Nano cryptocurrency C library with P2PoW/DPoW support for Embedded

1.0.0

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_bitcoin_serialize_t Struct Reference

```
#include <f_bitcoin.h>
```

Data Fields

- uint8_t **version_bytes** [4]
- uint8_t **master_node**
- uint8_t **finger_print** [4]
- uint8_t **child_number** [4]
- uint8_t **chain_code** [32]
- uint8_t **sk_or_pk_data** [33]
- uint8_t **chksum** [4]

4.1.1   Detailed Description

Definition at line **24** of file **f_bitcoin.h**.

4.1.2   Field Documentation

4.1.2.1   chain_code

```
uint8_t chain_code[32]
```

Definition at line **29** of file **f_bitcoin.h**.

4.1.2.2 child_number

```
uint8_t child_number[4]
```

Definition at line **28** of file **f_bitcoin.h**.

4.1.2.3 chksum

```
uint8_t chksum[4]
```

Definition at line **31** of file **f_bitcoin.h**.

4.1.2.4 finger_print

```
uint8_t finger_print[4]
```

Definition at line **27** of file **f_bitcoin.h**.

4.1.2.5 master_node

```
uint8_t master_node
```

Definition at line **26** of file **f_bitcoin.h**.

4.1.2.6 sk_or_pk_data

```
uint8_t sk_or_pk_data[33]
```

Definition at line **30** of file **f_bitcoin.h**.

4.1.2.7 version_bytes

```
uint8_t version_bytes[4]
```

Definition at line **25** of file **f_bitcoin.h**.

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

- **f_bitcoin.h**

4.2   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**

4.2.1   Detailed Description

Nano signed block raw data defined in this `reference`

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

4.2.2   Field Documentation

4.2.2.1   account

```
uint8_t account[32]
```

Account in raw binary data.

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

4.2.2.2   balance

f_uint128_t `balance`

Big number 128 bit raw balance.

See also

>   **f_uint128_t** (p. **??**)

Definition at line **280** of file **f_nano_crypto_util.h**.

4.2.2.3 link

```
uint8_t link[32]
```

link or destination account

Definition at line **282** of file **f_nano_crypto_util.h**.

4.2.2.4 preamble

```
uint8_t preamble[32]
```

Block preamble.

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

4.2.2.5 prefixes

```
uint8_t prefixes
```

Internal use for this API.

Definition at line **286** of file **f_nano_crypto_util.h**.

4.2.2.6 previous

```
uint8_t previous[32]
```

Previous block.

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

4.2.2.7 representative

```
uint8_t representative[32]
```

Representative for current account.

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

4.2.2.8 signature

```
uint8_t signature[64]
```

Signature of the block.

Definition at line **284** of file **f_nano_crypto_util.h**.

4.2.2.9 work

```
uint64_t work
```

Internal use for this API.

Definition at line **288** of file **f_nano_crypto_util.h**.

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

- **f_nano_crypto_util.h**

4.3 f_file_info_err_t Struct Reference

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

4.3.1 Detailed Description

Error enumerator for info file functions.

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

- **f_nano_crypto_util.h**

4.4 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**

4.4.1 Detailed Description

struct of the block of encrypted file to store Nano SEED

Definition at line **402** of file **f_nano_crypto_util.h**.

4.4.2 Field Documentation

4.4.2.1 description

```
uint8_t description[F_DESC_SZ]
```

File description.

Definition at line **408** of file **f_nano_crypto_util.h**.

4.4.2.2 iv

```
uint8_t iv[16]
```

Initial vector of first encryption layer.

Definition at line **412** of file **f_nano_crypto_util.h**.

4.4.2.3 nano_hdr

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

Header of the file.

Definition at line **404** of file **f_nano_crypto_util.h**.

4.4.2.4 salt

```
uint8_t salt[32]
```

Salt of the first encryption layer.

Definition at line **410** of file **f_nano_crypto_util.h**.

4.4.2.5 seed_block

`F_ENCRYPTED_BLOCK seed_block`

Second encrypted block for Nano SEED.

Definition at line **414** of file **f_nano_crypto_util.h**.

4.4.2.6 ver

`uint32_t ver`

Version of the file.

Definition at line **406** 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_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]

4.5.1 Detailed Description

struct of the block of encrypted file to store Nano SEED

Definition at line **374** of file **f_nano_crypto_util.h**.

4.5.2 Field Documentation

4.5.2.1 hash_sk_unencrypted

`uint8_t hash_sk_unencrypted[32]`

hash of Nano SEED when unencrypted

Definition at line **382** of file **f_nano_crypto_util.h**.

4.5.2.2 iv

`uint8_t iv[16]`

Initial sub vector.

Definition at line **378** of file **f_nano_crypto_util.h**.

4.5.2.3 reserved

`uint8_t reserved[16]`

Reserved (not used)

Definition at line **380** of file **f_nano_crypto_util.h**.

4.5.2.4 sk_encrypted

`uint8_t sk_encrypted[32]`

Secret.

SEED encrypted (second layer)

Definition at line **384** of file **f_nano_crypto_util.h**.

4.5.2.5 sub_salt

`uint8_t sub_salt[32]`

Salt of the sub block to be stored.

Definition at line **376** 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_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]

4.6.1   Detailed Description

struct of the body block of the info file

Definition at line **486** of file **f_nano_crypto_util.h**.

4.6.2   Field Documentation

4.6.2.1   last_used_wallet_number

```
uint32_t last_used_wallet_number
```

Last used wallet number.

Definition at line **490** of file **f_nano_crypto_util.h**.

4.6.2.2   max_fee

```
char max_fee[F_RAW_STR_MAX_SZ]
```

Custom preferred max fee of Proof of Work.

Definition at line **494** of file **f_nano_crypto_util.h**.

4.6.2.3   reserved

```
uint8_t reserved[44]
```

Reserved.

Definition at line **496** of file **f_nano_crypto_util.h**.

4.6.2.4   wallet_prefix

`uint8_t wallet_prefix`

Wallet prefix: 0 for NANO; 1 for XRB.

Definition at line **488** of file **f_nano_crypto_util.h**.

4.6.2.5   wallet_representative

`char wallet_representative[ `**`MAX_STR_NANO_CHAR`**`]`

Wallet representative.

Definition at line **492** of file **f_nano_crypto_util.h**.

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

- **f_nano_crypto_util.h**

4.7   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**

4.7.1   Detailed Description

struct of the body block of the info file

Definition at line **518** of file **f_nano_crypto_util.h**.

4.7.2   Field Documentation

4.7.2.1 body

`F_NANO_WALLET_INFO_BODY body`

Body of the file info.

Definition at line **530** of file **f_nano_crypto_util.h**.

4.7.2.2 desc

`char desc[F_NANO_DESC_SZ]`

Description.

Definition at line **524** of file **f_nano_crypto_util.h**.

4.7.2.3 file_info_integrity

`uint8_t file_info_integrity[32]`

File info integrity of the body block.

Definition at line **528** of file **f_nano_crypto_util.h**.

4.7.2.4 header

`uint8_t header[sizeof(F_NANO_WALLET_INFO_MAGIC)]`

Header magic.

Definition at line **520** of file **f_nano_crypto_util.h**.

4.7.2.5 nanoseed_hash

`uint8_t nanoseed_hash[32]`

Nano SEED hash file.

Definition at line **526** of file **f_nano_crypto_util.h**.

4.7.2.6 version

`uint16_t version`

Version.

Definition at line **522** 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 errors.h File Reference

Macros

- #define **ERROR_SUCCESS** 0
- #define **ERROR_GEN_TOKEN_NO_RAND_NUM_GEN** 3858
- #define **ERROR_INVALID_NANO_ADDRESS_VERIFY_CHKSUM** 23
- #define **INVALID_RAW_BALANCE** 8893
- #define **CANT_OPEN_DICTIONARY_FILE** 2580
- #define **MISSING_PASSWORD** 7153
- #define **EMPTY_PASSWORD** 7169
- #define **WRONG_PASSWORD** 7167
- #define **ERROR_25519_IS_NOT_CANONICAL_OR_HAS_NOT_SMALL_ORDER** 12621
- #define **ERROR_NANO_BLOCK** 13014
- #define **ERROR_P2POW_BLOCK** 13015
- #define **ERROR_BRAINWALLET_ALLOW_MODE_NOT_ACCEPTED** 0x3C00
- #define **ERROR_MISSING_BRAINWALLET** 0x3C01
- #define **ERROR_MISSING_SALT** 0x3C02

Enumerations

- enum **f_nano_account_or_pk_string_to_pk_util_err_t** {
  NANO_ACCOUNT_TO_PK_OK =0, **NANO_ACCOUNT_TO_PK_OVFL** =8100, **NANO_ACCOUNT_TO←**
  _PK_NULL_STRING, **NANO_ACCOUNT_WRONG_PK_STR_SZ**,
  NANO_ACCOUNT_WRONG_HEX_STRING, **NANO_ACCOUNT_BASE32_CONVERT_ERROR**, **NAN←**
  O_ACCOUNT_TO_PK_WRONG_ACCOUNT_LEN }

5.1.1 Macro Definition Documentation

5.1.1.1 CANT_OPEN_DICTIONARY_FILE

`#define CANT_OPEN_DICTIONARY_FILE 2580`

Dictionary file not found or filesystem error.

Definition at line **50** of file **errors.h**.

5.1.1.2 EMPTY_PASSWORD

`#define EMPTY_PASSWORD 7169`

Empty password error.

Definition at line **62** of file **errors.h**.

5.1.1.3 ERROR_25519_IS_NOT_CANONICAL_OR_HAS_NOT_SMALL_ORDER

`#define ERROR_25519_IS_NOT_CANONICAL_OR_HAS_NOT_SMALL_ORDER 12621`

Error in Elliptic Curve Ed25519: Is not canonical or has small order.

Definition at line **74** of file **errors.h**.

5.1.1.4 ERROR_BRAINWALLET_ALLOW_MODE_NOT_ACCEPTED

`#define ERROR_BRAINWALLET_ALLOW_MODE_NOT_ACCEPTED 0x3C00`

Definition at line **92** of file **errors.h**.

5.1.1.5 ERROR_GEN_TOKEN_NO_RAND_NUM_GEN

`#define ERROR_GEN_TOKEN_NO_RAND_NUM_GEN 3858`

No random number generation.

Add one to *myNanoEmbedded* library.

See also

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

Definition at line **15** of file **errors.h**.

5.1.1.6 ERROR_INVALID_NANO_ADDRESS_VERIFY_CHKSUM

```
#define ERROR_INVALID_NANO_ADDRESS_VERIFY_CHKSUM 23
```

Nano address checksum invalid.

Definition at line **22** of file **errors.h**.

5.1.1.7 ERROR_MISSING_BRAINWALLET

```
#define ERROR_MISSING_BRAINWALLET 0x3C01
```

Definition at line **98** of file **errors.h**.

5.1.1.8 ERROR_MISSING_SALT

```
#define ERROR_MISSING_SALT 0x3C02
```

Definition at line **104** of file **errors.h**.

5.1.1.9 ERROR_NANO_BLOCK

```
#define ERROR_NANO_BLOCK 13014
```

Nano block error.

Definition at line **80** of file **errors.h**.

5.1.1.10 ERROR_P2POW_BLOCK

```
#define ERROR_P2POW_BLOCK 13015
```

Nano P2PoW block error.

Definition at line **86** of file **errors.h**.

5.1.1.11 ERROR_SUCCESS

```
#define ERROR_SUCCESS 0
```

Error success.

Most of the *myNanoEmbedded* functions returns **ERROR_SUCCESS** when execution success.

Definition at line **8** of file **errors.h**.

5.1.1.12 INVALID_RAW_BALANCE

```
#define INVALID_RAW_BALANCE 8893
```

Invalid raw balance error.

Definition at line **43** of file **errors.h**.

5.1.1.13 MISSING_PASSWORD

```
#define MISSING_PASSWORD 7153
```

Missing password error.

Definition at line **56** of file **errors.h**.

5.1.1.14 WRONG_PASSWORD

```
#define WRONG_PASSWORD 7167
```

Wrong password error.

Definition at line **68** of file **errors.h**.

5.1.2 Enumeration Type Documentation

5.1.2.1 f_nano_account_or_pk_string_to_pk_util_err_t

```
enum  f_nano_account_or_pk_string_to_pk_util_err_t
```

Nano account or public key string error enumerator.

Enumerator

NANO_ACCOUNT_TO_PK_OK	
NANO_ACCOUNT_TO_PK_OVFL	
NANO_ACCOUNT_TO_PK_NULL_STRING	
NANO_ACCOUNT_WRONG_PK_STR_SZ	
NANO_ACCOUNT_WRONG_HEX_STRING	
NANO_ACCOUNT_BASE32_CONVERT_ERROR	
NANO_ACCOUNT_TO_PK_WRONG_ACCOUNT_LEN	

Definition at line **28** of file **errors.h**.

5.2 errors.h

```
00001 //mon apr 26 20:56:00 -03 2021
00002 #ifndef ERRORS_H
00003 #define ERRORS_H
00004
00008 #define ERROR_SUCCESS 0
00009
00015 #define ERROR_GEN_TOKEN_NO_RAND_NUM_GEN 3858
00016
00017 //nano_base_32_2_hex
00022 #define ERROR_INVALID_NANO_ADDRESS_VERIFY_CHKSUM 23
00023
00028 enum f_nano_account_or_pk_string_to_pk_util_err_t {
00029     NANO_ACCOUNT_TO_PK_OK=0,
00030     NANO_ACCOUNT_TO_PK_OVFL=8100,
00031     NANO_ACCOUNT_TO_PK_NULL_STRING,
00032     NANO_ACCOUNT_WRONG_PK_STR_SZ,
00033     NANO_ACCOUNT_WRONG_HEX_STRING,
00034     NANO_ACCOUNT_BASE32_CONVERT_ERROR,
00035     NANO_ACCOUNT_TO_PK_WRONG_ACCOUNT_LEN
00036 };
00037
00038 //valid_raw_balance
00043 #define INVALID_RAW_BALANCE 8893
00044
00045 //f_nano_seed_to_bip39
00050 #define CANT_OPEN_DICTIONARY_FILE 2580
00051
00056 #define MISSING_PASSWORD 7153
00057
00062 #define EMPTY_PASSWORD 7169
00063
00068 #define WRONG_PASSWORD 7167
00069
00074 #define ERROR_25519_IS_NOT_CANONICAL_OR_HAS_NOT_SMALL_ORDER 12621
00075
00080 #define ERROR_NANO_BLOCK 13014
00081
00086 #define ERROR_P2POW_BLOCK 13015
00087
00092 #define ERROR_BRAINWALLET_ALLOW_MODE_NOT_ACCEPTED 0x3C00
00093
00098 #define ERROR_MISSING_BRAINWALLET 0x3C01
00099
00104 #define ERROR_MISSING_SALT 0x3C02
00105 #endif
```

5.3 f_add_bn_288_le.h File Reference

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

Typedefs

- typedef uint8_t **F_ADD_288**[36]

5.3.1 Detailed Description

Low level implementation of Nano Cryptocurrency C library.

Definition in file **f_add_bn_288_le.h**.

