EF)

Description

Writes *Lenght* bytes to *Offset* of the currently selected EF.

Data updates are NOT protected with transactions.

Verify Key

Header

CLA	INS	P1	P2	P3
80	2A	00	00	10

Command data

Byte(s)	Description	Length
1 - 16	External Authentication Key	16

Status words

SW	Description
90 00	Command completed successfully
63 CX	Verification failed, retry counter specified
67 00	Wrong Lc
69 83	Authentication method blocked
6A 86	Incorrect parameters P1-P2
6A 88	Referenced data not found

Description

Verifies the External Authentication key as 16 byte plain text in the data part of the command. On success the AUTH_FLAG_KEY bit in authstate is set, on failure it is cleared.

Warning: Currently the retry counter is not handled in a secure way. For more information see e.g. Rankl/Effing.

Verify PIN

Header

CLA	INS	P1	P2	P3
80	20	00	00	08

Command data

Byte(s)	Description	Length
1 - 8	PIN	8

Status words

SW	Description
90 00	Command completed successfully
63 CX	Verification failed, retry counter specified
67 00	Wrong Lc
69 83	Authentication method blocked
6A 86	Incorrect parameters P1-P2
6A 88	Referenced data not found

Description

This command verifies the PIN. The four data bytes contain the user supplied PIN, which is checked for correctness. On success the AUTH_FLAG_PIN bit in authstate is set, on fauilure it is cleared.

Warning: Currently the retry counter is not handled in a secure way. For more information see e.g. Rankl/Effing.

1.9 Developers

- Matthias Bruestle <`m@mbsks.franken.de`>

1.10 License

2 SOSSE Data Structure Index

2.1 SOSSE Data Structures

Here are the data structures with brief descriptions:

3 SOSSE File Index

3.1 SOSSE File List

Here is a list of all files with brief descriptions:

4 SOSSE Page Index

4.1 SOSSE Related Pages

Here is a list of all related documentation pages:

5 SOSSE Data Structure Documentation

5.1 s_finfo Struct Reference

File info struct. This is directly written as a file header in the file system.

`#include <fs.h>`

Data Fields

- **iu16 size**

  Size of the file content.

- **iu16 fid**

  ID of the file.

- **iu8 type**

  Type of the file. (FS_TYPE_*).

- **iu8 ac** [1]

  Access conditions of the file. (See CREATE FILE dox.).

5.1.1 Detailed Description

File info struct. This is directly written as a file header in the file system.

5.1.2 Field Documentation

5.1.2.1 iu8 s_finfo::ac[1]

Access conditions of the file. (See CREATE FILE dox.).

5.1.2.2 iu16 s_finfo::fid

ID of the file.

5.1.2.3 iu16 s_finfo::size

Size of the file content.

5.1.2.4 iu8 s_finfo::type

Type of the file. (FS_TYPE_*).

The documentation for this struct was generated from the following file:

- **fs.h**

5.2 s_fpath Struct Reference

File path specifier.

`#include <fs.h>`

Data Fields

- **iu16 df**

  ID of one DF level. Set to FFFF if unused.

- **iu16 ef**

  IF of the EF. Set to FFFF if unused.

5.2.1 Detailed Description

File path specifier.

Todo:

There is probably a potential for simplification, when the file type (DF/EF) is not handled so rigid.

5.2.2 Field Documentation

5.2.2.1 iu16 s_fpath::df

ID of one DF level. Set to FFFF if unused.

5.2.2.2 iu16 s_fpath::ef

IF of the EF. Set to FFFF if unused.

The documentation for this struct was generated from the following file:

- **fs.h**

5.3 s_fstream Struct Reference

Stream struct.

`#include <fstream.h>`

Data Fields

- **iu16 start**
  EEPROM start address of the stream.
- **iu16 size**
  Data size of the stream.
- **iu16 pos**
  Current position in the stream.

5.3.1 Detailed Description

Stream struct.

5.3.2 Field Documentation

5.3.2.1 iu16 s_fstream::pos

Current position in the stream.

5.3.2.2 iu16 s_fstream::size

Data size of the stream.

5.3.2.3 iu16 s_fstream::start

EEPROM start address of the stream.

The documentation for this struct was generated from the following file:

- **fstream.h**

6 SOSSE File Documentation

6.1 auth.c File Reference

Authentication functions.

`#include <config.h>`

`#include <auth.h>`

`#include <crypt.h>`

`#include <fs.h>`

`#include <fstream.h>`

`#include <hal.h>`

`#include <string.h>`

`#include <sw.h>`

`#include <tea.h>`

`#include <tools.h>`

`#include <transaction.h>`

`#include <types.h>`

Functions

- **bool auth_init** (void)

  Initialize authentication variables.

- **bool auth_checkAc** (**iu8** ac)

  Check fullfillment of access condition.

- **bool auth_getCheckPinLen** (**iu8** pintype, **iu8** pinlen, **iu16** *addr, **iu8** *len, **iu16** *rcaddr, **iu8** *maxret)

  Verifies length of PIN and returns addresses of PIN in EEPROM.

- **bool auth_setPin** (**iu8** pintype, **iu8** *pin, **iu8** pinlen)

  Write PIN, PUK, External Authentication key or Internal Authentication key.

- **bool auth_verifyPin** (**iu8** pintype, **iu8** *pin, **iu8** pinlen)

  Verifies PIN, PUK or External Authentication key and sets flags in authstate.

- **bool auth_getChallenge** (**iu8** *dst, **iu8** rndlen)

  Returns 8 byte challenge for External Authentication.

- **bool auth_createVerifyCryptogram** (**iu8** *cry, **iu8** crylen, **bool** create, **bool** enc)

  Encrypt or decrypt challenge or verify cryptogram.

Variables

- **iu8 challenge** [CRYPT_BLOCK_LEN]

  Value of the last with Get Challenge fetched random challenge.

- **iu8 challvalidity**
- **iu8 authstate**

  Variable indicating current authentication state.

6.1.1 Detailed Description

Authentication functions.

Id:

auth.c,v 1.21 2003/03/30 19:43:15 m Exp

6.1.2 Function Documentation

6.1.2.1 bool auth_checkAc (iu8 *ac*)

Check fullfillment of access condition.

This function checks, if the current setting of authstate fullfills the condition specified with *ac*.

Parameters:

ac A nibble specifying an access condition.

Return values:

TRUE on success.

FALSE on failure. Error code given in sw.

6.1.2.2 bool auth_createVerifyCryptogram (iu8 * *cry*, iu8 *crylen*, bool *create*, bool *enc*)

Encrypt or decrypt challenge or verify cryptogram.

To include the decryption process CONF_WITH_DECRYPT must be set to 1.

Parameters:

cry Pointer to byte array, which supplies challenge and which is used to return the encrypted or decrypted challenge or a cryptogram.

crylen Length of challenge or cryptogram.

create If set to TRUE, a challenge is processed, else a cryptogram is verified.

enc If CONF_WITH_DECRYPT is not set to 1, this has no effect. Otherwise, if TRUE encrypt the challenge, else decrypt it. If *create* is FALSE, this parameter is ignored.

Return values:

TRUE on success.

FALSE on failure. Error code given in sw.

6.1.2.3 bool auth_getChallenge (iu8 * *dst*, iu8 *rndlen*)

Returns 8 byte challenge for External Authentication.

The challenge is also written to the challenge array and challvalidity is set to 2.

Parameters:

dst Destination byte array.

rndlen Length of requested data. This must be 8.

Return values:

TRUE on success.

FALSE on failure. Error code given in sw.

6.1.2.4 bool auth_getCheckPinLen (iu8 *pintype*, iu8 *pinlen*, iu16 * *addr*, iu8 * *len*, iu16 * *rcaddr*, iu8 * *maxret*)

Verifies length of PIN and returns addresses of PIN in EEPROM.

This functions does depending on *pintype*:

- Check the length of the PIN or key.
- Return the address of the PIN or key in EEPROM.
- Return the length of the PIN or key in EEPROM.
- Return the address of the retry counter in EEPROM.

- Return the default max. value for the retry counter.

Parameters:

pintype PIN type. One of the PIN_TYPE defines.

pinlen Length of supplied PIN or key.

addr Returns the address of the PIN or key in EEPROM.

len Returns the length ofthe PIN or key in EEPROM.

rcaddr Returns the address of the retry counter for the PIN or key.

maxret Returns the default max. value for the retry counter.

Return values:

TRUE on success.

FALSE on failure. Error code given in sw.

6.1.2.5 bool auth_init (void)

Initialize authentication variables.

This function resets authstate and challvalidity. It must be called after reset.

Return values:

TRUE on success.

FALSE on failure. Error code given in sw.

6.1.2.6 bool auth_setPin (iu8 *pintype*, iu8 * *pin*, iu8 *pinlen*)

Write PIN, PUK, External Authentication key or Internal Authentication key.

The process is secured by transactions.

Parameters:

pintype Type of the PIN/Key. May be one of the PIN_TYPE_* defines.

pin Pointer to byte array containing PIN/Key.

pinlen Length of PIN/Key.

Return values:

TRUE on success.

FALSE on failure. Error code given in sw.

6.1.2.7 bool auth_verifyPin (iu8 *pintype*, iu8 * *pin*, iu8 *pinlen*)

Verifies PIN, PUK or External Authentication key and sets flags in authstate.

The update of the retry counter is secured by transactions.

Warning: Currently the retry counter is not handled in a secure way. For more information see e.g. Rankl/Effing.

Parameters:

pintype Type of the PIN/Key. May be one of the PIN_TYPE_* defines.

pin Pointer to byte array containing PIN/Key.

pinlen Length of PIN/Key.

Return values:

TRUE on success.

FALSE on failure. Error code given in sw.

6.1.3 Variable Documentation

6.1.3.1 iu8 authstate

Variable indicating current authentication state.

Single bit flags indicate, which authorization process has been successful. The flag is cleared, at the reset of the card and after unsuccessful authorization processes.

Valid flags are:

- AUTH_FLAG_PIN
- AUTH_FLAG_KEY

6.1.3.2 iu8 challenge[CRYPT_BLOCK_LEN]

Value of the last with Get Challenge fetched random challenge.

6.1.3.3 iu8 challvalidity

6.2 auth.h File Reference

Authentication declarations.

`#include <types.h>`

Defines

- #define **AUTH_FLAG_PIN** 0x01

  Flag for authstate signalling a successful PIN verification.

- #define **AUTH_FLAG_KEY** 0x02

  Flag for authstate signalling a successful external authentication or key verification.

- #define **AUTH_AC_ALW** 0x0

  Access Condition: Always.

- #define **AUTH_AC_PIN** 0x1
  Access Condition: PIN.
- #define **AUTH_AC_ADM** 0x2
  Access Condition: Key.
- #define **AUTH_AC_POA** 0x3
  Access Condition: PIN or Key.
- #define **AUTH_AC_PAA** 0x4
  Access Condition: PIN and Key.
- #define **AUTH_AC_NEV** 0xF
  Access Condition: Never.
- #define **PIN_MAX_RETRIES** 3
  Default max. retries for PIN.
- #define **PUK_MAX_RETRIES** 10
  Default max. retries for PUK.
- #define **PIN_TYPE_PIN** 0
  PIN type: PIN.
- #define **PIN_TYPE_PUK** 1
  PIN type: PUK.
- #define **PIN_TYPE_EXT** 2
  PIN type: Key for external authentication.
- #define **PIN_TYPE_INT** 3
  PIN type: Key for internal authentication.
- #define **KEY_MAX_RETRIES** 10
  Default max. retries for keys.
- #define **AUTH_FID_EXTAUTH** 0xFF00
  FID for file with external authentication key.
- #define **AUTH_FID_INTAUTH** 0xFF01
  FID for file with internal authentication key.
- #define **AUTH_FID_PINPUK** 0xFF02
  FID for file with PIN and PUK.

Functions

- **bool auth_init** (void)

  Initialize authentication variables.

- **bool auth_checkAc** (**iu8** ac)

  Check fullfillment of access condition.

- **bool auth_getCheckPinLen** (**iu8** pintype, **iu8** pinlen, **iu16** ∗addr, **iu8** ∗len, **iu16** ∗rcaddr, **iu8** ∗maxret)

  Verifies length of PIN and returns addresses of PIN in EEPROM.

- **bool auth_setPin** (**iu8** pintype, **iu8** ∗pin, **iu8** pinlen)

  Write PIN, PUK, External Authentication key or Internal Authentication key.

- **bool auth_verifyPin** (**iu8** pintype, **iu8** ∗pin, **iu8** pinlen)

  Verifies PIN, PUK or External Authentication key and sets flags in authstate.

- **bool auth_getChallenge** (**iu8** ∗dst, **iu8** rndlen)

  Returns 8 byte challenge for External Authentication.

- **bool auth_createVerifyCryptogram** (**iu8** ∗cry, **iu8** crylen, **bool** create, **bool** enc)

  Encrypt or decrypt challenge or verify cryptogram.

Variables

- **iu8 challvalidity**

  Validity of challenge.

6.2.1 Detailed Description

Authentication declarations.

Here are the declarations and definitions for the authentication functions. The authentication types supported are:

- PIN/PUK
- External Authentication
- Internal Authentication

Todo:

Here are certainly some optimizations possible to reduce code size.

Id:

auth.h,v 1.11 2002/12/22 15:42:55 m Exp

6.2.2 Define Documentation

6.2.2.1 #define AUTH_AC_ADM 0x2

Access Condition: Key.

6.2.2.2 #define AUTH_AC_ALW 0x0

Access Condition: Always.

6.2.2.3 #define AUTH_AC_NEV 0xF

Access Condition: Never.

6.2.2.4 #define AUTH_AC_PAA 0x4

Access Condition: PIN and Key.

6.2.2.5 #define AUTH_AC_PIN 0x1

Access Condition: PIN.

6.2.2.6 #define AUTH_AC_POA 0x3

Access Condition: PIN or Key.

6.2.2.7 #define AUTH_FID_EXTAUTH 0xFF00

FID for file with external authentication key.

