Nano cryptocurrency C library with P2PoW/DPoW support for Embedded

1.0.0

Generated by Doxygen 1.8.13

Contents

Chapter 1

Overview

myNanoEmbedded is a lightweight C library of source files that integrates `Nano Cryptocurrency` to low complexity computational devices to send/receive digital money to anywhere in the world with fast trasnsaction and with a small fee by delegating a Proof of Work with your choice:

- DPoW (Distributed Proof of Work)
- P2PoW (a Descentralized P2P Proof of Work)

API features

- Attaches a random function to TRNG hardware (if available)
- Self entropy verifier to ensure excelent TRNG or PRNG entropy
- Creates a encrypted by password your stream or file to store your Nano SEED
- Bip39 and Brainwallet support
- Convert raw data to Base32
- Parse SEED and Bip39 to JSON
- Sign a block using Blake2b hash with Ed25519 algorithm
- ARM-A, ARM-M, Thumb, Xtensa-LX6 and IA64 compatible
- Linux desktop, Raspberry PI, ESP32 and Olimex A20 tested platforms
- Communication over `Fenix protocol` bridge over TLS
- Libsodium and mbedTLS libraries with smaller resources and best performance
- Optmized for size and speed
- Non static functions (all data is cleared before processed for security)
- Fully written in C for maximum performance and portability

To add this API in your project you must first:

1. Download the latest version.

   ```
   git clone https://github.com/devfabiosilva/myNanoEmbedded.git --recurse-submodules
   ```

2. Include the main library files in the client application.

   ```
   #include "f_nano_crypto_util.h"
   ```

Initialize API

Function	Description
f_random_attach() (p. **??**)	Initializes the PRNG or TRNG to be used in this API

Transmit/Receive transactions

To transmit/receive your transaction you must use `Fenix` protocol to stabilish a DPoW/P2PoW support

Examples using platforms

The repository has some examples with most common embedded and Linux systems

- `Native Linux`
- `Raspberry Pi`
- `ESP32`
- `Olimex A20`
- `STM`

Credits

Author

Fábio Pereira da Silva

Date

Feb 2020

Version

1.0

Copyright

License MIT `see here`

References:

[1] - Colin LeMahieu - *Nano: A Feeless Distributed Cryptocurrency Network* - (2015)

[2] - Z. S. Spakovszky - *7.3 A Statistical Definition of Entropy* - (2005) - NOTE: Entropy function for cryptography is implemented based on `Definition (7.12)` of this amazing topic

[3] - Kaique Anarkrypto - *Delegated Proof of Work* - (2019)

[4] - `docs.nano.org` - *Node RPCs documentation*

Chapter 2

Data Structure Index

2.1 Data Structures

Here are the data structures with brief descriptions:

Chapter 3

File Index

3.1 Files

Here is a list of all files with brief descriptions:

Chapter 4

Data Structure Documentation

4.1 f_block_transfer_t Struct Reference

`#include <f_nano_crypto_util.h>`

Data Fields

- uint8_t **preamble** [32]
- uint8_t **account** [32]
- uint8_t **previous** [32]
- uint8_t **representative** [32]
- **f_uint128_t balance**
- uint8_t **link** [32]
- uint8_t **signature** [64]
- uint8_t **prefixes**
- uint64_t **work**

4.1.1 Detailed Description

Nano signed block raw data defined in this `reference`

Definition at line **258** of file **f_nano_crypto_util.h**.

4.1.2 Field Documentation

4.1.2.1 account

`uint8_t account[32]`

Account in raw binary data.

Definition at line **262** of file **f_nano_crypto_util.h**.

4.1.2.2 balance

```
f_uint128_t balance
```

Big number 128 bit raw balance.

See also

f_uint128_t (p. **??**)

Definition at line **270** of file **f_nano_crypto_util.h**.

4.1.2.3 link

```
uint8_t link[32]
```

link or destination account

Definition at line **272** of file **f_nano_crypto_util.h**.

4.1.2.4 preamble

```
uint8_t preamble[32]
```

Block preamble.

Definition at line **260** of file **f_nano_crypto_util.h**.

4.1.2.5 prefixes

```
uint8_t prefixes
```

Internal use for this API.

Definition at line **276** of file **f_nano_crypto_util.h**.

4.1.2.6 previous

```
uint8_t previous[32]
```

Previous block.

Definition at line **264** of file **f_nano_crypto_util.h**.

4.1.2.7 representative

```
uint8_t representative[32]
```

Representative for current account.

Definition at line **266** of file **f_nano_crypto_util.h**.

4.1.2.8 signature

```
uint8_t signature[64]
```

Signature of the block.

Definition at line **274** of file **f_nano_crypto_util.h**.

4.1.2.9 work

```
uint64_t work
```

Internal use for this API.

Definition at line **278** of file **f_nano_crypto_util.h**.

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

- **f_nano_crypto_util.h**

4.2 f_file_info_err_t Struct Reference

```
#include <f_nano_crypto_util.h>
```

4.2.1 Detailed Description

Error enumerator for info file functions.

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

- **f_nano_crypto_util.h**

4.3 f_nano_crypto_wallet_t Struct Reference

```
#include <f_nano_crypto_util.h>
```

Data Fields

- uint8_t **nano_hdr** [sizeof(NANO_WALLET_MAGIC)]
- uint32_t **ver**
- uint8_t **description** [F_DESC_SZ]
- uint8_t **salt** [32]
- uint8_t **iv** [16]
- F_ENCRYPTED_BLOCK **seed_block**

4.3.1 Detailed Description

struct of the block of encrypted file to store Nano SEED

Definition at line **389** of file **f_nano_crypto_util.h**.

4.3.2 Field Documentation

4.3.2.1 description

```
uint8_t description[F_DESC_SZ]
```

File description.

Definition at line **395** of file **f_nano_crypto_util.h**.

4.3.2.2 iv

```
uint8_t iv[16]
```

Initial vector of first encryption layer.

Definition at line **399** of file **f_nano_crypto_util.h**.

4.3.2.3 nano_hdr

```
uint8_t nano_hdr[sizeof(NANO_WALLET_MAGIC)]
```

Header of the file.

Definition at line **391** of file **f_nano_crypto_util.h**.

4.3.2.4 salt

```
uint8_t salt[32]
```

Salt of the first encryption layer.

Definition at line **397** of file **f_nano_crypto_util.h**.

4.3.2.5 seed_block

```
F_ENCRYPTED_BLOCK seed_block
```

Second encrypted block for Nano SEED.

Definition at line **401** of file **f_nano_crypto_util.h**.

4.3.2.6 ver

```
uint32_t ver
```

Version of the file.

Definition at line **393** of file **f_nano_crypto_util.h**.

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

- **f_nano_crypto_util.h**

4.4 f_nano_encrypted_wallet_t Struct Reference

```
#include <f_nano_crypto_util.h>
```

Data Fields

- uint8_t **sub_salt** [32]
- uint8_t **iv** [16]
- uint8_t **reserved** [16]
- uint8_t **hash_sk_unencrypted** [32]
- uint8_t **sk_encrypted** [32]

4.4.1 Detailed Description

struct of the block of encrypted file to store Nano SEED

Definition at line **361** of file **f_nano_crypto_util.h**.

4.4.2 Field Documentation

4.4.2.1 hash_sk_unencrypted

```
uint8_t hash_sk_unencrypted[32]
```

hash of Nano SEED when unencrypted

Definition at line **369** of file **f_nano_crypto_util.h**.

4.4.2.2 iv

```
uint8_t iv[16]
```

Initial sub vector.

Definition at line **365** of file **f_nano_crypto_util.h**.

4.4.2.3 reserved

```
uint8_t reserved[16]
```

Reserved (not used)

Definition at line **367** of file **f_nano_crypto_util.h**.

4.4.2.4 sk_encrypted

```
uint8_t sk_encrypted[32]
```

Secret.

SEED encrypted (second layer)

Definition at line **371** of file **f_nano_crypto_util.h**.

4.4.2.5 sub_salt

```
uint8_t sub_salt[32]
```

Salt of the sub block to be stored.

Definition at line **363** of file **f_nano_crypto_util.h**.

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

- **f_nano_crypto_util.h**

4.5 f_nano_wallet_info_bdy_t Struct Reference

```
#include <f_nano_crypto_util.h>
```

Data Fields

- uint8_t **wallet_prefix**
- uint32_t **last_used_wallet_number**
- char **wallet_representative** [**MAX_STR_NANO_CHAR**]
- char **max_fee** [F_RAW_STR_MAX_SZ]
- uint8_t **reserved** [44]

4.5.1 Detailed Description

struct of the body block of the info file

Definition at line **473** of file **f_nano_crypto_util.h**.

4.5.2 Field Documentation

4.5.2.1 last_used_wallet_number

```
uint32_t last_used_wallet_number
```

Last used wallet number.

Definition at line **477** of file **f_nano_crypto_util.h**.

4.5.2.2 max_fee

```
char max_fee[F_RAW_STR_MAX_SZ]
```

Custom preferred max fee of Proof of Work.

Definition at line **481** of file **f_nano_crypto_util.h**.

4.5.2.3 reserved

```
uint8_t reserved[44]
```

Reserved.

Definition at line **483** of file **f_nano_crypto_util.h**.

4.5.2.4 wallet_prefix

```
uint8_t wallet_prefix
```

Wallet prefix: 0 for NANO; 1 for XRB.

Definition at line **475** of file **f_nano_crypto_util.h**.

4.5.2.5 wallet_representative

```
char wallet_representative[ MAX_STR_NANO_CHAR]
```

Wallet representative.

Definition at line **479** of file **f_nano_crypto_util.h**.

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

- **f_nano_crypto_util.h**

4.6 f_nano_wallet_info_t Struct Reference

```
#include <f_nano_crypto_util.h>
```

Data Fields

- uint8_t **header** [sizeof(F_NANO_WALLET_INFO_MAGIC)]
- uint16_t **version**
- char **desc** [F_NANO_DESC_SZ]
- uint8_t **nanoseed_hash** [32]
- uint8_t **file_info_integrity** [32]
- F_NANO_WALLET_INFO_BODY **body**

4.6.1 Detailed Description

struct of the body block of the info file

Definition at line **505** of file **f_nano_crypto_util.h**.

4.6.2 Field Documentation

4.6.2.1 body

```
F_NANO_WALLET_INFO_BODY body
```

Body of the file info.

Definition at line **517** of file **f_nano_crypto_util.h**.

4.6.2.2 desc

```
char desc[F_NANO_DESC_SZ]
```

Description.

Definition at line **511** of file **f_nano_crypto_util.h**.

4.6.2.3 file_info_integrity

```
uint8_t file_info_integrity[32]
```

File info integrity of the body block.

Definition at line **515** of file **f_nano_crypto_util.h**.

4.6.2.4 header

```
uint8_t header[sizeof(F_NANO_WALLET_INFO_MAGIC)]
```

Header magic.

Definition at line **507** of file **f_nano_crypto_util.h**.

4.6.2.5 nanoseed_hash

```
uint8_t nanoseed_hash[32]
```

Nano SEED hash file.

Definition at line **513** of file **f_nano_crypto_util.h**.

4.6.2.6 version

```
uint16_t version
```

Version.

Definition at line **509** of file **f_nano_crypto_util.h**.

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

- **f_nano_crypto_util.h**

Chapter 5

File Documentation

5.1 f_add_bn_288_le.h File Reference

```
#include <stdint.h>
```

Typedefs

- typedef uint8_t **F_ADD_288**[36]

5.1.1 Detailed Description

Low level implementation of Nano Cryptocurrency C library.

Definition in file **f_add_bn_288_le.h**.

5.1.2 Typedef Documentation

5.1.2.1 F_ADD_288

```
F_ADD_288
```

288 bit big number

Definition at line **19** of file **f_add_bn_288_le.h**.

5.2 f_add_bn_288_le.h

```
00001 /*
00002     AUTHOR: Fábio Pereira da Silva
00003     YEAR: 2019-20
00004     LICENSE: MIT
00005     EMAIL: fabioegel@gmail.com or fabioegel@protonmail.com
00006 */
00007
00008 #include <stdint.h>
00009
00019 typedef uint8_t F_ADD_288[36];
00020
00021
00022 #ifndef F_DOC_SKIP
00023
00033  void f_add_bn_288_le(F_ADD_288, F_ADD_288, F_ADD_288, int *, int);
00034  void f_sl_elv_add_le(F_ADD_288, int);
00035
00036 #endif
00037
```

5.3 f_nano_crypto_util.h File Reference

```
#include <stdint.h>
#include "f_util.h"
```

Data Structures

- struct **f_block_transfer_t**
- struct **f_nano_encrypted_wallet_t**
- struct **f_nano_crypto_wallet_t**
- struct **f_nano_wallet_info_bdy_t**
- struct **f_nano_wallet_info_t**

Macros

- #define **F_NANO_POW_MAX_THREAD** (size_t)10
- #define **MAX_STR_NANO_CHAR** (size_t)70
- #define **PUB_KEY_EXTENDED_MAX_LEN** (size_t)40
- #define **NANO_PREFIX** "nano_"
- #define **XRB_PREFIX** "xrb_"
- #define **NANO_ENCRYPTED_SEED_FILE** "/spiffs/secure/nano.nse"
- #define **NANO_PASSWD_MAX_LEN** (size_t)80
- #define **STR_NANO_SZ** (size_t)66
- #define **NANO_FILE_WALLETS_INFO** "/spiffs/secure/walletsinfo.i"
- #define **REP_XRB** (uint8_t)0x4
- #define **SENDER_XRB** (uint8_t)0x02
- #define **DEST_XRB** (uint8_t)0x01
- #define **F_BRAIN_WALLET_VERY_POOR** (uint32_t)0
- #define **F_BRAIN_WALLET_POOR** (uint32_t)1
- #define **F_BRAIN_WALLET_VERY_BAD** (uint32_t)2
- #define **F_BRAIN_WALLET_BAD** (uint32_t)3
- #define **F_BRAIN_WALLET_VERY_WEAK** (uint32_t)4
- #define **F_BRAIN_WALLET_WEAK** (uint32_t)5
- #define **F_BRAIN_WALLET_STILL_WEAK** (uint32_t)6
- #define **F_BRAIN_WALLET_MAYBE_GOOD** (uint32_t)7
- #define **F_BRAIN_WALLET_GOOD** (uint32_t)8
- #define **F_BRAIN_WALLET_VERY_GOOD** (uint32_t)9
- #define **F_BRAIN_WALLET_NICE** (uint32_t)10
- #define **F_BRAIN_WALLET_PERFECT** (uint32_t)11