5.3.2 Typedef Documentation

5.3.2.1 F_ADD_288

```
F_ADD_288
```

288 bit big number

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

5.4 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
```

5.5 f_bitcoin.h File Reference

```
#include <mbedtls/bignum.h>
```

Data Structures

- struct **f_bitcoin_serialize_t**

Macros

- #define **F_BITCOIN_WIF_MAINNET** (uint8_t)0x80
- #define **F_BITCOIN_WIF_TESTNET** (uint8_t)0xEF
- #define **F_BITCOIN_P2PKH** (uint8_t)0x00
- #define **F_BITCOIN_T2PKH** (uint8_t)0x6F
- #define **F_BITCOIN_BUF_SZ** (size_t)512
- #define **F_MAX_BASE58_LENGTH** (size_t)112
- #define **F_BITCOIN_SEED_GENERATOR** "Bitcoin seed"
- #define **MAINNET_PUBLIC** (size_t)0
- #define **MAINNET_PRIVATE** (size_t)1
- #define **TESTNET_PUBLIC** (size_t)2
- #define **TESTNET_PRIVATE** (size_t)3
- #define **F_VERSION_BYTES_IDX_LEN** (size_t)(sizeof(**F_VERSION_BYTES**)/(4∗sizeof(uint8_t)))
- #define **F_XPRIV_BASE58** (int)1
- #define **F_XPUB_BASE58** (int)2
- #define **DERIVE_XPRIV_XPUB_DYN_OUT_BASE58** (int)8
- #define **DERIVE_XPRIV_XPUB_DYN_OUT_XPRIV** (int)16
- #define **DERIVE_XPRIV_XPUB_DYN_OUT_XPUB** (int)32
- #define **F_GET_XKEY_IS_BASE58** (int)0x00008000

Typedefs

- typedef enum **error_load_master_key_from_entropy_bits_e** **ERROR_LOAD_FROM_MASTER_KEY**↩
  _FROM_ENTROPY_BITS
- typedef enum **master_key_entropy_bits_e** **MASTER_KEY_ENTROPY_BITS**

Enumerations

- enum **error_load_master_key_from_entropy_bits_e** { **ERR_INVALID_ENTROPY_BITS** = 20300, **ER**↩
  R_INVALID_VERSION_BYTES, **ERR_MASTER_KEY_FROM_ENTROPY_ALLOC** }
- enum **master_key_entropy_bits_e** { **MK_128** = 16, **MK_256** = 32, **MK_512** = 64 }

Functions

- struct **f_bitcoin_serialize_t** **__attribute__** ((packed)) BITCOIN_SERIALIZE
- int **f_decode_b58_util** (uint8_t ∗, size_t, size_t ∗, const char ∗)
- int **f_encode_b58** (char ∗, size_t, size_t ∗, uint8_t ∗, size_t)
- int **f_private_key_to_wif** (char ∗, size_t, size_t ∗, uint8_t, uint8_t ∗)
- int **f_wif_to_private_key** (uint8_t ∗, unsigned char ∗, const char ∗)
- int **f_generate_master_key** (BITCOIN_SERIALIZE ∗, size_t, uint32_t)
- int **f_bitcoin_valid_bip32** (BITCOIN_SERIALIZE ∗, int ∗, void ∗, int)
- int **f_uncompress_elliptic_curve** (uint8_t ∗, size_t, size_t ∗, mbedtls_ecp_group_id, uint8_t ∗, size_t)
- int **f_bip32_to_public_key_or_private_key** (uint8_t ∗, int ∗, uint8_t ∗, uint8_t ∗, uint8_t ∗, uint32_t, const
  void ∗, int)
- int **f_public_key_to_address** (char ∗, size_t, size_t ∗, uint8_t ∗, uint8_t)
- int **f_xpriv2xpub** (void ∗, size_t, size_t ∗, void ∗, int)
- int **load_master_private_key** (void ∗, unsigned char ∗, size_t)
- int **f_fingerprint** (uint8_t ∗, uint8_t ∗, uint8_t ∗)
- int **f_get_xkey_type** (void ∗)
- int **f_derive_xpriv_or_xpub_dynamic** (void ∗∗, uint8_t ∗, uint32_t ∗, void ∗, uint32_t, int)
- int **f_derive_xkey_dynamic** (void ∗∗, void ∗, const char ∗, int)
- int **f_check_if_invalid_btc_public_key** (uint8_t ∗)
- **ERROR_LOAD_FROM_MASTER_KEY_FROM_ENTROPY_BITS** **f_load_from_master_key_from_**↩
  entropy_bits (BITCOIN_SERIALIZE ∗, size_t, const uint8_t ∗, **MASTER_KEY_ENTROPY_BITS**)

Variables

- static const uint8_t **F_VERSION_BYTES** [ ][4]
- uint8_t **version_bytes** [4]
- uint8_t **master_node**
- uint8_t **finger_print** [4]
- uint8_t **child_number** [4]
- uint8_t **chain_code** [32]
- uint8_t **sk_or_pk_data** [33]
- uint8_t **chksum** [4]

5.5.1 Macro Definition Documentation

5.5.1.1 DERIVE_XPRIV_XPUB_DYN_OUT_BASE58

```
#define DERIVE_XPRIV_XPUB_DYN_OUT_BASE58 (int)8
```

Definition at line **58** of file **f_bitcoin.h**.

5.5.1.2 DERIVE_XPRIV_XPUB_DYN_OUT_XPRIV

```
#define DERIVE_XPRIV_XPUB_DYN_OUT_XPRIV (int)16
```

Definition at line **59** of file **f_bitcoin.h**.

5.5.1.3 DERIVE_XPRIV_XPUB_DYN_OUT_XPUB

```
#define DERIVE_XPRIV_XPUB_DYN_OUT_XPUB (int)32
```

Definition at line **60** of file **f_bitcoin.h**.

5.5.1.4 F_BITCOIN_BUF_SZ

```
#define F_BITCOIN_BUF_SZ (size_t)512
```

Definition at line **7** of file **f_bitcoin.h**.

5.5.1.5 F_BITCOIN_P2PKH

`#define F_BITCOIN_P2PKH (uint8_t)0x00`

Definition at line **5** of file **f_bitcoin.h**.

5.5.1.6 F_BITCOIN_SEED_GENERATOR

`#define F_BITCOIN_SEED_GENERATOR "Bitcoin seed"`

Definition at line **9** of file **f_bitcoin.h**.

5.5.1.7 F_BITCOIN_T2PKH

`#define F_BITCOIN_T2PKH (uint8_t)0x6F`

Definition at line **6** of file **f_bitcoin.h**.

5.5.1.8 F_BITCOIN_WIF_MAINNET

`#define F_BITCOIN_WIF_MAINNET (uint8_t)0x80`

Definition at line **3** of file **f_bitcoin.h**.

5.5.1.9 F_BITCOIN_WIF_TESTNET

`#define F_BITCOIN_WIF_TESTNET (uint8_t)0xEF`

Definition at line **4** of file **f_bitcoin.h**.

5.5.1.10 F_GET_XKEY_IS_BASE58

`#define F_GET_XKEY_IS_BASE58 (int)0x00008000`

Definition at line **62** of file **f_bitcoin.h**.

5.5.1.11 F_MAX_BASE58_LENGTH

```
#define F_MAX_BASE58_LENGTH (size_t)112
```

Definition at line **8** of file **f_bitcoin.h**.

5.5.1.12 F_VERSION_BYTES_IDX_LEN

```
#define F_VERSION_BYTES_IDX_LEN (size_t)(sizeof( F_VERSION_BYTES)/(4*sizeof(uint8_t)))
```

Definition at line **22** of file **f_bitcoin.h**.

5.5.1.13 F_XPRIV_BASE58

```
#define F_XPRIV_BASE58 (int)1
```

Definition at line **52** of file **f_bitcoin.h**.

5.5.1.14 F_XPUB_BASE58

```
#define F_XPUB_BASE58 (int)2
```

Definition at line **53** of file **f_bitcoin.h**.

5.5.1.15 MAINNET_PRIVATE

```
#define MAINNET_PRIVATE (size_t)1
```

Definition at line **12** of file **f_bitcoin.h**.

5.5.1.16 MAINNET_PUBLIC

```
#define MAINNET_PUBLIC (size_t)0
```

Definition at line **11** of file **f_bitcoin.h**.

5.5.1.17 TESTNET_PRIVATE

```
#define TESTNET_PRIVATE (size_t)3
```

Definition at line **14** of file **f_bitcoin.h**.

5.5.1.18 TESTNET_PUBLIC

```
#define TESTNET_PUBLIC (size_t)2
```

Definition at line **13** of file **f_bitcoin.h**.

5.5.2 Typedef Documentation

5.5.2.1 ERROR_LOAD_FROM_MASTER_KEY_FROM_ENTROPY_BITS

typedef enum **error_load_master_key_from_entropy_bits_e** **ERROR_LOAD_FROM_MASTER_KEY_FROM_ENT**↩
ROPY_BITS

5.5.2.2 MASTER_KEY_ENTROPY_BITS

typedef enum **master_key_entropy_bits_e** **MASTER_KEY_ENTROPY_BITS**

5.5.3 Enumeration Type Documentation

5.5.3.1 error_load_master_key_from_entropy_bits_e

enum **error_load_master_key_from_entropy_bits_e**

Enumerator

ERR_INVALID_ENTROPY_BITS	
ERR_INVALID_VERSION_BYTES	
ERR_MASTER_KEY_FROM_ENTROPY_ALLOC	

Definition at line **68** of file **f_bitcoin.h**.

5.5.3.2 master_key_entropy_bits_e

enum **master_key_entropy_bits_e**

Enumerator

MK_128	
MK_256	
MK_512	

Definition at line **74** of file **f_bitcoin.h**.

5.5.4 Function Documentation

5.5.4.1 __attribute__()

struct **f_nano_wallet_info_t** __attribute__ (
          (packed))

5.5.4.2 f_bip32_to_public_key_or_private_key()

int f_bip32_to_public_key_or_private_key (
          uint8_t * ,
          int * ,
          uint8_t * ,
          uint8_t * ,
          uint8_t * ,
          uint32_t ,
          const void * ,
          int)

5.5.4.3 f_bitcoin_valid_bip32()

int f_bitcoin_valid_bip32 (
          BITCOIN_SERIALIZE * ,
          int * ,
          void * ,
          int)

5.5.4.4 f_check_if_invalid_btc_public_key()

```
int f_check_if_invalid_btc_public_key (
            uint8_t *  )
```

5.5.4.5 f_decode_b58_util()

```
int f_decode_b58_util (
            uint8_t * ,
            size_t ,
            size_t * ,
            const char *  )
```

5.5.4.6 f_derive_xkey_dynamic()

```
int f_derive_xkey_dynamic (
            void ** ,
            void * ,
            const char * ,
            int  )
```

5.5.4.7 f_derive_xpriv_or_xpub_dynamic()

```
int f_derive_xpriv_or_xpub_dynamic (
            void ** ,
            uint8_t * ,
            uint32_t * ,
            void * ,
            uint32_t ,
            int  )
```

5.5.4.8 f_encode_b58()

```
int f_encode_b58 (
            char * ,
            size_t ,
            size_t * ,
            uint8_t * ,
            size_t  )
```

5.5.4.9 f_fingerprint()

```
int f_fingerprint (
            uint8_t * ,
            uint8_t * ,
            uint8_t *  )
```

5.5.4.10 f_generate_master_key()

```
int f_generate_master_key (
            BITCOIN_SERIALIZE * ,
            size_t ,
            uint32_t  )
```

5.5.4.11 f_get_xkey_type()

```
int f_get_xkey_type (
            void *  )
```

5.5.4.12 f_load_from_master_key_from_entropy_bits()

```
 ERROR_LOAD_FROM_MASTER_KEY_FROM_ENTROPY_BITS f_load_from_master_key_from_entropy_bits (
            BITCOIN_SERIALIZE * ,
            size_t ,
            const uint8_t * ,
             MASTER_KEY_ENTROPY_BITS  )
```

5.5.4.13 f_private_key_to_wif()

```
int f_private_key_to_wif (
            char * ,
            size_t ,
            size_t * ,
            uint8_t ,
            uint8_t *  )
```

5.5.4.14   f_public_key_to_address()

```
int f_public_key_to_address (
            char * ,
            size_t ,
            size_t * ,
            uint8_t * ,
            uint8_t  )
```

5.5.4.15   f_uncompress_elliptic_curve()

```
int f_uncompress_elliptic_curve (
            uint8_t * ,
            size_t ,
            size_t * ,
            mbedtls_ecp_group_id ,
            uint8_t * ,
            size_t  )
```

5.5.4.16   f_wif_to_private_key()

```
int f_wif_to_private_key (
            uint8_t * ,
            unsigned char * ,
            const char *  )
```

5.5.4.17   f_xpriv2xpub()

```
int f_xpriv2xpub (
            void * ,
            size_t ,
            size_t * ,
            void * ,
            int  )
```

5.5.4.18   load_master_private_key()

```
int load_master_private_key (
            void * ,
            unsigned char * ,
            size_t  )
```

5.5.5 Variable Documentation

5.5.5.1 chain_code

```
uint8_t chain_code[32]
```

Definition at line **21** of file **f_bitcoin.h**.

5.5.5.2 child_number

```
uint8_t child_number[4]
```

Definition at line **20** of file **f_bitcoin.h**.

5.5.5.3 chksum

```
uint8_t chksum[4]
```

Definition at line **23** of file **f_bitcoin.h**.

5.5.5.4 F_VERSION_BYTES

```
const uint8_t F_VERSION_BYTES[][4]  [static]
```

Initial value:

```
= {
    {0x04, 0x88, 0xB2, 0x1E},
    {0x04, 0x88, 0xAD, 0xE4},
    {0x04, 0x35, 0x87, 0xCF},
    {0x04, 0x35, 0x83, 0x94}
}
```

Definition at line **16** of file **f_bitcoin.h**.

5.5.5.5 finger_print

```
uint8_t finger_print[4]
```

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

5.5.5.6 master_node

```
uint8_t master_node
```

Definition at line **18** of file **f_bitcoin.h**.

5.5.5.7 sk_or_pk_data

```
uint8_t sk_or_pk_data[33]
```

Definition at line **22** of file **f_bitcoin.h**.

5.5.5.8 version_bytes

```
uint8_t version_bytes[4]
```

Definition at line **17** of file **f_bitcoin.h**.

5.6   f_bitcoin.h

```
00001 #include <mbedtls/bignum.h>
00002
00003 #define F_BITCOIN_WIF_MAINNET (uint8_t)0x80
00004 #define F_BITCOIN_WIF_TESTNET (uint8_t)0xEF
00005 #define F_BITCOIN_P2PKH (uint8_t)0x00 // P2PKH address
00006 #define F_BITCOIN_T2PKH (uint8_t)0x6F // Testnet Address
00007 #define F_BITCOIN_BUF_SZ (size_t)512
00008 #define F_MAX_BASE58_LENGTH (size_t)112//52 // including null char
00009 #define F_BITCOIN_SEED_GENERATOR "Bitcoin seed"
00010
00011 #define MAINNET_PUBLIC (size_t)0
00012 #define MAINNET_PRIVATE (size_t)1
00013 #define TESTNET_PUBLIC (size_t)2
00014 #define TESTNET_PRIVATE (size_t)3
00015
00016 static const uint8_t F_VERSION_BYTES[][4] = {
00017     {0x04, 0x88, 0xB2, 0x1E}, //mainnet public
00018     {0x04, 0x88, 0xAD, 0xE4}, //mainnet private
00019     {0x04, 0x35, 0x87, 0xCF}, //testnet public
00020     {0x04, 0x35, 0x83, 0x94} // testnet private
00021 };
00022 #define F_VERSION_BYTES_IDX_LEN (size_t)(sizeof(F_VERSION_BYTES)/(4*sizeof(uint8_t)))
00023
00024 typedef struct f_bitcoin_serialize_t {
00025     uint8_t version_bytes[4];
00026     uint8_t master_node;
00027     uint8_t finger_print[4];
00028     uint8_t child_number[4];
00029     uint8_t chain_code[32];
00030     uint8_t sk_or_pk_data[33];
00031     uint8_t chksum[4];
00032 } __attribute__((packed)) BITCOIN_SERIALIZE;
00033
00034 int f_decode_b58_util(uint8_t *, size_t, size_t *, const char *);
00035 int f_encode_b58(char *, size_t, size_t *, uint8_t *, size_t);
00036 int f_private_key_to_wif(char *, size_t, size_t *, uint8_t, uint8_t *);
00037 int f_wif_to_private_key(uint8_t *, unsigned char *, const char *);
00038 int f_generate_master_key(BITCOIN_SERIALIZE *, size_t, uint32_t);
00039 int f_bitcoin_valid_bip32(BITCOIN_SERIALIZE *, int *, void *, int);
00040 int f_uncompress_elliptic_curve(uint8_t *, size_t, size_t *, mbedtls_ecp_group_id, uint8_t *, size_t);
00041 int f_bip32_to_public_key_or_private_key(
00042     uint8_t *,
```