It is only searched for the file in the MF and not in any DF.

6.2.2.8 #define AUTH_FID_INTAUTH 0xFF01

FID for file with internal authentication key.

It is only searched for the file in the MF and not in any DF.

6.2.2.9 #define AUTH_FID_PINPUK 0xFF02

FID for file with PIN and PUK.

It is only searched for the file in the MF and not in any DF.

6.2.2.10 #define AUTH_FLAG_KEY 0x02

Flag for authstate signalling a successful external authentication or key verification.

6.2.2.11 #define AUTH_FLAG_PIN 0x01

Flag for authstate signalling a successful PIN verification.

6.2.2.12 #define KEY_MAX_RETRIES 10

Default max. retries for keys.

6.2.2.13 #define PIN_MAX_RETRIES 3

Default max. retries for PIN.

6.2.2.14 #define PIN_TYPE_EXT 2

PIN type: Key for external authentication.

6.2.2.15 #define PIN_TYPE_INT 3

PIN type: Key for internal authentication.

6.2.2.16 #define PIN_TYPE_PIN 0

PIN type: PIN.

6.2.2.17 #define PIN_TYPE_PUK 1

PIN type: PUK.

6.2.2.18 #define PUK_MAX_RETRIES 10

Default max. retries for PUK.

6.2.3 Function Documentation

6.2.3.1 bool auth_checkAc (iu8 *ac*)

Check fullfillment of access condition.

This function checks, if the current setting of authstate fullfills the condition specified with *ac*.

Parameters:

ac A nibble specifying an access condition.

Return values:

TRUE on success.

FALSE on failure. Error code given in sw.

6.2.3.2 bool auth_createVerifyCryptogram (iu8 * *cry*, iu8 *crylen*, bool *create*, bool *enc*)

Encrypt or decrypt challenge or verify cryptogram.

To include the decryption process CONF_WITH_DECRYPT must be set to 1.

Parameters:

- ***cry*** Pointer to byte array, which supplies challenge and which is used to return the encrypted or decrypted challenge or a cryptogram.
- ***crylen*** Length of challenge or cryptogram.
- ***create*** If set to TRUE, a challenge is processed, else a cryptogram is verified.
- ***enc*** If CONF_WITH_DECRYPT is not set to 1, this has no effect. Otherwise, if TRUE encrypt the challenge, else decrypt it. If *create* is FALSE, this parameter is ignored.

Return values:

- ***TRUE*** on success.
- ***FALSE*** on failure. Error code given in sw.

6.2.3.3 bool auth_getChallenge (iu8 * *dst*, iu8 *rndlen*)

Returns 8 byte challenge for External Authentication.

The challenge is also written to the challenge array and challvalidity is set to 2.

Parameters:

- ***dst*** Destination byte array.
- ***rndlen*** Length of requested data. This must be 8.

Return values:

- ***TRUE*** on success.
- ***FALSE*** on failure. Error code given in sw.

6.2.3.4 bool auth_getCheckPinLen (iu8 *pintype*, iu8 *pinlen*, iu16 * *addr*, iu8 * *len*, iu16 * *rcaddr*, iu8 * *maxret*)

Verifies length of PIN and returns addresses of PIN in EEPROM.

This functions does depending on *pintype*:

- Check the length of the PIN or key.
- Return the address of the PIN or key in EEPROM.
- Return the length of the PIN or key in EEPROM.
- Return the address of the retry counter in EEPROM.

- Return the default max. value for the retry counter.

Parameters:

pintype PIN type. One of the PIN_TYPE defines.

pinlen Length of supplied PIN or key.

addr Returns the address of the PIN or key in EEPROM.

len Returns the length ofthe PIN or key in EEPROM.

rcaddr Returns the address of the retry counter for the PIN or key.

maxret Returns the default max. value for the retry counter.

Return values:

TRUE on success.

FALSE on failure. Error code given in sw.

6.2.3.5 bool auth_init (void)

Initialize authentication variables.

This function resets authstate and challvalidity. It must be called after reset.

Return values:

TRUE on success.

FALSE on failure. Error code given in sw.

6.2.3.6 bool auth_setPin (iu8 *pintype*, iu8 ∗ *pin*, iu8 *pinlen*)

Write PIN, PUK, External Authentication key or Internal Authentication key.

The process is secured by transactions.

Parameters:

pintype Type of the PIN/Key. May be one of the PIN_TYPE_∗ defines.

pin Pointer to byte array containing PIN/Key.

pinlen Length of PIN/Key.

Return values:

TRUE on success.

FALSE on failure. Error code given in sw.

6.2.3.7 bool auth_verifyPin (iu8 *pintype*, iu8 ∗ *pin*, iu8 *pinlen*)

Verifies PIN, PUK or External Authentication key and sets flags in authstate.

The update of the retry counter is secured by transactions.

Warning: Currently the retry counter is not handled in a secure way. For more information see e.g. Rankl/Effing.

Parameters:

pintype Type of the PIN/Key. May be one of the PIN_TYPE_* defines.

pin Pointer to byte array containing PIN/Key.

pinlen Length of PIN/Key.

Return values:

TRUE on success.

FALSE on failure. Error code given in sw.

6.2.4 Variable Documentation

6.2.4.1 iu8 challvalidity

Validity of challenge.

The initial value if challvalidity is 2. This is decremented at the start of the command loop, so that it is 1 at the command following a Get Challenge.

6.3 authtest.c File Reference

Test program for authentication functions.

`#include <stdio.h>`

`#include <config.h>`

`#include <auth.h>`

`#include <sw.h>`

`#include <fs.h>`

`#include <hal.h>`

`#include <transaction.h>`

`#include <netinet/in.h>`

Functions

- int **main** (int argc, char *argv[])

Variables

- **iu8 authstate**
- **iu32 challenge** [2]

    Value of the last with Get Challenge fetched random challenge.

- **iu8 challvalidity**

6.3.1 Detailed Description

Test program for authentication functions.

Id:
authtest.c,v 1.11 2002/12/22 15:42:55 m Exp

6.3.2 Function Documentation

6.3.2.1 int main (int *argc*, char * *argv*[ ])

6.3.3 Variable Documentation

6.3.3.1 iu8 authstate

6.3.3.2 iu32 challenge[2]

Value of the last with Get Challenge fetched random challenge.

6.3.3.3 iu8 challvalidity

6.4 commands.c File Reference

Commands.

```
#include <auth.h>
#include <hal.h>
#include <commands.h>
#include <config.h>
#include <crypt.h>
#include <fs.h>
#include <log.h>
#include <stddef.h>
#include <string.h>
#include <sw.h>
#include <t0.h>
#include <tools.h>
#include <types.h>
```

Defines

- #define **pinssize** (PIN_LEN+PIN_LEN)

- #define **pin1len** PIN_LEN

Functions

- void **cmd_write** (void)
  Write EEPROM. (Test command).
- void **cmd_read** (void)
  Read EEPROM. (Test command).
- void **cmd_changeUnblockPIN** (void)
  Change or Unblock PIN.
- void **cmd_create** (void)
  Create File.
- void **cmd_delete** (void)
  Delete File.
- void **cmd_extAuth** (void)
  External Authentication.
- void **cmd_getChallenge** (void)
  Get Challenge.
- void **cmd_intAuth** (void)
  Internal Authentication.
- void **cmd_readBinary** (void)
  Read Binary.
- void **cmd_select** (void)
  Select File.
- void **cmd_updateBinary** (void)
  Update Binary.
- void **cmd_verifyKeyPIN** (void)
  Verify Key/Verify PIN.

6.4.1 Detailed Description

Commands.

The documentation of the commands is on the main page of the doxygen documentation.

Id:
commands.c,v 1.30 2003/05/10 17:26:58 m Exp

6.4.2 Define Documentation

6.4.2.1 #define pin1len PIN_LEN

6.4.2.2 #define pinssize (PIN_LEN+PIN_LEN)

6.4.3 Function Documentation

6.4.3.1 void cmd_changeUnblockPIN (void)

Change or Unblock PIN.

6.4.3.2 void cmd_create (void)

Create File.

6.4.3.3 void cmd_delete (void)

Delete File.

6.4.3.4 void cmd_extAuth (void)

External Authentication.

6.4.3.5 void cmd_getChallenge (void)

Get Challenge.

6.4.3.6 void cmd_intAuth (void)

Internal Authentication.

6.4.3.7 void cmd_read (void)

Read EEPROM. (Test command).

6.4.3.8 void cmd_readBinary (void)

Read Binary.

6.4.3.9 void cmd_select (void)

Select File.

6.4.3.10 void cmd_updateBinary (void)

Update Binary.

6.4.3.11 void cmd_verifyKeyPIN (void)

Verify Key/Verify PIN.

This function is a combined implementation of Verify Key and Verify PIN. This is done to save code size.

6.4.3.12 void cmd_write (void)

Write EEPROM. (Test command).

6.5 commands.h File Reference

Command declarations.

Defines

- #define **CLA_PROP** 0x80

  CLA byte: Proprietary.

- #define **INS_WRITE** 0x02

  INS byte: Write EEPROM.

- #define **INS_READ** 0x04

  INS byte: Read EEPROM.

- #define **INS_LED** 0x06

  INS byte: LED Effects.

- #define **INS_CHANGE_PIN** 0x24

  INS byte: Change PIN.

- #define **INS_CREATE** 0xE0

  INS byte: Create File.

- #define **INS_DELETE** 0xE4

  INS byte: Delete File.

- #define **INS_EXTERNAL_AUTH** 0x82

  INS byte: External Authentication.

- #define **INS_GET_CHALLENGE** 0x84

  INS byte: Get Challenge.

- #define **INS_GET_RESPONSE** 0xC0
  INS byte: Get Response.
- #define **INS_INTERNAL_AUTH** 0x88
  INS byte: Internal Authentication.
- #define **INS_READ_BINARY** 0xB0
  INS byte: Read Binary.
- #define **INS_SELECT** 0xA4
  INS byte: Select File.
- #define **INS_UNBLOCK_PIN** 0x2C
  INS byte: Unblock PIN.
- #define **INS_UPDATE_BINARY** 0xD6
  INS byte: Update Binary.
- #define **INS_VERIFY_KEY** 0x2A
  INS byte: Verify Key.
- #define **INS_VERIFY_PIN** 0x20
  INS byte: Verify PIN.

Functions

- void **cmd_write** (void)
  Write EEPROM. (Test command).
- void **cmd_read** (void)
  Read EEPROM. (Test command).
- void **cmd_led** (void)
  LED Effects.
- void **cmd_changeUnblockPIN** (void)
  Change or Unblock PIN.
- void **cmd_create** (void)
  Create File.
- void **cmd_delete** (void)
  Delete File.

- void **cmd_extAuth** (void)
  External Authentication.
- void **cmd_getChallenge** (void)
  Get Challenge.
- void **cmd_getResponse** (void)
  Get Response.
- void **cmd_intAuth** (void)
  Internal Authentication.
- void **cmd_readBinary** (void)
  Read Binary.
- void **cmd_select** (void)
  Select File.
- void **cmd_updateBinary** (void)
  Update Binary.
- void **cmd_verifyKeyPIN** (void)
  Verify Key/Verify PIN.

Variables

- **iu8 resplen**
  Valid data in response array. Invalid if zero.

6.5.1 Detailed Description

Command declarations.

The documentation of the commands is on the main page of the doxygen documentation.

Id:
commands.h,v 1.15 2002/12/22 15:42:55 m Exp

6.5.2 Define Documentation

6.5.2.1 #define CLA_PROP 0x80

CLA byte: Proprietary.

6.5.2.2 #define INS_CHANGE_PIN 0x24

INS byte: Change PIN.

6.5.2.3 #define INS_CREATE 0xE0

INS byte: Create File.

6.5.2.4 #define INS_DELETE 0xE4

INS byte: Delete File.

6.5.2.5 #define INS_EXTERNAL_AUTH 0x82

INS byte: External Authentication.

6.5.2.6 #define INS_GET_CHALLENGE 0x84

INS byte: Get Challenge.

6.5.2.7 #define INS_GET_RESPONSE 0xC0

INS byte: Get Response.

6.5.2.8 #define INS_INTERNAL_AUTH 0x88

INS byte: Internal Authentication.

6.5.2.9 #define INS_LED 0x06

INS byte: LED Effects.

6.5.2.10 #define INS_READ 0x04

INS byte: Read EEPROM.

6.5.2.11 #define INS_READ_BINARY 0xB0

INS byte: Read Binary.

6.5.2.12 #define INS_SELECT 0xA4

INS byte: Select File.

6.5.2.13 #define INS_UNBLOCK_PIN 0x2C

INS byte: Unblock PIN.

6.5.2.14 #define INS_UPDATE_BINARY 0xD6

INS byte: Update Binary.

6.5.2.15 #define INS_VERIFY_KEY 0x2A

INS byte: Verify Key.

6.5.2.16 #define INS_VERIFY_PIN 0x20

INS byte: Verify PIN.

6.5.2.17 #define INS_WRITE 0x02

INS byte: Write EEPROM.

6.5.3 Function Documentation

6.5.3.1 void cmd_changeUnblockPIN (void)

Change or Unblock PIN.

6.5.3.2 void cmd_create (void)

Create File.

6.5.3.3 void cmd_delete (void)

Delete File.

6.5.3.4 void cmd_extAuth (void)

External Authentication.

6.5.3.5 void cmd_getChallenge (void)

Get Challenge.

6.5.3.6 void cmd_getResponse (void)

Get Response.

6.5.3.7 void cmd_intAuth (void)

Internal Authentication.

6.5.3.8 void cmd_led (void)

LED Effects.

6.5.3.9 void cmd_read (void)

Read EEPROM. (Test command).

6.5.3.10 void cmd_readBinary (void)

Read Binary.

6.5.3.11 void cmd_select (void)

Select File.

6.5.3.12 void cmd_updateBinary (void)

Update Binary.