Typedefs

- typedef uint8_t **NANO_SEED**[crypto_sign_SEEDBYTES]
- typedef uint8_t **f_uint128_t**[16]
- typedef uint8_t **NANO_PRIVATE_KEY**[sizeof(**NANO_SEED**)]
- typedef uint8_t **NANO_PRIVATE_KEY_EXTENDED**[crypto_sign_ed25519_SECRETKEYBYTES]
- typedef uint8_t **NANO_PUBLIC_KEY**[crypto_sign_ed25519_PUBLICKEYBYTES]
- typedef uint8_t **NANO_PUBLIC_KEY_EXTENDED**[**PUB_KEY_EXTENDED_MAX_LEN**]
- typedef enum **f_nano_err_t** **f_nano_err**
- typedef enum **f_write_seed_err_t** **f_write_seed_err**
- typedef enum **f_file_info_err_t** **F_FILE_INFO_ERR**

Enumerations

- enum **f_nano_err_t** {
  NANO_ERR_OK =0, **NANO_ERR_CANT_PARSE_BN_STR** =5151, **NANO_ERR_MALLOC**, **NANO_E↩RR_CANT_PARSE_FACTOR**,
  NANO_ERR_MPI_MULT, **NANO_ERR_CANT_PARSE_TO_BLK_TRANSFER**, **NANO_ERR_EMPTY_↩STR**, **NANO_ERR_CANT_PARSE_VALUE**,
  NANO_ERR_PARSE_MPI_TO_STR, **NANO_ERR_CANT_COMPLETE_NULL_CHAR**, **NANO_ERR_C↩ANT_PARSE_TO_MPI**, **NANO_ERR_INSUFICIENT_FUNDS**,
  NANO_ERR_SUB_MPI, **NANO_ERR_ADD_MPI**, **NANO_ERR_NO_SENSE_VALUE_TO_SEND_NEG↩ATIVE**, **NANO_ERR_NO_SENSE_VALUE_TO_SEND_ZERO**,
  NANO_ERR_NO_SENSE_BALANCE_NEGATIVE, **NANO_ERR_VAL_A_INVALID_MODE**, **NANO_ER↩R_CANT_PARSE_TO_TEMP_UINT128_T**, **NANO_ERR_VAL_B_INVALID_MODE**,
  NANO_ERR_CANT_PARSE_RAW_A_TO_MPI, **NANO_ERR_CANT_PARSE_RAW_B_TO_MPI**, **NAN↩O_ERR_UNKNOWN_ADD_SUB_MODE**, **NANO_ERR_INVALID_RES_OUTPUT** }
- enum **f_write_seed_err_t** {
  WRITE_ERR_OK =0, **WRITE_ERR_NULL_PASSWORD** =7180, **WRITE_ERR_EMPTY_STRING**, **WRI↩TE_ERR_MALLOC**,
  WRITE_ERR_ENCRYPT_PRIV_KEY, **WRITE_ERR_GEN_SUB_PRIV_KEY**, **WRITE_ERR_GEN_MAIN↩_PRIV_KEY**, **WRITE_ERR_ENCRYPT_SUB_BLOCK**,
  WRITE_ERR_UNKNOWN_OPTION, **WRITE_ERR_FILE_ALREDY_EXISTS**, **WRITE_ERR_CREATING↩_FILE**, **WRITE_ERR_WRITING_FILE** }
- enum **f_file_info_err_t** {
  F_FILE_INFO_ERR_OK =0, **F_FILE_INFO_ERR_CANT_OPEN_INFO_FILE** =7001, **F_FILE_INFO_ER↩R_NANO_SEED_ENCRYPTED_FILE_NOT_FOUND**, **F_FILE_INFO_ERR_CANT_DELETE_NANO_IN↩FO_FILE**,
  F_FILE_INFO_ERR_MALLOC, **F_FILE_INFO_ERR_CANT_READ_NANO_SEED_ENCRYPTED_FILE**, **F_FILE_INFO_ERR_CANT_READ_INFO_FILE**, **F_FILE_INFO_INVALID_HEADER_FILE**,
  F_FILE_INFO_ERR_INVALID_SHA256_INFO_FILE, **F_FILE_INFO_ERR_NANO_SEED_HASH_FAIL**, **F_FILE_INFO_ERR_NANO_INVALID_REPRESENTATIVE**, **F_FILE_INFO_ERR_NANO_INVALID_MA↩X_FEE_VALUE**,
  F_FILE_INFO_ERR_OPEN_FOR_WRITE_INFO, **F_FILE_INFO_ERR_EXISTING_FILE**, **F_FILE_INFO↩_ERR_CANT_WRITE_FILE_INFO** }

Functions

- struct **f_block_transfer_t** **__attribute__** ((packed)) F_BLOCK_TRANSFER
- int **f_cloud_crypto_wallet_nano_create_seed** (size_t, char *, char *)
- int **f_generate_nano_seed** (**NANO_SEED**, uint32_t)
- int **pk_to_wallet** (char *, char *, **NANO_PUBLIC_KEY_EXTENDED**)
- int **f_seed_to_nano_wallet** (**NANO_PRIVATE_KEY**, **NANO_PUBLIC_KEY**, **NANO_SEED**, uint32_t)
- char * **f_nano_key_to_str** (char *, unsigned char *)
- int **f_nano_seed_to_bip39** (char *, size_t, size_t *, **NANO_SEED**, char *)

- int **f_bip39_to_nano_seed** (uint8_t *, char *, char *)
- int **f_parse_nano_seed_and_bip39_to_JSON** (char *, size_t, size_t *, void *, int, const char *)
- int **f_read_seed** (uint8_t *, const char *, void *, int, int)
- int **f_nano_raw_to_string** (char *, size_t *, size_t, void *, int)
- int **f_nano_valid_nano_str_value** (const char *)
- int **valid_nano_wallet** (const char *)
- int **nano_base_32_2_hex** (uint8_t *, char *)
- int **f_nano_transaction_to_JSON** (char *, size_t, size_t *, **NANO_PRIVATE_KEY_EXTENDED**, F_BL↩ OCK_TRANSFER *)
- int **valid_raw_balance** (const char *)
- int **is_null_hash** (uint8_t *)
- int **is_nano_prefix** (const char *, const char *)
- **F_FILE_INFO_ERR f_get_nano_file_info** (F_NANO_WALLET_INFO *)
- **F_FILE_INFO_ERR f_set_nano_file_info** (F_NANO_WALLET_INFO *, int)
- **f_nano_err f_nano_value_compare_value** (void *, void *, uint32_t *)
- **f_nano_err f_nano_verify_nano_funds** (void *, void *, void *, uint32_t)
- **f_nano_err f_nano_parse_raw_str_to_raw128_t** (uint8_t *, const char *)
- **f_nano_err f_nano_parse_real_str_to_raw128_t** (uint8_t *, const char *)
- **f_nano_err f_nano_add_sub** (void *, void *, void *, uint32_t)
- int **f_nano_sign_block** (F_BLOCK_TRANSFER *, F_BLOCK_TRANSFER *, **NANO_PRIVATE_KEY_E↩ XTENDED**)
- **f_write_seed_err f_write_seed** (void *, int, uint8_t *, char *)
- **f_nano_err f_nano_balance_to_str** (char *, size_t, size_t *, **f_uint128_t**)
- int **f_extract_seed_from_brainwallet** (uint8_t *, char **, uint32_t, const char *, const char *)
- int **f_verify_work** (uint64_t *, const unsigned char *, uint64_t *, uint64_t)
- int **f_nano_pow** (uint64_t *, unsigned char *, const uint64_t, int)

Variables

- uint8_t **preamble** [32]
- uint8_t **account** [32]
- uint8_t **previous** [32]
- uint8_t **representative** [32]
- **f_uint128_t balance**
- uint8_t **link** [32]
- uint8_t **signature** [64]
- uint8_t **prefixes**
- uint64_t **work**
- uint8_t **sub_salt** [32]
- uint8_t **iv** [16]
- uint8_t **reserved** [16]
- uint8_t **hash_sk_unencrypted** [32]
- uint8_t **sk_encrypted** [32]
- uint8_t **nano_hdr** [sizeof(NANO_WALLET_MAGIC)]
- uint32_t **ver**
- uint8_t **description** [F_DESC_SZ]
- uint8_t **salt** [32]
- F_ENCRYPTED_BLOCK **seed_block**
- uint8_t **wallet_prefix**
- uint32_t **last_used_wallet_number**
- char **wallet_representative** [**MAX_STR_NANO_CHAR**]
- char **max_fee** [F_RAW_STR_MAX_SZ]
- uint8_t **header** [sizeof(F_NANO_WALLET_INFO_MAGIC)]
- uint16_t **version**
- char **desc** [F_NANO_DESC_SZ]
- uint8_t **nanoseed_hash** [32]
- uint8_t **file_info_integrity** [32]
- F_NANO_WALLET_INFO_BODY **body**

5.3.1 Detailed Description

This API Integrates Nano Cryptocurrency to low computational devices.

Definition in file **f_nano_crypto_util.h**.

5.3.2 Macro Definition Documentation

5.3.2.1 DEST_XRB

```
#define DEST_XRB (uint8_t)0x01
```

Definition at line **427** of file **f_nano_crypto_util.h**.

5.3.2.2 F_BRAIN_WALLET_BAD

```
#define F_BRAIN_WALLET_BAD (uint32_t)3
```

[bad].

Crack within one day

Definition at line **1025** of file **f_nano_crypto_util.h**.

5.3.2.3 F_BRAIN_WALLET_GOOD

```
#define F_BRAIN_WALLET_GOOD (uint32_t)8
```

[good].

Crack within one thousand year

Definition at line **1056** of file **f_nano_crypto_util.h**.

5.3.2.4 F_BRAIN_WALLET_MAYBE_GOOD

```
#define F_BRAIN_WALLET_MAYBE_GOOD (uint32_t)7
```

[maybe good for you].

Crack within one century

Definition at line **1049** of file **f_nano_crypto_util.h**.

5.3.2.5 F_BRAIN_WALLET_NICE

```
#define F_BRAIN_WALLET_NICE (uint32_t)10
```

[very nice].

Crack withing one hundred thousand year

Definition at line **1068** of file **f_nano_crypto_util.h**.

5.3.2.6 F_BRAIN_WALLET_PERFECT

```
#define F_BRAIN_WALLET_PERFECT (uint32_t)11
```

[Perfect!] 3.34x10^53 Years to crack

Definition at line **1074** of file **f_nano_crypto_util.h**.

5.3.2.7 F_BRAIN_WALLET_POOR

```
#define F_BRAIN_WALLET_POOR (uint32_t)1
```

[poor].

Crack within minutes

Definition at line **1013** of file **f_nano_crypto_util.h**.

5.3.2.8 F_BRAIN_WALLET_STILL_WEAK

```
#define F_BRAIN_WALLET_STILL_WEAK (uint32_t)6
```

[still weak].

Crack within one year

Definition at line **1043** of file **f_nano_crypto_util.h**.

5.3.2.9 F_BRAIN_WALLET_VERY_BAD

```
#define F_BRAIN_WALLET_VERY_BAD (uint32_t)2
```

[very bad].

Crack within one hour

Definition at line **1019** of file **f_nano_crypto_util.h**.

5.3.2.10 F_BRAIN_WALLET_VERY_GOOD

```
#define F_BRAIN_WALLET_VERY_GOOD (uint32_t)9
```

[very good].

Crack within ten thousand year

Definition at line **1062** of file **f_nano_crypto_util.h**.

5.3.2.11 F_BRAIN_WALLET_VERY_POOR

```
#define F_BRAIN_WALLET_VERY_POOR (uint32_t)0
```

[very poor].

Crack within seconds or less

Definition at line **1007** of file **f_nano_crypto_util.h**.

5.3.2.12 F_BRAIN_WALLET_VERY_WEAK

```
#define F_BRAIN_WALLET_VERY_WEAK (uint32_t)4
```

[very weak].

Crack within one week

Definition at line **1031** of file **f_nano_crypto_util.h**.

5.3.2.13 F_BRAIN_WALLET_WEAK

```
#define F_BRAIN_WALLET_WEAK (uint32_t)5
```

[weak].

Crack within one month

Definition at line **1037** of file **f_nano_crypto_util.h**.

5.3.2.14 F_NANO_POW_MAX_THREAD

```
#define F_NANO_POW_MAX_THREAD (size_t)10
```

Definition at line **136** of file **f_nano_crypto_util.h**.

5.3.2.15 MAX_STR_NANO_CHAR

```
#define MAX_STR_NANO_CHAR (size_t)70
```

(desktop only) Number of threads for Proof of Work routines.

Defines a max size of Nano char (70 bytes)

Default 10

Definition at line **148** of file **f_nano_crypto_util.h**.

5.3.2.16 NANO_ENCRYPTED_SEED_FILE

```
#define NANO_ENCRYPTED_SEED_FILE "/spiffs/secure/nano.nse"
```

Path to non deterministic encrypted file with password.

File containing the SEED of the Nano wallets generated by TRNG (if available in your Hardware) or PRNG.
Default name: "nano.nse"

Definition at line **190** of file **f_nano_crypto_util.h**.

5.3.2.17 NANO_FILE_WALLETS_INFO

```
#define NANO_FILE_WALLETS_INFO "/spiffs/secure/walletsinfo.i"
```

Custom information file path about Nano SEED wallet stored in "walletsinfo.i".

Definition at line **208** of file **f_nano_crypto_util.h**.

5.3.2.18 NANO_PASSWD_MAX_LEN

```
#define NANO_PASSWD_MAX_LEN (size_t)80
```

Password max length.

Definition at line **196** of file **f_nano_crypto_util.h**.

5.3.2.19 NANO_PREFIX

```
#define NANO_PREFIX "nano_"
```

Nano prefix.

Definition at line **160** of file **f_nano_crypto_util.h**.

5.3.2.20 PUB_KEY_EXTENDED_MAX_LEN

```
#define PUB_KEY_EXTENDED_MAX_LEN (size_t)40
```

Max size of public key (extended)

Definition at line **154** of file **f_nano_crypto_util.h**.

5.3.2.21 REP_XRB

```
#define REP_XRB (uint8_t)0x4
```

Representative XRB flag.

Destination XRB flag.

Sender XRB flag.