```
00043    int *,
00044    uint8_t *,
00045    uint8_t *,
00046    uint8_t *,
00047    uint32_t,
00048    const void *,
00049    int
00050 );
00051 int f_public_key_to_address(char *, size_t, size_t *, uint8_t *, uint8_t);
00052 #define F_XPRIV_BASE58 (int)1
00053 #define F_XPUB_BASE58 (int)2
00054 int f_xpriv2xpub(void *, size_t, size_t *, void *, int);
00055 int load_master_private_key(void *, unsigned char *, size_t);
00056 int f_fingerprint(uint8_t *, uint8_t *, uint8_t *);
00057
00058 #define DERIVE_XPRIV_XPUB_DYN_OUT_BASE58 (int)8
00059 #define DERIVE_XPRIV_XPUB_DYN_OUT_XPRIV (int)16
00060 #define DERIVE_XPRIV_XPUB_DYN_OUT_XPUB (int)32
00061
00062 #define F_GET_XKEY_IS_BASE58 (int)0x00008000
00063 int f_get_xkey_type(void *);
00064 int f_derive_xpriv_or_xpub_dynamic(void **, uint8_t *, uint32_t *, void *, uint32_t, int);
00065 int f_derive_xkey_dynamic(void **, void *, const char *, int);
00066 int f_check_if_invalid_btc_public_key(uint8_t *);
00067
00068 typedef enum error_load_master_key_from_entropy_bits_e {
00069    ERR_INVALID_ENTROPY_BITS = 20300,
00070    ERR_INVALID_VERSION_BYTES,
00071    ERR_MASTER_KEY_FROM_ENTROPY_ALLOC
00072 } ERROR_LOAD_FROM_MASTER_KEY_FROM_ENTROPY_BITS;
00073
00074 typedef enum master_key_entropy_bits_e {
00075    MK_128 = 16,
00076    MK_256 = 32,
00077    MK_512 = 64
00078 } MASTER_KEY_ENTROPY_BITS;
00079
00080 ERROR_LOAD_FROM_MASTER_KEY_FROM_ENTROPY_BITS
00081 f_load_from_master_key_from_entropy_bits(
00082    BITCOIN_SERIALIZE *,
00083    size_t,
00084    const uint8_t *,
00085    MASTER_KEY_ENTROPY_BITS
00086 );
00087
```

5.7 f_nano_crypto_util.h File Reference

```
#include <errors.h>
#include <stdint.h>
#include <f_util.h>
#include <f_bitcoin.h>
#include <version.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 **F_BLOCK_TRANSFER_SIZE** (size_t)sizeof(F_BLOCK_TRANSFER)
- #define **F_P2POW_BLOCK_TRANSFER_SIZE** $2*$ **F_BLOCK_TRANSFER_SIZE**
- #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
- #define **F_SIGNATURE_RAW** (uint32_t)1
- #define **F_SIGNATURE_STRING** (uint32_t)2
- #define **F_SIGNATURE_OUTPUT_RAW_PK** (uint32_t)4
- #define **F_SIGNATURE_OUTPUT_STRING_PK** (uint32_t)8
- #define **F_SIGNATURE_OUTPUT_XRB_PK** (uint32_t)16
- #define **F_SIGNATURE_OUTPUT_NANO_PK** (uint32_t)32
- #define **F_IS_SIGNATURE_RAW_HEX_STRING** (uint32_t)64
- #define **F_MESSAGE_IS_HASH_STRING** (uint32_t)128
- #define **F_DEFAULT_THRESHOLD** (uint64_t) 0xfffffc000000000
- #define **F_VERIFY_SIG_NANO_WALLET** (uint32_t)1
- #define **F_PUBLIC_KEY_RAW_HEX** (uint32_t)2
- #define **F_PUBLIC_KEY_ASCII_HEX** (uint32_t)4
- #define **F_BALANCE_RAW_128** F_NANO_A_RAW_128
- #define **F_BALANCE_REAL_STRING** F_NANO_A_REAL_STRING
- #define **F_BALANCE_RAW_STRING** F_NANO_A_RAW_STRING
- #define **F_VALUE_SEND_RECEIVE_RAW_128** F_NANO_B_RAW_128
- #define **F_VALUE_SEND_RECEIVE_REAL_STRING** F_NANO_B_REAL_STRING
- #define **F_VALUE_SEND_RECEIVE_RAW_STRING** F_NANO_B_RAW_STRING
- #define **F_VALUE_TO_SEND** (int)(1<<0)
- #define **F_VALUE_TO_RECEIVE** (int)(1<<1)
- #define **F_FEE_VALUE_RAW_128** F_NANO_B_RAW_128
- #define **F_FEE_VALUE_REAL_STRING** F_NANO_B_REAL_STRING
- #define **F_FEE_VALUE_RAW_STRING** F_NANO_B_RAW_STRING

Typedefs

- typedef uint8_t **F_TOKEN**[16]
- 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**
- typedef enum **f_nano_create_block_dyn_err_t** **F_NANO_CREATE_BLOCK_DYN_ERR**
- typedef enum **f_nano_p2pow_block_dyn_err_t** **F_NANO_P2POW_BLOCK_DYN_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 }
- enum **f_nano_create_block_dyn_err_t** {
  NANO_CREATE_BLK_DYN_OK = 0, **NANO_CREATE_BLK_DYN_BLOCK_NULL** = 8000, **NANO_CR**↩
  EATE_BLK_DYN_ACCOUNT_NULL, **NANO_CREATE_BLK_DYN_COMPARE_BALANCE**,
  NANO_CREATE_BLK_DYN_GENESIS_WITH_NON_EMPTY_BALANCE, **NANO_CREATE_BLK_DY**↩
  N_CANT_SEND_IN_GENESIS_BLOCK, **NANO_CREATE_BLK_DYN_REP_NULL**, **NANO_CREATE_**↩
  BLK_DYN_BALANCE_NULL,
  NANO_CREATE_BLK_DYN_SEND_RECEIVE_NULL, **NANO_CREATE_BLK_DYN_LINK_NULL**, **NA**↩
  NO_CREATE_BLK_DYN_BUF_MALLOC, **NANO_CREATE_BLK_DYN_MALLOC**,

NANO_CREATE_BLK_DYN_WRONG_PREVIOUS_SZ, NANO_CREATE_BLK_DYN_WRONG_PREVI←
OUS_STR_SZ, NANO_CREATE_BLK_DYN_PARSE_STR_HEX_ERR, NANO_CREATE_BLK_DYN_←
FORBIDDEN_AMOUNT_TYPE,
NANO_CREATE_BLK_DYN_COMPARE, NANO_CREATE_BLK_DYN_EMPTY_VAL_TO_SEND_OR_←
REC, NANO_CREATE_BLK_DYN_INVALID_DIRECTION_OPTION }

- enum **f_nano_p2pow_block_dyn_err_t** {
NANO_P2POW_CREATE_BLOCK_OK = 0, NANO_P2POW_CREATE_BLOCK_INVALID_USER_BLO←
CK = 8400, NANO_P2POW_CREATE_BLOCK_MALLOC, NANO_P2POW_CREATE_BLOCK_NULL,
NANO_P2POW_CREATE_OUTPUT, NANO_P2POW_CREATE_OUTPUT_MALLOC }

Functions

- struct **f_block_transfer_t** **__attribute__** ((packed)) F_BLOCK_TRANSFER
- double **to_multiplier** (uint64_t, uint64_t)
- uint64_t **from_multiplier** (double, uint64_t)
- void **f_set_dictionary_path** (const char ∗)
- char ∗ **f_get_dictionary_path** (void)
- int **f_generate_token** (**F_TOKEN**, void ∗, size_t, const char ∗)
- int **f_verify_token** (**F_TOKEN**, void ∗, size_t, const char ∗)
- 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)
- int **f_nano_is_valid_block** (F_BLOCK_TRANSFER ∗)
- int **f_nano_block_to_json** (char ∗, size_t ∗, size_t, F_BLOCK_TRANSFER ∗)
- int **f_nano_get_block_hash** (uint8_t ∗, F_BLOCK_TRANSFER ∗)
- int **f_nano_get_p2pow_block_hash** (uint8_t ∗, uint8_t ∗, F_BLOCK_TRANSFER ∗)
- int **f_nano_p2pow_to_JSON** (char ∗, size_t ∗, size_t, F_BLOCK_TRANSFER ∗)
- 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_sign_data** (unsigned char ∗ **signature**, void ∗out_public_key, uint32_t ouput_type, const unsigned char ∗message, size_t msg_len, const unsigned char ∗private_key)
- int **f_verify_signed_data** (const unsigned char ∗, const unsigned char ∗, size_t, const void ∗, uint32_t)
- int **f_is_valid_nano_seed_encrypted** (void ∗, size_t, int)
- int **nano_create_block_dynamic** (F_BLOCK_TRANSFER ∗∗, const void ∗, size_t, const void ∗, size_t, const void ∗, size_t, const void ∗, const void ∗, uint32_t, const void ∗, size_t, int)
- int **nano_create_p2pow_block_dynamic** (F_BLOCK_TRANSFER ∗∗, F_BLOCK_TRANSFER ∗, const void ∗, size_t, const void ∗, uint32_t, const void ∗, size_t)
- int **f_verify_signed_block** (F_BLOCK_TRANSFER ∗)
- 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**

5.7.1 Detailed Description

This API Integrates Nano Cryptocurrency to low computational devices.

Definition in file **f_nano_crypto_util.h**.

5.7.2 Macro Definition Documentation

5.7.2.1 DEST_XRB

`#define DEST_XRB (uint8_t)0x01`

Definition at line **440** of file **f_nano_crypto_util.h**.

5.7.2.2 F_BALANCE_RAW_128

`#define F_BALANCE_RAW_128 F_NANO_A_RAW_128`

Balance is RAW 128 bit.

Definition at line **1452** of file **f_nano_crypto_util.h**.

5.7.2.3 F_BALANCE_RAW_STRING

`#define F_BALANCE_RAW_STRING F_NANO_A_RAW_STRING`

Balance is raw string.

Definition at line **1464** of file **f_nano_crypto_util.h**.

5.7.2.4 F_BALANCE_REAL_STRING

`#define F_BALANCE_REAL_STRING F_NANO_A_REAL_STRING`

Balance is real string.

Definition at line **1458** of file **f_nano_crypto_util.h**.

5.7.2.5 F_BLOCK_TRANSFER_SIZE

`#define F_BLOCK_TRANSFER_SIZE (size_t)sizeof(F_BLOCK_TRANSFER)`

Definition at line **291** of file **f_nano_crypto_util.h**.

5.7.2.6 F_BRAIN_WALLET_BAD

`#define F_BRAIN_WALLET_BAD (uint32_t)3`

[bad].

Crack within one day

Definition at line **1209** of file **f_nano_crypto_util.h**.

5.7.2.7 F_BRAIN_WALLET_GOOD

`#define F_BRAIN_WALLET_GOOD (uint32_t)8`

[good].

Crack within one thousand year

Definition at line **1240** of file **f_nano_crypto_util.h**.

5.7.2.8 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 **1233** of file **f_nano_crypto_util.h**.

5.7.2.9 F_BRAIN_WALLET_NICE

`#define F_BRAIN_WALLET_NICE (uint32_t)10`

[very nice].

Crack withing one hundred thousand year

Definition at line **1252** of file **f_nano_crypto_util.h**.

5.7.2.10 F_BRAIN_WALLET_PERFECT

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

[Perfect!] 3.34x10^{53} Years to crack

Definition at line **1258** of file **f_nano_crypto_util.h**.

5.7.2.11 F_BRAIN_WALLET_POOR

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

[poor].

Crack within minutes

Definition at line **1197** of file **f_nano_crypto_util.h**.

5.7.2.12 F_BRAIN_WALLET_STILL_WEAK

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

[still weak].

Crack within one year

Definition at line **1227** of file **f_nano_crypto_util.h**.

5.7.2.13 F_BRAIN_WALLET_VERY_BAD

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

[very bad].

Crack within one hour

Definition at line **1203** of file **f_nano_crypto_util.h**.

5.7.2.14 F_BRAIN_WALLET_VERY_GOOD

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

[very good].

Crack within ten thousand year

Definition at line **1246** of file **f_nano_crypto_util.h**.

5.7.2.15 F_BRAIN_WALLET_VERY_POOR

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

[very poor].

Crack within seconds or less

Definition at line **1191** of file **f_nano_crypto_util.h**.

5.7.2.16 F_BRAIN_WALLET_VERY_WEAK

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

[very weak].

Crack within one week

Definition at line **1215** of file **f_nano_crypto_util.h**.

5.7.2.17 F_BRAIN_WALLET_WEAK

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

[weak].

Crack within one month

Definition at line **1221** of file **f_nano_crypto_util.h**.

5.7.2.18 F_DEFAULT_THRESHOLD

```
#define F_DEFAULT_THRESHOLD (uint64_t) 0xffffffc000000000
```

Default Nano Proof of Work Threshold.

Definition at line **1361** of file **f_nano_crypto_util.h**.

5.7.2.19 F_FEE_VALUE_RAW_128

```
#define F_FEE_VALUE_RAW_128 F_NANO_B_RAW_128
```

P2PoW fee value is raw 128 bit.

Definition at line **1500** of file **f_nano_crypto_util.h**.

5.7.2.20 F_FEE_VALUE_RAW_STRING

```
#define F_FEE_VALUE_RAW_STRING F_NANO_B_RAW_STRING
```

P2PoW fee value is raw string.

Definition at line **1512** of file **f_nano_crypto_util.h**.

5.7.2.21 F_FEE_VALUE_REAL_STRING

```
#define F_FEE_VALUE_REAL_STRING F_NANO_B_REAL_STRING
```

P2PoW fee value is real string.

Definition at line **1506** of file **f_nano_crypto_util.h**.

5.7.2.22 F_IS_SIGNATURE_RAW_HEX_STRING

```
#define F_IS_SIGNATURE_RAW_HEX_STRING (uint32_t)64
```

Signature is raw hex string flag.

See also

> **f_sign_data()** (p. **??**)

Definition at line **1348** of file **f_nano_crypto_util.h**.

5.7.2.23 F_MESSAGE_IS_HASH_STRING

```
#define F_MESSAGE_IS_HASH_STRING (uint32_t)128
```

Message is raw hex hash string.

See also

> **f_sign_data()** (p. **??**)

Definition at line **1355** of file **f_nano_crypto_util.h**.

5.7.2.24 F_NANO_POW_MAX_THREAD

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

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

Default 10

Definition at line **140** of file **f_nano_crypto_util.h**.

5.7.2.25 F_P2POW_BLOCK_TRANSFER_SIZE

```
#define F_P2POW_BLOCK_TRANSFER_SIZE 2* F_BLOCK_TRANSFER_SIZE
```

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

5.7.2.26 F_PUBLIC_KEY_ASCII_HEX

```
#define F_PUBLIC_KEY_ASCII_HEX (uint32_t)4
```

Public key is a hex ASCII encoded string.

See also

> **f_verify_signed_data()** (p. **??**)

Definition at line **1413** of file **f_nano_crypto_util.h**.

5.7.2.27 F_PUBLIC_KEY_RAW_HEX

`#define F_PUBLIC_KEY_RAW_HEX (uint32_t)2`

Public key raw 32 bytes data.

See also

> **f_verify_signed_data()** (p. **??**)

Definition at line **1406** of file **f_nano_crypto_util.h**.

5.7.2.28 F_SIGNATURE_OUTPUT_NANO_PK

`#define F_SIGNATURE_OUTPUT_NANO_PK (uint32_t)32`

Public key is a NANO wallet encoded base32 string.

See also

> **f_sign_data()** (p. **??**)

Definition at line **1341** of file **f_nano_crypto_util.h**.

5.7.2.29 F_SIGNATURE_OUTPUT_RAW_PK

`#define F_SIGNATURE_OUTPUT_RAW_PK (uint32_t)4`

Public key is raw data.

See also

> **f_sign_data()** (p. **??**)

Definition at line **1320** of file **f_nano_crypto_util.h**.