6.5.3.13 void cmd_verifyKeyPIN (void)

Verify Key/Verify PIN.

This function is a combined implementation of Verify Key and Verify PIN. This is done to save code size.

6.5.3.14 void cmd_write (void)

Write EEPROM. (Test command).

6.5.4 Variable Documentation

6.5.4.1 iu8 resplen

Valid data in response array. Invalid if zero.

6.6 config.h File Reference

Configuration settings.

Defines

- #define **CONF_IODATA_SIZE** 256
  Max. I/O data size.
- #define **CONF_WITH_FILESYSTEM** 1
  En-/disables file system.
- #define **CONF_WITH_RECORD_FILES** 0
  En-/disables support for fixed record files.

- #define **CONF_WITH_TRANSACTIONS** 1

  En-/disables transactions.

- #define **CONF_WITH_TESTCMDS** 1

  En-/disables test commands.

- #define **CONF_WITH_DELETECMD** 1

  En-/disables DELETE.

- #define **CONF_WITH_CREATECMD** 1

  En-/disables CREATE.

- #define **CONF_WITH_PINAUTH** 1

  En-/disables PIN authentication functions.

- #define **CONF_WITH_PINCMDS** 1

  En-/disables PIN commands.

- #define **CONF_WITH_KEYAUTH** 1

  En-/disables key authentication functions.

- #define **CONF_WITH_KEYCMDS** 1

  En-/disables key related commands.

- #define **CONF_WITH_I2CEEPROM** 1

  En-/disables support for the external I2C EEPROM.

- #define **CONF_WITH_DECRYPT** 1

  En-/disables support for decryption.

- #define **CONF_WITH_LOGGING** 1

  En-/disables support for logging.

- #define **CONF_WITH_RETCODEMGR** 0

  En-/disables return code manager.

- #define **CONF_WITH_FUNNY** 0

  En-/disables funny commands.

- #define **CONF_WITH_TRNG** 0

  En-/disables true random generator. **Highly experimental!**.

- #define **CONF_WITH_CRYPT_ALGO** 0

  Set crypto algorithm.

- #define **CARD_STATE_ADDR** 0x0000

EEPROM address of the state of the card.

- #define **ATR_LEN_ADDR** (CARD_STATE_ADDR+1)

  EEPROM address of the ATR length.

- #define **ATR_MAXLEN** 28

  Max. length of the ATR.

- #define **ATR_ADDR** (ATR_LEN_ADDR+1)

  EEPROM address of the ATR.

- #define **SERNUM_ADDR** (ATR_ADDR+ATR_MAXLEN)

  EEPROM address of the serial number of the card.

- #define **SERNUM_LEN** 8

  Length of the serial number.

- #define **RAND_STATE_ADDR** (SERNUM_ADDR+SERNUM_-LEN)

  EEPROM address of the state of the PRNG.

- #define **RAND_STATE_LEN** 8+8+16

  Length of the state of the PRNG. (Counter[8], State[8], Key[16]).

- #define **TRANSAC_STATE_ADDR** (RAND_STATE_-ADDR+RAND_STATE_LEN)

  EEPROM address of the transaction state.

- #define **TRANSAC_DATA_ADDR** (TRANSAC_STATE_-ADDR+1)

  EEPROM address of the transaction data.

- #define **TRANSAC_DATA_LEN** 19

  Length of the transaction data.

- #define **ADM_KEY_LEN** 16

  Length of the External Authentication key.

- #define **INT_KEY_LEN** 16

  Length of the Internal Authentication key.

- #define **PIN_LEN** 8

  Length of the PIN.

- #define **PUK_LEN** 8

  Length of the PUK.

- #define **DATA_END_1** (TRANSAC_DATA_ADDR+TRANSAC_-DATA_LEN)

  Dummy address to handle the ifdefs here.

- #define **FS_START_PTR_ADDR** DATA_END_1

  EEPROM address of pointer to the file system start.

- #define **ENDIAN_LITTLE**

  This is a little endian architecture.

- #define **EEPROM_SIZE** 0x200

  Size of the internal EEPROM.

- #define **RAM_SIZE** 0x200

  Size of the RAM.

- #define **CHIP** 0x01

  Chip ID.

- #define **ESIZ** 0x03

  External EEPROM ID.

- #define **ARCH** avr2

  AVR architecture. Needed for assembler.

6.6.1 Detailed Description

Configuration settings.

Id:

config.h,v 1.28 2003/05/10 17:26:58 m Exp

6.6.2 Define Documentation

6.6.2.1 #define ADM_KEY_LEN 16

Length of the External Authentication key.

6.6.2.2 #define ARCH avr2

AVR architecture. Needed for assembler.

6.6.2.3 #define ATR_ADDR (ATR_LEN_ADDR+1)

EEPROM address of the ATR.

6.6.2.4 #define ATR_LEN_ADDR (CARD_STATE_ADDR+1)

EEPROM address of the ATR length.

6.6.2.5 #define ATR_MAXLEN 28

Max. length of the ATR.

6.6.2.6 #define CARD_STATE_ADDR 0x0000

EEPROM address of the state of the card.

This is currently unused.

Warning: When using interrupts on the AVR, this byte may accidently be changed. See errata sheets of AVR.

6.6.2.7 #define CHIP 0x01

Chip ID.

6.6.2.8 #define CONF_IODATA_SIZE 256

Max. I/O data size.

6.6.2.9 #define CONF_WITH_CREATECMD 1

En-/disables CREATE.

6.6.2.10 #define CONF_WITH_CRYPT_ALGO 0

Set crypto algorithm.

- 0: TEA
  - 1: Skipjack
  - 2: NEWDES-SK

6.6.2.11 #define CONF_WITH_DECRYPT 1

En-/disables support for decryption.

If set to 1, support for decryption side of the cryptographic algorithms are included.

6.6.2.12 #define CONF_WITH_DELETECMD 1

En-/disables DELETE.

6.6.2.13 #define CONF_WITH_FILESYSTEM 1

En-/disables file system.

Todo:

Make this define something happen.

6.6.2.14 #define CONF_WITH_FUNNY 0

En-/disables funny commands.

If set to 1, funny commands are included.

6.6.2.15 #define CONF_WITH_I2CEEPROM 1

En-/disables support for the external I2C EEPROM.

6.6.2.16 #define CONF_WITH_KEYAUTH 1

En-/disables key authentication functions.

If set to 1, functions impelementing calculations with challenges are included. This is a requirement for CONF_WITH_KEY_CMDS.

6.6.2.17 #define CONF_WITH_KEYCMDS 1

En-/disables key related commands.

If set to 1, commands EXTERNAL KEYENTIFICATION, GET CHALLENGE, INTERNAL KEYENTIFICATION and VERIFY KEY are included.

6.6.2.18 #define CONF_WITH_LOGGING 1

En-/disables support for logging.

If set to 1, support for logging is included.

6.6.2.19 #define CONF_WITH_PINAUTH 1

En-/disables PIN authentication functions.

If set to 1, functions to set and test PIN and PUK are included. This is a requirement for CONF_WITH_PINCMDS.

6.6.2.20 #define CONF_WITH_PINCMDS 1

En-/disables PIN commands.

If set to 1, commands CHANGE PIN and VERIFY PIN are included.

6.6.2.21 #define CONF_WITH_RECORD_FILES 0

En-/disables support for fixed record files.

6.6.2.22 #define CONF_WITH_RETCODEMGR 0

En-/disables return code manager.

If set to 1, the return code manager is included. This increases code size on the AVR.

6.6.2.23 #define CONF_WITH_TESTCMDS 1

En-/disables test commands.

If set to 1, commands READ EEPROM and WRITE EEPROM are included.

6.6.2.24 #define CONF_WITH_TRANSACTIONS 1

En-/disables transactions.

6.6.2.25 #define CONF_WITH_TRNG 0

En-/disables true random generator. **Highly experimental!**.

If set to 1, a true random number generator is included. This is currently only for test purposes and highly experimental!

6.6.2.26 #define DATA_END_1 (TRANSAC_DATA_ADDR+TRANSAC_DATA_LEN)

Dummy address to handle the ifdefs here.

6.6.2.27 #define EEPROM_SIZE 0x200

Size of the internal EEPROM.

6.6.2.28 #define ENDIAN_LITTLE

This is a little endian architecture.

6.6.2.29 #define ESIZ 0x03

External EEPROM ID.

6.6.2.30 #define FS_START_PTR_ADDR DATA_END_1

EEPROM address of pointer to the file system start.

6.6.2.31 #define INT_KEY_LEN 16

Length of the Internal Authentication key.

6.6.2.32 #define PIN_LEN 8

Length of the PIN.

6.6.2.33 #define PUK_LEN 8

Length of the PUK.

6.6.2.34 #define RAM_SIZE 0x200

Size of the RAM.

6.6.2.35 #define RAND_STATE_ADDR (SERNUM_-ADDR+SERNUM_LEN)

EEPROM address of the state of the PRNG.

6.6.2.36 #define RAND_STATE_LEN 8+8+16

Length of the state of the PRNG. (Counter[8], State[8], Key[16]).

6.6.2.37 #define SERNUM_ADDR (ATR_ADDR+ATR_-MAXLEN)

EEPROM address of the serial number of the card.

6.6.2.38 #define SERNUM_LEN 8

Length of the serial number.

6.6.2.39 #define TRANSAC_DATA_ADDR (TRANSAC_STATE_-ADDR+1)

EEPROM address of the transaction data.

6.6.2.40 #define TRANSAC_DATA_LEN 19

Length of the transaction data.

6.6.2.41 #define TRANSAC_STATE_ADDR (RAND_STATE_-ADDR+RAND_STATE_LEN)

EEPROM address of the transaction state.

6.7 crypt.h File Reference

Crypto declarations.

#include <config.h>

#include <sw.h>

#include <string.h>

#include <types.h>

#include <tea.h>

Defines

- #define **CRYPT_ALGO_TEA** 0

  Algorithm ID: TEA.

- #define **CRYPT_ALGO_SKIPJACK** 1

  Algorithm ID: Skipjack.

- #define **CRYPT_ALGO_NEWDESSK** 2

  Algorithm ID: NEWDES-SK.

- #define **CRYPT_KEY_LEN** TEA_KEY_LEN

  Length of key in octets.

- #define **CRYPT_BLOCK_LEN** TEA_BLOCK_LEN

  Length of cipher block in octets.

- #define **crypt_enc**(v, k) tea_enc((**iu32***)(v),(**iu32***)(k))

  Single block encryption function.

- #define **crypt_dec**(v, k) tea_dec((**iu32***)(v),(**iu32***)(k))

  Single block decryption function.

6.7.1 Detailed Description

Crypto declarations.

Id:

crypt.h,v 1.1 2003/03/30 12:42:21 m Exp

6.7.2 Define Documentation

6.7.2.1 #define CRYPT_ALGO_NEWDESSK 2

Algorithm ID: NEWDES-SK.

6.7.2.2 #define CRYPT_ALGO_SKIPJACK 1

Algorithm ID: Skipjack.

6.7.2.3 #define CRYPT_ALGO_TEA 0

Algorithm ID: TEA.

6.7.2.4 #define CRYPT_BLOCK_LEN TEA_BLOCK_LEN

Length of cipher block in octets.

6.7.2.5 #define crypt_dec(v, k) tea_dec((iu32*)(v),(iu32*)(k))

Single block decryption function.

6.7.2.6 #define crypt_enc(v, k) tea_enc((iu32*)(v),(iu32*)(k))

Single block encryption function.

6.7.2.7 #define CRYPT_KEY_LEN TEA_KEY_LEN

Length of key in octets.

6.8 ctapi.c File Reference

Virtual CT-API library.

`#include <unistd.h>`

`#include <stdio.h>`

`#include <stdlib.h>`

`#include <string.h>`

`#include <sys/types.h>`

`#include <ctapi.h>`

`#include <main.h>`

`#include <sw.h>`

Functions

- **IS8 CT_forkIcc** (void)
- **IS8 CT_stopIcc** (void)
- **IS8 CT_ifdCmdResetCT** (**IU16** lenc, **IU8** *command, **IU16** *lenr, **IU8** *response)
- **IS8 CT_ifdCmdRequestICC** (**IU16** lenc, **IU8** *command, **IU16** *lenr, **IU8** *response)

- **IS8 CT_ifdCmdGetStatus** (**IU16** lenc, **IU8** *command, **IU16** *lenr, **IU8** *response)
- **IS8 CT_ifdCmdEjectICC** (**IU16** lenc, **IU8** *command, **IU16** *lenr, **IU8** *response)
- **IS8 CT_ifdCmdEepromFile** (**IU16** lenc, **IU8** *command, **IU16** *lenr, **IU8** *response)

  Parses EEPROM FILE command.

- **IS8 CT_ifdCmd** (**IU16** lenc, **IU8** *command, **IU16** *lenr, **IU8** *response)
- int **CT_t0** (**IU16** lenc, **IU8** *command, **IU16** *lenr, **IU8** *response)
- **IS8 CT_iccCmd** (**IU8** *dad, **IU8** *sad, **IU16** lenc, **IU8** *command, **IU16** *lenr, **IU8** *response)
- **IS8 CT_init** (**IU16** ctn, **IU16** pn)
- **IS8 CT_data** (**IU16** ctn, **IU8** *dad, **IU8** *sad, **IU16** lenc, **IU8** *command, **IU16** *lenr, **IU8** *response)
- **IS8 CT_close** (**IU16** ctn)

Variables

- **IS32 cthandle** = -1
- int **fd_fromCard** [2]
- int **fd_toCard** [2]
- pid_t **pidVirtuaCard** = 0
- int **loadEepromFile** = FALSE
- int **saveEepromFile** = FALSE
- char **eepromFile** [256]
- int **fdEepromFile**
- char **atr** [32]
- int **atrlen** = 0

6.8.1 Detailed Description

Virtual CT-API library.