5.3.2.22 SENDER_XRB

```
#define SENDER_XRB (uint8_t)0x02
```

Definition at line **421** of file **f_nano_crypto_util.h**.

5.3.2.23 STR_NANO_SZ

```
#define STR_NANO_SZ (size_t)66
```

String size of Nano encoded Base32 including NULL char.

Definition at line **202** of file **f_nano_crypto_util.h**.

5.3.2.24 XRB_PREFIX

```
#define XRB_PREFIX "xrb_"
```

XRB (old Raiblocks) prefix.

Definition at line **166** of file **f_nano_crypto_util.h**.

5.3.3 Typedef Documentation

5.3.3.1 F_FILE_INFO_ERR

F_FILE_INFO_ERR

Typedef Error enumerator for info file functions.

5.3.3.2 f_nano_err

f_nano_err

Error function enumerator.

See also

f_nano_err_t (p. **??**)

5.3.3.3 f_uint128_t

```
f_uint128_t
```

128 bit big number of Nano balance

Definition at line **220** of file **f_nano_crypto_util.h**.

5.3.3.4 f_write_seed_err

```
typedef enum f_write_seed_err_t f_write_seed_err
```

5.3.3.5 NANO_PRIVATE_KEY

```
NANO_PRIVATE_KEY
```

Size of Nano Private Key.

Definition at line **230** of file **f_nano_crypto_util.h**.

5.3.3.6 NANO_PRIVATE_KEY_EXTENDED

```
NANO_PRIVATE_KEY_EXTENDED
```

Size of Nano Private Key extended.

Definition at line **236** of file **f_nano_crypto_util.h**.

5.3.3.7 NANO_PUBLIC_KEY

```
NANO_PUBLIC_KEY
```

Size of Nano Public Key.

Definition at line **242** of file **f_nano_crypto_util.h**.

5.3.3.8 NANO_PUBLIC_KEY_EXTENDED

`NANO_PUBLIC_KEY_EXTENDED`

Size of Public Key Extended.

Definition at line **248** of file **f_nano_crypto_util.h**.

5.3.3.9 NANO_SEED

`NANO_SEED`

Size of Nano SEED.

Definition at line **214** of file **f_nano_crypto_util.h**.

5.3.4 Enumeration Type Documentation

5.3.4.1 f_file_info_err_t

`enum` **f_file_info_err_t**

Enumerator

F_FILE_INFO_ERR_OK	SUCCESS.
F_FILE_INFO_ERR_CANT_OPEN_INFO_FILE	Can't open info file.
F_FILE_INFO_ERR_NANO_SEED_ENCRYPTED_FILE_NO↩ T_FOUND	Encrypted file with Nano SEED not found.
F_FILE_INFO_ERR_CANT_DELETE_NANO_INFO_FILE	Can not delete Nano info file.
F_FILE_INFO_ERR_MALLOC	Fatal Error MALLOC.
F_FILE_INFO_ERR_CANT_READ_NANO_SEED_ENCRYP↩ TED_FILE	Can not read encrypted Nano SEED in file.
F_FILE_INFO_ERR_CANT_READ_INFO_FILE	Can not read info file.
F_FILE_INFO_INVALID_HEADER_FILE	Invalid info file header.
F_FILE_INFO_ERR_INVALID_SHA256_INFO_FILE	Invalid SHA256 info file.
F_FILE_INFO_ERR_NANO_SEED_HASH_FAIL	Nano SEED hash failed.
F_FILE_INFO_ERR_NANO_INVALID_REPRESENTATIVE	Invalid representative.
F_FILE_INFO_ERR_NANO_INVALID_MAX_FEE_VALUE	Invalid max fee value.
F_FILE_INFO_ERR_OPEN_FOR_WRITE_INFO	Can not open info file for write.
F_FILE_INFO_ERR_EXISTING_FILE	Error File Exists.
F_FILE_INFO_ERR_CANT_WRITE_FILE_INFO	Can not write info file.

Definition at line **533** of file **f_nano_crypto_util.h**.

5.3.4.2 f_nano_err_t

enum **f_nano_err_t**

Enumerator

NANO_ERR_OK	SUCCESS.
NANO_ERR_CANT_PARSE_BN_STR	Can not parse string big number.
NANO_ERR_MALLOC	Fatal ERROR MALLOC.
NANO_ERR_CANT_PARSE_FACTOR	Can not parse big number factor.
NANO_ERR_MPI_MULT	Error multiplication MPI.
NANO_ERR_CANT_PARSE_TO_BLK_TRANSFER	Can not parse to block transfer.
NANO_ERR_EMPTY_STR	Error empty string.
NANO_ERR_CANT_PARSE_VALUE	Can not parse value.
NANO_ERR_PARSE_MPI_TO_STR	Can not parse MPI to string.
NANO_ERR_CANT_COMPLETE_NULL_CHAR	Can not complete NULL char.
NANO_ERR_CANT_PARSE_TO_MPI	Can not parse to MPI.
NANO_ERR_INSUFICIENT_FUNDS	Insuficient funds.
NANO_ERR_SUB_MPI	Error subtract MPI.
NANO_ERR_ADD_MPI	Error add MPI.
NANO_ERR_NO_SENSE_VALUE_TO_SEND_NEGATIVE	Does not make sense send negativative balance.
NANO_ERR_NO_SENSE_VALUE_TO_SEND_ZERO	Does not make sense send empty value.
NANO_ERR_NO_SENSE_BALANCE_NEGATIVE	Does not make sense negative balance.
NANO_ERR_VAL_A_INVALID_MODE	Invalid A mode value.
NANO_ERR_CANT_PARSE_TO_TEMP_UINT128_T	Can not parse temporary memory to uint_128_t.
NANO_ERR_VAL_B_INVALID_MODE	Invalid A mode value.
NANO_ERR_CANT_PARSE_RAW_A_TO_MPI	Can not parse raw A value to MPI.
NANO_ERR_CANT_PARSE_RAW_B_TO_MPI	Can not parse raw B value to MPI.
NANO_ERR_UNKNOWN_ADD_SUB_MODE	Unknown ADD/SUB mode.
NANO_ERR_INVALID_RES_OUTPUT	Invalid output result.

Definition at line **292** of file **f_nano_crypto_util.h**.

5.3.4.3 f_write_seed_err_t

enum **f_write_seed_err_t**

Enumerator

WRITE_ERR_OK	Error SUCCESS.
WRITE_ERR_NULL_PASSWORD	Error NULL password.
WRITE_ERR_EMPTY_STRING	Empty string.
WRITE_ERR_MALLOC	Error MALLOC.
WRITE_ERR_ENCRYPT_PRIV_KEY	Error encrypt private key.
WRITE_ERR_GEN_SUB_PRIV_KEY	Can not generate sub private key.
WRITE_ERR_GEN_MAIN_PRIV_KEY	Can not generate main private key.
WRITE_ERR_ENCRYPT_SUB_BLOCK	Can not encrypt sub block.

Enumerator

WRITE_ERR_UNKNOWN_OPTION	Unknown option.
WRITE_ERR_FILE_ALREDY_EXISTS	File already exists.
WRITE_ERR_CREATING_FILE	Can not create file.
WRITE_ERR_WRITING_FILE	Can not write file.

Definition at line **429** of file **f_nano_crypto_util.h**.

5.3.5 Function Documentation

5.3.5.1 __attribute__()

```
struct  f_nano_wallet_info_t __attribute__ (
              (packed)  )
```

5.3.5.2 f_bip39_to_nano_seed()

```
int f_bip39_to_nano_seed (
             uint8_t * seed,
             char * str,
             char * dictionary )
```

Parse Nano Bip39 encoded string to raw Nano SEED given a dictionary file.

Parameters

out	*seed*	Nano SEED
in	*str*	A encoded Bip39 string pointer
in	*dictionary*	A string pointer path to file

WARNING Sensive data. Do not share any SEED or Bip39 encoded string !

Return values

0	On Success, otherwise Error

See also

f_nano_seed_to_bip39() (p. **??**)

5.3.5.3 f_cloud_crypto_wallet_nano_create_seed()

```
int f_cloud_crypto_wallet_nano_create_seed (
            size_t entropy,
            char * file_name,
            char * password )
```

Generates a new SEED and saves it to an non deterministic encrypted file.

password is mandatory

Parameters

in	*entropy*	Entropy type. Entropy type are: F_ENTROPY_TYPE_PARANOIC F_ENTROPY_TYPE_EXCELENT F_ENTROPY_TYPE_GOOD F_ENTROPY_TYPE_NOT_ENOUGH F_ENTROPY_TYPE_NOT_RECOMENDED
in	*file_name*	The file and path to be stored in your file system directory. It can be *NULL*. If you parse a *NULL* value then file will be stored in *NANO_ENCRYPTED_SEED_FILE* variable file system pointer.
in	*password*	Password of the encrypted file. It can NOT be *NULL* or EMPTY

WARNING

f_cloud_crypto_wallet_nano_create_seed() *(*p. **??***)* does not verify your password. It is recommended to use a strong password like symbols, capital letters and numbers to keep your SEED safe and avoid brute force attacks.

You can use ***f_pass_must_have_at_least()*** *(*p. **??***)* function to check passwords strength

Return values

0	On Success, otherwise Error

5.3.5.4 f_extract_seed_from_brainwallet()

```
int f_extract_seed_from_brainwallet (
            uint8_t * seed,
            char ** warning_msg,
            uint32_t allow_mode,
            const char * brainwallet,
            const char * salt )
```

Analyzes a text given a *mode* and if pass then the text in *braiwallet* is translated to a Nano SEED.

Parameters

out	*seed*	Output Nano SEED extracted from *brainwallet*

Parameters

out	*warning_msg*	Warning message parsed to application. It can be NULL
in	*allow_mode*	Allow *mode*. Funtion will return SUCCESS only if permitted mode set by user Allow mode are: • *F_BRAIN_WALLET_VERY_POOR* Crack within seconds or less • *F_BRAIN_WALLET_POOR* Crack within minutes • *F_BRAIN_WALLET_VERY_BAD* Crack within one hour • *F_BRAIN_WALLET_BAD* Crack within one day • *F_BRAIN_WALLET_VERY_WEAK* Crack within one week • *F_BRAIN_WALLET_WEAK* Crack within one month • *F_BRAIN_WALLET_STILL_WEAK* Crack within one year • *F_BRAIN_WALLET_MAYBE_GOOD* Crack within one century • *F_BRAIN_WALLET_GOOD* Crack within one thousand year • *F_BRAIN_WALLET_VERY_GOOD* Crack within ten thousand year • *F_BRAIN_WALLET_NICE* Crack withing one hundred thousand year • *F_BRAIN_WALLET_PERFECT* 3.34x10^53 Years to crack
in	*brainwallet*	Brainwallet text to be parsed. It can be NOT NULL or null string
in	*salt*	Salt of the Braiwallet. It can be NOT NULL or null string

Return values

0	If success, otherwise error.

See also

f_bip39_to_nano_seed() (p. **??**)

5.3.5.5 f_generate_nano_seed()

```
int f_generate_nano_seed (
            NANO_SEED seed,
           uint32_t entropy )
```

Generates a new SEED and stores it to *seed* pointer.

Parameters

out	*seed*	SEED generated in system PRNG or TRNG

Parameters

in	*entropy*	Entropy type. Entropy type are: F_ENTROPY_TYPE_PARANOIC F_ENTROPY_TYPE_EXCELENT F_ENTROPY_TYPE_GOOD F_ENTROPY_TYPE_NOT_ENOUGH F_ENTROPY_TYPE_NOT_RECOMENDED

Return values

0	On Success, otherwise Error

5.3.5.6 f_get_nano_file_info()

```
F_FILE_INFO_ERR f_get_nano_file_info (
            F_NANO_WALLET_INFO * info )
```

Opens default file *walletsinfo.i* (if exists) containing information *F_NANO_WALLET_INFO* structure and parsing to pointer *info* if success.

Parameters

out	*info*	Pointer to buffer to be parsed struct from *$PATH/walletsinfo.i* file.

Return values

F_FILE_INFO_ERR_OK	If Success, otherwise *F_FILE_INFO_ERR* enum type error

See also

F_FILE_INFO_ERR (p. **??**) enum type error for detailed error and **f_nano_wallet_info_t** (p. **??**) for info type details

5.3.5.7 f_nano_add_sub()

```
f_nano_err f_nano_add_sub (
            void * res,
            void * valA,
            void * valB,
            uint32_t mode )
```

Add/Subtract two Nano balance values and stores value in *res*

Parameters

out	*res*	Result value res = valA + valB or res = valA - valB
in	*valA*	Input balance A value
in	*valB*	Input balance B value
in	*mode*	Mode type: • *F_NANO_ADD_A_B* valA + valB • *F_NANO_SUB_A_B* valA - valB • *F_NANO_RES_RAW_128* Output is a raw data 128 bit big number result • *F_NANO_RES_RAW_STRING* Output is a 128 bit Big Integer string • *F_NANO_RES_REAL_STRING* Output is a Real string value • *F_NANO_A_RAW_128* if *balance* is big number raw buffer type • *F_NANO_A_RAW_STRING* if *balance* is big number raw string type • *F_NANO_A_REAL_STRING* if *balance* is real number string type • *F_NANO_B_RAW_128* if *value_to_send* is big number raw buffer type • *F_NANO_B_RAW_STRING* if *value_to_send* is big number raw string type • *F_NANO_B_REAL_STRING* if *value_to_send* is real number string type

Return values

NANO_ERR_OK	If Success, otherwise f_nano_err_t enum type error

See also

f_nano_err_t (p. **??**) for **f_nano_err** (p. **??**) enum error type

5.3.5.8 f_nano_balance_to_str()

```
f_nano_err f_nano_balance_to_str (
            char * str,
            size_t str_len,
            size_t * out_len,
             f_uint128_t value )
```

Converts a raw Nano balance to string raw balance.

Parameters

out	*str*	Output string pointer
in	*str_len*	Size of string pointer memory
out	*out_len*	Output length of converted value to string. If *out_len* is NULL then *str* returns converted value with NULL terminated string
in	*value*	Raw Nano balance value

Return values

0	If success, otherwise error.

See also

function **f_nano_parse_raw_str_to_raw128_t()** (p. **??**) and return errors **f_nano_err** (p. **??**)

5.3.5.9 f_nano_key_to_str()

```
char * f_nano_key_to_str (
            char * out,
            unsigned char * key )
```

Parse a raw binary public key to string.