5.7.2.30 F_SIGNATURE_OUTPUT_STRING_PK

`#define F_SIGNATURE_OUTPUT_STRING_PK (uint32_t)8`

Public key is hex ASCII encoded string.

See also

> **f_sign_data()** (p. **??**)

Definition at line **1327** of file **f_nano_crypto_util.h**.

5.7.2.31 F_SIGNATURE_OUTPUT_XRB_PK

```
#define F_SIGNATURE_OUTPUT_XRB_PK (uint32_t)16
```

Public key is a XRB wallet encoded base32 string.

See also

> **f_sign_data()** (p. **??**)

Definition at line **1334** of file **f_nano_crypto_util.h**.

5.7.2.32 F_SIGNATURE_RAW

```
#define F_SIGNATURE_RAW (uint32_t)1
```

Signature is raw data.

See also

> **f_sign_data()** (p. **??**)

Definition at line **1306** of file **f_nano_crypto_util.h**.

5.7.2.33 F_SIGNATURE_STRING

```
#define F_SIGNATURE_STRING (uint32_t)2
```

Signature is hex ASCII encoded string.

See also

> **f_sign_data()** (p. **??**)

Definition at line **1313** of file **f_nano_crypto_util.h**.

5.7.2.34 F_VALUE_SEND_RECEIVE_RAW_128

```
#define F_VALUE_SEND_RECEIVE_RAW_128 F_NANO_B_RAW_128
```

Value to send or receive is RAW 128 bit.

Definition at line **1470** of file **f_nano_crypto_util.h**.

5.7.2.35 F_VALUE_SEND_RECEIVE_RAW_STRING

```
#define F_VALUE_SEND_RECEIVE_RAW_STRING F_NANO_B_RAW_STRING
```

Value to send or receive is raw string.

Definition at line **1482** of file **f_nano_crypto_util.h**.

5.7.2.36 F_VALUE_SEND_RECEIVE_REAL_STRING

```
#define F_VALUE_SEND_RECEIVE_REAL_STRING F_NANO_B_REAL_STRING
```

Value to send or receive is real string.

Definition at line **1476** of file **f_nano_crypto_util.h**.

5.7.2.37 F_VALUE_TO_RECEIVE

```
#define F_VALUE_TO_RECEIVE (int)(1<<1)
```

Value to receive.

Definition at line **1494** of file **f_nano_crypto_util.h**.

5.7.2.38 F_VALUE_TO_SEND

```
#define F_VALUE_TO_SEND (int)(1<<0)
```

Value to send.

Definition at line **1488** of file **f_nano_crypto_util.h**.

5.7.2.39 F_VERIFY_SIG_NANO_WALLET

```
#define F_VERIFY_SIG_NANO_WALLET (uint32_t)1
```

Public key is a NANO wallet with *XRB* or *NANO* prefixes encoded base32 string.

See also

> **f_verify_signed_data()** (p. **??**)

Definition at line **1399** of file **f_nano_crypto_util.h**.

5.7.2.40 MAX_STR_NANO_CHAR

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

Defines a max size of Nano char (70 bytes)

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

5.7.2.41 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 **194** of file **f_nano_crypto_util.h**.

5.7.2.42 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 **212** of file **f_nano_crypto_util.h**.

5.7.2.43 NANO_PASSWD_MAX_LEN

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

Password max length.

Definition at line **200** of file **f_nano_crypto_util.h**.

5.7.2.44 NANO_PREFIX

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

Nano prefix.

Definition at line **164** of file **f_nano_crypto_util.h**.

5.7.2.45 PUB_KEY_EXTENDED_MAX_LEN

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

Max size of public key (extended)

Definition at line **158** of file **f_nano_crypto_util.h**.

5.7.2.46 REP_XRB

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

Representative XRB flag.

Destination XRB flag.

Sender XRB flag.

5.7.2.47 SENDER_XRB

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

Definition at line **434** of file **f_nano_crypto_util.h**.

5.7.2.48 STR_NANO_SZ

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

String size of Nano encoded Base32 including NULL char.

Definition at line **206** of file **f_nano_crypto_util.h**.

5.7.2.49 XRB_PREFIX

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

XRB (old Raiblocks) prefix.

Definition at line **170** of file **f_nano_crypto_util.h**.

5.7.3 Typedef Documentation

5.7.3.1 F_FILE_INFO_ERR

`F_FILE_INFO_ERR`

Typedef Error enumerator for info file functions.

5.7.3.2 F_NANO_CREATE_BLOCK_DYN_ERR

`typedef enum` **`f_nano_create_block_dyn_err_t`** **`F_NANO_CREATE_BLOCK_DYN_ERR`**

5.7.3.3 f_nano_err

`f_nano_err`

Error function enumerator.

See also

> **f_nano_err_t** (p. **??**)

5.7.3.4 F_NANO_P2POW_BLOCK_DYN_ERR

`typedef enum` **`f_nano_p2pow_block_dyn_err_t`** **`F_NANO_P2POW_BLOCK_DYN_ERR`**

5.7.3.5 F_TOKEN

`typedef uint8_t F_TOKEN[16]`

Definition at line **218** of file **f_nano_crypto_util.h**.

5.7.3.6 f_uint128_t

`f_uint128_t`

128 bit big number of Nano balance

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

5.7.3.7 f_write_seed_err

typedef enum **f_write_seed_err_t** **f_write_seed_err**

5.7.3.8 NANO_PRIVATE_KEY

NANO_PRIVATE_KEY

Size of Nano Private Key.

Definition at line **240** of file **f_nano_crypto_util.h**.

5.7.3.9 NANO_PRIVATE_KEY_EXTENDED

NANO_PRIVATE_KEY_EXTENDED

Size of Nano Private Key extended.

Definition at line **246** of file **f_nano_crypto_util.h**.

5.7.3.10 NANO_PUBLIC_KEY

NANO_PUBLIC_KEY

Size of Nano Public Key.

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

5.7.3.11 NANO_PUBLIC_KEY_EXTENDED

NANO_PUBLIC_KEY_EXTENDED

Size of Public Key Extended.

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

5.7.3.12 NANO_SEED

NANO_SEED

Size of Nano SEED.

Definition at line **224** of file **f_nano_crypto_util.h**.

5.7.4 Enumeration Type Documentation

5.7.4.1 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 **546** of file **f_nano_crypto_util.h**.

5.7.4.2 f_nano_create_block_dyn_err_t

enum **f_nano_create_block_dyn_err_t**

Enumerator

NANO_CREATE_BLK_DYN_OK	
NANO_CREATE_BLK_DYN_BLOCK_NULL	
NANO_CREATE_BLK_DYN_ACCOUNT_NULL	
NANO_CREATE_BLK_DYN_COMPARE_BALANCE	
NANO_CREATE_BLK_DYN_GENESIS_WITH_NON_EMPTY_BALANCE	
NANO_CREATE_BLK_DYN_CANT_SEND_IN_GENESIS_BLOCK	
NANO_CREATE_BLK_DYN_REP_NULL	
NANO_CREATE_BLK_DYN_BALANCE_NULL	
NANO_CREATE_BLK_DYN_SEND_RECEIVE_NULL	
NANO_CREATE_BLK_DYN_LINK_NULL	
NANO_CREATE_BLK_DYN_BUF_MALLOC	
NANO_CREATE_BLK_DYN_MALLOC	
NANO_CREATE_BLK_DYN_WRONG_PREVIOUS_SZ	
NANO_CREATE_BLK_DYN_WRONG_PREVIOUS_STR_SZ	
NANO_CREATE_BLK_DYN_PARSE_STR_HEX_ERR	
NANO_CREATE_BLK_DYN_FORBIDDEN_AMOUNT_TYPE	
NANO_CREATE_BLK_DYN_COMPARE	
NANO_CREATE_BLK_DYN_EMPTY_VAL_TO_SEND_OR_REC	
NANO_CREATE_BLK_DYN_INVALID_DIRECTION_OPTION	

Definition at line **606** of file **f_nano_crypto_util.h**.

5.7.4.3   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 **305** of file **f_nano_crypto_util.h**.

5.7.4.4   f_nano_p2pow_block_dyn_err_t

enum   **f_nano_p2pow_block_dyn_err_t**

Enumerator

NANO_P2POW_CREATE_BLOCK_OK	
NANO_P2POW_CREATE_BLOCK_INVALID_USER_BLOCK	
NANO_P2POW_CREATE_BLOCK_MALLOC	
NANO_P2POW_CREATE_BLOCK_NULL	
NANO_P2POW_CREATE_OUTPUT	
NANO_P2POW_CREATE_OUTPUT_MALLOC	

Definition at line **629** of file **f_nano_crypto_util.h**.

5.7.4.5 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.
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 **442** of file **f_nano_crypto_util.h**.

5.7.5 Function Documentation

5.7.5.1 __attribute__()

struct **f_block_transfer_t** __attribute__ (
          (packed))

5.7.5.2 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. **??**)

5.7.5.3 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

5.7.5.4 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*
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* 3.34×10^{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. **??**)

5.7.5.5   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
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

5.7.5.6   f_generate_token()

```
int f_generate_token (
            F_TOKEN signature,
            void * data,
            size_t data_sz,
            const char * password )
```

Generates a non deterministic token given a message data and a password.

Parameters

out	*signature*	128 bit non deterministic token
in	*data*	Data to be signed in token
in	*data_sz*	Size of data
in	*password*	Password

Return values

0	On Success, otherwise Error

See also

> **f_verify_token()** (p. **??**)

5.7.5.7   f_get_dictionary_path()

```
char * f_get_dictionary_path (
            void  )
```

Get default dictionary path in **myNanoEmbedded** library.

Return values

Path	and name of the dictionary file

See also

> **f_set_dictionary_path()** (p. **??**)

5.7.5.8   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

5.7.5.9 f_is_valid_nano_seed_encrypted()

```
int f_is_valid_nano_seed_encrypted (
            void * stream,
            size_t stream_len,
            int read_from )
```

Verifies if ecrypted Nano SEED is valid.

Parameters

in	*stream*	Encrypted binary data block coming from memory or file
in	*stream_len*	size of *stream* data
in	*read_from*	Source *READ_SEED_FROM_STREAM* if encrypted binary data is in memory or *READ_SEED_FROM_FILE* is in a file.

Return values

0	If invalid, greater than zero if is valid or error if less than zero.

5.7.5.10 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

Parameters

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

5.7.5.11   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. **??**)

5.7.5.12 f_nano_block_to_json()

```
int f_nano_block_to_json (
            char * dest,
            size_t * olen,
            size_t dest_size,
            F_BLOCK_TRANSFER * user_block )
```

Parse a Nano Block to JSON.

Parameters

out	*dest*	Destination of the converted JSON block
out	*olen*	Output length of the converted JSON block. *olen* can be NULL. If NULL, destination size contains a NULL char
in	*dest_size*	Size of *destmemory buffer*
in	*user_block*	*User Nano block*

Returns

> *0 if success, non zero if error*

5.7.5.13 f_nano_get_block_hash()

```
int f_nano_get_block_hash (
            uint8_t * hash,
            F_BLOCK_TRANSFER * block )
```

Gets a hash from Nano block.

Parameters

out	*hash*	Output hash
in	*block*	Nano Block

Returns

0 if success, non zero if error

5.7.5.14 f_nano_get_p2pow_block_hash()

```
int f_nano_get_p2pow_block_hash (
            uint8_t * user_hash,
            uint8_t * fee_hash,
            F_BLOCK_TRANSFER * block )
```

Get Nano user block hash and Nano fee block hashes from P2PoW block.

Parameters

out	*user_hash*	Hash of the user block
out	*fee_hash*	Hash of the P2PoW block
in	*block*	Input Nano Block

Returns

0 if success, non zero if error

5.7.5.15 f_nano_is_valid_block()

```
int f_nano_is_valid_block (
            F_BLOCK_TRANSFER * block )
```

Checks if Binary Nano Block is valid.

Parameters

in	*block*	Nano Block

Returns

0 if is invalid block or 1 if is valid block

5.7.5.16 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

5.7.5.17 f_nano_p2pow_to_JSON()

```
int f_nano_p2pow_to_JSON (
            char * buffer,
            size_t * olen,
            size_t buffer_sz,
            F_BLOCK_TRANSFER * block )
```

Parse binary P2PoW block to JSON.

Parameters

out	*buffer*	Output JSON string
out	*olen*	Output JSON string size. *olen* can be NULL. If NULL, *buffer* will be terminated with a NULL char
in	*buffer_sz*	Size of memory buffer
in	*block*	P2PoW block

Returns

> 0 if success, non zero if error

5.7.5.18 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

5.7.5.19 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

5.7.5.20 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. **??**)

Mandatory: You need to enable attach a random function to your project using **f_random_attach()** (p. **??**)

Return values

0	If success, otherwise error.

See also

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

5.7.5.21 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. **??**)

5.7.5.22 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. **??**)

5.7.5.23   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. **??**)

5.7.5.24 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

5.7.5.25 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. **??**)

5.7.5.26   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 equal *valB* • *F_NANO_COMPARE_LT* if *valA* is lesser than *valB* • *F_NANO_COMPARE_GT* if *valA* is greater 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

5.7.5.27 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

5.7.5.28 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. **??**)

5.7.5.29 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. **??**)

5.7.5.30   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

5.7.5.31 f_set_dictionary_path()

```
void f_set_dictionary_path (
            const char * path )
```

Set default dictionary file and path to **myNanoEmbedded** library.

Parameters

in	path	Path to dictionary file

If **f_set_dictionary_path()** (p. **??**) is not used in **myNanoEmbedded** library then default path stored in *BIP39_D←↩ICTIONARY* is used

See also

> **f_get_dictionary_path()** (p. **??**)

5.7.5.32 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

5.7.5.33 f_sign_data()

```
int f_sign_data (
            unsigned char * signature,
            void * out_public_key,
            uint32_t ouput_type,
            const unsigned char * message,
            size_t msg_len,
            const unsigned char * private_key )
```

Signs a *message* with a deterministic signature given a *private key*

Parameters

out	*signature*	Output signature
out	*out_public_key*	Output public key. It can be NULL
in	*output_type*	Output type of public key. Public key types are: • *F_SIGNATURE_RAW* Signature is raw 64 bytes long • *F_SIGNATURE_STRING* Singnature is hex ASCII encoded string • *F_SIGNATURE_OUTPUT_RAW_PK* Public key is raw 32 bytes data • *F_SIGNATURE_OUTPUT_STRING_PK* Public key is hes ASCII encoded string • *F_SIGNATURE_OUTPUT_XRB_PK* Public key is a XRB wallet encoded base32 string • *F_SIGNATURE_OUTPUT_NANO_PK* Public key is a NANO wallet encoded base32 string
in	*message*	Message to be signed with Elliptic Curve Ed25519 with blake2b hash
in	*msg_len*	Size of message to be signed
in	*private_key*	Private key to sign message

Return values

0	If success, otherwise error.

See also

> **f_verify_signed_data()** (p. **??**)

5.7.5.34 f_verify_signed_block()

```
int f_verify_signed_block (
          F_BLOCK_TRANSFER *  )
```

5.7.5.35 f_verify_signed_data()

```
int f_verify_signed_data (
          const unsigned char * signature,
          const unsigned char * message,
          size_t message_len,
          const void * public_key,
          uint32_t pk_type )
```

Verifies if a signed message is valid.

Parameters

in	*signature*	Signature of the *message*
in	*message*	Message to be verified
in	*message_len*	Length of the message
in	*public_key*	Public key to verify signed message
in	*pk_type*	Type of the public key. Types are: • *F_VERIFY_SIG_NANO_WALLET* Public key is a NANO wallet with *XRB* or *NANO* prefixes encoded base32 string • *F_VERIFY_SIG_RAW_HEX* Public key is raw 32 bytes data • *F_PUBLIC_KEY_ASCII_HEX* Public key is a hex ASCII encoded string

Return value are

• Greater than zero if *signature* is VALID

• 0 (zero) if *signature* is INVALID

• Negative if ERROR occurred

See also

>   **f_sign_data()** (p. **??**)

5.7.5.36 f_verify_token()

```
int f_verify_token (
            F_TOKEN signature,
            void * data,
            size_t data_sz,
            const char * password )
```

Verifies if a token is valid given data and password.