Id:

ctapi.c,v 1.4 2002/12/24 13:29:57 m Exp

6.8.2 Function Documentation

6.8.2.1 IS8 CT_close (IU16 *ctn*)

6.8.2.2 IS8 CT_data (IU16 *ctn*, IU8 * *dad*, IU8 * *sad*, IU16 *lenc*, IU8 * *command*, IU16 * *lenr*, IU8 * *response*)

6.8.2.3 IS8 CT_forkIcc (void)

6.8.2.4 IS8 CT_iccCmd (IU8 * *dad*, IU8 * *sad*, IU16 *lenc*, IU8 * *command*, IU16 * *lenr*, IU8 * *response*)

6.8.2.5 IS8 CT_ifdCmd (IU16 *lenc*, IU8 * *command*, IU16 * *lenr*, IU8 * *response*)

6.8.2.6 IS8 CT_ifdCmdEepromFile (IU16 *lenc*, IU8 * *command*, IU16 * *lenr*, IU8 * *response*)

Parses EEPROM FILE command.

- CLA: '20'
  - INS: 'E0'
  - P1:
    - bit 1: Save EEPROM to file at eject or reset.
    - bit 2: Load EEPROM from file at reset.
  - P2: RFU
  - Lc: Length of file name
  - Data: File name (possibly including path) without null-termination

6.8.2.7 IS8 CT_ifdCmdEjectICC (IU16 *lenc*, IU8 * *command*, IU16 * *lenr*, IU8 * *response*)

6.8.2.8 IS8 CT_ifdCmdGetStatus (IU16 *lenc*, IU8 * *command*, IU16 * *lenr*, IU8 * *response*)

6.8.2.9 IS8 CT_ifdCmdRequestICC (IU16 *lenc*, IU8 * *command*, IU16 * *lenr*, IU8 * *response*)

6.8.2.10 IS8 CT_ifdCmdResetCT (IU16 *lenc*, IU8 * *command*, IU16 * *lenr*, IU8 * *response*)

6.8.2.11 IS8 CT_init (IU16 *ctn*, IU16 *pn*)

6.8.2.12 IS8 CT_stopIcc (void)

6.8.2.13 int CT_t0 (IU16 *lenc*, IU8 * *command*, IU16 * *lenr*, IU8 * *response*)

6.8.3 Variable Documentation

6.8.3.1 char atr[32]

6.8.3.2 int atrlen = 0

6.8.3.3 IS32 cthandle = -1

6.8.3.4 char eepromFile[256]

6.8.3.5 int fd_fromCard[2]

6.8.3.6 int fd_toCard[2]

6.8.3.7 int fdEepromFile

6.8.3.8 int loadEepromFile = FALSE

6.8.3.9 pid_t pidVirtuaCard = 0

6.8.3.10 int saveEepromFile = FALSE

6.9 ctapi.h File Reference

Declarations for virtual CT-API library.

`#include <types.h>`

Defines

- #define **CTAPI_AD_ICC1** 0
- #define **CTAPI_AD_CT** 1
- #define **CTAPI_AD_HOST** 2
- #define **CTAPI_RV_OK** 0
- #define **CTAPI_RV_ERR_INVALID** -1
- #define **CTAPI_RV_ERR_CT** -8
- #define **CTAPI_RV_ERR_TRANS** -10
- #define **CTAPI_RV_ERR_MEMORY** -11
- #define **CTAPI_RV_ERR_HOST** -127
- #define **CTAPI_RV_ERR_HTSI** -128
- #define **CTAPI_PORT** 1
- #define **CTAPI_CLA** 0x20
- #define **CTAPI_INS_RESET_CT** 0x11
- #define **CTAPI_INS_REQUEST_ICC** 0x12
- #define **CTAPI_INS_GET_STATUS** 0x13
- #define **CTAPI_INS_EJECT_ICC** 0x14
- #define **CTAPI_INS_EEPROM** 0xE0

Typedefs

- typedef unsigned char **IU8**
- typedef signed char **IS8**
- typedef unsigned short **IU16**
- typedef signed short **IS16**
- typedef unsigned long **IU32**
- typedef signed long **IS32**

Functions

- **IS8 CT_forkIcc** (void)
- **IS8 CT_stopIcc** (void)
- **IS8 CT_ifdCmdEepromFile** (**IU16** lenc, **IU8** *command, **IU16** *lenr, **IU8** *response)

    Parses EEPROM FILE command.

- **IS8 CT_ifdCmdResetCT** (**IU16** lenc, **IU8** *command, **IU16** *lenr, **IU8** *response)
- **IS8 CT_ifdCmdRequestICC** (**IU16** lenc, **IU8** *command, **IU16** *lenr, **IU8** *response)
- **IS8 CT_ifdCmdGetStatus** (**IU16** lenc, **IU8** *command, **IU16** *lenr, **IU8** *response)
- **IS8 CT_ifdCmdEjectICC** (**IU16** lenc, **IU8** *command, **IU16** *lenr, **IU8** *response)
- **IS8 CT_ifdCmd** (**IU16** lenc, **IU8** *command, **IU16** *lenr, **IU8** *response)
- **IS8 CT_iccCmd** (**IU8** *dad, **IU8** *sad, **IU16** lenc, **IU8** *command, **IU16** *lenr, **IU8** *response)
- **IS8 CT_init** (**IU16** ctn, **IU16** pn)
- **IS8 CT_data** (**IU16** ctn, **IU8** *dad, **IU8** *sad, **IU16** lenc, **IU8** *command, **IU16** *lenr, **IU8** *response)
- **IS8 CT_close** (**IU16** ctn)

Variables

- int **fd_fromCard** [2]
- int **fd_toCard** [2]
- int **loadEepromFile**
- int **saveEepromFile**
- char **eepromFile** [256]
- int **fdEepromFile**

6.9.1 Detailed Description

Declarations for virtual CT-API library.

Id:
ctapi.h,v 1.2 2002/12/22 23:16:15 m Exp

6.9.2 Define Documentation

6.9.2.1 #define CTAPI_AD_CT 1

6.9.2.2 #define CTAPI_AD_HOST 2

6.9.2.3 #define CTAPI_AD_ICC1 0

6.9.2.4 #define CTAPI_CLA 0x20

6.9.2.5 #define CTAPI_INS_EEPROM 0xE0

6.9.2.6 #define CTAPI_INS_EJECT_ICC 0x14

6.9.2.7 #define CTAPI_INS_GET_STATUS 0x13

6.9.2.8 #define CTAPI_INS_REQUEST_ICC 0x12

6.9.2.9 #define CTAPI_INS_RESET_CT 0x11

6.9.2.10 #define CTAPI_PORT 1

6.9.2.11 #define CTAPI_RV_ERR_CT -8

6.9.2.12 #define CTAPI_RV_ERR_HOST -127

6.9.2.13 #define CTAPI_RV_ERR_HTSI -128

6.9.2.14 #define CTAPI_RV_ERR_INVALID -1

6.9.2.15 #define CTAPI_RV_ERR_MEMORY -11

6.9.2.16 #define CTAPI_RV_ERR_TRANS -10

6.9.2.17 #define CTAPI_RV_OK 0

6.9.3 Typedef Documentation

6.9.3.1 typedef signed short IS16

6.9.3.2 typedef signed long IS32

6.9.3.3 typedef signed char IS8

6.9.3.4 typedef unsigned short IU16

6.9.3.5 typedef unsigned long IU32

6.9.3.6 typedef unsigned char IU8

6.9.4 Function Documentation

6.9.4.1 IS8 CT_close (IU16 *ctn*)

6.9.4.2 IS8 CT_data (IU16 *ctn*, IU8 * *dad*, IU8 * *sad*, IU16 *lenc*, IU8 * *command*, IU16 * *lenr*, IU8 * *response*)

6.9.4.3 IS8 CT_forkIcc (void)

6.9.4.4 IS8 CT_iccCmd (IU8 * *dad*, IU8 * *sad*, IU16 *lenc*, IU8 * *command*, IU16 * *lenr*, IU8 * *response*)

6.9.4.5 IS8 CT_ifdCmd (IU16 *lenc*, IU8 * *command*, IU16 * *lenr*, IU8 * *response*)

6.9.4.6 IS8 CT_ifdCmdEepromFile (IU16 *lenc*, IU8 * *command*, IU16 * *lenr*, IU8 * *response*)

Parses EEPROM FILE command.

- CLA: '20'
  - INS: 'E0'
  - P1:
    - bit 1: Save EEPROM to file at eject or reset.
    - bit 2: Load EEPROM from file at reset.
  - P2: RFU

- Lc: Length of file name
- Data: File name (possibly including path) without null-termination

6.9.4.7 IS8 CT_ifdCmdEjectICC (IU16 *lenc*, IU8 * *command*, IU16 * *lenr*, IU8 * *response*)

6.9.4.8 IS8 CT_ifdCmdGetStatus (IU16 *lenc*, IU8 * *command*, IU16 * *lenr*, IU8 * *response*)

6.9.4.9 IS8 CT_ifdCmdRequestICC (IU16 *lenc*, IU8 * *command*, IU16 * *lenr*, IU8 * *response*)

6.9.4.10 IS8 CT_ifdCmdResetCT (IU16 *lenc*, IU8 * *command*, IU16 * *lenr*, IU8 * *response*)

6.9.4.11 IS8 CT_init (IU16 *ctn*, IU16 *pn*)

6.9.4.12 IS8 CT_stopIcc (void)

6.9.5 Variable Documentation

6.9.5.1 char eepromFile[256]

6.9.5.2 int fd_fromCard[2]

6.9.5.3 int fd_toCard[2]

6.9.5.4 int fdEepromFile

6.9.5.5 int loadEepromFile

6.9.5.6 int saveEepromFile

6.10 fs.c File Reference

File system.

`#include <config.h>`

`#include <sw.h>`

`#include <fs.h>`

```
#include <hal.h>
#include <log.h>
#include <string.h>
#include <transaction.h>
#include <types.h>
```

Functions

- **bool fs_init** (void)

  Initialize file system variables.

- **bool fs_seek** (**iu16** fid, **S_FSTREAM** *fs, **S_FINFO** *fi, **iu8** type)

  Look for a file in a stream.

- **bool fs_seekEnd** (**S_FSTREAM** *fs)

  Look for the end of files in the stream.

- **bool fs_getData** (**S_FPATH** *fp, **S_FSTREAM** *fs, **S_FINFO** *fi)

  Get stream and info for a file specified as a path.

- **bool fs_getFPath** (**iu16** fid, **S_FPATH** *fp, **S_FSTREAM** *fs, **S_-FINFO** *fi)

  Get path, stream and info for a file specified as a FID.

- **bool fs_delete** (**iu16** fid)

  Delete file specified as fid.

- **bool fs_create** (**S_FINFO** *fi)

  The function creates a file.

Variables

- **iu16 fsstart**

  Start address of the MF file header.

- **iu16 fssize**

  Size of the content area of the MF.

- **S_FPATH selected**

  Path specified of the currently selected file.

6.10.1 Detailed Description

File system.

Id:

fs.c,v 1.23 2002/12/22 15:42:55 m Exp

6.10.2 Function Documentation

6.10.2.1 bool fs_create (S_FINFO * *fi*)

The function creates a file.

The functions creates a file with the data in *fi*. The current file must be a MF or DF. In case the current file is a DF, the file, which should be created, can't be a DF.

Parameters:

fi Pointer to S_FINFO, which contains the data of the file to create.

Return values:

TRUE on success.

FALSE on failure. Error code given in sw.

6.10.2.2 bool fs_delete (iu16 *fid*)

Delete file specified as fid.

The function looks for the fid in the current file, which must be a MF or DF. The file, which should be deleted must be the last one in the current MF or DF.

Parameters:

fid ID of the file to delete.

Return values:

TRUE on success.

FALSE on failure. Error code given in sw.

6.10.2.3 bool fs_getData (S_FPATH * *fp*, S_FSTREAM * *fs*, S_FINFO * *fi*)

Get stream and info for a file specified as a path.

Parameters:

fp Pointer to S_FPATH, which is used to supply the information about the file wanted.

fs Pointer to S_FSTREAM, which will be filled with data on success.

fi Pointer to S_FINFO, which will be filled with data on success.

Return values:

TRUE on success.

FALSE on failure. Error code given in sw.

6.10.2.4 bool fs_getFPath (iu16 *fid*, S_FPATH ∗ *fp*, S_FSTREAM ∗ *fs*, S_FINFO ∗ *fi*)

Get path, stream and info for a file specified as a FID.

The file is searched for in the order described in the SELECT FILE documentation on the main page.

Parameters:

fid File ID to look for.

fp Pointer to S_FPATH, which will be filled with data on success.

fs Pointer to S_FSTREAM, which will be filled with data on success. This can be NULL when no data is wanted.

fi Pointer to S_FINFO, which will be filled with data on success. This can be NULL when no data is wanted.

Return values:

TRUE on success.

FALSE on failure. Error code given in sw.

6.10.2.5 bool fs_init (void)

Initialize file system variables.

This looks e.g. for the start of the file system and it's size. It must be called after each reset.

Return values:

TRUE on success.

FALSE on failure. Error code given in sw.

6.10.2.6 bool fs_seek (iu16 *fid*, S_FSTREAM ∗ *fs*, S_FINFO ∗ *fi*, iu8 *type*)

Look for a file in a stream.

Parameters:

fid File ID to look for.

fs Pointer to S_FSTREAM, where the FID is searched for. It returns on success a stream on the file found.

fi Pointer to S_FINFO, which will be filled with data on success.

type Type of file wanted.

Return values:

TRUE on success.

FALSE on failure. Error code given in sw.

Todo:

There is probably a potential for simplification, when the file type is not handled so rigid.

6.10.2.7 bool fs_seekEnd (S_FSTREAM * *fs*)

Look for the end of files in the stream.

Parameters:

fs Pointer to S_FSTREAM, which supplies a directory stream and which will be used to return the end position on success.

Return values:

TRUE on success.

FALSE on failure. Error code given in sw.

6.10.3 Variable Documentation

6.10.3.1 iu16 fssize

Size of the content area of the MF.