Parameters

out	*out*	Pointer to outuput string
in	*in*	Pointer to raw public key

Returns

A pointer to output string

5.3.5.10 f_nano_parse_raw_str_to_raw128_t()

```
f_nano_err f_nano_parse_raw_str_to_raw128_t (
            uint8_t * res,
            const char * raw_str_value )
```

Parse a raw string balance to raw big number 128 bit.

Parameters

out	*res*	Binary raw balance
in	*raw_str_value*	Raw balance string

Return values

NANO_ERR_OK	If Success, otherwise f_nano_err_t enum type error

See also

f_nano_err_t (p. **??**) for **f_nano_err** (p. **??**) enum error type

5.3.5.11 f_nano_parse_real_str_to_raw128_t()

```
f_nano_err f_nano_parse_real_str_to_raw128_t (
            uint8_t * res,
            const char * real_str_value )
```

Parse a real string balance to raw big number 128 bit.

Parameters

out	*res*	Binary raw balance
in	*real_str_value*	Real balance string

Return values

NANO_ERR_OK	If Success, otherwise f_nano_err_t enum type error

See also

f_nano_err_t (p. **??**) for **f_nano_err** (p. **??**) enum error type

5.3.5.12 f_nano_pow()

```
int f_nano_pow (
            uint64_t * PoW_res,
            unsigned char * hash,
            const uint64_t threshold,
            int n_thr )
```

Calculates a Proof of Work given a *hash*, *threshold* and number of threads *n_thr*

Parameters

out	*PoW_res*	Output Proof of Work
in	*hash*	Input *hash*
in	*threshold*	Input *threshold*
in	*n_thr*	Number of threads. Default maximum value: 10. You can modify *F_NANO_POW_MAX_THREAD* in **f_nano_crypto_util.h** (p. **??**)

Return values

0	If success, otherwise error.

See also

f_verify_work() (p. **??**)

5.3.5.13 f_nano_raw_to_string()

```
int f_nano_raw_to_string (
            char * str,
            size_t * olen,
            size_t str_sz,
            void * raw,
            int raw_type )
```

Converts Nano raw balance [string | f_uint128_t] to real string value.

Parameters

out	*str*	Output real string value
out	*olen*	Size of output real string value. It can be NULL. If NULL output *str* will have a NULL char at the end.
in	*str_sz*	Size of *str* buffer
in	*raw*	Raw balance.
in	*raw_type*	Raw balance type: • F_RAW_TO_STR_UINT128 for raw **f_uint128_t** balance • F_RAW_TO_STR_STRING for raw **char** balance

Return values

0	On Success, otherwise Error

See also

f_nano_valid_nano_str_value() (p. **??**)

5.3.5.14 f_nano_seed_to_bip39()

```
int f_nano_seed_to_bip39 (
            char * buf,
            size_t buf_sz,
```

```
            size_t * out_buf_len,
             NANO_SEED seed,
            char * dictionary_file )
```

Parse Nano SEED to Bip39 encoding given a dictionary file.

Parameters

out	*buf*	Output string containing encoded Bip39 SEED
in	*buf_sz*	Size of memory of buf pointer
out	*out_buf_len*	If *out_buf_len* is NOT NULL then *out_buf_len* returns the size of string encoded Bip39 and *out* with non NULL char. If *out_buf_len* is NULL then *out* has a string encoded Bip39 with a NULL char.
in	*seed*	Nano SEED
in	*dictionary_file*	Path to dictionary file

WARNING Sensive data. Do not share any SEED or Bip39 encoded string !

Return values

0	On Success, otherwise Error

See also

f_bip39_to_nano_seed() (p. **??**)

5.3.5.15 f_nano_sign_block()

```
int f_nano_sign_block (
            F_BLOCK_TRANSFER * user_block,
            F_BLOCK_TRANSFER * fee_block,
             NANO_PRIVATE_KEY_EXTENDED private_key )
```

Signs *user_block* and worker *fee_block* given a private key *private_key*

Parameters

in,out	*user_block*	User block to be signed with a private key *private_key*
in,out	*fee_block*	Fee block to be signed with a private key *private_key*. Can be NULL if worker does not require fee
in	*private_key*	Private key to sign block(s)

Return values

0	If Success, otherwise error

See also

f_nano_transaction_to_JSON() (p. **??**)

5.3.5.16 f_nano_transaction_to_JSON()

```
int f_nano_transaction_to_JSON (
            char * str,
            size_t str_len,
            size_t * str_out,
             NANO_PRIVATE_KEY_EXTENDED private_key,
            F_BLOCK_TRANSFER * block_transfer )
```

Sign a block pointed in *block_transfer* with a given *private_key* and stores signed block to *block_transfer* and parse to JSON Nano RPC.

Parameters

out	*str*	A string pointer to store JSON Nano RPC
in	*str_len*	Size of buffer in *str* pointer
out	*str_out*	Size of JSON string. *str_out* can be NULL
in	*private_key*	Private key to sign the block *block_transfer*
in,out	*block_transfer*	Nano block containing raw data to be stored in Nano Blockchain

WARNING Sensive data. Do not share any PRIVATE KEY

Return values

0	On Success, otherwise Error

5.3.5.17 f_nano_valid_nano_str_value()

```
int f_nano_valid_nano_str_value (
            const char * str )
```

Check if a real string or raw string are valid Nano balance.

Parameters

in	*str*	Value to be checked

Return values

0	If valid, otherwise is invalid

See also

f_nano_raw_to_string() (p. **??**)

5.3.5.18 f_nano_value_compare_value()

```
f_nano_err f_nano_value_compare_value (
            void * valA,
            void * valB,
            uint32_t * mode_compare )
```

Comparare two Nano balance.

Parameters

in	*valA*	Nano balance value A
in	*valB*	Nano balance value B
in,out	*mode_compare*	Input mode and output result Input mode: • *F_NANO_A_RAW_128* if *valA* is big number raw buffer type • *F_NANO_A_RAW_STRING* if *valA* is big number raw string type • *F_NANO_A_REAL_STRING* if *valA* is real number string type • *F_NANO_B_RAW_128* if *valB* is big number raw buffer type • *F_NANO_B_RAW_STRING* if *valB* is big number raw string type • *F_NANO_B_REAL_STRING* if *valB* is real number string type Output type: • *F_NANO_COMPARE_EQ* If *valA* is greater than *valB* • *F_NANO_COMPARE_LT* if *valA* is lesser than *valB* • *F_NANO_COMPARE_LEQ* if *valA* is lesser or equal than *valB* • *F_NANO_COMPARE_GT* if *valA* is greater than *valB* • *F_NANO_COMPARE_GEQ* If *valA* is greater or equal than *valB*

Return values

NANO_ERR_OK	If Success, otherwise f_nano_err_t enum type error

See also

f_nano_err_t (p. **??**) for **f_nano_err** (p. **??**) enum error type

5.3.5.19 f_nano_verify_nano_funds()

```
f_nano_err f_nano_verify_nano_funds (
            void * balance,
            void * value_to_send,
            void * fee,
            uint32_t mode )
```

Check if Nano balance has sufficient funds.

Parameters

in	*balance*	Nano balance
in	*value_to_send*	Value to send
in	*fee*	Fee value (it can be NULL)
in	*mode*	Value type mode • *F_NANO_A_RAW_128* if *balance* is big number raw buffer type • *F_NANO_A_RAW_STRING* if *balance* is big number raw string type • *F_NANO_A_REAL_STRING* if *balance* is real number string type • *F_NANO_B_RAW_128* if *value_to_send* is big number raw buffer type • *F_NANO_B_RAW_STRING* if *value_to_send* is big number raw string type • *F_NANO_B_REAL_STRING* if *value_to_send* is real number string type • *F_NANO_C_RAW_128* if *fee* is big number raw buffer type (can be ommited if *fee* is NULL) • *F_NANO_C_RAW_STRING* if *fee* is big number raw string type (can be ommited if *fee* is NULL) • *F_NANO_C_REAL_STRING* if *fee* is real number string type (can be ommited if *fee* is NULL)

Return values

NANO_ERR_OK	If Success, otherwise f_nano_err_t enum type error

See also

f_nano_err_t (p. **??**) for **f_nano_err** (p. **??**) enum error type

5.3.5.20 f_parse_nano_seed_and_bip39_to_JSON()

```
int f_parse_nano_seed_and_bip39_to_JSON (
            char * dest,
            size_t dest_sz,
            size_t * olen,
            void * source_data,
```

```
            int source,
            const char * password )
```

Parse Nano SEED and Bip39 to JSON given a encrypted data in memory or encrypted data in file or unencrypted seed in memory.

Parameters

out	*dest*	Destination JSON string pointer
in	*dest_sz*	Buffer size of *dest* pointer
out	*olen*	Size of the output JSON string. If NULL string JSON returns a NULL char at the end of string otherwise it will return the size of the string is stored into *olen* variable without NULL string in *dest*
in	*source_data*	Input data source (encrypted file | encrypted data in memory | unencrypted seed in memory)
in	*source*	Source data type: • PARSE_JSON_READ_SEED_GENERIC: If seed are in memory pointed in *source_data*. Password is ignored. Can be NULL. • READ_SEED_FROM_STREAM: Read encrypted data from stream pointed in *source_data*. Password is required. • READ_SEED_FROM_FILE: Read encrypted data stored in a file where *source_data* is path to file. Password is required.
in	*password*	Required for READ_SEED_FROM_STREAM and READ_SEED_FROM_FILE sources

WARNING Sensive data. Do not share any SEED or Bip39 encoded string !

Return values

0	On Success, otherwise Error

See also

f_read_seed() (p. **??**)

5.3.5.21 f_read_seed()

```
int f_read_seed (
            uint8_t * seed,
            const char * passwd,
            void * source_data,
            int force_read,
            int source )
```

Extracts a Nano SEED from encrypted stream in memory or in a file.

Parameters

out	*seed*	Output Nano SEED
in	*passwd*	Password (always required)
in	*source_data*	Encrypted source data from memory or path pointed in *source_data*
in	*force_read*	If non zero value then forces reading from a corrupted file. This param is ignored when reading *source_data* from memory
in	*source*	Source data type: • READ_SEED_FROM_STREAM: Read encrypted data from stream pointed in *source_data*. Password is required. • READ_SEED_FROM_FILE: Read encrypted data stored in a file where *source_data* is path to file. Password is required.

WARNING Sensive data. Do not share any SEED !

Return values

0	On Success, otherwise Error

See also

f_parse_nano_seed_and_bip39_to_JSON() (p. **??**) **f_write_seed()** (p. **??**)

5.3.5.22 f_seed_to_nano_wallet()

```
int f_seed_to_nano_wallet (
              NANO_PRIVATE_KEY private_key,
              NANO_PUBLIC_KEY public_key,
              NANO_SEED seed,
             uint32_t wallet_number )
```

Extracts one key pair from Nano SEED given a wallet number.

Parameters

out	*private_key*	Private key of the *wallet_number* from given *seed*
out	*public_key*	Public key of the *wallet_number* from given *seed*
in,out	*seed*	Nano SEED
in	*wallet_number*	Wallet number of key pair to be extracted from Nano SEED

WARNING 1:

- Seed must be read from memory
- Seed is destroyed when extracting public and private keys

WARNING 2:

- Never expose SEED and private key. This function destroys seed and any data after execution and finally parse public and private keys to output.

Return values

0	On Success, otherwise Error

5.3.5.23 f_set_nano_file_info()

```
F_FILE_INFO_ERR f_set_nano_file_info (
            F_NANO_WALLET_INFO * info,
            int overwrite_existing_file )
```

Saves wallet information stored at buffer struct *info* to file *walletsinfo.i*

Parameters

in	*info*	Pointer to data to be saved at *$PATH/walletsinfo.i* file.
in	*overwrite_existing_file*	If non zero then overwrites file *$PATH/walletsinfo.i*

Return values

F_FILE_INFO_ERR_OK	If Success, otherwise *F_FILE_INFO_ERR* enum type error

See also

F_FILE_INFO_ERR (p. **??**) enum type error for detailed error and **f_nano_wallet_info_t** (p. **??**) for info type details

5.3.5.24 f_verify_work()

```
int f_verify_work (
            uint64_t * result,
            const unsigned char * hash,
            uint64_t * work,
            uint64_t threshold )
```

Verifies if Proof of Work of a given *hash* is valid.

Parameters

out	*result*	Result of work. It can be NULL
in	*hash*	Input *hash* for verification
in	*work*	Work previously calculated to be checked
in	*threshold*	Input *threshold*

Return values

0	If is not valid or less than zero if error or greater than zero if is valid

See also

f_nano_pow() (p. **??**)

5.3.5.25 f_write_seed()

```
f_write_seed_err f_write_seed (
            void * source_data,
            int source,
            uint8_t * seed,
            char * passwd )
```

Writes a SEED into a ecrypted with password with non deterministic stream in memory or file.

Parameters

out	*source_data*	Memory pointer or file name
in	*source*	Source of output data: • *WRITE_SEED_TO_STREAM* Output data is a pointer to memory to store encrypted Nano SEED data • *WRITE_SEED_TO_FILE* Output is a string filename to store encrypted Nano SEED data
in	*seed*	Nano SEED to be stored in encrypted stream or file
in	*passwd*	(Mandatory) It can not be null string or NULL. See ***f_pass_must_have_at_least()*** *(*p. **??***)* function to check passwords strength

Return values

0	If Success, otherwise error

See also

f_read_seed() (p. **??**)

5.3.5.26 is_nano_prefix()

```
int is_nano_prefix (
            const char * nano_wallet,
            const char * prefix )
```

Checks *prefix* in *nano_wallet*

Parameters

in	*nano_wallet*	Base32 Nano wallet encoded string
in	*prefix*	Prefix type • NANO_PREFIX for nano_ • XRB_PREFIX for xrb_

Return values

1	If *prefix* in *nano_wallet*, otherwise 0

5.3.5.27 is_null_hash()

```
int is_null_hash (
            uint8_t * hash )
```

Check if 32 bytes hash is filled with zeroes.

Parameters

in	*hash*	32 bytes binary *hash*

Return values

1	If zero filled buffer, otherwise 0

5.3.5.28 nano_base_32_2_hex()

```
int nano_base_32_2_hex (
            uint8_t * res,
            char * str_wallet )
```

Parse Nano Base32 wallet string to public key binary.