Parameters

in	*signature*	128 bit non deterministic token
in	*data*	Data to be signed in token
in	*data_sz*	Size of data
in	*password*	Password

Return values

0	On if invalid; 1 if valid ; less than zero if an error occurs

See also

> **f_generate_token()** (p. **??**)

5.7.5.37 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. **??**)

5.7.5.38 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. **??**)

5.7.5.39 from_multiplier()

```
uint64_t from_multiplier (
        double multiplier,
        uint64_t base_difficulty )
```

Calculates a PoW given a multiplier and base difficulty.

Parameters

in	*multiplier*	Multiplier of the work
in	*base_difficulty*	Base difficulty Details here

See also

> **to_multiplier()** (p. **??**)

Return values

Calculated	value

5.7.5.40 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

5.7.5.41 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

5.7.5.42 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. **??**)

5.7.5.43 nano_create_block_dynamic()

```
int nano_create_block_dynamic (
            F_BLOCK_TRANSFER ** block,
            const void * account,
            size_t account_len,
            const void * previous,
            size_t previous_len,
            const void * representative,
            size_t representative_len,
            const void * balance,
            const void * value_to_send_or_receive,
            uint32_t balance_and_val_to_send_or_rec_types,
            const void * link,
            size_t link_len,
            int direction )
```

Creates a Nano block dynamically in memory.

Parameters

out	*block*	Pointer to new allocated Nano block
in	*account*	*nano* or *xrb* or *raw hex string* or raw *hex binary* **account** (public key)
in	*account_len*	Account length. If zero it is assumed as *nano*, *xrb* or *raw hex string* public key

Parameters

in	*previous*	*Hex string* or *raw hex binary* **previous** block
in	*previous_len*	Previous length size. If zero it is assumed **previous** is a *NULL* terminated string
in	*representative*	*nano* or *xrb* or *raw hex string* or raw *hex binary* **representative** account
in	*representative_len*	Representative length. If zero it is assumed as *nano*, *xrb* or *raw hex string* representative
in	*balance*	*Real balance* or *raw string balance* or *raw binary balance*
in	*value_to_send_or_receive*	*Real value to send or receive* or *raw string value to send or receive* or *raw value to send or receive*
in	*balance_and_val_to_send_or_rec_types*	Balance and value to send/receive types: • *F_BALANCE_RAW_128* Balance is *raw binary 128 bit* • *F_BALANCE_REAL_STRING* Balance is *real string* • *F_BALANCE_RAW_STRING* Balance is *raw string* • *F_VALUE_SEND_RECEIVE_RAW_128* Value to send/receive is *raw binary 128 bit* • *F_VALUE_SEND_RECEIVE_REAL_STRING* Value to send/receive is *real string* • *F_VALUE_SEND_RECEIVE_RAW_STRING* Value to send/receive is *raw string*
in	*link*	*nano* or *xrb* or *raw hex string* or raw *hex binary* **link**
in	*link_len*	Link length. If zero it is assumed as *nano*, *xrb* or *raw hex string* link
in	*direction*	Direction of the Nano block: • *F_VALUE_TO_SEND* Value to send to **link** = *destination account* • *F_VALUE_TO_RECEIVE* Value to receive from **link** = *receive amount*

WARNING: *block* must be free after used.

Return values

ERROR_SUCCESS	when success or non zero otherwise

See also

> **nano_create_p2pow_block_dynamic()** (p. **??**)

5.7.5.44 nano_create_p2pow_block_dynamic()

```
int nano_create_p2pow_block_dynamic (
            F_BLOCK_TRANSFER ** ,
            F_BLOCK_TRANSFER * ,
            const void * ,
            size_t ,
            const void * ,
            uint32_t ,
            const void * ,
            size_t  )
```

5.7.5.45 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. **??**)

5.7.5.46 to_multiplier()

```
double to_multiplier (
            uint64_t difficulty,
            uint64_t base_difficulty )
```

Calculates a relative difficulty compared PoW with another.

Parameters

in	*dificulty*	Work difficulty
in	*base_difficulty*	Base difficulty Details `here`

See also

> **from_multiplier()** (p. **??**)

Return values

Calculated	value

5.7.5.47 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

5.7.5.48 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

5.7.6 Variable Documentation

5.7.6.1 account

`uint8_t account[32]`

Account in raw binary data.

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

5.7.6.2 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**.

5.7.6.3 body

`F_NANO_WALLET_INFO_BODY body`

Body of the file info.

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

5.7.6.4 desc

`char desc[F_NANO_DESC_SZ]`

Description.

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

5.7.6.5 description

```
uint8_t description[F_DESC_SZ]
```

File description.

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

5.7.6.6 file_info_integrity

```
uint8_t file_info_integrity[32]
```

File info integrity of the body block.

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

5.7.6.7 hash_sk_unencrypted

```
uint8_t hash_sk_unencrypted[32]
```

hash of Nano SEED when unencrypted

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

5.7.6.8 header

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

Header magic.

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

5.7.6.9 iv

```
uint8_t iv
```

Initial sub vector.

Initial vector of first encryption layer.

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

5.7.6.10 last_used_wallet_number

```
uint32_t last_used_wallet_number
```

Last used wallet number.

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

5.7.6.11 link

```
uint8_t link[32]
```

link or destination account

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

5.7.6.12 max_fee

```
char max_fee[F_RAW_STR_MAX_SZ]
```

Custom preferred max fee of Proof of Work.

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

5.7.6.13 nano_hdr

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

Header of the file.

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

5.7.6.14 nanoseed_hash

```
uint8_t nanoseed_hash[32]
```

Nano SEED hash file.

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

5.7.6.15 preamble

```
uint8_t preamble[32]
```

Block preamble.

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

5.7.6.16 prefixes

```
uint8_t prefixes
```

Internal use for this API.

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

5.7.6.17 previous

```
uint8_t previous[32]
```

Previous block.

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

5.7.6.18 representative

```
uint8_t representative[32]
```

Representative for current account.

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

5.7.6.19 reserved

```
uint8_t reserved
```

Reserved (not used)

Reserved.

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

5.7.6.20 salt

```
uint8_t salt[32]
```

Salt of the first encryption layer.

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

5.7.6.21 seed_block

```
F_ENCRYPTED_BLOCK seed_block
```

Second encrypted block for Nano SEED.

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

5.7.6.22 signature

```
uint8_t signature[64]
```

Signature of the block.

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

5.7.6.23 sk_encrypted

```
uint8_t sk_encrypted[32]
```

Secret.

SEED encrypted (second layer)

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

5.7.6.24 sub_salt

```
uint8_t sub_salt[32]
```

Salt of the sub block to be stored.

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

5.7.6.25 ver

`uint32_t ver`

Version of the file.

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

5.7.6.26 version

`uint16_t version`

Version.

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

5.7.6.27 wallet_prefix

`uint8_t wallet_prefix`

Wallet prefix: 0 for NANO; 1 for XRB.

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

5.7.6.28 wallet_representative

`char wallet_representative[ `**MAX_STR_NANO_CHAR**`]`

Wallet representative.

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

5.7.6.29 work

`uint64_t work`

Internal use for this API.

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

5.8 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 <errors.h>
00009 #include <stdint.h>
00010 #include <f_util.h>
00011 #include <f_bitcoin.h>
00012
00013 #ifndef F_DOC_SKIP
00014
00015  #ifdef F_XTENSA
00016
00017   #ifndef F_ESP32
00018    #define F_ESP32
00019   #endif
00020
00021   #include "esp_system.h"
00022
00023  #endif
00024
00025  #include "sodium/crypto_generichash.h"
00026  #include "sodium/crypto_sign.h"
00027  #include "sodium.h"
00028
00029  #ifdef F_ESP32
00030
00031   #include "sodium/private/curve25519_ref10.h"
00032
00033  #else
00034
00035   #include "sodium/private/ed25519_ref10.h"
00036
00037   #define ge_p3 ge25519_p3
00038   #define sc_reduce sc25519_reduce
00039   #define sc_muladd sc25519_muladd
00040   #define ge_scalarmult_base ge25519_scalarmult_base
00041   #define ge_p3_tobytes ge25519_p3_tobytes
00042
00043  #endif
00044
00045 #endif
00046
00047 #include <version.h>
00048
00131 #ifdef __cplusplus
00132 extern "C" {
00133 #endif
00134
00135
00140 #define F_NANO_POW_MAX_THREAD (size_t)10
00141
00142 #ifndef F_DOC_SKIP
00143  #ifdef F_ESP32
00144   #undef F_NANO_POW_MAX_THREAD
00145  #endif
00146 #endif
00147
00152 #define MAX_STR_NANO_CHAR (size_t)70 //5+56+8+1
00153
00158 #define PUB_KEY_EXTENDED_MAX_LEN (size_t)40
00159
00164 #define NANO_PREFIX "nano_"
00165
00170 #define XRB_PREFIX "xrb_"
00171
00172 #ifdef F_ESP32
00173
00178 #define BIP39_DICTIONARY "/spiffs/dictionary.dic"
00179 #else
00180
00181  #ifndef F_DOC_SKIP
00182   #define BIP39_DICTIONARY_SAMPLE "../../dictionary.dic"
00183   #define BIP39_DICTIONARY "dictionary.dic"
00184  #endif
00185
00186 #endif
00187
00194 #define NANO_ENCRYPTED_SEED_FILE "/spiffs/secure/nano.nse"
00195
00200 #define NANO_PASSWD_MAX_LEN (size_t)80
```

```
00201
00206 #define STR_NANO_SZ (size_t)66// 65+1 Null included
00207
00212 #define NANO_FILE_WALLETS_INFO "/spiffs/secure/walletsinfo.i"
00213
00218 typedef uint8_t F_TOKEN[16];
00219
00224 typedef uint8_t NANO_SEED[crypto_sign_SEEDBYTES];
00225
00230 typedef uint8_t f_uint128_t[16];
00231
00232 #ifndef F_DOC_SKIP
00233  #define EXPORT_KEY_TO_CHAR_SZ (size_t)sizeof(NANO_SEED)+1
00234 #endif
00235
00240 typedef uint8_t NANO_PRIVATE_KEY[sizeof(NANO_SEED)];
00241
00246 typedef uint8_t NANO_PRIVATE_KEY_EXTENDED[crypto_sign_ed25519_SECRETKEYBYTES];
00247
00252 typedef uint8_t NANO_PUBLIC_KEY[crypto_sign_ed25519_PUBLICKEYBYTES];
00253
00258 typedef uint8_t NANO_PUBLIC_KEY_EXTENDED[PUB_KEY_EXTENDED_MAX_LEN];
00259
00268 typedef struct f_block_transfer_t {
00270     uint8_t preamble[32];
00272     uint8_t account[32];
00274     uint8_t previous[32];
00276     uint8_t representative[32];
00280     f_uint128_t balance;
00282     uint8_t link[32];
00284     uint8_t signature[64];
00286     uint8_t prefixes;
00288     uint64_t work;
00289 } __attribute__((packed)) F_BLOCK_TRANSFER;
00290
00291 #define F_BLOCK_TRANSFER_SIZE (size_t)sizeof(F_BLOCK_TRANSFER)
00292 #define F_P2POW_BLOCK_TRANSFER_SIZE 2*F_BLOCK_TRANSFER_SIZE
00293
00294 #ifndef F_DOC_SKIP
00295  #define F_BLOCK_TRANSFER_SIGNABLE_SZ
     (size_t)(sizeof(F_BLOCK_TRANSFER)-64-sizeof(uint64_t)-sizeof(uint8_t))
00296 #endif
00297
00305 typedef enum f_nano_err_t {
00307     NANO_ERR_OK=0,
00309     NANO_ERR_CANT_PARSE_BN_STR=5151,
00311     NANO_ERR_MALLOC,
00313     NANO_ERR_CANT_PARSE_FACTOR,
00315     NANO_ERR_MPI_MULT,
00317     NANO_ERR_CANT_PARSE_TO_BLK_TRANSFER,
00319     NANO_ERR_EMPTY_STR,
00321     NANO_ERR_CANT_PARSE_VALUE,
00323     NANO_ERR_PARSE_MPI_TO_STR,
00325     NANO_ERR_CANT_COMPLETE_NULL_CHAR,
00327     NANO_ERR_CANT_PARSE_TO_MPI,
00329     NANO_ERR_INSUFICIENT_FUNDS,
00331     NANO_ERR_SUB_MPI,
00333     NANO_ERR_ADD_MPI,
00335     NANO_ERR_NO_SENSE_VALUE_TO_SEND_NEGATIVE,
00337     NANO_ERR_NO_SENSE_VALUE_TO_SEND_ZERO,
00339     NANO_ERR_NO_SENSE_BALANCE_NEGATIVE,
00341     NANO_ERR_VAL_A_INVALID_MODE,
00343     NANO_ERR_CANT_PARSE_TO_TEMP_UINT128_T,
00345     NANO_ERR_VAL_B_INVALID_MODE,
00347     NANO_ERR_CANT_PARSE_RAW_A_TO_MPI,
00349     NANO_ERR_CANT_PARSE_RAW_B_TO_MPI,
00351     NANO_ERR_UNKNOWN_ADD_SUB_MODE,
00353     NANO_ERR_INVALID_RES_OUTPUT
00354 } f_nano_err;
00355
00356 #ifndef F_DOC_SKIP
00357
00358  #define READ_SEED_FROM_STREAM (int)1
00359  #define READ_SEED_FROM_FILE (int)2
00360  #define WRITE_SEED_TO_STREAM (int)4
00361  #define WRITE_SEED_TO_FILE (int)8
00362  #define PARSE_JSON_READ_SEED_GENERIC (int)16
00363  #define F_STREAM_DATA_FILE_VERSION (uint32_t)((1<<16)|0)
00364
00365 #endif
00366
00374 typedef struct f_nano_encrypted_wallet_t {
00376     uint8_t sub_salt[32];
00378     uint8_t iv[16];
00380     uint8_t reserved[16];
00382     uint8_t hash_sk_unencrypted[32];
00384     uint8_t sk_encrypted[32];
```