6.10.3.2 iu16 fsstart

Start address of the MF file header.

6.10.3.3 S_FPATH selected

Path specified of the currently selected file.

Todo:

Saving here a S_FINFO and/or S_FSTREAM of the selected file might reduce the code size.

6.11 fs.h File Reference

File system declarations.

`#include <types.h>`

`#include <fstream.h>`

Data Structures

- struct **s_finfo**
  File info struct. This is directly written as a file header in the file system.
- struct **s_fpath**
  File path specifier.

Defines

- #define **FS_TYPE_DF** 0x38
  File type DF.
- #define **FS_TYPE_EF** 0x00
  File type EF.
- #define **FS_TYPE_EF_TRANSP** 0x00
  File type transparent EF.
- #define **FS_TYPE_EF_FIXED** 0x01
  File type fixed record EF.
- #define **FS_HEADER_SIZE** sizeof(**S_FINFO**)
  Size of the file header, which is currently the same as S_FINFO.

Typedefs

- typedef **s_fpath S_FPATH**
  File path specifier.
- typedef **s_finfo S_FINFO**
  File info struct. This is directly written as a file header in the file system.

Functions

- **bool fs_init** (void)
  Initialize file system variables.
- **bool fs_seek** (**iu16** fid, **S_FSTREAM** ∗fs, **S_FINFO** ∗fi, **iu8** type)
  Look for a file in a stream.
- **bool fs_seekEnd** (**S_FSTREAM** ∗fs)
  Look for the end of files in the stream.

- **bool fs_getData** (**S_FPATH** *fp, **S_FSTREAM** *fs, **S_FINFO** *fi)

  Get stream and info for a file specified as a path.

- **bool fs_getFPath** (**iu16** fid, **S_FPATH** *fp, **S_FSTREAM** *fs, **S_-FINFO** *fi)

  Get path, stream and info for a file specified as a FID.

- **bool fs_delete** (**iu16** fid)

  Delete file specified as fid.

- **bool fs_create** (**S_FINFO** *fi)

  The function creates a file.

Variables

- **S_FPATH selected**

  Path specified of the currently selected file.

6.11.1 Detailed Description

File system declarations.

SOSSE implements a very simple file system. Each file starts with following header:

- 2 Byte: Content size
- 2 Byte: ID
- 1 Byte: Type
- 1 Byte: Access conditions

Directly after that the content area is located. If the file is a DF, this content area is directly used to add header/content sequences.

After the content area of a file, a content size of 0 must be writen as an end of data marker. This means the data capacity of a MF or DF is the total content size of it minus 2.

Currently the file system support only one level of DFs, i.e. the maximum depth is MF:DF:EF.

There are probably a lot optimization possibilities in this file system, but I think this is not bad for my first designed and implemented file system.

Todo:

Here are certainly some optimizations possible to reduce code size.

Id:

fs.h,v 1.14 2003/05/10 17:26:58 m Exp

6.11.2 Define Documentation

6.11.2.1 #define FS_HEADER_SIZE sizeof(S_FINFO)

Size of the file header, which is currently the same as S_FINFO.

6.11.2.2 #define FS_TYPE_DF 0x38

File type DF.

6.11.2.3 #define FS_TYPE_EF 0x00

File type EF.

6.11.2.4 #define FS_TYPE_EF_FIXED 0x01

File type fixed record EF.

6.11.2.5 #define FS_TYPE_EF_TRANSP 0x00

File type transparent EF.

6.11.3 Typedef Documentation

6.11.3.1 typedef struct s_finfo S_FINFO

File info struct. This is directly written as a file header in the file system.

6.11.3.2 typedef struct s_fpath S_FPATH

File path specifier.

Todo:

There is probably a potential for simplification, when the file type (DF/EF) is not handled so rigid.

6.11.4 Function Documentation

6.11.4.1 bool fs_create (S_FINFO * *fi*)

The function creates a file.

The functions creates a file with the data in *fi*. The current file must be a MF or DF. In case the current file is a DF, the file, which should be created, can't be a DF.

Parameters:

fi Pointer to S_FINFO, which contains the data of the file to create.

Return values:

TRUE on success.

FALSE on failure. Error code given in sw.

6.11.4.2 bool fs_delete (iu16 *fid*)

Delete file specified as fid.

The function looks for the fid in the current file, which must be a MF or DF. The file, which should be deleted must be the last one in the current MF or DF.

Parameters:

fid ID of the file to delete.

Return values:

TRUE on success.

FALSE on failure. Error code given in sw.

6.11.4.3 bool fs_getData (S_FPATH * *fp*, S_FSTREAM * *fs*, S_FINFO * *fi*)

Get stream and info for a file specified as a path.

Parameters:

fp Pointer to S_FPATH, which is used to supply the information about the file wanted.

fs Pointer to S_FSTREAM, which will be filled with data on success.

fi Pointer to S_FINFO, which will be filled with data on success.

Return values:

TRUE on success.

FALSE on failure. Error code given in sw.

6.11.4.4 bool fs_getFPath (iu16 *fid*, S_FPATH * *fp*, S_FSTREAM * *fs*, S_FINFO * *fi*)

Get path, stream and info for a file specified as a FID.

The file is searched for in the order described in the SELECT FILE documentation on the main page.

Parameters:

fid File ID to look for.

fp Pointer to S_FPATH, which will be filled with data on success.

fs Pointer to S_FSTREAM, which will be filled with data on success. This can be NULL when no data is wanted.

fi Pointer to S_FINFO, which will be filled with data on success. This can be NULL when no data is wanted.

Return values:

TRUE on success.

FALSE on failure. Error code given in sw.

6.11.4.5 bool fs_init (void)

Initialize file system variables.

This looks e.g. for the start of the file system and it's size. It must be called after each reset.

Return values:

TRUE on success.

FALSE on failure. Error code given in sw.

6.11.4.6 bool fs_seek (iu16 *fid*, S_FSTREAM * *fs*, S_FINFO * *fi*, iu8 *type*)

Look for a file in a stream.

Parameters:

fid File ID to look for.

fs Pointer to S_FSTREAM, where the FID is searched for. It returns on success a stream on the file found.

fi Pointer to S_FINFO, which will be filled with data on success.

type Type of file wanted.

Return values:

TRUE on success.

FALSE on failure. Error code given in sw.

Todo:

There is probably a potential for simplification, when the file type is not handled so rigid.

6.11.4.7 bool fs_seekEnd (S_FSTREAM * *fs*)

Look for the end of files in the stream.

Parameters:

fs Pointer to S_FSTREAM, which supplies a directory stream and which will be used to return the end position on success.

Return values:

TRUE on success.

FALSE on failure. Error code given in sw.

6.11.5 Variable Documentation

6.11.5.1 S_FPATH selected

Path specified of the currently selected file.

Todo:
Saving here a S_FINFO and/or S_FSTREAM of the selected file might reduce the code size.

6.12 fstest.c File Reference

Test program for file system functions.

`#include <stdio.h>`

`#include <auth.h>`

`#include <fs.h>`

`#include <fstream.h>`

`#include <hal.h>`

Functions

- int **main** (int argc, char *argv[ ])

Variables

- **iu16 fsstart**
  Start address of the MF file header.
- **iu16 fssize**
  Size of the content area of the MF.
- **iu8 authstate**
  Variable indicating current authentication state.

6.12.1 Detailed Description

Test program for file system functions.

Id:
fstest.c,v 1.11 2002/12/22 15:42:55 m Exp

6.12.2 Function Documentation

6.12.2.1 int main (int *argc*, char * *argv*[ ])

6.12.3 Variable Documentation

6.12.3.1 iu8 authstate ()

Variable indicating current authentication state.

Single bit flags indicate, which authorization process has been successful. The flag is cleared, at the reset of the card and after unsuccessful authorization processes.

Valid flags are:

- AUTH_FLAG_PIN
- AUTH_FLAG_KEY

6.12.3.2 iu16 fssize

Size of the content area of the MF.

6.12.3.3 iu16 fsstart

Start address of the MF file header.

6.13 fstream.c File Reference

File stream functions.

`#include <sw.h>`

`#include <fstream.h>`

`#include <hal.h>`

`#include <types.h>`

Functions

- **bool fstream_read** (**S_FSTREAM** ∗fs, **iu8** ∗dst, **iu8** len)

  Read data from a stream.

- **bool fstream_write** (**S_FSTREAM** ∗fs, **iu8** ∗src, **iu8** len)

  Write data to a stream.

- **bool fstream_test** (**S_FSTREAM** ∗fs, **iu16** len)

  This function tests, if enough space is available to read or write len *bytes.*

6.13.1 Detailed Description

File stream functions.

Id:

fstream.c,v 1.11 2002/12/22 15:42:55 m Exp

6.13.2 Function Documentation

6.13.2.1 bool fstream_read (S_FSTREAM * *fs*, iu8 * *dst*, iu8 *len*)

Read data from a stream.

Either this functions reads all data or nothing.

Parameters:

fs Pointer to S_FSTREAM, on which it should be operated.

dst Pointer to the data destination.

len Length of the data

Return values:

TRUE on success.

FALSE on failure. Error code given in sw.

6.13.2.2 bool fstream_test (S_FSTREAM * *fs*, iu16 *len*)

This function tests, if enough space is available to read or write *len* bytes.

Parameters:

fs Pointer to S_FSTREAM, on which it should be operated.

len Length of the data

Return values:

TRUE on success.

FALSE on failure. Error code given in sw.

6.13.2.3 bool fstream_write (S_FSTREAM * *fs*, iu8 * *src*, iu8 *len*)

Write data to a stream.

This functions tries to write all or nothing.

Parameters:

fs Pointer to S_FSTREAM, on which it should be operated.

src Pointer to data source.

len Length of the data

Return values:

TRUE on success.

FALSE on failure. Error code given in sw.

6.14 fstream.h File Reference

File stream declarations.

`#include <types.h>`

Data Structures

- struct **s_fstream**
  Stream struct.

Typedefs

- typedef **s_fstream S_FSTREAM**
  Stream struct.

Functions

- **bool fstream_read** (**S_FSTREAM** *fs, **iu8** *dst, **iu8** len)
  Read data from a stream.
- **bool fstream_write** (**S_FSTREAM** *fs, **iu8** *src, **iu8** len)
  Write data to a stream.
- **bool fstream_test** (**S_FSTREAM** *fs, **iu16** len)
  This function tests, if enough space is available to read or write len *bytes.*

6.14.1 Detailed Description

File stream declarations.

These streams are used to pass more easily EEPROM areas around and to reduce the chances of overwriting unwanted EEPROM areas, e.g. by overflowing the data area.

This is a very slim version of the streams implemented in cryptlib from Peter Gutmann.

Id:
fstream.h,v 1.8 2002/12/22 15:42:55 m Exp

6.14.2 Typedef Documentation

6.14.2.1 typedef struct s_fstream S_FSTREAM

Stream struct.

6.14.3 Function Documentation

6.14.3.1 bool fstream_read (S_FSTREAM * *fs*, iu8 * *dst*, iu8 *len*)

Read data from a stream.

Either this functions reads all data or nothing.

Parameters:

fs Pointer to S_FSTREAM, on which it should be operated.

dst Pointer to the data destination.

len Length of the data

Return values:

TRUE on success.

FALSE on failure. Error code given in sw.

6.14.3.2 bool fstream_test (S_FSTREAM * *fs*, iu16 *len*)

This function tests, if enough space is available to read or write *len* bytes.

Parameters:

fs Pointer to S_FSTREAM, on which it should be operated.

len Length of the data

Return values:

TRUE on success.

FALSE on failure. Error code given in sw.

6.14.3.3 bool fstream_write (S_FSTREAM * *fs*, iu8 * *src*, iu8 *len*)

Write data to a stream.

This functions tries to write all or nothing.

Parameters:

fs Pointer to S_FSTREAM, on which it should be operated.

src Pointer to data source.

len Length of the data

Return values:

TRUE on success.

FALSE on failure. Error code given in sw.

6.15 hal-emu.c File Reference

HAL emulation for Linux.

`#include <config.h>`

`#include <sio/sio.h>`

`#include <stdio.h>`

`#include <stdlib.h>`

`#include <string.h>`

`#include <crypt.h>`

`#include <sw.h>`

`#include <types.h>`

`#include <sys/stat.h>`

`#include <fcntl.h>`

`#include <unistd.h>`

`#include <errno.h>`

Defines

- #define **PORT** "/dev/ttyS0"

Functions

- void **hal_init** (void)

  Initializes the HAL.

- void **hal_destroy** (void)
- **bool hal_eeprom_read** (**iu8** *dst, **iu16** src, **iu8** len)

  Read data from EEPROM.

- **bool hal_eeprom_write** (**iu16** dst, **iu8** *src, **iu8** len)

  Write data to EEPROM.

- void **hal_io_sendByteT0** (**iu8** b)

  Send a byte with T=0 error detection and recovery.

- **iu8 hal_io_recByteT0** (void)

  Receive a byte with T=0 error detection and recovery.

- **bool hal_rnd_getBlock** (**iu8** *dst)

  Returns 8 random bytes.

Variables

- **iu8 eeprom** [1 *1024]
- int **halsend**

6.15.1 Detailed Description

HAL emulation for Linux.

Id:
hal-emu.c,v 1.18 2003/03/30 12:42:21 m Exp

6.15.2 Define Documentation

6.15.2.1 #define PORT "/dev/ttyS0"

6.15.3 Function Documentation

6.15.3.1 void hal_destroy (void)

6.15.3.2 bool hal_eeprom_read (iu8 * *dst*, iu16 *src*, iu8 *len*)

Read data from EEPROM.

The internal EEPROM begins at address 0, the external EEPROM is located subsequently.

Parameters:
dst Pointer to destination area.

src EEPROM source address.

len Length of data in bytes.

Return values:
TRUE on success.

FALSE on failure. Error code given in sw.

6.15.3.3 bool hal_eeprom_write (iu16 *dst*, iu8 * *src*, iu8 *len*)

Write data to EEPROM.