Parameters

out	*res*	Output raw binary public key
in	*str_wallet*	Valid Base32 encoded Nano string to be parsed

Return values

0	On Success, otherwise Error

See also

pk_to_wallet() (p. **??**)

5.3.5.29 pk_to_wallet()

```
int pk_to_wallet (
            char * out,
            char * prefix,
             NANO_PUBLIC_KEY_EXTENDED pubkey_extended )
```

Parse a Nano public key to Base32 Nano wallet string.

Parameters

out	*out*	Output string containing the wallet
in	*prefix*	Nano prefix. *NANO_PREFIX* for nano_ *XRB_PREFIX* for xrb_
in,out	*pubkey_extended*	Public key to be parsed to string

WARNING: *pubkey_extended* is destroyed when parsing to Nano base32 encoding

Return values

0	On Success, otherwise Error

See also

nano_base_32_2_hex() (p. **??**)

5.3.5.30 valid_nano_wallet()

```
int valid_nano_wallet (
            const char * wallet )
```

Check if a string containing a Base32 Nano wallet is valid.

Parameters

in	*wallet*	Base32 Nano wallet encoded string

Return values

0	If valid wallet otherwise is invalid

5.3.5.31 valid_raw_balance()

```
int valid_raw_balance (
            const char * balance )
```

Checks if a string buffer pointed in *balance* is a valid raw balance.

Parameters

in	*balance*	Pointer containing a string buffer

Return values

0	On Success, otherwise Error

5.3.6 Variable Documentation

5.3.6.1 account

```
uint8_t account[32]
```

Account in raw binary data.

Definition at line **252** of file **f_nano_crypto_util.h**.

5.3.6.2 balance

```
f_uint128_t balance
```

Big number 128 bit raw balance.

See also

f_uint128_t (p. **??**)

Definition at line **260** of file **f_nano_crypto_util.h**.

5.3.6.3 body

```
F_NANO_WALLET_INFO_BODY body
```

Body of the file info.

Definition at line **260** of file **f_nano_crypto_util.h**.

5.3.6.4 desc

```
char desc[F_NANO_DESC_SZ]
```

Description.

Definition at line **254** of file **f_nano_crypto_util.h**.

5.3.6.5 description

```
uint8_t description[F_DESC_SZ]
```

File description.

Definition at line **254** of file **f_nano_crypto_util.h**.

5.3.6.6 file_info_integrity

```
uint8_t file_info_integrity[32]
```

File info integrity of the body block.

Definition at line **258** of file **f_nano_crypto_util.h**.

5.3.6.7 hash_sk_unencrypted

```
uint8_t hash_sk_unencrypted[32]
```

hash of Nano SEED when unencrypted

Definition at line **256** of file **f_nano_crypto_util.h**.

5.3.6.8 header

```
uint8_t header[sizeof(F_NANO_WALLET_INFO_MAGIC)]
```

Header magic.

Definition at line **250** of file **f_nano_crypto_util.h**.

5.3.6.9 iv

```
uint8_t iv
```

Initial sub vector.

Initial vector of first encryption layer.

Definition at line **252** of file **f_nano_crypto_util.h**.

5.3.6.10 last_used_wallet_number

```
uint32_t last_used_wallet_number
```

Last used wallet number.

Definition at line **252** of file **f_nano_crypto_util.h**.

5.3.6.11 link

```
uint8_t link[32]
```

link or destination account

Definition at line **262** of file **f_nano_crypto_util.h**.

5.3.6.12 max_fee

```
char max_fee[F_RAW_STR_MAX_SZ]
```

Custom preferred max fee of Proof of Work.

Definition at line **256** of file **f_nano_crypto_util.h**.

5.3.6.13 nano_hdr

```
uint8_t nano_hdr[sizeof(NANO_WALLET_MAGIC)]
```

Header of the file.

Definition at line **250** of file **f_nano_crypto_util.h**.

5.3.6.14 nanoseed_hash

```
uint8_t nanoseed_hash[32]
```

Nano SEED hash file.

Definition at line **256** of file **f_nano_crypto_util.h**.

5.3.6.15 preamble

```
uint8_t preamble[32]
```

Block preamble.

Definition at line **250** of file **f_nano_crypto_util.h**.

5.3.6.16 prefixes

```
uint8_t prefixes
```

Internal use for this API.

Definition at line **266** of file **f_nano_crypto_util.h**.

5.3.6.17 previous

```
uint8_t previous[32]
```

Previous block.

Definition at line **254** of file **f_nano_crypto_util.h**.

5.3.6.18 representative

```
uint8_t representative[32]
```

Representative for current account.

Definition at line **256** of file **f_nano_crypto_util.h**.

5.3.6.19 reserved

```
uint8_t reserved
```

Reserved (not used)

Reserved.

Definition at line **254** of file **f_nano_crypto_util.h**.

5.3.6.20 salt

```
uint8_t salt[32]
```

Salt of the first encryption layer.

Definition at line **256** of file **f_nano_crypto_util.h**.

5.3.6.21 seed_block

```
F_ENCRYPTED_BLOCK seed_block
```

Second encrypted block for Nano SEED.

Definition at line **260** of file **f_nano_crypto_util.h**.

5.3.6.22 signature

```
uint8_t signature[64]
```

Signature of the block.

Definition at line **264** of file **f_nano_crypto_util.h**.

5.3.6.23 sk_encrypted

```
uint8_t sk_encrypted[32]
```

Secret.

SEED encrypted (second layer)

Definition at line **258** of file **f_nano_crypto_util.h**.

5.3.6.24 sub_salt

```
uint8_t sub_salt[32]
```

Salt of the sub block to be stored.

Definition at line **250** of file **f_nano_crypto_util.h**.

5.3.6.25 ver

```
uint32_t ver
```

Version of the file.

Definition at line **252** of file **f_nano_crypto_util.h**.

5.3.6.26 version

```
uint16_t version
```

Version.

Definition at line **252** of file **f_nano_crypto_util.h**.

5.3.6.27 wallet_prefix

```
uint8_t wallet_prefix
```

Wallet prefix: 0 for NANO; 1 for XRB.

Definition at line **250** of file **f_nano_crypto_util.h**.

5.3.6.28 wallet_representative

```
char wallet_representative[ MAX_STR_NANO_CHAR]
```

Wallet representative.

Definition at line **254** of file **f_nano_crypto_util.h**.

5.3.6.29 work

```
uint64_t work
```

Internal use for this API.

Definition at line **268** of file **f_nano_crypto_util.h**.