```
00385 } __attribute__ ((packed)) F_ENCRYPTED_BLOCK;
00386
00387 #ifndef F_DOC_SKIP
00388
00389  static const uint8_t NANO_WALLET_MAGIC[] = {'_', 'n', 'a', 'n', 'o', 'w', 'a', 'l', 'l', 'e', 't', 'f',
      'i', 'l', 'e', '_'};
00390  #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 !!!"
00391  #define F_DESC_SZ (size_t) (160-sizeof(uint32_t))
00392
00393 #endif
00394
00402 typedef struct f_nano_crypto_wallet_t {
00404     uint8_t nano_hdr[sizeof(NANO_WALLET_MAGIC)];
00406     uint32_t ver;
00408     uint8_t description[F_DESC_SZ];
00410     uint8_t salt[32];
00412     uint8_t iv[16];
00414     F_ENCRYPTED_BLOCK seed_block;
00415 } __attribute__ ((packed)) F_NANO_CRYPTOWALLET;
00416
00417 #ifndef F_DOC_SKIP
00418
00419 _Static_assert((sizeof(F_NANO_CRYPTOWALLET)&0x1F)==0, "Error 1");
00420 _Static_assert((sizeof(F_ENCRYPTED_BLOCK)&0x1F)==0, "Error 2");
00421
00422 #endif
00423
00428 #define REP_XRB (uint8_t)0x4
00429
00434 #define SENDER_XRB (uint8_t)0x02
00435
00440 #define DEST_XRB (uint8_t)0x01
00441
00442 typedef enum f_write_seed_err_t {
00444     WRITE_ERR_OK=0,
00446     WRITE_ERR_NULL_PASSWORD=7180,
00448     WRITE_ERR_EMPTY_STRING,
00450     WRITE_ERR_MALLOC,
00452     WRITE_ERR_ENCRYPT_PRIV_KEY,
00454     WRITE_ERR_GEN_SUB_PRIV_KEY,
00456     WRITE_ERR_GEN_MAIN_PRIV_KEY,
00458     WRITE_ERR_ENCRYPT_SUB_BLOCK,
00460     WRITE_ERR_UNKNOWN_OPTION,
00462     WRITE_ERR_FILE_ALREDY_EXISTS,
00464     WRITE_ERR_CREATING_FILE,
00466     WRITE_ERR_WRITING_FILE
00467 } f_write_seed_err;
00468
00469 #ifndef F_DOC_SKIP
00470
00471  #define F_RAW_TO_STR_UINT128 (int)1
00472  #define F_RAW_TO_STR_STRING (int)2
00473  #define F_RAW_STR_MAX_SZ (size_t)41 // 39 + '\0' + '.' -> 39 = log10(2^128)
00474  #define F_MAX_STR_RAW_BALANCE_MAX (size_t)40 //39+'\0'
00475  #define F_NANO_EMPTY_BALANCE "0.0"
00476
00477 #endif
00478
00486 typedef struct f_nano_wallet_info_bdy_t {
00488     uint8_t wallet_prefix; // 0 for NANO; 1 for XRB
00490     uint32_t last_used_wallet_number;
00492     char wallet_representative[MAX_STR_NANO_CHAR];
00494     char max_fee[F_RAW_STR_MAX_SZ];
00496     uint8_t reserved[44];
00497 } __attribute__((packed)) F_NANO_WALLET_INFO_BODY;
00498
00499 #ifndef F_DOC_SKIP
00500
00501  _Static_assert((sizeof(F_NANO_WALLET_INFO_BODY)&0x1F)==0, "Error F_NANO_WALLET_INFO_BODY is not byte
      aligned");
00502
00503  #define F_NANO_WALLET_INFO_DESC "Nano file descriptor used for fast custom access. BUY BITCOIN AND NANO."
00504  #define F_NANO_WALLET_INFO_VERSION (uint16_t)((1<<8)|1)
00505  static const uint8_t F_NANO_WALLET_INFO_MAGIC[] = {'_', 'n', 'a', 'n', 'o', 'w', 'a', 'l', 'l', 'e', 't',
      '_', 'n', 'f', 'o', '_'};
00506
00507  #define F_NANO_DESC_SZ (size_t)78
00508
00509 #endif
00510
00518 typedef struct f_nano_wallet_info_t {
00520     uint8_t header[sizeof(F_NANO_WALLET_INFO_MAGIC)];
00522     uint16_t version;
00524     char desc[F_NANO_DESC_SZ];
00526     uint8_t nanoseed_hash[32];
00528     uint8_t file_info_integrity[32];
```

```
00530     F_NANO_WALLET_INFO_BODY body;
00531 } __attribute__((packed)) F_NANO_WALLET_INFO;
00532
00533 #ifndef F_DOC_SKIP
00534
00535   _Static_assert((sizeof(F_NANO_WALLET_INFO)&0x1F)==0, "Error F_NANO_WALLET_INFO is not byte aligned");
00536
00537 #endif
00538
00546 typedef enum f_file_info_err_t {
00548     F_FILE_INFO_ERR_OK=0,
00550     F_FILE_INFO_ERR_CANT_OPEN_INFO_FILE=7001,
00552     F_FILE_INFO_ERR_NANO_SEED_ENCRYPTED_FILE_NOT_FOUND,
00554     F_FILE_INFO_ERR_CANT_DELETE_NANO_INFO_FILE,
00556     F_FILE_INFO_ERR_MALLOC,
00558     F_FILE_INFO_ERR_CANT_READ_NANO_SEED_ENCRYPTED_FILE,
00560     F_FILE_INFO_ERR_CANT_READ_INFO_FILE,
00562     F_FILE_INFO_INVALID_HEADER_FILE,
00564     F_FILE_INFO_ERR_INVALID_SHA256_INFO_FILE,
00566     F_FILE_INFO_ERR_NANO_SEED_HASH_FAIL,
00568     F_FILE_INFO_ERR_NANO_INVALID_REPRESENTATIVE,
00570     F_FILE_INFO_ERR_NANO_INVALID_MAX_FEE_VALUE,
00572     F_FILE_INFO_ERR_OPEN_FOR_WRITE_INFO,
00574     F_FILE_INFO_ERR_EXISTING_FILE,
00576     F_FILE_INFO_ERR_CANT_WRITE_FILE_INFO
00577 } F_FILE_INFO_ERR;
00578
00579 #ifndef F_DOC_SKIP
00580
00581   #define F_NANO_ADD_A_B (uint32_t)(1<<0)
00582   #define F_NANO_SUB_A_B (uint32_t)(1<<1)
00583   #define F_NANO_A_RAW_128 (uint32_t)(1<<2)
00584   #define F_NANO_A_RAW_STRING (uint32_t)(1<<3)
00585   #define F_NANO_A_REAL_STRING (uint32_t)(1<<4)
00586   #define F_NANO_B_RAW_128 (uint32_t)(1<<5)
00587   #define F_NANO_B_RAW_STRING (uint32_t)(1<<6)
00588   #define F_NANO_B_REAL_STRING (uint32_t)(1<<7)
00589   #define F_NANO_RES_RAW_128 (uint32_t)(1<<8)
00590   #define F_NANO_RES_RAW_STRING (uint32_t)(1<<9)
00591   #define F_NANO_RES_REAL_STRING (uint32_t)(1<<10)
00592   #define F_NANO_C_RAW_128 (uint32_t)(F_NANO_B_RAW_128<<16)
00593   #define F_NANO_C_RAW_STRING (uint32_t)(F_NANO_B_RAW_STRING<<16)
00594   #define F_NANO_C_REAL_STRING (uint32_t)(F_NANO_B_REAL_STRING<<16)
00595
00596   #define F_NANO_COMPARE_EQ (uint32_t)(1<<16) //Equal
00597   #define F_NANO_COMPARE_LT (uint32_t)(1<<17) // Lesser than
00598   #define F_NANO_COMPARE_LEQ (F_NANO_COMPARE_LT|F_NANO_COMPARE_EQ) // Less or equal
00599   #define F_NANO_COMPARE_GT (uint32_t)(1<<18) // Greater
00600   #define F_NANO_COMPARE_GEQ (F_NANO_COMPARE_GT|F_NANO_COMPARE_EQ) // Greater or equal
00601   #define DEFAULT_MAX_FEE "0.001"
00602
00603 #endif
00604
00605 #ifndef F_ESP32
00606 typedef enum f_nano_create_block_dyn_err_t {
00607     NANO_CREATE_BLK_DYN_OK = 0,
00608     NANO_CREATE_BLK_DYN_BLOCK_NULL = 8000,
00609     NANO_CREATE_BLK_DYN_ACCOUNT_NULL,
00610 //   NANO_CREATE_BLK_DYN_PREV_NULL,
00611     NANO_CREATE_BLK_DYN_COMPARE_BALANCE,
00612     NANO_CREATE_BLK_DYN_GENESIS_WITH_NON_EMPTY_BALANCE,
00613     NANO_CREATE_BLK_DYN_CANT_SEND_IN_GENESIS_BLOCK,
00614     NANO_CREATE_BLK_DYN_REP_NULL,
00615     NANO_CREATE_BLK_DYN_BALANCE_NULL,
00616     NANO_CREATE_BLK_DYN_SEND_RECEIVE_NULL,
00617     NANO_CREATE_BLK_DYN_LINK_NULL,
00618     NANO_CREATE_BLK_DYN_BUF_MALLOC,
00619     NANO_CREATE_BLK_DYN_MALLOC,
00620     NANO_CREATE_BLK_DYN_WRONG_PREVIOUS_SZ,
00621     NANO_CREATE_BLK_DYN_WRONG_PREVIOUS_STR_SZ,
00622     NANO_CREATE_BLK_DYN_PARSE_STR_HEX_ERR,
00623     NANO_CREATE_BLK_DYN_FORBIDDEN_AMOUNT_TYPE,
00624     NANO_CREATE_BLK_DYN_COMPARE,
00625     NANO_CREATE_BLK_DYN_EMPTY_VAL_TO_SEND_OR_REC,
00626     NANO_CREATE_BLK_DYN_INVALID_DIRECTION_OPTION
00627 } F_NANO_CREATE_BLOCK_DYN_ERR;
00628
00629 typedef enum f_nano_p2pow_block_dyn_err_t {
00630     NANO_P2POW_CREATE_BLOCK_OK = 0,
00631     NANO_P2POW_CREATE_BLOCK_INVALID_USER_BLOCK = 8400,
00632     NANO_P2POW_CREATE_BLOCK_MALLOC,
00633     NANO_P2POW_CREATE_BLOCK_NULL,
00634     NANO_P2POW_CREATE_OUTPUT,
00635     NANO_P2POW_CREATE_OUTPUT_MALLOC
00636 } F_NANO_P2POW_BLOCK_DYN_ERR;
00637
00638 #endif
```

```
00639
00651 double to_multiplier(uint64_t, uint64_t);
00652
00664 uint64_t from_multiplier(double, uint64_t);
00665
00675 void f_set_dictionary_path(const char *);
00676
00684 char *f_get_dictionary_path(void);
00685
00698 int f_generate_token(F_TOKEN, void *, size_t, const char *);
00699
00712 int f_verify_token(F_TOKEN, void *, size_t, const char *);
00713
00736 int f_cloud_crypto_wallet_nano_create_seed(size_t, char *, char *);
00737
00750 int f_generate_nano_seed(NANO_SEED, uint32_t);
00751
00766 int pk_to_wallet(char *, char *, NANO_PUBLIC_KEY_EXTENDED);
00767
00785 int f_seed_to_nano_wallet(NANO_PRIVATE_KEY, NANO_PUBLIC_KEY, NANO_SEED, uint32_t);
00786
00796 int f_nano_is_valid_block(F_BLOCK_TRANSFER *);
00797
00810 int f_nano_block_to_json(char *, size_t *, size_t, F_BLOCK_TRANSFER *);
00811
00822 int f_nano_get_block_hash(uint8_t *, F_BLOCK_TRANSFER *);
00823
00835 int f_nano_get_p2pow_block_hash(uint8_t *, uint8_t *, F_BLOCK_TRANSFER *);
00836
00849 int f_nano_p2pow_to_JSON(char *, size_t *, size_t, F_BLOCK_TRANSFER *);
00850
00860 char *f_nano_key_to_str(char *, unsigned char *);
00861
00880 int f_nano_seed_to_bip39(char *, size_t, size_t *, NANO_SEED, char *);
00881
00896 int f_bip39_to_nano_seed(uint8_t *, char *, char *);
00897
00919 int f_parse_nano_seed_and_bip39_to_JSON(char *, size_t, size_t *, void *, int, const char *);
00920
00938 int f_read_seed(uint8_t *, const char *, void *, int, int);
00939
00954 int f_nano_raw_to_string(char *, size_t *, size_t, void *, int);
00955
00964 int f_nano_valid_nano_str_value(const char *);
00965
00973 int valid_nano_wallet(const char *);
00974
00984 int nano_base_32_2_hex(uint8_t *, char *);
00985
01000 int f_nano_transaction_to_JSON(char *, size_t, size_t *, NANO_PRIVATE_KEY_EXTENDED, F_BLOCK_TRANSFER *);
01001
01009 int valid_raw_balance(const char *);
01010
01018 int is_null_hash(uint8_t *);
01019
01031 int is_nano_prefix(const char *, const char *);
01032
01041 F_FILE_INFO_ERR f_get_nano_file_info(F_NANO_WALLET_INFO *);
01042
01052 F_FILE_INFO_ERR f_set_nano_file_info(F_NANO_WALLET_INFO *, int);
01053
01075 f_nano_err f_nano_value_compare_value(void *, void *, uint32_t *);
01076
01097 f_nano_err f_nano_verify_nano_funds(void *, void *, void *, uint32_t);
01098
01108 f_nano_err f_nano_parse_raw_str_to_raw128_t(uint8_t *, const char *);
01109
01119 f_nano_err f_nano_parse_real_str_to_raw128_t(uint8_t *, const char *);
01120
01143 f_nano_err f_nano_add_sub(void *, void *, void *, uint32_t);
01144
01155 int f_nano_sign_block(F_BLOCK_TRANSFER *, F_BLOCK_TRANSFER *, NANO_PRIVATE_KEY_EXTENDED);
01156
01170 f_write_seed_err f_write_seed(void *, int, uint8_t *, char *);
01171
01184 f_nano_err f_nano_balance_to_str(char *, size_t, size_t *, f_uint128_t);
01185
01186
01191 #define F_BRAIN_WALLET_VERY_POOR (uint32_t)0
01192
01197 #define F_BRAIN_WALLET_POOR (uint32_t)1
01198
01203 #define F_BRAIN_WALLET_VERY_BAD (uint32_t)2
01204
01209 #define F_BRAIN_WALLET_BAD (uint32_t)3
01210
01215 #define F_BRAIN_WALLET_VERY_WEAK (uint32_t)4
```

```
01216
01221 #define F_BRAIN_WALLET_WEAK (uint32_t)5
01222
01227 #define F_BRAIN_WALLET_STILL_WEAK (uint32_t)6
01228
01233 #define F_BRAIN_WALLET_MAYBE_GOOD (uint32_t)7
01234
01235
01240 #define F_BRAIN_WALLET_GOOD (uint32_t)8
01241
01246 #define F_BRAIN_WALLET_VERY_GOOD (uint32_t)9
01247
01252 #define F_BRAIN_WALLET_NICE (uint32_t)10
01253
01258 #define F_BRAIN_WALLET_PERFECT (uint32_t)11
01259
01286 int f_extract_seed_from_brainwallet(uint8_t *, char **, uint32_t, const char *, const char *);
01287
01299 int f_verify_work(uint64_t *, const unsigned char *, uint64_t *, uint64_t);
01300
01306 #define F_SIGNATURE_RAW (uint32_t)1
01307
01313 #define F_SIGNATURE_STRING (uint32_t)2
01314
01320 #define F_SIGNATURE_OUTPUT_RAW_PK (uint32_t)4
01321
01327 #define F_SIGNATURE_OUTPUT_STRING_PK (uint32_t)8
01328
01334 #define F_SIGNATURE_OUTPUT_XRB_PK (uint32_t)16
01335
01341 #define F_SIGNATURE_OUTPUT_NANO_PK (uint32_t)32
01342
01348 #define F_IS_SIGNATURE_RAW_HEX_STRING (uint32_t)64
01349
01355 #define F_MESSAGE_IS_HASH_STRING (uint32_t)128
01356
01361 #define F_DEFAULT_THRESHOLD (uint64_t) 0xfffffffc000000000
01362
01386 int f_sign_data(
01387     unsigned char *signature,
01388     void *out_public_key,
01389     uint32_t ouput_type,
01390     const unsigned char *message,
01391     size_t msg_len,
01392     const unsigned char *private_key);
01393
01399 #define F_VERIFY_SIG_NANO_WALLET (uint32_t)1
01400
01406 #define F_PUBLIC_KEY_RAW_HEX (uint32_t)2
01407
01413 #define F_PUBLIC_KEY_ASCII_HEX (uint32_t)4
01414
01435 int f_verify_signed_data( const unsigned char *, const unsigned char *, size_t, const void *, uint32_t);
01436
01446 int f_is_valid_nano_seed_encrypted(void *, size_t, int);
01447
01452 #define F_BALANCE_RAW_128 F_NANO_A_RAW_128
01453
01458 #define F_BALANCE_REAL_STRING F_NANO_A_REAL_STRING
01459
01464 #define F_BALANCE_RAW_STRING F_NANO_A_RAW_STRING
01465
01470 #define F_VALUE_SEND_RECEIVE_RAW_128 F_NANO_B_RAW_128
01471
01476 #define F_VALUE_SEND_RECEIVE_REAL_STRING F_NANO_B_REAL_STRING
01477
01482 #define F_VALUE_SEND_RECEIVE_RAW_STRING F_NANO_B_RAW_STRING
01483
01488 #define F_VALUE_TO_SEND (int)(1<<0)
01489
01494 #define F_VALUE_TO_RECEIVE (int)(1<<1)
01495
01500 #define F_FEE_VALUE_RAW_128 F_NANO_B_RAW_128
01501
01506 #define F_FEE_VALUE_REAL_STRING F_NANO_B_REAL_STRING
01507
01512 #define F_FEE_VALUE_RAW_STRING F_NANO_B_RAW_STRING
01513
01560 int nano_create_block_dynamic(
01561     F_BLOCK_TRANSFER **,
01562     const void *,
01563     size_t,
01564     const void *,
01565     size_t,
01566     const void *,
01567     size_t,
01568     const void *,
```