The internal EEPROM begins at address 0, the external EEPROM is located subsequently.

Parameters:
dst EEPROM destination address

src Pointer to source area.

len Length of data in bytes.

Return values:

TRUE on success.

FALSE on failure. Error code given in sw.

6.15.3.4 void hal_init (void)

Initializes the HAL.

Must be called after each reset.

6.15.3.5 iu8 hal_io_recByteT0 (void)

Receive a byte with T=0 error detection and recovery.

Currently only 9600 bps at 3.58MHz with direct convention is supported.

Returns:

This function returns a received byte. There can't be an error, because T=0 tries endlessly to receive a characters. (And if there is really an error, what should SOSSE do with it anyway?)

6.15.3.6 void hal_io_sendByteT0 (iu8 *b*)

Send a byte with T=0 error detection and recovery.

Currently only 9600 bps at 3.58MHz with direct convention is supported.

Parameters:

b Byte to send.

6.15.3.7 bool hal_rnd_getBlock (iu8 * *dst*)

Returns 8 random bytes.

The current AVR implementation does not produce real random numbers, but pseudo random numbers. These are produced by a X9.17 generator modified to use a counter instead of a timer. The state of this PRNG is held at RAND_-STATE_ADDR. The start value (at least the key) at this address should be unique for every card.

Parameters:

dst Pointer to the destination, where the 8 random bytes should be written.

Return values:

TRUE on success.

FALSE on failure. Error code given in sw.

Note:

There is still a simpler PRNG in the code, which uses 16 bytes RAM and 126 bytes ROM less than the X9.17 generator.

6.15.4 Variable Documentation

6.15.4.1 iu8 eeprom[1*1024]

6.15.4.2 int halsend

6.16 hal.c File Reference

HAL (Hardware Abstraction Layer).

```
#include <config.h>
#include <crypt.h>
#include <hal.h>
#include <avr/eeprom.h>
#include <avr/io.h>
#include <log.h>
#include <string.h>
#include <tools.h>
#include <types.h>
```

Functions

- void **hal_init** (void)

  Initializes the HAL.

- **iu8 xeread** (**iu16** addr)
- void **xewrt** (**iu16** addr, **iu8** b)
- **bool hal_eeprom_read** (**iu8** *dst, **iu16** src, **iu8** len)

  Read data from EEPROM.

- **bool hal_eeprom_write** (**iu16** dst, **iu8** *src, **iu8** len)

  Write data to EEPROM.

- void **sendbytet0** (**iu8** b)
- **iu8 recbytet0** (void)
- void **hal_io_sendByteT0** (**iu8** b)

  Send a byte with T=0 error detection and recovery.

- **iu8 hal_io_recByteT0** (void)

  Receive a byte with T=0 error detection and recovery.

- **bool hal_rnd_getBlock** (**iu8** *dst)

  Returns 8 random bytes.

6.16.1 Detailed Description

HAL (Hardware Abstraction Layer).

Id:
hal.c,v 1.22 2003/03/30 12:42:21 m Exp

6.16.2 Function Documentation

6.16.2.1 bool hal_eeprom_read (iu8 * *dst*, iu16 *src*, iu8 *len*)

Read data from EEPROM.

The internal EEPROM begins at address 0, the external EEPROM is located subsequently.

Parameters:

dst Pointer to destination area.

src EEPROM source address.

len Length of data in bytes.

Return values:

TRUE on success.

FALSE on failure. Error code given in sw.

6.16.2.2 bool hal_eeprom_write (iu16 *dst*, iu8 * *src*, iu8 *len*)

Write data to EEPROM.

The internal EEPROM begins at address 0, the external EEPROM is located subsequently.

Parameters:

dst EEPROM destination address

src Pointer to source area.

len Length of data in bytes.

Return values:

TRUE on success.

FALSE on failure. Error code given in sw.

6.16.2.3 void hal_init (void)

Initializes the HAL.

Must be called after each reset.

6.16.2.4 iu8 hal_io_recByteT0 (void)

Receive a byte with T=0 error detection and recovery.

Currently only 9600 bps at 3.58MHz with direct convention is supported.

Returns:
This function returns a received byte. There can't be an error, because T=0 tries endlessly to receive a characters. (And if there is really an error, what should SOSSE do with it anyway?)

6.16.2.5 void hal_io_sendByteT0 (iu8 *b*)

Send a byte with T=0 error detection and recovery.

Currently only 9600 bps at 3.58MHz with direct convention is supported.

Parameters:
b Byte to send.

6.16.2.6 bool hal_rnd_getBlock (iu8 * *dst*)

Returns 8 random bytes.

The current AVR implementation does not produce real random numbers, but pseudo random numbers. These are produced by a X9.17 generator modified to use a counter instead of a timer. The state of this PRNG is held at RAND_-STATE_ADDR. The start value (at least the key) at this address should be unique for every card.

Parameters:
dst Pointer to the destination, where the 8 random bytes should be written.

Return values:
TRUE on success.

FALSE on failure. Error code given in sw.

Note:
There is still a simpler PRNG in the code, which uses 16 bytes RAM and 126 bytes ROM less than the X9.17 generator.

6.16.2.7 iu8 recbytet0 (void)

6.16.2.8 void sendbytet0 (iu8 *b*)

6.16.2.9 iu8 xeread (iu16 *addr*)

6.16.2.10 void xewrt (iu16 *addr*, iu8 *b*)

6.17 hal.h File Reference

HAL (Hardware Abstraction Layer) declarations.

`#include <types.h>`

Functions

- void **hal_init** (void)
  Initializes the HAL.
- **bool hal_eeprom_read** (**iu8** ∗dst, **iu16** src, **iu8** len)
  Read data from EEPROM.
- **bool hal_eeprom_write** (**iu16** dst, **iu8** ∗src, **iu8** len)
  Write data to EEPROM.
- void **hal_io_sendByteT0** (**iu8** b)
  Send a byte with T=0 error detection and recovery.
- **iu8 hal_io_recByteT0** (void)
  Receive a byte with T=0 error detection and recovery.
- **bool hal_rnd_addEntropy** (void)
  Adds entropy to the random number generator state.
- **bool hal_rnd_getBlock** (**iu8** ∗dst)
  Returns 8 random bytes.
- **bool hal_led** (char set)
  Sets the LEDs on Jupiter 2 cards.

6.17.1 Detailed Description

HAL (Hardware Abstraction Layer) declarations.

Id:
hal.h,v 1.15 2002/12/22 15:42:55 m Exp

6.17.2 Function Documentation

6.17.2.1 bool hal_eeprom_read (iu8 * *dst*, iu16 *src*, iu8 *len*)

Read data from EEPROM.

The internal EEPROM begins at address 0, the external EEPROM is located subsequently.

Parameters:

dst Pointer to destination area.

src EEPROM source address.

len Length of data in bytes.

Return values:

TRUE on success.

FALSE on failure. Error code given in sw.

6.17.2.2 bool hal_eeprom_write (iu16 *dst*, iu8 * *src*, iu8 *len*)

Write data to EEPROM.

The internal EEPROM begins at address 0, the external EEPROM is located subsequently.

Parameters:

dst EEPROM destination address

src Pointer to source area.

len Length of data in bytes.

Return values:

TRUE on success.

FALSE on failure. Error code given in sw.

6.17.2.3 void hal_init (void)

Initializes the HAL.

Must be called after each reset.

6.17.2.4 iu8 hal_io_recByteT0 (void)

Receive a byte with T=0 error detection and recovery.

Currently only 9600 bps at 3.58MHz with direct convention is supported.

Returns:

This function returns a received byte. There can't be an error, because T=0 tries endlessly to receive a characters. (And if there is really an error, what should SOSSE do with it anyway?)

6.17.2.5 void hal_io_sendByteT0 (iu8 *b*)

Send a byte with T=0 error detection and recovery.

Currently only 9600 bps at 3.58MHz with direct convention is supported.

Parameters:

b Byte to send.

6.17.2.6 bool hal_led (char *set*)

Sets the LEDs on Jupiter 2 cards.

This can be used for example for debugging to show some state or to get a Nightrider feeling.

Parameters:

set Bits 0-6 are currently used for specifying LEDs.

Return values:

TRUE is currently returned

Note:

This function is always included when compiling for the AT90S8535, which is AFAIK the only processor, which is available as PCB with LEDs.

6.17.2.7 bool hal_rnd_addEntropy (void)

Adds entropy to the random number generator state.

The idea: Adding real randomness to the RAND_STATE via random events and a high speed timer.

The implementation: Timer 0 is started at the beginning and incremented every clock cycle, i.e. normally 3.58 million times per second. The timer is read at the beginning and at the end of the execution of every command. After every 8 timer values gathered in this way, RAND_STATE is read and encrypted with RAND_STATE and timer values as key. The result is written back to RAND_STATE.

The analysis: The analysis has been done by calculating the difference of each consecutive timer value pair. From these value two distributions are build. The first is the frequency for the timer value differences within the command and the second is the frequency for the timer value differences outside the command, i.e. while communicating with the terminal. The command used for this evaluation is the Read EEPROM command (80 04 18 00 10), which should have a relatively constant execution time and does not receive bytes from the terminal.

Terminal randomness: As expected the randomness of the distribution depends on the external system, i.e. card reader and host. Different distributions have been seen e.g. for Towitoko Chipdrive and Intertext IX2. This randomness can increas the randomness in the RAND_STATE buffer, when the card is in the hand of the owner of the card. By this, it changes RAND_STATE in a non-deterministic way for the institution writing the starting value of RAND_-STATE

Command randomness: Because there is no obvious possibility for different runtimes, the expected result would be to have exactly the same delta values for all command runs, but this is not the case. The result is a 9 - 10 ticks wide curved distribution. The reason for this might be the I2C routine communicating with the external EEPROM, because Get Response and Read EE in the internal EEPROM area have the expected result with a single delta value. Independed of the source of the clock tick variations, there is probably not much non-determinism for a skilled attacker, because he can count the clocks a command has executed.

Note:

This function and the initialization in **hal_init**() (p. 76) is currently commented out.

Return values:

TRUE on success.

FALSE on failure. Error code given in sw.

6.17.2.8 bool hal_rnd_getBlock (iu8 * *dst*)

Returns 8 random bytes.

The current AVR implementation does not produce real random numbers, but pseudo random numbers. These are produced by a X9.17 generator modified to use a counter instead of a timer. The state of this PRNG is held at RAND_-STATE_ADDR. The start value (at least the key) at this address should be unique for every card.

Parameters:

dst Pointer to the destination, where the 8 random bytes should be written.

Return values:

TRUE on success.

FALSE on failure. Error code given in sw.

Note:

There is still a simpler PRNG in the code, which uses 16 bytes RAM and 126 bytes ROM less than the X9.17 generator.

6.18 log.c File Reference

Logging functions.

`#include <config.h>`

`#include <hal.h>`

`#include <log.h>`

`#include <types.h>`

6.18.1 Detailed Description

Logging functions.

Id:

log.c,v 1.6 2002/12/22 15:42:55 m Exp

6.19 log.h File Reference

Logging declarations.

`#include <types.h>`

Defines

- #define **LOG_START** 0x1800
- #define **LOG_TAG_JOKER** 0xFE
- #define **LOG_TAG_EEWRITE_DST** 0x01
- #define **LOG_TAG_EEWRITE_SRC** 0x02
- #define **LOG_TAG_EEWRITE_LEN** 0x03
- #define **LOG_TAG_EEREAD_DST** 0x04
- #define **LOG_TAG_EEREAD_SRC** 0x05
- #define **LOG_TAG_EEREAD_LEN** 0x06
- #define **LOG_TAG_FSCREATE** 0x07
- #define **LOG_TAG_RANDOM** 0x08
- #define **log_init**()

  Initialize global logging variables.

- #define **log_add**(tag, data, len)

  Log data.

Variables

- **bool log_enable**

  Enables/disables logging.

6.19.1 Detailed Description

Logging declarations.

This logging facility can be used to assist debugging. It appends TLV objects (Tag, Length, Value) to logging area starting at LOG_START+2. At LOG_-START the address of the end of data is maintained. This is in native byte order. (Native of architecture or compiler.)

Tags can be used to specify what data has been logged. This is meant to assist testing, so be free to change the tags as needed.

Id:
log.h,v 1.5 2002/12/22 15:42:55 m Exp

6.19.2 Define Documentation

6.19.2.1 #define log_add(tag, data, len)

Log data.

Tag, length and then the data itself is append to the log data. It can be called even, when CONF_WITH_LOGGING is not 1, because it is then neutralized with a define.

Parameters:

tag Tag descriping the data.

data Pointer to the data.

len Length of the data in bytes.

6.19.2.2 #define log_init()

Initialize global logging variables.

This function must be called before using log_add. It can be called even, when CONF_WITH_LOGGING is not 1, because it is then neutralized with a define.

6.19.2.3 #define LOG_START 0x1800

6.19.2.4 #define LOG_TAG_EEREAD_DST 0x04

6.19.2.5 #define LOG_TAG_EEREAD_LEN 0x06

6.19.2.6 #define LOG_TAG_EEREAD_SRC 0x05

6.19.2.7 #define LOG_TAG_EEWRITE_DST 0x01

6.19.2.8 #define LOG_TAG_EEWRITE_LEN 0x03

6.19.2.9 #define LOG_TAG_EEWRITE_SRC 0x02

6.19.2.10 #define LOG_TAG_FSCREATE 0x07

6.19.2.11 #define LOG_TAG_JOKER 0xFE

6.19.2.12 #define LOG_TAG_RANDOM 0x08

6.19.3 Variable Documentation

6.19.3.1 bool log_enable

Enables/disables logging.

This must be set the TRUE, before any log_add has an effect. It can be also changed in between to switch off and on logging to reduce unwanted data in the logging area.

6.20 main.c File Reference

main() (p. 35) function with command loop.