5.4 f_nano_crypto_util.h

```
00001 /*
00002     AUTHOR: Fábio Pereira da Silva
00003     YEAR: 2019-20
00004     LICENSE: MIT
00005     EMAIL: fabioegel@gmail.com or fabioegel@protonmail.com
00006 */
00007
00008 #include <stdint.h>
00009 #include "f_util.h"
00010
00011 #ifndef F_DOC_SKIP
00012
00013  #ifdef F_XTENSA
00014
00015   #ifndef F_ESP32
00016    #define F_ESP32
00017   #endif
00018
00019   #include "esp_system.h"
00020
00021  #endif
00022
00023  #include "sodium/crypto_generichash.h"
00024  #include "sodium/crypto_sign.h"
00025  #include "sodium.h"
00026
00027  #ifdef F_ESP32
00028
00029   #include "sodium/private/curve25519_ref10.h"
00030
00031  #else
00032
00033   #include "sodium/private/ed25519_ref10.h"
00034
00035   #define ge_p3 ge25519_p3
00036   #define sc_reduce sc25519_reduce
00037   #define sc_muladd sc25519_muladd
00038   #define ge_scalarmult_base ge25519_scalarmult_base
00039   #define ge_p3_tobytes ge25519_p3_tobytes
00040
00041  #endif
00042
00043 #endif
00044
00127 #ifdef __cplusplus
00128 extern "C" {
00129 #endif
00130
00131
00136 #define F_NANO_POW_MAX_THREAD (size_t)10
00137
```

```
00138 #ifndef F_DOC_SKIP
00139  #ifdef F_ESP32
00140   #undef F_NANO_POW_MAX_THREAD
00141  #endif
00142 #endif
00143
00148 #define MAX_STR_NANO_CHAR (size_t)70 //5+56+8+1
00149
00154 #define PUB_KEY_EXTENDED_MAX_LEN (size_t)40
00155
00160 #define NANO_PREFIX "nano_"
00161
00166 #define XRB_PREFIX "xrb_"
00167
00168 #ifdef F_ESP32
00169
00174 #define BIP39_DICTIONARY "/spiffs/dictionary.dic"
00175 #else
00176
00177  #ifndef F_DOC_SKIP
00178   #define BIP39_DICTIONARY_SAMPLE "../../dictionary.dic"
00179   #define BIP39_DICTIONARY "dictionary.dic"
00180  #endif
00181
00182 #endif
00183
00190 #define NANO_ENCRYPTED_SEED_FILE "/spiffs/secure/nano.nse"
00191
00196 #define NANO_PASSWD_MAX_LEN (size_t)80
00197
00202 #define STR_NANO_SZ (size_t)66// 65+1 Null included
00203
00208 #define NANO_FILE_WALLETS_INFO "/spiffs/secure/walletsinfo.i"
00209
00214 typedef uint8_t NANO_SEED[crypto_sign_SEEDBYTES];
00215
00220 typedef uint8_t f_uint128_t[16];
00221
00222 #ifndef F_DOC_SKIP
00223  #define EXPORT_KEY_TO_CHAR_SZ (size_t)sizeof(NANO_SEED)+1
00224 #endif
00225
00230 typedef uint8_t NANO_PRIVATE_KEY[sizeof(NANO_SEED)];
00231
00236 typedef uint8_t NANO_PRIVATE_KEY_EXTENDED[crypto_sign_ed25519_SECRETKEYBYTES];
00237
00242 typedef uint8_t NANO_PUBLIC_KEY[crypto_sign_ed25519_PUBLICKEYBYTES];
00243
00248 typedef uint8_t NANO_PUBLIC_KEY_EXTENDED[PUB_KEY_EXTENDED_MAX_LEN];
00249
00258 typedef struct f_block_transfer_t {
00260    uint8_t preamble[32];
00262    uint8_t account[32];
00264    uint8_t previous[32];
00266    uint8_t representative[32];
00270    f_uint128_t balance;
00272    uint8_t link[32];
00274    uint8_t signature[64];
00276    uint8_t prefixes;
00278    uint64_t work;
00279 } __attribute__((packed)) F_BLOCK_TRANSFER;
00280
00281 #ifndef F_DOC_SKIP
00282  #define F_BLOCK_TRANSFER_SIGNABLE_SZ
       (size_t)(sizeof(F_BLOCK_TRANSFER)-64-sizeof(uint64_t)-sizeof(uint8_t))
00283 #endif
00284
00292 typedef enum f_nano_err_t {
00294    NANO_ERR_OK=0,
00296    NANO_ERR_CANT_PARSE_BN_STR=5151,
00298    NANO_ERR_MALLOC,
00300    NANO_ERR_CANT_PARSE_FACTOR,
00302    NANO_ERR_MPI_MULT,
00304    NANO_ERR_CANT_PARSE_TO_BLK_TRANSFER,
00306    NANO_ERR_EMPTY_STR,
00308    NANO_ERR_CANT_PARSE_VALUE,
00310    NANO_ERR_PARSE_MPI_TO_STR,
00312    NANO_ERR_CANT_COMPLETE_NULL_CHAR,
00314    NANO_ERR_CANT_PARSE_TO_MPI,
00316    NANO_ERR_INSUFICIENT_FUNDS,
00318    NANO_ERR_SUB_MPI,
00320    NANO_ERR_ADD_MPI,
00322    NANO_ERR_NO_SENSE_VALUE_TO_SEND_NEGATIVE,
00324    NANO_ERR_NO_SENSE_VALUE_TO_SEND_ZERO,
00326    NANO_ERR_NO_SENSE_BALANCE_NEGATIVE,
00328    NANO_ERR_VAL_A_INVALID_MODE,
00330    NANO_ERR_CANT_PARSE_TO_TEMP_UINT128_T,
```

```
00332    NANO_ERR_VAL_B_INVALID_MODE,
00334    NANO_ERR_CANT_PARSE_RAW_A_TO_MPI,
00336    NANO_ERR_CANT_PARSE_RAW_B_TO_MPI,
00338    NANO_ERR_UNKNOWN_ADD_SUB_MODE,
00340    NANO_ERR_INVALID_RES_OUTPUT
00341 } f_nano_err;
00342
00343 #ifndef F_DOC_SKIP
00344
00345  #define READ_SEED_FROM_STREAM (int)1
00346  #define READ_SEED_FROM_FILE (int)2
00347  #define WRITE_SEED_TO_STREAM (int)4
00348  #define WRITE_SEED_TO_FILE (int)8
00349  #define PARSE_JSON_READ_SEED_GENERIC (int)16
00350  #define F_STREAM_DATA_FILE_VERSION (uint32_t)((1<<16)|0)
00351
00352 #endif
00353
00361 typedef struct f_nano_encrypted_wallet_t {
00363    uint8_t sub_salt[32];
00365    uint8_t iv[16];
00367    uint8_t reserved[16];
00369    uint8_t hash_sk_unencrypted[32];
00371    uint8_t sk_encrypted[32];
00372 } __attribute__ ((packed)) F_ENCRYPTED_BLOCK;
00373
00374 #ifndef F_DOC_SKIP
00375
00376  static const uint8_t NANO_WALLET_MAGIC[] = {'_', 'n', 'a', 'n', 'o', 'w', 'a', 'l', 'l', 'e', 't', 'f',
    'i', 'l', 'e', '_'};
00377  #define F_NANO_FILE_DESC "NANO Seed Encrypted file/stream. Keep it safe and backup it. This file is
     protected by password. BUY BITCOIN and NANO !!!"
00378  #define F_DESC_SZ (size_t) (160-sizeof(uint32_t))
00379
00380 #endif
00381
00389 typedef struct f_nano_crypto_wallet_t {
00391    uint8_t nano_hdr[sizeof(NANO_WALLET_MAGIC)];
00393    uint32_t ver;
00395    uint8_t description[F_DESC_SZ];
00397    uint8_t salt[32];
00399    uint8_t iv[16];
00401    F_ENCRYPTED_BLOCK seed_block;
00402 } __attribute__ ((packed)) F_NANO_CRYPTOWALLET;
00403
00404 #ifndef F_DOC_SKIP
00405
00406 _Static_assert((sizeof(F_NANO_CRYPTOWALLET)&0x1F)==0, "Error 1");
00407 _Static_assert((sizeof(F_ENCRYPTED_BLOCK)&0x1F)==0, "Error 2");
00408
00409 #endif
00410
00415 #define REP_XRB (uint8_t)0x4
00416
00421 #define SENDER_XRB (uint8_t)0x02
00422
00427 #define DEST_XRB (uint8_t)0x01
00428
00429 typedef enum f_write_seed_err_t {
00431    WRITE_ERR_OK=0,
00433    WRITE_ERR_NULL_PASSWORD=7180,
00435    WRITE_ERR_EMPTY_STRING,
00437    WRITE_ERR_MALLOC,
00439    WRITE_ERR_ENCRYPT_PRIV_KEY,
00441    WRITE_ERR_GEN_SUB_PRIV_KEY,
00443    WRITE_ERR_GEN_MAIN_PRIV_KEY,
00445    WRITE_ERR_ENCRYPT_SUB_BLOCK,
00447    WRITE_ERR_UNKNOWN_OPTION,
00449    WRITE_ERR_FILE_ALREDY_EXISTS,
00451    WRITE_ERR_CREATING_FILE,
00453    WRITE_ERR_WRITING_FILE
00454 } f_write_seed_err;
00455
00456 #ifndef F_DOC_SKIP
00457
00458  #define F_RAW_TO_STR_UINT128 (int)1
00459  #define F_RAW_TO_STR_STRING (int)2
00460  #define F_RAW_STR_MAX_SZ (size_t)41 // 39 + '\0' + '.' -> 39 = log10(2^128)
00461  #define F_MAX_STR_RAW_BALANCE_MAX (size_t)40 //39+'\0'
00462  #define F_NANO_EMPTY_BALANCE "0.0"
00463
00464 #endif
00465
00473 typedef struct f_nano_wallet_info_bdy_t {
00475    uint8_t wallet_prefix; // 0 for NANO; 1 for XRB
00477    uint32_t last_used_wallet_number;
00479    char wallet_representative[MAX_STR_NANO_CHAR];
```

```
00481    char max_fee[F_RAW_STR_MAX_SZ];
00483    uint8_t reserved[44];
00484 } __attribute__((packed)) F_NANO_WALLET_INFO_BODY;
00485
00486 #ifndef F_DOC_SKIP
00487
00488  _Static_assert((sizeof(F_NANO_WALLET_INFO_BODY)&0x1F)==0, "Error F_NANO_WALLET_INFO_BODY is not byte
      aligned");
00489
00490  #define F_NANO_WALLET_INFO_DESC "Nano file descriptor used for fast custom access. BUY BITCOIN AND NANO."
00491  #define F_NANO_WALLET_INFO_VERSION (uint16_t)((1<<8)|1)
00492  static const uint8_t F_NANO_WALLET_INFO_MAGIC[] = {'_', 'n', 'a', 'n', 'o', 'w', 'a', 'l', 'l', 'e', 't',
     '_', 'n', 'f', 'o', '_'};
00493
00494  #define F_NANO_DESC_SZ (size_t)78
00495
00496 #endif
00497
00505 typedef struct f_nano_wallet_info_t {
00507    uint8_t header[sizeof(F_NANO_WALLET_INFO_MAGIC)];
00509    uint16_t version;
00511    char desc[F_NANO_DESC_SZ];
00513    uint8_t nanoseed_hash[32];
00515    uint8_t file_info_integrity[32];
00517    F_NANO_WALLET_INFO_BODY body;
00518 } __attribute__((packed)) F_NANO_WALLET_INFO;
00519
00520 #ifndef F_DOC_SKIP
00521
00522  _Static_assert((sizeof(F_NANO_WALLET_INFO)&0x1F)==0, "Error F_NANO_WALLET_INFO is not byte aligned");
00523
00524 #endif
00525
00533 typedef enum f_file_info_err_t {
00535    F_FILE_INFO_ERR_OK=0,
00537    F_FILE_INFO_ERR_CANT_OPEN_INFO_FILE=7001,
00539    F_FILE_INFO_ERR_NANO_SEED_ENCRYPTED_FILE_NOT_FOUND,
00541    F_FILE_INFO_ERR_CANT_DELETE_NANO_INFO_FILE,
00543    F_FILE_INFO_ERR_MALLOC,
00545    F_FILE_INFO_ERR_CANT_READ_NANO_SEED_ENCRYPTED_FILE,
00547    F_FILE_INFO_ERR_CANT_READ_INFO_FILE,
00549    F_FILE_INFO_INVALID_HEADER_FILE,
00551    F_FILE_INFO_ERR_INVALID_SHA256_INFO_FILE,
00553    F_FILE_INFO_ERR_NANO_SEED_HASH_FAIL,
00555    F_FILE_INFO_ERR_NANO_INVALID_REPRESENTATIVE,
00557    F_FILE_INFO_ERR_NANO_INVALID_MAX_FEE_VALUE,
00559    F_FILE_INFO_ERR_OPEN_FOR_WRITE_INFO,
00561    F_FILE_INFO_ERR_EXISTING_FILE,
00563    F_FILE_INFO_ERR_CANT_WRITE_FILE_INFO
00564 } F_FILE_INFO_ERR;
00565
00566 #ifndef F_DOC_SKIP
00567
00568  #define F_NANO_ADD_A_B (uint32_t)(1<<0)
00569  #define F_NANO_SUB_A_B (uint32_t)(1<<1)
00570  #define F_NANO_A_RAW_128 (uint32_t)(1<<2)
00571  #define F_NANO_A_RAW_STRING (uint32_t)(1<<3)
00572  #define F_NANO_A_REAL_STRING (uint32_t)(1<<4)
00573  #define F_NANO_B_RAW_128 (uint32_t)(1<<5)
00574  #define F_NANO_B_RAW_STRING (uint32_t)(1<<6)
00575  #define F_NANO_B_REAL_STRING (uint32_t)(1<<7)
00576  #define F_NANO_RES_RAW_128 (uint32_t)(1<<8)
00577  #define F_NANO_RES_RAW_STRING (uint32_t)(1<<9)
00578  #define F_NANO_RES_REAL_STRING (uint32_t)(1<<10)
00579  #define F_NANO_C_RAW_128 (uint32_t)(F_NANO_B_RAW_128<<16)
00580  #define F_NANO_C_RAW_STRING (uint32_t)(F_NANO_B_RAW_STRING<<16)
00581  #define F_NANO_C_REAL_STRING (uint32_t)(F_NANO_B_REAL_STRING<<16)
00582
00583  #define F_NANO_COMPARE_EQ (uint32_t)(1<<16) //Equal
00584  #define F_NANO_COMPARE_LT (uint32_t)(1<<17) // Lesser than
00585  #define F_NANO_COMPARE_LEQ (F_NANO_COMPARE_LT|F_NANO_COMPARE_EQ) // Less or equal
00586  #define F_NANO_COMPARE_GT (uint32_t)(1<<18) // Greater
00587  #define F_NANO_COMPARE_GEQ (F_NANO_COMPARE_GT|F_NANO_COMPARE_EQ) // Greater or equal
00588  #define DEFAULT_MAX_FEE "0.001"
00589
00590 #endif
00591
00614 int f_cloud_crypto_wallet_nano_create_seed(size_t, char *, char *);
00615
00628 int f_generate_nano_seed(NANO_SEED, uint32_t);
00629
00644 int pk_to_wallet(char *, char *, NANO_PUBLIC_KEY_EXTENDED);
00645
00663 int f_seed_to_nano_wallet(NANO_PRIVATE_KEY, NANO_PUBLIC_KEY, NANO_SEED, uint32_t);
00664
00674 char *f_nano_key_to_str(char *, unsigned char *);
00675
```

```
00694 int f_nano_seed_to_bip39(char *, size_t, size_t *, NANO_SEED, char *);
00695
00710 int f_bip39_to_nano_seed(uint8_t *, char *, char *);
00711
00733 int f_parse_nano_seed_and_bip39_to_JSON(char *, size_t, size_t *, void *, int, const char *);
00734
00752 int f_read_seed(uint8_t *, const char *, void *, int, int);
00753
00768 int f_nano_raw_to_string(char *, size_t *, size_t, void *, int);
00769
00778 int f_nano_valid_nano_str_value(const char *);
00779
00787 int valid_nano_wallet(const char *);
00788
00798 int nano_base_32_2_hex(uint8_t *, char *);
00799
00814 int f_nano_transaction_to_JSON(char *, size_t, size_t *, NANO_PRIVATE_KEY_EXTENDED, F_BLOCK_TRANSFER *);
00815
00823 int valid_raw_balance(const char *);
00824
00832 int is_null_hash(uint8_t *);
00833
00845 int is_nano_prefix(const char *, const char *);
00846
00855 F_FILE_INFO_ERR f_get_nano_file_info(F_NANO_WALLET_INFO *);
00856
00866 F_FILE_INFO_ERR f_set_nano_file_info(F_NANO_WALLET_INFO *, int);
00867
00891 f_nano_err f_nano_value_compare_value(void *, void *, uint32_t *);
00892
00913 f_nano_err f_nano_verify_nano_funds(void *, void *, void *, uint32_t);
00914
00924 f_nano_err f_nano_parse_raw_str_to_raw128_t(uint8_t *, const char *);
00925
00935 f_nano_err f_nano_parse_real_str_to_raw128_t(uint8_t *, const char *);
00936
00959 f_nano_err f_nano_add_sub(void *, void *, void *, uint32_t);
00960
00971 int f_nano_sign_block(F_BLOCK_TRANSFER *, F_BLOCK_TRANSFER *, NANO_PRIVATE_KEY_EXTENDED);
00972
00986 f_write_seed_err f_write_seed(void *, int, uint8_t *, char *);
00987
01000 f_nano_err f_nano_balance_to_str(char *, size_t, size_t *, f_uint128_t);
01001
01002
01007 #define F_BRAIN_WALLET_VERY_POOR (uint32_t)0
01008
01013 #define F_BRAIN_WALLET_POOR (uint32_t)1
01014
01019 #define F_BRAIN_WALLET_VERY_BAD (uint32_t)2
01020
01025 #define F_BRAIN_WALLET_BAD (uint32_t)3
01026
01031 #define F_BRAIN_WALLET_VERY_WEAK (uint32_t)4
01032
01037 #define F_BRAIN_WALLET_WEAK (uint32_t)5
01038
01043 #define F_BRAIN_WALLET_STILL_WEAK (uint32_t)6
01044
01049 #define F_BRAIN_WALLET_MAYBE_GOOD (uint32_t)7
01050
01051
01056 #define F_BRAIN_WALLET_GOOD (uint32_t)8
01057
01062 #define F_BRAIN_WALLET_VERY_GOOD (uint32_t)9
01063
01068 #define F_BRAIN_WALLET_NICE (uint32_t)10
01069
01074 #define F_BRAIN_WALLET_PERFECT (uint32_t)11
01075
01102 int f_extract_seed_from_brainwallet(uint8_t *, char **, uint32_t, const char *, const char *);
01103
01116 int f_verify_work(uint64_t *, const unsigned char *, uint64_t *, uint64_t);
01117
01118 #ifndef F_ESP32
01119
01132 int f_nano_pow(uint64_t *, unsigned char *, const uint64_t, int);
01133 #endif
01134
01135 #ifdef __cplusplus
01136 }
01137 #endif
01138
```

5.5 f_util.h File Reference

```
#include <stdint.h>
#include "mbedtls/sha256.h"
#include "mbedtls/aes.h"
```

Macros

- #define **F_ENTROPY_TYPE_PARANOIC** (uint32_t)1477682819
- #define **F_ENTROPY_TYPE_EXCELENT** (uint32_t)1476885281
- #define **F_ENTROPY_TYPE_GOOD** (uint32_t)1472531015
- #define **F_ENTROPY_TYPE_NOT_ENOUGH** (uint32_t)1471001808
- #define **F_ENTROPY_TYPE_NOT_RECOMENDED** (uint32_t)1470003345
- #define **ENTROPY_BEGIN** f_verify_system_entropy_begin();
- #define **ENTROPY_END** f_verify_system_entropy_finish();
- #define **F_PASS_MUST_HAVE_AT_LEAST_NONE** (int)0
- #define **F_PASS_MUST_HAVE_AT_LEAST_ONE_NUMBER** (int)1
- #define **F_PASS_MUST_HAVE_AT_LEAST_ONE_SYMBOL** (int)2
- #define **F_PASS_MUST_HAVE_AT_LEAST_ONE_UPPER_CASE** (int)4
- #define **F_PASS_MUST_HAVE_AT_LEAST_ONE_LOWER_CASE** (int)8
- #define **F_PASS_IS_TOO_LONG** (int)256
- #define **F_PASS_IS_TOO_SHORT** (int)512
- #define **F_PASS_IS_OUT_OVF** (int)1024
- #define **F_GET_CH_MODE_NO_ECHO** (int)(1<<16)
- #define **F_GET_CH_MODE_ANY_KEY** (int)(1<<17)

Typedefs

- typedef void(* **rnd_fn**) (void *, size_t)

Functions

- int **f_verify_system_entropy** (uint32_t, void *, size_t, int)
- int **f_pass_must_have_at_least** (char *, size_t, size_t, size_t, int)
- int **f_passwd_comp_safe** (char *, char *, size_t, size_t, size_t)
- char * **f_get_entropy_name** (uint32_t)
- uint32_t **f_sel_to_entropy_level** (int)
- int **f_str_to_hex** (uint8_t *, char *)
- void **f_random_attach** (**rnd_fn**)
- void **f_random** (void *, size_t)
- int **get_console_passwd** (char *, size_t)
- int **f_get_char_no_block** (int)
- int **f_convert_to_long_int** (unsigned long int *, char *, size_t)
- int **f_convert_to_unsigned_int** (unsigned int *, char *, size_t)
- int **f_convert_to_long_int0x** (unsigned long int *, char *, size_t)
- int **f_convert_to_long_int0** (unsigned long int *, char *, size_t)
- int **f_convert_to_long_int_std** (unsigned long int *, char *, size_t)
- void * **f_is_random_attached** ()
- void **f_random_detach** ()

5.5.1 Detailed Description

This ABI is a utility for myNanoEmbedded library and sub routines are implemented here.

Definition in file **f_util.h**.

5.5.2 Macro Definition Documentation

5.5.2.1 ENTROPY_BEGIN

```
#define ENTROPY_BEGIN f_verify_system_entropy_begin();
```

Begins and prepares a entropy function.

See also

f_verify_system_entropy() (p. **??**)

Definition at line **152** of file **f_util.h**.