```
01569      const void *,
01570      uint32_t,
01571      const void *,
01572      size_t,
01573      int
01574 );
01575
01576 int nano_create_p2pow_block_dynamic(
01577      F_BLOCK_TRANSFER **,
01578      F_BLOCK_TRANSFER *,
01579      const void *,
01580      size_t,
01581      const void *,
01582      uint32_t,
01583      const void *,
01584      size_t
01585 );
01586
01587 int f_verify_signed_block(F_BLOCK_TRANSFER *);
01588
01601 int f_nano_pow(uint64_t *, unsigned char *, const uint64_t, int);
01602
01603 #ifdef __cplusplus
01604 }
01605 #endif
01606
```

5.9 f_util.h File Reference

```
#include <stdint.h>
#include "mbedtls/sha256.h"
#include "mbedtls/aes.h"
#include "mbedtls/ecdsa.h"
#include "errors.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)
- #define **CLEAR_AND_FREE**(ptr, size) free(memset(ptr, 0, size));

Typedefs

- typedef void(∗ **rnd_fn**) (void ∗, size_t)
- typedef int(∗ **fn_det**) (void ∗, unsigned char ∗, size_t)
- typedef enum **err_ecdsa_secret_key_valid_e ERR_ECDSA_SECRET_KEY_VALID**
- typedef enum **err_ecdsa_public_key_valid_e ERR_ECDSA_PUBLIC_KEY_VALID**

Enumerations

- enum **err_ecdsa_secret_key_valid_e** {
  ERR_KEY_SK_SIZE_ZERO = 476, **ERR_KEY_WRONG_SIZE**, **ERR_SK_MALLOC**, **ERR_SK_READ**↩
  _BINARY,
  ERR_SK_CHECK }
- enum **err_ecdsa_public_key_valid_e** { **ERR_KEY_PK_SIZE_ZERO** = 501, **ERR_PK_MALLOC**, **ERR**↩
  _PK_CHECK }

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** ()
- int **f_convert_to_unsigned_int0x** (unsigned int ∗val, char ∗value, size_t value_sz)
- int **f_convert_to_unsigned_int0** (unsigned int ∗val, char ∗value, size_t value_sz)
- int **f_convert_to_unsigned_int_std** (unsigned int ∗val, char ∗value, size_t value_sz)
- int **f_convert_to_double** (double ∗, const char ∗)
- uint32_t **crc32_init** (unsigned char ∗, size_t, uint32_t)
- int **f_reverse** (unsigned char ∗, size_t)
- f_md_hmac_sha512 **f_hmac_sha512** (unsigned char ∗, const unsigned char ∗, size_t, const unsigned char
  ∗, size_t)
- **ERR_ECDSA_SECRET_KEY_VALID f_ecdsa_secret_key_valid** (mbedtls_ecp_group_id, unsigned char
  ∗, size_t)
- **ERR_ECDSA_PUBLIC_KEY_VALID f_ecdsa_public_key_valid** (mbedtls_ecp_group_id, unsigned char
  ∗, size_t)
- f_ecdsa_key_pair_err **f_gen_ecdsa_key_pair** (f_ecdsa_key_pair ∗, int, **fn_det**, void ∗)
- int **f_uncompress_elliptic_curve** (uint8_t ∗, size_t, size_t ∗, mbedtls_ecp_group_id, uint8_t ∗, size_t)
- uint8_t ∗ **f_ripemd160** (const uint8_t ∗, size_t)
- int **f_url_encode** (char ∗, size_t, size_t ∗, uint8_t ∗, size_t)
- int **f_encode_to_base64_dynamic** (char ∗∗, size_t ∗, void ∗, size_t)
- int **f_base64_decode_dynamic** (void ∗∗, size_t ∗, const char ∗, size_t)
- int **f_base64url_encode_dynamic** (void ∗∗, size_t ∗, void ∗, size_t)
- int **f_encode_to_base64** (char ∗, size_t, size_t ∗, void ∗, size_t)
- int **f_base64url_encode** (char ∗, size_t, size_t ∗, void ∗, size_t)
- int **f_base64url_decode** (void ∗, size_t, size_t ∗, const char ∗, size_t)
- int **f_url_base64_to_base64_dynamic** (char ∗∗, size_t ∗, const char ∗, size_t)
- int **f_url_decode** (void ∗, size_t, size_t ∗, const char ∗, size_t)

5.9.1 Detailed Description

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

Definition in file **f_util.h**.

5.9.2 Macro Definition Documentation

5.9.2.1 CLEAR_AND_FREE

```
#define CLEAR_AND_FREE(
            ptr,
            size ) free(memset(ptr, 0, size));
```

Definition at line **642** of file **f_util.h**.

5.9.2.2 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 **154** of file **f_util.h**.

5.9.2.3 ENTROPY_END

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

Ends a entropy function.

See also

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

Definition at line **161** of file **f_util.h**.

5.9.2.4 F_ENTROPY_TYPE_EXCELENT

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

Type of the excelent entropy used for verifier.

Slow

Definition at line **126** of file **f_util.h**.

5.9.2.5 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 **133** of file **f_util.h**.

5.9.2.6 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 **140** of file **f_util.h**.

5.9.2.7 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 **147** of file **f_util.h**.

5.9.2.8   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 **119** of file **f_util.h**.

5.9.2.9   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 **381** of file **f_util.h**.

5.9.2.10   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 **375** of file **f_util.h**.

5.9.2.11   F_PASS_IS_OUT_OVF

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

Password is overflow and cannot be stored.

Definition at line **209** of file **f_util.h**.

5.9.2.12 F_PASS_IS_TOO_LONG

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

Password is too long.

Definition at line **197** of file **f_util.h**.

5.9.2.13 F_PASS_IS_TOO_SHORT

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

Password is too short.

Definition at line **203** of file **f_util.h**.

5.9.2.14 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 **167** of file **f_util.h**.

5.9.2.15 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 **191** of file **f_util.h**.

5.9.2.16 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 **173** of file **f_util.h**.

5.9.2.17 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 **179** of file **f_util.h**.

5.9.2.18 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 **185** of file **f_util.h**.

5.9.3 Typedef Documentation

5.9.3.1 ERR_ECDSA_PUBLIC_KEY_VALID

```
typedef enum  err_ecdsa_public_key_valid_e  ERR_ECDSA_PUBLIC_KEY_VALID
```

Typedef of err_ecdsa_public_key_valid_e.

5.9.3.2 ERR_ECDSA_SECRET_KEY_VALID

```
typedef enum  err_ecdsa_secret_key_valid_e  ERR_ECDSA_SECRET_KEY_VALID
```

Typedef of err_ecdsa_secret_key_valid_e.

5.9.3.3 fn_det

```
typedef int(* fn_det) (void *, unsigned char *, size_t)
```

Definition at line **545** of file **f_util.h**.

5.9.3.4 rnd_fn

`rnd_fn`

Pointer caller for random function.

Definition at line **340** of file **f_util.h**.

5.9.4 Enumeration Type Documentation

5.9.4.1 err_ecdsa_public_key_valid_e

`enum` **err_ecdsa_public_key_valid_e**

ECDSA public key error checker enumerator.

Enumerator

ERR_KEY_PK_SIZE_ZERO	Public key size is zero.
ERR_PK_MALLOC	Error alloc memory for check.
ERR_PK_CHECK	Error binary read.

Definition at line **593** of file **f_util.h**.

5.9.4.2 err_ecdsa_secret_key_valid_e

enum **err_ecdsa_secret_key_valid_e**

ECDSA secret key error checker enumerator.

Enumerator

ERR_KEY_SK_SIZE_ZERO	Key size is zero.
ERR_KEY_WRONG_SIZE	Error key wrong size.
ERR_SK_MALLOC	Error alloc memory for check.
ERR_SK_READ_BINARY	Error binary read.
ERR_SK_CHECK	Secret key invalid.

Definition at line **576** of file **f_util.h**.

5.9.5 Function Documentation

5.9.5.1 crc32_init()

```
uint32_t crc32_init (
        unsigned char * p,
        size_t len,
        uint32_t crcinit )
```

Performs a CRC32 of a given data.

Parameters

in	*p*	Pointer of the data
in	*len*	Size of data in pointer *p*
in	*crcinit*	Init vector of the CRC32

Return values

CRC32	hash

5.9.5.2 f_base64_decode_dynamic()

```
int f_base64_decode_dynamic (
            void ** ,
            size_t * ,
            const char * ,
            size_t  )
```

5.9.5.3 f_base64url_decode()

```
int f_base64url_decode (
            void * ,
            size_t ,
            size_t * ,
            const char * ,
            size_t  )
```

5.9.5.4 f_base64url_encode()

```
int f_base64url_encode (
            char * ,
            size_t ,
            size_t * ,
            void * ,
            size_t  )
```

5.9.5.5 f_base64url_encode_dynamic()

```
int f_base64url_encode_dynamic (
            void ** ,
            size_t * ,
            void * ,
            size_t  )
```

5.9.5.6 f_convert_to_double()

```
int f_convert_to_double (
            double * val,
            const char * value )
```

Convert any valid number im *value* and converts it to double *val*

Parameters

out	*val*	Value converted to double
in	*value*	Value in string to be converted

Return values

0	On Success, Otherwise error

5.9.5.7 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. **??**)

5.9.5.8 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. **??**)

5.9.5.9 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. **??**)

5.9.5.10 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

Parameters

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. **??**)

5.9.5.11 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. **??**)

5.9.5.12 f_convert_to_unsigned_int0()

```
int f_convert_to_unsigned_int0 (
            unsigned int * val,
            char * value,
            size_t value_sz )
```

Converts a octal value in ASCII string 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_unsigned_int0x()** (p. **??**)

5.9.5.13 f_convert_to_unsigned_int0x()

```
int f_convert_to_unsigned_int0x (
            unsigned int * val,
            char * value,
            size_t value_sz )
```

Converts a hex value in ASCII string 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_unsigned_int0()** (p. **??**)

5.9.5.14 f_convert_to_unsigned_int_std()

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

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

Parameters

out	*val*	Value stored in a unsigned int variable
in	*value*	Input value to be parsed to unsigned int • If a string contains only numbers, it will be parsed to unsigned 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_unsigned_int()** (p. **??**)

5.9.5.15 f_ecdsa_public_key_valid()

```
ERR_ECDSA_PUBLIC_KEY_VALID f_ecdsa_public_key_valid (
            mbedtls_ecp_group_id gid,
            unsigned char * public_key,
            size_t public_key_len )
```

Checks is ECDSA public key is valid.

Parameters

in	*gid*	Input EC group *id*
in	*public_key*	Public key
in	*public_key_len*	Size of public key

Return values

ERROR_SUCCESS	(0) if success or non zero if error

5.9.5.16   f_ecdsa_secret_key_valid()

```
ERR_ECDSA_SECRET_KEY_VALID f_ecdsa_secret_key_valid (
            mbedtls_ecp_group_id gid,
            unsigned char * secret_key,
            size_t secret_key_len )
```

Checks is ECDSA secret key is valid.

Parameters

in	*gid*	Input EC group *id*
in	*secret_key*	Secret key
in	*secret_key_len*	Size of secret key

Return values

ERROR_SUCCESS	(0) if success or non zero if error

5.9.5.17   f_encode_to_base64()

```
int f_encode_to_base64 (
            char * ,
            size_t ,
            size_t * ,
            void * ,
            size_t  )
```

5.9.5.18   f_encode_to_base64_dynamic()

```
int f_encode_to_base64_dynamic (
            char ** ,
            size_t * ,
            void * ,
            size_t  )
```

5.9.5.19   f_gen_ecdsa_key_pair()

```
f_ecdsa_key_pair_err f_gen_ecdsa_key_pair (
            f_ecdsa_key_pair * ,
            int ,
             fn_det ,
            void * )
```

5.9.5.20 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\<br\>

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

5.9.5.21 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

5.9.5.22 f_hmac_sha512()

```
f_md_hmac_sha512 f_hmac_sha512 (
            unsigned char * result,
            const unsigned char * key,
            size_t key_len,
            const unsigned char * data,
            size_t data_len )
```

Calculates SHA512 HMAC.

Parameters

out	*result*	Output result
in	*key*	Pointer of the key address
in	*key_len*	Size of *key*
in	*data*	Data pointer address
in	*data_len*	Size of *data*

Return values

ERROR_SUCCESS	(0) if success or non zero if error

5.9.5.23 f_is_random_attached()

```
void * f_is_random_attached ( )
```

Verifies if system random function is attached in myNanoEmbedded API.

Return values

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

See also

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

5.9.5.24 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)

5.9.5.25 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

5.9.5.26 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. **??**)

5.9.5.27 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. **??**)

5.9.5.28 f_random_detach()

```
void f_random_detach ( )
```

Detaches system random numeber genarator from myNanoEmbedded API.

See also

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

5.9.5.29 f_reverse()

```
int f_reverse (
          unsigned char * val,
          size_t val_sz )
```

Reverse bytes.

Parameters

in	*val*	Pointer to be reversed
in	*val_sz*	Size of *val*

Return values

ERROR_SUCCESS	(0) if success or non zero if error

5.9.5.30 f_ripemd160()

```
uint8_t* f_ripemd160 (
          const uint8_t * ,
          size_t  )
```

5.9.5.31 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*

5.9.5.32 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

5.9.5.33 f_uncompress_elliptic_curve()

```
int f_uncompress_elliptic_curve (
            uint8_t * ,
            size_t ,
            size_t * ,
            mbedtls_ecp_group_id ,
            uint8_t * ,
            size_t  )
```

5.9.5.34   f_url_base64_to_base64_dynamic()

```
int f_url_base64_to_base64_dynamic (
          char ** ,
          size_t * ,
          const char * ,
          size_t  )
```

5.9.5.35   f_url_decode()

```
int f_url_decode (
          void * ,
          size_t ,
          size_t * ,
          const char * ,
          size_t  )
```

5.9.5.36   f_url_encode()

```
int f_url_encode (
          char * ,
          size_t ,
          size_t * ,
          uint8_t * ,
          size_t  )
```