`#include <config.h>`

`#include <auth.h>`

`#include <commands.h>`

`#include <sw.h>`

`#include <fs.h>`

`#include <hal.h>`

`#include <t0.h>`

`#include <transaction.h>`

Functions

- int **main** (void)

  Main function containing command interpreter loop.

6.20.1 Detailed Description

main() (p. 35) function with command loop.

6.20.2 Function Documentation

6.20.2.1 int main (void)

Main function containing command interpreter loop.

At the end of the loop, sw is sent as the status word.

This function does never return.

6.21 main.h File Reference

main() (p. 35) declarations and documentation.

6.21.1 Detailed Description

main() (p. 35) declarations and documentation.

Id:
main.h,v 1.24 2003/03/30 19:50:58 m Exp

6.22 newdes-sk.c File Reference

`#include <config.h>`

`#include <types.h>`

6.23 newdes-sk.h File Reference

NEWDES-SK declarations.

`#include <types.h>`

Defines

- #define **NEWDESSK_KEY_LEN** 15
  NEWDES-SK key size.
- #define **NEWDESSK_BLOCK_LEN** 8
  NEWDES-SK block length.

Functions

- void **newdessk_enc** (**iu8** *v, **iu8** *k)
  NEWDES-SK encryption function.

- void **newdessk_dec** (**iu8** *v, **iu8** *k)
  NEWDES-SK decryption function.

6.23.1 Detailed Description

NEWDES-SK declarations.

Id:
newdes-sk.h,v 1.1 2003/03/30 12:42:21 m Exp

6.23.2 Define Documentation

6.23.2.1 #define NEWDESSK_BLOCK_LEN 8

NEWDES-SK block length.

6.23.2.2 #define NEWDESSK_KEY_LEN 15

NEWDES-SK key size.

6.23.3 Function Documentation

6.23.3.1 void newdessk_dec (iu8 * *v*, iu8 * *k*)

NEWDES-SK decryption function.

This function decrypts *v* with *k* and returns the decrypted data in *v*.

Parameters:
v Array of eight iu8 values containing the data block.
k Array of 15 iu8 values containing the key.

6.23.3.2 void newdessk_enc (iu8 * *v*, iu8 * *k*)

NEWDES-SK encryption function.

This function encrypts *v* with *k* and returns the encrypted data in *v*.

Parameters:
v Array of eight iu8 values containing the data block.
k Array of 15 iu8 values containing the key.

6.24 skipjack.c File Reference

Skipjack declarations.

`#include <config.h>`

#include <skipjack.h>

#include <types.h>

6.24.1 Detailed Description

Skipjack declarations.

Id:
skipjack.c,v 1.1 2003/03/30 12:42:21 m Exp

6.25 skipjack.h File Reference

Skipjack declarations.

#include <types.h>

Defines

- #define **SKIPJACK_KEY_LEN** 10
  SKIPJACK key size.
- #define **SKIPJACK_BLOCK_LEN** 8
  SKIPJACK block length.

Functions

- void **skipjack_enc** (**iu8** *v, **iu8** *k)
  Skipjack encryption function.
- void **skipjack_dec** (**iu8** *v, **iu8** *k)
  Skipjack decryption function.

6.25.1 Detailed Description

Skipjack declarations.

Documentation for Skipjack is available at `http://csrc.nist.gov/encryption/skipjack/skipjack-kea.htm`.

Id:
skipjack.h,v 1.1 2003/03/30 12:42:21 m Exp

6.25.2 Define Documentation

6.25.2.1 #define SKIPJACK_BLOCK_LEN 8

SKIPJACK block length.

6.25.2.2 #define SKIPJACK_KEY_LEN 10

SKIPJACK key size.

6.25.3 Function Documentation

6.25.3.1 void skipjack_dec (iu8 * *v*, iu8 * *k*)

Skipjack decryption function.

This function decrypts *v* with *k* and returns the decrypted data in *v*.

Parameters:

- ***v*** Array of eight iu8 values containing the data block.
- ***k*** Array of ten iu8 values containing the key.

6.25.3.2 void skipjack_enc (iu8 * *v*, iu8 * *k*)

Skipjack encryption function.

This function encrypts *v* with *k* and returns the encrypted data in *v*.

Parameters:

- ***v*** Array of eight iu8 values containing the data block.
- ***k*** Array of ten iu8 values containing the key.

6.26 sw.c File Reference

Errno functions.

`#include <config.h>`

`#include <sw.h>`

`#include <types.h>`

Functions

- void **sw_set** (**iu16** value)

  Function for setting sw.

Variables

- **iu16 sw**

  Status word returned by functions.

6.26.1 Detailed Description

Errno functions.

Id:
sw.c,v 1.1 2002/12/22 15:42:55 m Exp

6.26.2 Function Documentation

6.26.2.1 void sw_set (iu16 *value*)

Function for setting sw.

This function reduces the code size on the AVR architecture.

6.26.3 Variable Documentation

6.26.3.1 iu16 sw

Status word returned by functions.

The state of sw is undefined, on return value TRUE.

The content of this variable is returned at the end of the command interpreter loop. Functions falled from the command interpreter loop should set it to SW_-OK if they return without error.

6.27 sw.h File Reference

Errno declarations.

`#include <types.h>`

Defines

- #define **sw_get**() **sw**

Enumerations

- enum **SW** { **SW_ACCESS_DENIED** = 0x6982, **SW_AUTH_-BLOCKED** = 0x6983, **SW_AVAILABLE** = 0x6100, **SW_-CHANGE_MEMORY_ERROR** = 0x6581, **SW_COUNTER** =

0x63C0, **SW_EOD** = 0x6282, **SW_FILE_EXISTS** = 0x6A89, **SW_-FILE_FILLED** = 0x6381, **SW_FILE_NOT_FOUND** = 0x6A82, **SW_FILE_TO_SHORT** = 0x6A84, **SW_INCOMPATIBLE_FILE** = 0x6981, **SW_LC_INCONSISTEND_WITH_P1P2** = 0x6A87, **SW_-NOT_ALLOWED** = 0x6986, **SW_NOT_SUPPORTED** = 0x6A81, **SW_OK** = 0x9000, **SW_OTHER** = 0x6F00, **SW_REF_DATA_-INVALID** = 0x6984, **SW_REF_DATA_NOT_FOUND** = 0x6A88, **SW_VERIFCATION_FAILED** = 0x6300, **SW_WRONG_CLA** = 0x6E00, **SW_WRONG_CONDITION** = 0x6985, **SW_WRONG_-DATA** = 0x6A80, **SW_WRONG_INS** = 0x6D00, **SW_WRONG_LE** = 0x6C00, **SW_WRONG_LEN** = 0x6700, **SW_WRONG_P1P2** = 0x6A86, **SW_WRONG_REFERENCE** = 0x6B00 }

Status word values.

Functions

- void **sw_set** (**iu16** value)

  Function for setting sw.

Variables

- **iu16 sw**

  Status word returned by functions.

6.27.1 Detailed Description

Errno declarations.

See also for the man page of sw.

Id:
sw.h,v 1.2 2003/04/24 16:47:00 m Exp

6.27.2 Define Documentation

6.27.2.1 #define sw_get() sw

6.27.3 Enumeration Type Documentation

6.27.3.1 enum SW

Status word values.

Note:
Implement this as defines, because doxygen apparently can't handle this?

Enumeration values:

- **SW_ACCESS_DENIED**
- **SW_AUTH_BLOCKED**
- **SW_AVAILABLE**
- **SW_CHANGE_MEMORY_ERROR**
- **SW_COUNTER**
- **SW_EOD**
- **SW_FILE_EXISTS**
- **SW_FILE_FILLED**
- **SW_FILE_NOT_FOUND**
- **SW_FILE_TO_SHORT**
- **SW_INCOMPATIBLE_FILE**
- **SW_LC_INCONSISTEND_WITH_P1P2**
- **SW_NOT_ALLOWED**
- **SW_NOT_SUPPORTED**
- **SW_OK**
- **SW_OTHER**
- **SW_REF_DATA_INVALID**
- **SW_REF_DATA_NOT_FOUND**
- **SW_VERIFCATION_FAILED**
- **SW_WRONG_CLA**
- **SW_WRONG_CONDITION**
- **SW_WRONG_DATA**
- **SW_WRONG_INS**
- **SW_WRONG_LE**
- **SW_WRONG_LEN**
- **SW_WRONG_P1P2**
- **SW_WRONG_REFERENCE**

6.27.4 Function Documentation

6.27.4.1 void sw_set (iu16 *value*)

Function for setting sw.

This function reduces the code size on the AVR architecture.

6.27.5 Variable Documentation

6.27.5.1 iu16 sw

Status word returned by functions.

The state of sw is undefined, on return value TRUE.

The content of this variable is returned at the end of the command interpreter loop. Functions falled from the command interpreter loop should set it to SW_-OK if they return without error.

6.28 t0.c File Reference

T=0 functions.

`#include <config.h>`

`#include <sw.h>`

`#include <commands.h>`

`#include <hal.h>`

`#include <t0.h>`

`#include <types.h>`

`#include <string.h>`

Defines

- #define **SW sw**

Functions

- void **t0_init** (void)

    Initialize T=0 data.

- void **t0_recBlock** (**iu8** *dst, **iu16** len)

    Receive multiple bytes.

- void **t0_wtx** (void)

    Send a WTX.

- **bool t0_dir** (**bool** in)

    Set IN/OUT direction if command.

- void **t0_exchApdu** (void)

    Exchanged APDUs with IFD.

- void **t0_getResponse** (void)

Get Response command.

- **bool t0_testP1P2** (**iu16** p1p2)

  Test P1P2 for correct content.

- **bool t0_testP3** (**iu8** p3)

  Test P3 for correct content.

Variables

- **bool iosend**
- **bool ioin**
- **iu8 ioheader** [5]
- **iu8 iodata** [sizeof(iodata)]
- **iu16 iodatalen**

6.28.1 Detailed Description

T=0 functions.

Id:

t0.c,v 1.14 2003/05/10 17:26:58 m Exp

6.28.2 Define Documentation

6.28.2.1 #define SW sw

6.28.3 Function Documentation

6.28.3.1 bool t0_dir (bool *in*)

Set IN/OUT direction if command.

This should be called by each command to set the command case. With IN commands the command data is only available after this function has been called.

Only in an error case, where only a status word is to be returned this command doesn't have to be called.

Parameters:

in Boolean value signaling direction.

Return values:

TRUE on success.

FALSE on failure. Error code given in sw.

6.28.3.2 void t0_exchApdu (void)

Exchanged APDUs with IFD.

This function handles also the Get Response case.

6.28.3.3 void t0_getResponse (void)

Get Response command.

Is called from t0_exchApdu.

6.28.3.4 void t0_init (void)

Initialize T=0 data.

Must be called before the first t0_exchApdu.

6.28.3.5 void t0_recBlock (iu8 * *dst*, iu16 *len*)

Receive multiple bytes.

Receive *len* bytes and write them to *dst*.

Parameters:

dst Pointer to destination area in RAM.

len Data bytes to receive.

6.28.3.6 bool t0_testP1P2 (iu16 *p1p2*)

Test P1P2 for correct content.

Compares the values of P1 and P2 in *header* to *p1p2*. If they differ, sw is set to SW_WRONG_P1P2 and FALSE is returned.

Parameters:

p1p2 Target value of P1P2.

Return values:

TRUE on success.

FALSE on failure. Error code given in sw.

6.28.3.7 bool t0_testP3 (iu8 *p3*)

Test P3 for correct content.

Compares the value of P3 in *header* to *p3*. If they differ, sw is set to SW_-WRONG_P1P2 and FALSE is returned.

Parameters:

p3 Target value of P3.

Return values:

TRUE on success.

FALSE on failure. Error code given in sw.

6.28.3.8 void t0_wtx (void)

Send a WTX.

6.28.4 Variable Documentation

6.28.4.1 iu8 iodata[sizeof(iodata)]

6.28.4.2 iu16 iodatalen

6.28.4.3 iu8 ioheader[5]

6.28.4.4 bool ioin

6.28.4.5 bool iosend `[static]`

6.29 t0.h File Reference

T=0 declarations.

Defines

- #define **CLA ioheader**[0]
- #define **INS ioheader**[1]
- #define **P1 ioheader**[2]
- #define **P2 ioheader**[3]
- #define **P3 ioheader**[4]
- #define **T0_DIR_IN** TRUE

  Data from IFD to ICC.

- #define **T0_DIR_OUT** FALSE

  Data from ICC to IFD.

Functions

- void **t0_init** (void)
  Initialize T=0 data.
- void **t0_recBlock** (**iu8** *dst, **iu16** len)
  Receive multiple bytes.
- void **t0_wtx** (void)
  Send a WTX.
- **bool t0_dir** (**bool** in)
  Set IN/OUT direction if command.
- void **t0_exchApdu** (void)
  Exchanged APDUs with IFD.
- void **t0_getResponse** (void)
  Get Response command.
- **bool t0_testP1P2** (**iu16** p1p2)
  Test P1P2 for correct content.
- **bool t0_testP3** (**iu8** p3)
  Test P3 for correct content.

Variables

- **bool ioin**
- **iu8 ioheader** [5]
- **iu8 iodata** [CONF_IODATA_SIZE]
- **iu16 iodatalen**

6.29.1 Detailed Description

T=0 declarations.

T=0 is not implemented fully as library, but must be partly done in the commands itself. This reduces the RAM requirements. E.g. when doing an Update Binary the data must not be received in total before writing, but can be received and written in single bytes.

Id:
t0.h,v 1.13 2003/05/10 17:26:58 m Exp

6.29.2 Define Documentation

6.29.2.1 #define CLA ioheader[0]

6.29.2.2 #define INS ioheader[1]

6.29.2.3 #define P1 ioheader[2]

6.29.2.4 #define P2 ioheader[3]

6.29.2.5 #define P3 ioheader[4]

6.29.2.6 #define T0_DIR_IN TRUE

Data from IFD to ICC.

6.29.2.7 #define T0_DIR_OUT FALSE

Data from ICC to IFD.