5.5.2.2 ENTROPY_END

```
#define ENTROPY_END f_verify_system_entropy_finish();
```

Ends a entropy function.

See also

f_verify_system_entropy() (p. **??**)

Definition at line **159** of file **f_util.h**.

5.5.2.3 F_ENTROPY_TYPE_EXCELENT

```
#define F_ENTROPY_TYPE_EXCELENT (uint32_t)1476885281
```

Type of the excelent entropy used for verifier.

Slow

Definition at line **124** of file **f_util.h**.

5.5.2.4 F_ENTROPY_TYPE_GOOD

```
#define F_ENTROPY_TYPE_GOOD (uint32_t)1472531015
```

Type of the good entropy used for verifier.

Not so slow

Definition at line **131** of file **f_util.h**.

5.5.2.5 F_ENTROPY_TYPE_NOT_ENOUGH

```
#define F_ENTROPY_TYPE_NOT_ENOUGH (uint32_t)1471001808
```

Type of the moderate entropy used for verifier.

Fast

Definition at line **138** of file **f_util.h**.

5.5.2.6 F_ENTROPY_TYPE_NOT_RECOMENDED

```
#define F_ENTROPY_TYPE_NOT_RECOMENDED (uint32_t)1470003345
```

Type of the not recommended entropy used for verifier.

Very fast

Definition at line **145** of file **f_util.h**.

5.5.2.7 F_ENTROPY_TYPE_PARANOIC

```
#define F_ENTROPY_TYPE_PARANOIC (uint32_t)1477682819
```

Type of the very excelent entropy used for verifier.

Very slow

Definition at line **117** of file **f_util.h**.

5.5.2.8 F_GET_CH_MODE_ANY_KEY

```
#define F_GET_CH_MODE_ANY_KEY (int)(1<<17)
```

See also

f_get_char_no_block() (p. **??**)

Definition at line **334** of file **f_util.h**.

5.5.2.9 F_GET_CH_MODE_NO_ECHO

```
#define F_GET_CH_MODE_NO_ECHO (int)(1<<16)
```

See also

f_get_char_no_block() (p. **??**)

Definition at line **328** of file **f_util.h**.

5.5.2.10 F_PASS_IS_OUT_OVF

```
#define F_PASS_IS_OUT_OVF (int)1024
```

Password is overflow and cannot be stored.

Definition at line **207** of file **f_util.h**.

5.5.2.11 F_PASS_IS_TOO_LONG

```
#define F_PASS_IS_TOO_LONG (int)256
```

Password is too long.

Definition at line **195** of file **f_util.h**.

5.5.2.12 F_PASS_IS_TOO_SHORT

```
#define F_PASS_IS_TOO_SHORT (int)512
```

Password is too short.

Definition at line **201** of file **f_util.h**.

5.5.2.13 F_PASS_MUST_HAVE_AT_LEAST_NONE

```
#define F_PASS_MUST_HAVE_AT_LEAST_NONE (int)0
```

Password does not need any criteria to pass.

Definition at line **165** of file **f_util.h**.

5.5.2.14 F_PASS_MUST_HAVE_AT_LEAST_ONE_LOWER_CASE

```
#define F_PASS_MUST_HAVE_AT_LEAST_ONE_LOWER_CASE (int)8
```

Password must have at least one lower case.

Definition at line **189** of file **f_util.h**.

5.5.2.15 F_PASS_MUST_HAVE_AT_LEAST_ONE_NUMBER

```
#define F_PASS_MUST_HAVE_AT_LEAST_ONE_NUMBER (int)1
```

Password must have at least one number.

Definition at line **171** of file **f_util.h**.

5.5.2.16 F_PASS_MUST_HAVE_AT_LEAST_ONE_SYMBOL

```
#define F_PASS_MUST_HAVE_AT_LEAST_ONE_SYMBOL (int)2
```

Password must have at least one symbol.

Definition at line **177** of file **f_util.h**.

5.5.2.17 F_PASS_MUST_HAVE_AT_LEAST_ONE_UPPER_CASE

```
#define F_PASS_MUST_HAVE_AT_LEAST_ONE_UPPER_CASE (int)4
```

Password must have at least one upper case.

Definition at line **183** of file **f_util.h**.

5.5.3 Typedef Documentation

5.5.3.1 rnd_fn

```
rnd_fn
```

Pointer caller for random function.

Definition at line **293** of file **f_util.h**.

5.5.4 Function Documentation

5.5.4.1 f_convert_to_long_int()

```
int f_convert_to_long_int (
            unsigned long int * val,
            char * value,
            size_t value_sz )
```

Converts a string value to unsigned long int.

Parameters

out	*val*	Value stored in a unsigned long int variable
in	*value*	Input value to be parsed to unsigned long int
in	*value_sz*	Max size allowed in *value* string.

Return values

0	On Success, Otherwise error

See also

f_convert_to_unsigned_int() (p. **??**)

5.5.4.2 f_convert_to_long_int0()

```
int f_convert_to_long_int0 (
            unsigned long int * val,
            char * value,
            size_t value_sz )
```

Converts a octal value in ASCII string to unsigned long int.

Parameters

out	*val*	Value stored in a unsigned long int variable
in	*value*	Input value to be parsed to unsigned long int
in	*value_sz*	Max size allowed in *value* string.

Return values

0	On Success, Otherwise error

See also

f_convert_to_long_int0x() (p. **??**)

5.5.4.3 f_convert_to_long_int0x()

```
int f_convert_to_long_int0x (
            unsigned long int * val,
            char * value,
            size_t value_sz )
```

Converts a hex value in ASCII string to unsigned long int.

Parameters

out	*val*	Value stored in a unsigned long int variable
in	*value*	Input value to be parsed to unsigned long int
in	*value_sz*	Max size allowed in *value* string.

Return values

0	On Success, Otherwise error

See also

f_convert_to_long_int0() (p. **??**)

5.5.4.4 f_convert_to_long_int_std()

```
int f_convert_to_long_int_std (
            unsigned long int * val,
            char * value,
            size_t value_sz )
```

Converts a actal/decimal/hexadecimal into ASCII string to unsigned long int.

Parameters

out	*val*	Value stored in a unsigned long int variable
in	*value*	Input value to be parsed to unsigned long int • If a string contains only numbers, it will be parsed to unsigned long int decimal • If a string begins with 0 it will be parsed to octal EX.: 010(octal) = 08(decimal) • If a string contais 0x or 0X it will be parsed to hexadecimal. EX.: 0x10(hexadecimal) = 16 (decimal)
in	*value_sz*	Max size allowed in *value* string.

Return values

0	On Success, Otherwise error

See also

f_convert_to_long_int() (p. **??**)

5.5.4.5 f_convert_to_unsigned_int()

```
int f_convert_to_unsigned_int (
            unsigned int * val,
            char * value,
            size_t value_sz )
```

Converts a string value to unsigned int.

Parameters

out	*val*	Value stored in a unsigned int variable
in	*value*	Input value to be parsed to unsigned int
in	*value_sz*	Max size allowed in *value* string.

Return values

0	On Success, Otherwise error

See also

f_convert_to_long_int() (p. **??**)

5.5.4.6 f_get_char_no_block()

```
int f_get_char_no_block (
            int mode )
```

Reads a char from console.

Waits a char and returns its value

Parameters

in	*mode*	Mode and/or character to be returned • *F_GET_CH_MODE_NO_ECHO* No echo is on the console string • *F_GET_CH_MODE_ANY_KEY* Returns any key pressed

Example:

```
key=f_get_char_no_block(F_GET_CH_MODE_NO_ECHO|'c'); // Waits 'c' char key and returns value 0x00000063
      without echo 'c' on the screen
```

Return values

key	code: On Success, Negative value on error

5.5.4.7 f_get_entropy_name()

```
char * f_get_entropy_name (
            uint32_t val )
```

Returns a entropy name given a index/ASCII index or entropy value.

Parameters

in	*val*	Index/ASCII index or entropy value

Return values:

- *NULL* If no entropy index/ASCII/entropy found in *val*
- *F_ENTROPY_TYPE_*∗ name if found in index/ASCII or entropy value

5.5.4.8 f_is_random_attached()

```
void * f_is_random_attached ( )
```

Verifies if system random function is attached in myNanoEmbedded API.

Return values

0	if not attached, Otherwise returns the pointer of random number genarator function

See also

f_random_attach() (p. **??**)

5.5.4.9 f_pass_must_have_at_least()

```
int f_pass_must_have_at_least (
            char * password,
            size_t n,
            size_t min,
            size_t max,
            int must_have )
```

Checks if a given password has enought requirements to be parsed to a function.

Parameters

in	*password*	Password string
in	*n*	Max buffer string permitted to store password including NULL char
in	*min*	Minimum size allowed in password string
in	*max*	Maximum size allowed in password
in	*must_have*	Must have a type: • F_PASS_MUST_HAVE_AT_LEAST_NONE Not need any special characters or number • F_PASS_MUST_HAVE_AT_LEAST_ONE_NUMBER Must have at least one number • F_PASS_MUST_HAVE_AT_LEAST_ONE_SYMBOL Must have at least one symbol • F_PASS_MUST_HAVE_AT_LEAST_ONE_UPPER_CASE Must have at least one upper case • F_PASS_MUST_HAVE_AT_LEAST_ONE_LOWER_CASE Must have at least one lower case

Return values:

- *0 (zero):* If password is passed in the test

- *F_PASS_IS_OUT_OVF:* If password lenght exceeds *n* value
- *F_PASS_IS_TOO_SHORT:* If password length is less than *min* value
- *F_PASS_IS_TOO_LONG:* If password length is greater tham *m* value
- *F_PASS_MUST_HAVE_AT_LEAST_ONE_UPPER_CASE:* If password is required in *must_have* type upper case characters
- *F_PASS_MUST_HAVE_AT_LEAST_ONE_LOWER_CASE:* If password is required in *must_have* type lower case characters
- *F_PASS_MUST_HAVE_AT_LEAST_ONE_SYMBOL:* If password is required in *must_have* type to have symbol(s)
- *F_PASS_MUST_HAVE_AT_LEAST_ONE_NUMBER:* if password is required in *must_have* type to have number(s)

5.5.4.10 f_passwd_comp_safe()

```
int f_passwd_comp_safe (
            char * pass1,
            char * pass2,
            size_t n,
            size_t min,
            size_t max )
```

Compares two passwords values with safe buffer.

Parameters

in	*pass1*	First password to compare with *pass2*
in	*pass2*	Second password to compare with *pass1*
in	*n*	Size of Maximum buffer of both *pass1* and *pass2*
in	*min*	Minimun value of both *pass1* and *pass2*
in	*max*	Maximum value of both *pass1* and *pass2*

Return values

0	If *pass1* is equal to *pass2*, otherwise value is less than 0 (zero) if password does not match

5.5.4.11 f_random()

```
void f_random (
            void * random,
            size_t random_sz )
```

Random function to be called to generate a *random* data with *random_sz*

Parameters

out	*random*	Random data to be parsed
in	*random_sz*	Size of random data to be filled

See also

f_random_attach() (p. **??**)

5.5.4.12 f_random_attach()

```
void f_random_attach (
              rnd_fn fn )
```

Attachs a function to be called by ***f_random()*** (p. **??**)

Parameters

in	*fn*	A function to be called

See also

rnd_fn() (p. **??**)

5.5.4.13 f_random_detach()

```
void f_random_detach ( )
```

Detaches system random numeber genarator from myNanoEmbedded API.

See also

f_random_attach() (p. **??**)

5.5.4.14 f_sel_to_entropy_level()

```
uint32_t f_sel_to_entropy_level (
             int sel )
```

Return a given entropy number given a number encoded ASCII or index number.

Parameters

in	*sel*	ASCII or index value

Return values:

- *0 (zero):* If no entropy number found in *sel*
- *F_ENTROPY_TYPE_PARANOIC*
- *F_ENTROPY_TYPE_EXCELENT*
- *F_ENTROPY_TYPE_GOOD*
- *F_ENTROPY_TYPE_NOT_ENOUGH*
- *F_ENTROPY_TYPE_NOT_RECOMENDED*

5.5.4.15 f_str_to_hex()

```
int f_str_to_hex (
            uint8_t * hex_stream,
            char * str )
```

Converts a *str* string buffer to raw *hex_stream* value stream.

Parameters

out	*hex*	Raw hex value
in	*str*	String buffer terminated with NULL char

Return values

0	On Success, otherwise Error

5.5.4.16 f_verify_system_entropy()

```
int f_verify_system_entropy (
            uint32_t type,
            void * rand,
            size_t rand_sz,
            int turn_on_wdt )
```

Take a random number generator function and returns random value only if randomized data have a desired entropy value.

Parameters

in	*type*	Entropy type. Entropy type values are: • F_ENTROPY_TYPE_PARANOIC Highest level entropy recommended for generate a Nano SEED with a paranoic entropy. Very slow • F_ENTROPY_TYPE_EXCELENT Gives a very excellent entropy for generating Nano SEED. Slow • F_ENTROPY_TYPE_GOOD Good entropy type for generating Nano SEED. Normal. • F_ENTROPY_TYPE_NOT_ENOUGH Moderate entropy for generating Nano SEED. Usually fast to create a temporary Nano SEED. Fast • F_ENTROPY_TYPE_NOT_RECOMENDED Fast but not recommended for generating Nano SEED.
out	*rand*	Random data with a satisfied type of entropy
in	*rand_sz*	Size of random data output
in	*turn_on_wdt*	For ESP32, Arduino platform and other microcontrollers only. Turns on/off WATCH DOG (0: OFF, NON ZERO: ON). For Raspberry PI and Linux native is ommited.

This implementation is based on topic in `Definition 7.12` in MIT opencourseware (7.3 A Statistical Definition of Entropy - 2005)
Many thanks to **Professor Z. S. Spakovszky** for this amazing topic

Return values

0	On Success, otherwise Error

5.5.4.17 get_console_passwd()

```
int get_console_passwd (
            char * pass,
            size_t pass_sz )
```

Reads a password from console.