6.29.3 Function Documentation

6.29.3.1 bool t0_dir (bool *in*)

Set IN/OUT direction if command.

This should be called by each command to set the command case. With IN commands the command data is only available after this function has been called.

Only in an error case, where only a status word is to be returned this command doesn't have to be called.

Parameters:

in Boolean value signaling direction.

Return values:

TRUE on success.

FALSE on failure. Error code given in sw.

6.29.3.2 void t0_exchApdu (void)

Exchanged APDUs with IFD.

This function handles also the Get Response case.

6.29.3.3 void t0_getResponse (void)

Get Response command.

Is called from t0_exchApdu.

6.29.3.4 void t0_init (void)

Initialize T=0 data.

Must be called before the first t0_exchApdu.

6.29.3.5 void t0_recBlock (iu8 * *dst*, iu16 *len*)

Receive multiple bytes.

Receive *len* bytes and write them to *dst*.

Parameters:

dst Pointer to destination area in RAM.

len Data bytes to receive.

6.29.3.6 bool t0_testP1P2 (iu16 *p1p2*)

Test P1P2 for correct content.

Compares the values of P1 and P2 in *header* to *p1p2*. If they differ, sw is set to SW_WRONG_P1P2 and FALSE is returned.

Parameters:

p1p2 Target value of P1P2.

Return values:

TRUE on success.

FALSE on failure. Error code given in sw.

6.29.3.7 bool t0_testP3 (iu8 *p3*)

Test P3 for correct content.

Compares the value of P3 in *header* to *p3*. If they differ, sw is set to SW_-WRONG_P1P2 and FALSE is returned.

Parameters:

p3 Target value of P3.

Return values:

TRUE on success.

FALSE on failure. Error code given in sw.

6.29.3.8 void t0_wtx (void)

Send a WTX.

6.29.4 Variable Documentation

6.29.4.1 iu8 iodata[CONF_IODATA_SIZE]

6.29.4.2 iu16 iodatalen

6.29.4.3 iu8 ioheader[5]

6.29.4.4 bool ioin

6.30 tea.c File Reference

TEA functions.

`#include <config.h>`

`#include <tea.h>`

`#include <tools.h>`

Defines

- #define **TEA_SMALL**
- #define **hton_ul**(x, y)

Functions

- **iu32 tea_func** (**iu32** ∗in, **iu32** ∗sum, **iu32** ∗k)
- void **tea_enc** (**iu32** ∗v, **iu32** ∗k)

  TEA encryption function.

- void **tea_dec** (**iu32** ∗v, **iu32** ∗k)

  TEA decryption function.

6.30.1 Detailed Description

TEA functions.

Id:

tea.c,v 1.6 2003/04/02 23:57:54 m Exp

6.30.2 Define Documentation

6.30.2.1 #define hton_ul(x, y)

6.30.2.2 #define TEA_SMALL

6.30.3 Function Documentation

6.30.3.1 void tea_dec (iu32 * *v*, iu32 * *k*)

TEA decryption function.

This function decrypts *v* with *k* and returns the decrypted data in *v*.

Parameters:

- ***v*** Array of two long values containing the data block.
- ***k*** Array of four long values containing the key.

6.30.3.2 void tea_enc (iu32 * *v*, iu32 * *k*)

TEA encryption function.

This function encrypts *v* with *k* and returns the encrypted data in *v*.

Parameters:

- ***v*** Array of two long values containing the data block.
- ***k*** Array of four long values containing the key.

6.30.3.3 iu32 tea_func (iu32 * *in*, iu32 * *sum*, iu32 * *k*)

6.31 tea.h File Reference

TEA declarations.

`#include <types.h>`

Defines

- #define **TEA_KEY_LEN** 16

  TEA key size.

- #define **TEA_BLOCK_LEN** 8

  TEA block length.

- #define **DELTA** 0x9E3779B9

  *Magic value. (Golden number * 2^31).*

- #define **ROUNDS** 32

  Number of rounds.

Functions

- void **tea_enc** (**iu32** ∗v, **iu32** ∗k)

  TEA encryption function.

- void **tea_dec** (**iu32** ∗v, **iu32** ∗k)

  TEA decryption function.

6.31.1 Detailed Description

TEA declarations.

Documentation for TEA is available at `http://www.cl.cam.ac.uk/ftp/users/djw3/tea.ps.`

Id:

tea.h,v 1.5 2002/12/22 15:42:55 m Exp

6.31.2 Define Documentation

6.31.2.1 #define DELTA 0x9E3779B9

Magic value. (Golden number ∗ 2^31).

6.31.2.2 #define ROUNDS 32

Number of rounds.

6.31.2.3 #define TEA_BLOCK_LEN 8

TEA block length.

6.31.2.4 #define TEA_KEY_LEN 16

TEA key size.

6.31.3 Function Documentation

6.31.3.1 void tea_dec (iu32 ∗ *v*, iu32 ∗ *k*)

TEA decryption function.

This function decrypts *v* with *k* and returns the decrypted data in *v*.

Parameters:

v Array of two long values containing the data block.

k Array of four long values containing the key.

6.31.3.2 void tea_enc (iu32 ∗ *v*, iu32 ∗ *k*)

TEA encryption function.

This function encrypts *v* with *k* and returns the encrypted data in *v*.

Parameters:

v Array of two long values containing the data block.

k Array of four long values containing the key.

6.32 tools.c File Reference

Tool functions.

`#include <config.h>`

`#include <tools.h>`

`#include <types.h>`

6.32.1 Detailed Description

Tool functions.

Id:

tools.c,v 1.7 2002/12/22 15:42:55 m Exp

6.33 tools.h File Reference

Tool declarations.

`#include <config.h>`

`#include <types.h>`

Defines

- #define **swap_us**(us) (((us&0x00FF)<<8)|((us&0xFF00)>>8))

  Byte swap single short.

- #define **swap_ul**(ul) (((ul&0x000000FF)<<24)|((ul&0x0000FF00)<<8)|((ul&0x00FF0000)>>8)|((ul&0x-FF000000)>>24))

  Byte swap single long.

- #define **hton_us**(us)

- #define **hton_ul**(ul)
- #define **min**(a, b) ((a)<(b)?(a):(b))
- #define **max**(a, b) ((a)>(b)?(a):(b))

Functions

- void **hton_us** (**iu16** *us, **iu8** num)

  Byte swap multiple shorts.

- void **hton_ul** (**iu32** *ul, **iu8** num)

  Byte swap multiple longs.

6.33.1 Detailed Description

Tool declarations.

Id:

tools.h,v 1.11 2002/12/22 15:42:56 m Exp

6.33.2 Define Documentation

6.33.2.1 #define hton_ul(ul)

6.33.2.2 #define hton_us(us)

6.33.2.3 #define max(a, b) ((a)>(b)?(a):(b))

Return maximum value

6.33.2.4 #define min(a, b) ((a)<(b)?(a):(b))

Return minimum value

6.33.2.5 #define swap_ul(ul) (((ul&0x000000FF)<<24)|((ul&0x0000FF00)<<8)|((ul&0x00FF0000)>>8)|((ul&0xFF000000)>>24))

Byte swap single long.

6.33.2.6 #define swap_us(us) (((us&0x00FF)<<8)|((us&0xFF00)>>8))

Byte swap single short.

6.33.3 Function Documentation

6.33.3.1 void hton_ul (iu32 * *ul*, iu8 *num*)

Byte swap multiple longs.

Parameters:

ul Pointer to an array of longs.

num Number of longs to process.

6.33.3.2 void hton_us (iu16 * *us*, iu8 *num*)

Byte swap multiple shorts.

Parameters:

us Pointer to an array of shorts.

num Number of shorts to process.

6.34 transaction.c File Reference

Transaction functions.

`#include <config.h>`

`#include <sw.h>`

`#include <hal.h>`

`#include <transaction.h>`

`#include <types.h>`

Functions

- **bool ta_setdata** (**iu8** *data, **iu8** len, **iu16** dst)

  Begins a transaction and sets the data.

- **bool ta_commit** (void)

  Commits the data in the transaction buffer.

6.34.1 Detailed Description

Transaction functions.

Id:

transaction.c,v 1.11 2002/12/22 15:42:56 m Exp

6.34.2 Function Documentation

6.34.2.1 bool ta_commit (void)

Commits the data in the transaction buffer.

Data is written to the destination specified in **ta_setdata**() (p. 107). The state is then set to clean.

This function must be called after each reset to commit pending transactions.

Return values:

TRUE on success.

FALSE on failure. Error code given in sw.

6.34.2.2 bool ta_setdata (iu8 * *data*, iu8 *len*, iu16 *dst*)

Begins a transaction and sets the data.

data/len bytes are written to the transaction buffer. The destination address is also stored there. Then the state is set to TRANSAC_STATE_DATA.

Parameters:

data Pointer to the data to write into the transaction buffer.

len Length of the data. (Max.: TRANSAC_DATA_LEN)

dst Destination of the data in EEPROM.

Return values:

TRUE on success.

FALSE on failure. Error code given in sw.

6.35 transaction.h File Reference

Transaction declarations.

`#include <types.h>`

Defines

- #define **TRANSAC_STATE_CLEAN** 0x00
- #define **TRANSAC_STATE_DATA** 0xA5

  Transaction buffer contains data.

Functions

- **bool ta_setdata** (**iu8** *data, **iu8** len, **iu16** dst)

  Begins a transaction and sets the data.

- **bool ta_commit** (void)

  Commits the data in the transaction buffer.

- **bool ta_rollback** (void)

  Do a rollback and discard the data in the transaction buffer.

6.35.1 Detailed Description

Transaction declarations.

Note:

Is the current transaction implementation secure against tearing? IMO it should be nearly be. Let's consider following scenarios:

1) Card is removed while writing the data for the transaction: This is no problem, because the state is still TRANSAC_STATE_CLEAN.

2) Card is removed while writing the status byte: This is most likely no problem. If the state is anything other than TRANSAC_STATE_DATA, it is regarded as clean, which it is. The only problem is, when it accidently is TRANSAC_STATE_DATA, which is IMO very unlikely, because the erased state is totally different. But on the other side, if the card tearing occured shortly after the state update, the state really is TRANSAC_STATE_DATA.

3) Card is removed while doing a commit and writing the data therein: This is no problem, because after the next reset the commit is tried again.

4) Card is removed after a commit, while reseting the status byte: This is no problem, because most likely the new state is interpreted as TRANSAC_STATE_CLEAN, which it is, and if the new state is really TRANSAC_STATE_DATA, it is just a second time written.

5) Card is removed while writing status byte in a rollback: This might be a problem, because the rollback might file, when the unlikely event occures, that the state is TRANSAC_STATE_DATA. But this can also happen, if the card is removed before the rollback, because the state is there already TRANSAC_STATE_DATA.

Id:

transaction.h,v 1.9 2002/12/22 15:42:56 m Exp

6.35.2 Define Documentation

6.35.2.1 #define TRANSAC_STATE_CLEAN 0x00

Transaction buffer is clean. (Also all other values not being 0xA5 are interpreted as clean.

6.35.2.2 #define TRANSAC_STATE_DATA 0xA5

Transaction buffer contains data.

6.35.3 Function Documentation

6.35.3.1 bool ta_commit (void)

Commits the data in the transaction buffer.

Data is written to the destination specified in **ta_setdata**() (p. 107). The state is then set to clean.

This function must be called after each reset to commit pending transactions.

Return values:

TRUE on success.

FALSE on failure. Error code given in sw.

6.35.3.2 bool ta_rollback (void)

Do a rollback and discard the data in the transaction buffer.

The state is set to clean.

Return values:

TRUE on success.

FALSE on failure. Error code given in sw.

6.35.3.3 bool ta_setdata (iu8 * *data*, iu8 *len*, iu16 *dst*)

Begins a transaction and sets the data.

data/len bytes are written to the transaction buffer. The destination address is also stored there. Then the state is set to TRANSAC_STATE_DATA.

Parameters:

data Pointer to the data to write into the transaction buffer.

len Length of the data. (Max.: TRANSAC_DATA_LEN)

dst Destination of the data in EEPROM.

Return values:

TRUE on success.

FALSE on failure. Error code given in sw.

6.36 types.h File Reference

Type declarations.

Defines

- #define **FALSE** 0
- #define **TRUE** !FALSE
- #define **PRG_RDB**(x) (*((**iu8***)(x)))
- #define **CODE**

Typedefs

- typedef unsigned char **iu8**

    1 byte unsiged data type.

- typedef unsigned short **iu16**

    2 byte unsigned data type.

- typedef unsigned long **iu32**

    4 byte unsigned data type.

- typedef unsigned char **bool**

    Boolean data type.

6.36.1 Detailed Description

Type declarations.

Id:
types.h,v 1.12 2003/03/30 19:36:48 m Exp

6.36.2 Define Documentation

6.36.2.1 #define CODE

6.36.2.2 #define FALSE 0

6.36.2.3 #define PRG_RDB(x) (*((iu8*)(x)))

6.36.2.4 #define TRUE !FALSE

6.36.3 Typedef Documentation

6.36.3.1 typedef unsigned char bool

Boolean data type.

6.36.3.2 typedef unsigned short iu16

2 byte unsigned data type.

6.36.3.3 typedef unsigned long iu32

4 byte unsigned data type.

6.36.3.4 typedef unsigned char iu8

1 byte unsigied data type.

7 SOSSE Page Documentation

7.1 Todo List

Class s_fpath There is probably a potential for simplification, when the file type (DF/EF) is not handled so rigid.

File auth.h Here are certainly some optimizations possible to reduce code size.

Global CONF_WITH_FILESYSTEM Make this define something happen.

File fs.h Here are certainly some optimizations possible to reduce code size.

Global fs_seek(iu16 fid, S_FSTREAM *fs, S_FINFO *fi, iu8 type) There is probably a potential for simplification, when the file type is not handled so rigid.

Global selected Saving here a S_FINFO and/or S_FSTREAM of the selected file might reduce the code size.

Index