Parameters

out	*pass*	Password to be parsed to pointer
in	*pass_sz*	Size of buffer *pass*

Return values

0	On Success, otherwise Error

5.6 f_util.h

```
00001 /*
00002     AUTHOR: Fábio Pereira da Silva
00003     YEAR: 2019-20
00004     LICENSE: MIT
00005     EMAIL: fabioegel@gmail.com or fabioegel@protonmail.com
00006 */
00007
00013 #include <stdint.h>
00014 #include "mbedtls/sha256.h"
00015 #include "mbedtls/aes.h"
00016
00017 #ifdef __cplusplus
00018 extern "C" {
00019 #endif
00020
00021 #ifndef F_DOC_SKIP
00022
00023  #define F_LOG_MAX 8*256
00024  #define LICENSE \
00025 "MIT License\n\n\
00026 Copyright (c) 2019 Fábio Pereira da Silva\n\n\
00027 Permission is hereby granted, free of charge, to any person obtaining a copy\n\
00028 of this software and associated documentation files (the \"Software\"), to deal\n\
00029 in the Software without restriction, including without limitation the rights\n\
00030 to use, copy, modify, merge, publish, distribute, sublicense, and/or sell\n\
00031 copies of the Software, and to permit persons to whom the Software is\n\
00032 furnished to do so, subject to the following conditions:\n\n\
00033 The above copyright notice and this permission notice shall be included in all\n\
00034 copies or substantial portions of the Software.\n\n\
00035 THE SOFTWARE IS PROVIDED \"AS IS\", WITHOUT WARRANTY OF ANY KIND, EXPRESS OR\n\
00036 IMPLIED, INCLUDING BUT NOT LIMITED TO THE WARRANTIES OF MERCHANTABILITY,\n\
00037 FITNESS FOR A PARTICULAR PURPOSE AND NONINFRINGEMENT. IN NO EVENT SHALL THE\n\
00038 AUTHORS OR COPYRIGHT HOLDERS BE LIABLE FOR ANY CLAIM, DAMAGES OR OTHER\n\
00039 LIABILITY, WHETHER IN AN ACTION OF CONTRACT, TORT OR OTHERWISE, ARISING FROM,\n\
00040 OUT OF OR IN CONNECTION WITH THE SOFTWARE OR THE USE OR OTHER DEALINGS IN THE\n\
00041 SOFTWARE.\n\n\n"
00042
00043 #endif
00044
00045 #ifdef F_ESP32
00046
00047  #define F_WDT_MAX_ENTROPY_TIME 2*120
00048  #define F_WDT_PANIC true
00049  #define F_WDT_MIN_TIME 20//4
00050
00051 #endif
00052
00070 int f_verify_system_entropy(uint32_t, void *, size_t, int);
00071
00098 int f_pass_must_have_at_least(char *, size_t, size_t, size_t, int);
00099
00100 #ifndef F_DOC_SKIP
00101
00102 int f_verify_system_entropy_begin();
00103 void f_verify_system_entropy_finish();
00104 int f_file_exists(char *);
00105 int f_find_str(size_t *, char *, size_t, char *);
00106 int f_find_replace(char *, size_t *, size_t, char *, size_t, char *, char *);
00107 int f_is_integer(char *, size_t);
00108 int is_filled_with_value(uint8_t *, size_t, uint8_t);
00109
00110 #endif
00111
00112 //#define F_ENTROPY_TYPE_PARANOIC (uint32_t)1476682819
00117 #define F_ENTROPY_TYPE_PARANOIC (uint32_t)1477682819
00118
00119 //#define F_ENTROPY_TYPE_EXCELENT (uint32_t)1475885281
00124 #define F_ENTROPY_TYPE_EXCELENT (uint32_t)1476885281
00125
00126 //#define F_ENTROPY_TYPE_GOOD (uint32_t)1471531015
00131 #define F_ENTROPY_TYPE_GOOD (uint32_t)1472531015
00132
00133 //#define F_ENTROPY_TYPE_NOT_ENOUGH (uint32_t)1470001808
00138 #define F_ENTROPY_TYPE_NOT_ENOUGH (uint32_t)1471001808
00139
00140 //#define F_ENTROPY_TYPE_NOT_RECOMENDED (uint32_t)1469703345
00145 #define F_ENTROPY_TYPE_NOT_RECOMENDED (uint32_t)1470003345
00146
00152 #define ENTROPY_BEGIN f_verify_system_entropy_begin();
00153
00159 #define ENTROPY_END f_verify_system_entropy_finish();
00160
00165 #define F_PASS_MUST_HAVE_AT_LEAST_NONE (int)0
00166
```

```
00171 #define F_PASS_MUST_HAVE_AT_LEAST_ONE_NUMBER (int)1
00172
00177 #define F_PASS_MUST_HAVE_AT_LEAST_ONE_SYMBOL (int)2
00178
00183 #define F_PASS_MUST_HAVE_AT_LEAST_ONE_UPPER_CASE (int)4
00184
00189 #define F_PASS_MUST_HAVE_AT_LEAST_ONE_LOWER_CASE (int)8
00190
00195 #define F_PASS_IS_TOO_LONG (int)256
00196
00201 #define F_PASS_IS_TOO_SHORT (int)512
00202
00207 #define F_PASS_IS_OUT_OVF (int)1024//768
00208
00209 #ifndef F_DOC_SKIP
00210
00211  #define F_PBKDF2_ITER_SZ 2*4096
00212
00213 typedef enum f_pbkdf2_err_t {
00214     F_PBKDF2_RESULT_OK=0,
00215     F_PBKDF2_ERR_CTX=95,
00216     F_PBKDF2_ERR_PKCS5,
00217     F_PBKDF2_ERR_INFO_SHA
00218 } f_pbkdf2_err;
00219
00220 typedef enum f_aes_err {
00221     F_AES_RESULT_OK=0,
00222     F_AES_ERR_ENCKEY=30,
00223     F_AES_ERR_DECKEY,
00224     F_AES_ERR_MALLOC,
00225     F_AES_UNKNOW_DIRECTION,
00226     F_ERR_ENC_DECRYPT_FAILED
00227 } f_aes_err;
00228
00229 char *fhex2strv2(char *, const void *, size_t, int);
00230 uint8_t *f_sha256_digest(uint8_t *, size_t);
00231 f_pbkdf2_err f_pbkdf2_hmac(unsigned char *, size_t, unsigned char *, size_t, uint8_t *);
00232 f_aes_err f_aes256cipher(uint8_t *, uint8_t *, void *, size_t, void *, int);
00233
00234 #endif
00235
00247 int f_passwd_comp_safe(char *, char *, size_t, size_t, size_t);
00248
00259 char *f_get_entropy_name(uint32_t);
00260
00275 uint32_t f_sel_to_entropy_level(int);
00276
00285 int f_str_to_hex(uint8_t *, char *);
00286
00287 #ifndef F_ESP32
00288
00293 typedef void (*rnd_fn)(void *, size_t);
00294
00302 void f_random_attach(rnd_fn);
00303
00312 void f_random(void *, size_t);
00313
00322 int get_console_passwd(char *, size_t);
00323
00328 #define F_GET_CH_MODE_NO_ECHO (int)(1<<16)
00329
00334 #define F_GET_CH_MODE_ANY_KEY (int)(1<<17)
00335
00351 int f_get_char_no_block(int);
00352
00353 #endif
00354
00365 int f_convert_to_long_int(unsigned long int *, char *, size_t);
00366
00367
00378 int f_convert_to_unsigned_int(unsigned int *, char *, size_t);
00379
00390 int f_convert_to_long_int0x(unsigned long int *, char *, size_t);
00391
00402 int f_convert_to_long_int0(unsigned long int *, char *, size_t);
00403
00417 int f_convert_to_long_int_std(unsigned long int *, char *, size_t);
00418
00426 void *f_is_random_attached();
00427
00434 void f_random_detach();
00435
00436 #ifdef __cplusplus
00437 }
00438 #endif
```

5.7 sodium.h File Reference

```
#include "sodium/version.h"
#include "sodium/core.h"
#include "sodium/crypto_aead_aes256gcm.h"
#include "sodium/crypto_aead_chacha20poly1305.h"
#include "sodium/crypto_aead_xchacha20poly1305.h"
#include "sodium/crypto_auth.h"
#include "sodium/crypto_auth_hmacsha256.h"
#include "sodium/crypto_auth_hmacsha512.h"
#include "sodium/crypto_auth_hmacsha512256.h"
#include "sodium/crypto_box.h"
#include "sodium/crypto_box_curve25519xsalsa20poly1305.h"
#include "sodium/crypto_core_hsalsa20.h"
#include "sodium/crypto_core_hchacha20.h"
#include "sodium/crypto_core_salsa20.h"
#include "sodium/crypto_core_salsa2012.h"
#include "sodium/crypto_core_salsa208.h"
#include "sodium/crypto_generichash.h"
#include "sodium/crypto_generichash_blake2b.h"
#include "sodium/crypto_hash.h"
#include "sodium/crypto_hash_sha256.h"
#include "sodium/crypto_hash_sha512.h"
#include "sodium/crypto_kdf.h"
#include "sodium/crypto_kdf_blake2b.h"
#include "sodium/crypto_kx.h"
#include "sodium/crypto_onetimeauth.h"
#include "sodium/crypto_onetimeauth_poly1305.h"
#include "sodium/crypto_pwhash.h"
#include "sodium/crypto_pwhash_argon2i.h"
#include "sodium/crypto_scalarmult.h"
#include "sodium/crypto_scalarmult_curve25519.h"
#include "sodium/crypto_secretbox.h"
#include "sodium/crypto_secretbox_xsalsa20poly1305.h"
#include "sodium/crypto_secretstream_xchacha20poly1305.h"
#include "sodium/crypto_shorthash.h"
#include "sodium/crypto_shorthash_siphash24.h"
#include "sodium/crypto_sign.h"
#include "sodium/crypto_sign_ed25519.h"
#include "sodium/crypto_stream.h"
#include "sodium/crypto_stream_chacha20.h"
#include "sodium/crypto_stream_salsa20.h"
#include "sodium/crypto_stream_xsalsa20.h"
#include "sodium/crypto_verify_16.h"
#include "sodium/crypto_verify_32.h"
#include "sodium/crypto_verify_64.h"
#include "sodium/randombytes.h"
#include "sodium/randombytes_salsa20_random.h"
#include "sodium/randombytes_sysrandom.h"
#include "sodium/runtime.h"
#include "sodium/utils.h"
#include "sodium/crypto_box_curve25519xchacha20poly1305.h"
#include "sodium/crypto_core_ed25519.h"
#include "sodium/crypto_scalarmult_ed25519.h"
#include "sodium/crypto_secretbox_xchacha20poly1305.h"
#include "sodium/crypto_pwhash_scryptsalsa208sha256.h"
#include "sodium/crypto_stream_salsa2012.h"
#include "sodium/crypto_stream_salsa208.h"
```

```
#include "sodium/crypto_stream_xchacha20.h"
```

5.7.1 Detailed Description

This header file is an implementation of Libsodium library.

Definition in file **sodium.h**.

5.8 sodium.h

```
00001
00005 #ifndef sodium_H
00006 #define sodium_H
00007
00008 #include "sodium/version.h"
00009
00010 #include "sodium/core.h"
00011 #include "sodium/crypto_aead_aes256gcm.h"
00012 #include "sodium/crypto_aead_chacha20poly1305.h"
00013 #include "sodium/crypto_aead_xchacha20poly1305.h"
00014 #include "sodium/crypto_auth.h"
00015 #include "sodium/crypto_auth_hmacsha256.h"
00016 #include "sodium/crypto_auth_hmacsha512.h"
00017 #include "sodium/crypto_auth_hmacsha512256.h"
00018 #include "sodium/crypto_box.h"
00019 #include "sodium/crypto_box_curve25519xsalsa20poly1305.h"
00020 #include "sodium/crypto_core_hsalsa20.h"
00021 #include "sodium/crypto_core_hchacha20.h"
00022 #include "sodium/crypto_core_salsa20.h"
00023 #include "sodium/crypto_core_salsa2012.h"
00024 #include "sodium/crypto_core_salsa208.h"
00025 #include "sodium/crypto_generichash.h"
00026 #include "sodium/crypto_generichash_blake2b.h"
00027 #include "sodium/crypto_hash.h"
00028 #include "sodium/crypto_hash_sha256.h"
00029 #include "sodium/crypto_hash_sha512.h"
00030 #include "sodium/crypto_kdf.h"
00031 #include "sodium/crypto_kdf_blake2b.h"
00032 #include "sodium/crypto_kx.h"
00033 #include "sodium/crypto_onetimeauth.h"
00034 #include "sodium/crypto_onetimeauth_poly1305.h"
00035 #include "sodium/crypto_pwhash.h"
00036 #include "sodium/crypto_pwhash_argon2i.h"
00037 #include "sodium/crypto_scalarmult.h"
00038 #include "sodium/crypto_scalarmult_curve25519.h"
00039 #include "sodium/crypto_secretbox.h"
00040 #include "sodium/crypto_secretbox_xsalsa20poly1305.h"
00041 #include "sodium/crypto_secretstream_xchacha20poly1305.h"
00042 #include "sodium/crypto_shorthash.h"
00043 #include "sodium/crypto_shorthash_siphash24.h"
00044 #include "sodium/crypto_sign.h"
00045 #include "sodium/crypto_sign_ed25519.h"
00046 #include "sodium/crypto_stream.h"
00047 #include "sodium/crypto_stream_chacha20.h"
00048 #include "sodium/crypto_stream_salsa20.h"
00049 #include "sodium/crypto_stream_xsalsa20.h"
00050 #include "sodium/crypto_verify_16.h"
00051 #include "sodium/crypto_verify_32.h"
00052 #include "sodium/crypto_verify_64.h"
00053 #include "sodium/randombytes.h"
00054 #ifdef __native_client__
00055 # include "sodium/randombytes_nativeclient.h"
00056 #endif
00057 #include "sodium/randombytes_salsa20_random.h"
00058 #include "sodium/randombytes_sysrandom.h"
00059 #include "sodium/runtime.h"
00060 #include "sodium/utils.h"
00061
00062 #ifndef SODIUM_LIBRARY_MINIMAL
00063 # include "sodium/crypto_box_curve25519xchacha20poly1305.h"
00064 # include "sodium/crypto_core_ed25519.h"
00065 # include "sodium/crypto_scalarmult_ed25519.h"
00066 # include "sodium/crypto_secretbox_xchacha20poly1305.h"
00067 # include "sodium/crypto_pwhash_scryptsalsa208sha256.h"
```

```
00068 # include "sodium/crypto_stream_salsa2012.h"
00069 # include "sodium/crypto_stream_salsa208.h"
00070 # include "sodium/crypto_stream_xchacha20.h"
00071 #endif
00072
00073 #endif
```

Index