5.9.5.37   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

5.9.5.38 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

5.10 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 #include "mbedtls/ecdsa.h"
00017 #include "errors.h"
00018
00019 #ifdef __cplusplus
00020 extern "C" {
00021 #endif
00022
00023 #ifndef F_DOC_SKIP
00024
00025  #define F_LOG_MAX 8*256
00026  #define LICENSE \
00027 "MIT License\n\n\
00028 Copyright (c) 2019 Fábio Pereira da Silva\n\n\
00029 Permission is hereby granted, free of charge, to any person obtaining a copy\n\
00030 of this software and associated documentation files (the \"Software\"), to deal\n\
00031 in the Software without restriction, including without limitation the rights\n\
00032 to use, copy, modify, merge, publish, distribute, sublicense, and/or sell\n\
00033 copies of the Software, and to permit persons to whom the Software is\n\
00034 furnished to do so, subject to the following conditions:\n\n\
00035 The above copyright notice and this permission notice shall be included in all\n\
00036 copies or substantial portions of the Software.\n\n\
00037 THE SOFTWARE IS PROVIDED \"AS IS\", WITHOUT WARRANTY OF ANY KIND, EXPRESS OR\n\
00038 IMPLIED, INCLUDING BUT NOT LIMITED TO THE WARRANTIES OF MERCHANTABILITY,\n\
00039 FITNESS FOR A PARTICULAR PURPOSE AND NONINFRINGEMENT. IN NO EVENT SHALL THE\n\
00040 AUTHORS OR COPYRIGHT HOLDERS BE LIABLE FOR ANY CLAIM, DAMAGES OR OTHER\n\
00041 LIABILITY, WHETHER IN AN ACTION OF CONTRACT, TORT OR OTHERWISE, ARISING FROM,\n\
00042 OUT OF OR IN CONNECTION WITH THE SOFTWARE OR THE USE OR OTHER DEALINGS IN THE\n\
00043 SOFTWARE.\n\n\n"
00044
00045 #endif
00046
00047 #ifdef F_ESP32
00048
00049  #define F_WDT_MAX_ENTROPY_TIME 2*120
00050  #define F_WDT_PANIC true
00051  #define F_WDT_MIN_TIME 20//4
00052
00053 #endif
00054
00072 int f_verify_system_entropy(uint32_t, void *, size_t, int);
00073
00100 int f_pass_must_have_at_least(char *, size_t, size_t, size_t, int);
00101
00102 #ifndef F_DOC_SKIP
00103
00104 int f_verify_system_entropy_begin();
00105 void f_verify_system_entropy_finish();
00106 int f_file_exists(char *);
00107 int f_find_str(size_t *, char *, size_t, char *);
00108 int f_find_replace(char *, size_t *, size_t, char *, size_t, char *, char *);
00109 int f_is_integer(char *, size_t);
00110 int is_filled_with_value(uint8_t *, size_t, uint8_t);
00111
00112 #endif
00113
00114 //#define F_ENTROPY_TYPE_PARANOIC (uint32_t)1476682819
00119 #define F_ENTROPY_TYPE_PARANOIC (uint32_t)1477682819
00120
00121 //#define F_ENTROPY_TYPE_EXCELENT (uint32_t)1475885281
00126 #define F_ENTROPY_TYPE_EXCELENT (uint32_t)1476885281
00127
00128 //#define F_ENTROPY_TYPE_GOOD (uint32_t)1471531015
00133 #define F_ENTROPY_TYPE_GOOD (uint32_t)1472531015
00134
00135 //#define F_ENTROPY_TYPE_NOT_ENOUGH (uint32_t)1470001808
00140 #define F_ENTROPY_TYPE_NOT_ENOUGH (uint32_t)1471001808
00141
00142 //#define F_ENTROPY_TYPE_NOT_RECOMENDED (uint32_t)1469703345
00147 #define F_ENTROPY_TYPE_NOT_RECOMENDED (uint32_t)1470003345
00148
00154 #define ENTROPY_BEGIN f_verify_system_entropy_begin();
00155
00161 #define ENTROPY_END f_verify_system_entropy_finish();
00162
```

```
00167 #define F_PASS_MUST_HAVE_AT_LEAST_NONE (int)0
00168
00173 #define F_PASS_MUST_HAVE_AT_LEAST_ONE_NUMBER (int)1
00174
00179 #define F_PASS_MUST_HAVE_AT_LEAST_ONE_SYMBOL (int)2
00180
00185 #define F_PASS_MUST_HAVE_AT_LEAST_ONE_UPPER_CASE (int)4
00186
00191 #define F_PASS_MUST_HAVE_AT_LEAST_ONE_LOWER_CASE (int)8
00192
00197 #define F_PASS_IS_TOO_LONG (int)256
00198
00203 #define F_PASS_IS_TOO_SHORT (int)512
00204
00209 #define F_PASS_IS_OUT_OVF (int)1024//768
00210
00211 #ifndef F_DOC_SKIP
00212
00213  #define F_PBKDF2_ITER_SZ 2*4096
00214
00215 typedef enum f_pbkdf2_err_t {
00216     F_PBKDF2_RESULT_OK=0,
00217     F_PBKDF2_ERR_CTX=95,
00218     F_PBKDF2_ERR_PKCS5,
00219     F_PBKDF2_ERR_INFO_SHA
00220 } f_pbkdf2_err;
00221
00222 typedef enum f_aes_err {
00223     F_AES_RESULT_OK=0,
00224     F_AES_ERR_ENCKEY=30,
00225     F_AES_ERR_DECKEY,
00226     F_AES_ERR_MALLOC,
00227     F_AES_UNKNOW_DIRECTION,
00228     F_ERR_ENC_DECRYPT_FAILED
00229 } f_aes_err;
00230
00231 typedef enum f_md_hmac_sha512_t {
00232    F_HMAC_SHA512_OK = 0,
00233    F_HMAC_SHA512_MALLOC = 304,
00234    F_HMAC_SHA512_ERR_INFO,
00235    F_HMAC_SHA512_ERR_SETUP,
00236    F_HMAC_SHA512_DIGEST_ERROR
00237 } f_md_hmac_sha512;
00239 typedef enum f_ecdsa_key_pair_err_t {
00240     F_ECDSA_KEY_PAIR_OK = 0,
00241     F_ECDSA_KEY_PAIR_NULL = 330,
00242     F_ECDSA_KEY_PAIR_MALLOC
00243 } f_ecdsa_key_pair_err;
00244
00245 typedef struct f_ecdsa_key_pair_t {
00246     size_t public_key_sz;
00247     size_t private_key_sz;
00248     mbedtls_ecdsa_context *ctx;
00249     mbedtls_ecp_group_id gid;
00250     unsigned char public_key[MBEDTLS_ECDSA_MAX_LEN];
00251     unsigned char private_key[MBEDTLS_ECDSA_MAX_LEN];
00252 } f_ecdsa_key_pair;
00253
00258 enum f_encode_decode_error_t {
00259    F_URL_ENCODE_OK = 0,
00260    F_ENCODE_BASE64_DEST_SMALL=11300,
00261    F_ENCODE_TO_BASE64_MALLOC,
00262    F_BASE64_DECODE_MALLOC,
00263    F_URL_ENCODE_EMPTY,
00264    F_URL_ENCODE_DEST_SMALL,
00265    F_BASE64_URL_DECODE_MALLOC,
00266    F_BASE64_URL_DECODE_MEMORY_SMALL,
00267    F_BASE64_URL_TO_BASE64_EMPTY_BASE64,
00268    F_BASE64_URL_TO_BASE64_MALLOC,
00269    F_URL_ENCODE_EMPTY_STRING,
00270    F_URL_ENCODE_WAITING_NEXT_NIBBLE,
00271    F_URL_INVALID_HEX_STRING,
00272    F_URL_NO_SPACE_IN_MEMORY_BUFFER,
00273    F_URL_ENCODE_INVALID_STRING
00274 };
00275
00276 char *fhex2strv2(char *, const void *, size_t, int);
00277 int f_sha256_digest(void **, int, uint8_t *, size_t);
00278 f_pbkdf2_err f_pbkdf2_hmac(unsigned char *, size_t, unsigned char *, size_t, uint8_t *);
00279 f_aes_err f_aes256cipher(uint8_t *, uint8_t *, void *, size_t, void *, int);
00280
00281 #endif
00282
00294 int f_passwd_comp_safe(char *, char *, size_t, size_t, size_t);
00295
00306 char *f_get_entropy_name(uint32_t);
00307
```

```
00322 uint32_t f_sel_to_entropy_level(int);
00323
00332 int f_str_to_hex(uint8_t *, char *);
00333
00334 #ifndef F_ESP32
00335
00340 typedef void (*rnd_fn)(void *, size_t);
00341
00349 void f_random_attach(rnd_fn);
00350
00359 void f_random(void *, size_t);
00360
00369 int get_console_passwd(char *, size_t);
00370
00375 #define F_GET_CH_MODE_NO_ECHO (int)(1<<16)
00376
00381 #define F_GET_CH_MODE_ANY_KEY (int)(1<<17)
00382
00398 int f_get_char_no_block(int);
00399
00400 #endif
00401
00412 int f_convert_to_long_int(unsigned long int *, char *, size_t);
00413
00414
00425 int f_convert_to_unsigned_int(unsigned int *, char *, size_t);
00426
00437 int f_convert_to_long_int0x(unsigned long int *, char *, size_t);
00438
00449 int f_convert_to_long_int0(unsigned long int *, char *, size_t);
00450
00464 int f_convert_to_long_int_std(unsigned long int *, char *, size_t);
00465
00473 void *f_is_random_attached();
00474
00481 void f_random_detach();
00482
00493 int f_convert_to_unsigned_int0x(unsigned int *val, char *value, size_t value_sz);
00494
00505 int f_convert_to_unsigned_int0(unsigned int *val, char *value, size_t value_sz);
00506
00520 int f_convert_to_unsigned_int_std(unsigned int *val, char *value, size_t value_sz);
00521
00531 int f_convert_to_double(double *, const char *);
00532
00543 uint32_t crc32_init(unsigned char *, size_t, uint32_t);
00544 //
00545 typedef int (*fn_det)(void *, unsigned char *, size_t);
00546
00556 int f_reverse(unsigned char *, size_t);
00557
00570 f_md_hmac_sha512 f_hmac_sha512(unsigned char *, const unsigned char *, size_t, const unsigned char *,
     size_t);
00571
00576 typedef enum err_ecdsa_secret_key_valid_e {
00578    ERR_KEY_SK_SIZE_ZERO = 476,
00580    ERR_KEY_WRONG_SIZE,
00582    ERR_SK_MALLOC,
00584    ERR_SK_READ_BINARY,
00586    ERR_SK_CHECK
00587 }  ERR_ECDSA_SECRET_KEY_VALID;
00588
00593 typedef enum err_ecdsa_public_key_valid_e {
00595    ERR_KEY_PK_SIZE_ZERO = 501,
00597    ERR_PK_MALLOC,
00599    ERR_PK_CHECK
00600 }  ERR_ECDSA_PUBLIC_KEY_VALID;
00601
00602
00613 ERR_ECDSA_SECRET_KEY_VALID
00614 f_ecdsa_secret_key_valid(mbedtls_ecp_group_id, unsigned char *, size_t);
00615
00626 ERR_ECDSA_PUBLIC_KEY_VALID
00627 f_ecdsa_public_key_valid(mbedtls_ecp_group_id, unsigned char *, size_t);
00628
00629 f_ecdsa_key_pair_err f_gen_ecdsa_key_pair(f_ecdsa_key_pair *, int, fn_det, void *);
00630 int f_uncompress_elliptic_curve(uint8_t *, size_t, size_t *, mbedtls_ecp_group_id, uint8_t *, size_t);
00631 uint8_t *f_ripemd160(const uint8_t *, size_t);
00632 int f_url_encode(char *, size_t, size_t *, uint8_t *, size_t);
00633 int f_encode_to_base64_dynamic(char **, size_t *, void *, size_t );
00634 int f_base64_decode_dynamic(void **, size_t *, const char *, size_t);
00635 int f_base64url_encode_dynamic(void **, size_t *, void *, size_t);
00636 int f_encode_to_base64(char *, size_t, size_t *, void *, size_t);
00637 int f_base64url_encode(char *, size_t, size_t *, void *, size_t);
00638 int f_base64url_decode(void *, size_t, size_t *, const char *, size_t);
00639 int f_url_base64_to_base64_dynamic(char **, size_t *, const char *, size_t);
00640 int f_url_decode(void *, size_t, size_t *, const char *, size_t);
```

```
00641
00642 #define CLEAR_AND_FREE(ptr, size) free(memset(ptr, 0, size));
00643 #ifdef __cplusplus
00644 }
00645 #endif
```

5.11   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_internal_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_core_ristretto255.h"
#include "sodium/crypto_scalarmult_ed25519.h"
#include "sodium/crypto_scalarmult_ristretto255.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"
```

5.12   sodium.h

```
00001
00002 #ifndef sodium_H
00003 #define sodium_H
00004
00005 #include "sodium/version.h"
00006
00007 #include "sodium/core.h"
00008 #include "sodium/crypto_aead_aes256gcm.h"
00009 #include "sodium/crypto_aead_chacha20poly1305.h"
00010 #include "sodium/crypto_aead_xchacha20poly1305.h"
00011 #include "sodium/crypto_auth.h"
00012 #include "sodium/crypto_auth_hmacsha256.h"
00013 #include "sodium/crypto_auth_hmacsha512.h"
00014 #include "sodium/crypto_auth_hmacsha512256.h"
00015 #include "sodium/crypto_box.h"
00016 #include "sodium/crypto_box_curve25519xsalsa20poly1305.h"
00017 #include "sodium/crypto_core_hsalsa20.h"
00018 #include "sodium/crypto_core_hchacha20.h"
00019 #include "sodium/crypto_core_salsa20.h"
00020 #include "sodium/crypto_core_salsa2012.h"
00021 #include "sodium/crypto_core_salsa208.h"
00022 #include "sodium/crypto_generichash.h"
00023 #include "sodium/crypto_generichash_blake2b.h"
00024 #include "sodium/crypto_hash.h"
00025 #include "sodium/crypto_hash_sha256.h"
00026 #include "sodium/crypto_hash_sha512.h"
00027 #include "sodium/crypto_kdf.h"
00028 #include "sodium/crypto_kdf_blake2b.h"
00029 #include "sodium/crypto_kx.h"
00030 #include "sodium/crypto_onetimeauth.h"
00031 #include "sodium/crypto_onetimeauth_poly1305.h"
00032 #include "sodium/crypto_pwhash.h"
00033 #include "sodium/crypto_pwhash_argon2i.h"
00034 #include "sodium/crypto_scalarmult.h"
00035 #include "sodium/crypto_scalarmult_curve25519.h"
00036 #include "sodium/crypto_secretbox.h"
00037 #include "sodium/crypto_secretbox_xsalsa20poly1305.h"
00038 #include "sodium/crypto_secretstream_xchacha20poly1305.h"
00039 #include "sodium/crypto_shorthash.h"
00040 #include "sodium/crypto_shorthash_siphash24.h"
00041 #include "sodium/crypto_sign.h"
00042 #include "sodium/crypto_sign_ed25519.h"
00043 #include "sodium/crypto_stream.h"
00044 #include "sodium/crypto_stream_chacha20.h"
00045 #include "sodium/crypto_stream_salsa20.h"
00046 #include "sodium/crypto_stream_xsalsa20.h"
00047 #include "sodium/crypto_verify_16.h"
00048 #include "sodium/crypto_verify_32.h"
00049 #include "sodium/crypto_verify_64.h"
00050 #include "sodium/randombytes.h"
00051 #include "sodium/randombytes_internal_random.h"
00052 #include "sodium/randombytes_sysrandom.h"
00053 #include "sodium/runtime.h"
00054 #include "sodium/utils.h"
00055
00056 #ifndef SODIUM_LIBRARY_MINIMAL
00057 # include "sodium/crypto_box_curve25519xchacha20poly1305.h"
00058 # include "sodium/crypto_core_ed25519.h"
00059 # include "sodium/crypto_core_ristretto255.h"
00060 # include "sodium/crypto_scalarmult_ed25519.h"
00061 # include "sodium/crypto_scalarmult_ristretto255.h"
00062 # include "sodium/crypto_secretbox_xchacha20poly1305.h"
00063 # include "sodium/crypto_pwhash_scryptsalsa208sha256.h"
```

```
00064 # include "sodium/crypto_stream_salsa2012.h"
00065 # include "sodium/crypto_stream_salsa208.h"
00066 # include "sodium/crypto_stream_xchacha20.h"
00067 #endif
00068
00069 #endif
```

5.13 version.h File Reference

Functions

- char ∗ **getTextInfoVersion** ()
- char ∗ **releaseDateVersion** ()
- char ∗ **getTextVersion** ()
- uint32_t **getVersion** ()
- uint16_t **getVersionMinor1** ()
- uint16_t **getVersionMinor2** ()
- uint16_t **getVersionMajor** ()

5.13.1 Function Documentation

5.13.1.1 getTextInfoVersion()

```
char * getTextInfoVersion ( )
```

Get information and version about *myNanoEmbedded*

Returns

Text string of *myNanoEmbedded*

5.13.1.2 getTextVersion()

```
char* getTextVersion ( )
```

5.13.1.3 getVersion()

```
uint32_t getVersion ( )
```

Returns

Numeric value of *myNanoEmbedded* version

5.13.1.4 getVersionMajor()

```
uint16_t getVersionMajor ( )
```

Returns

Numeric value of major version of *myNanoEmbedded*

5.13.1.5 getVersionMinor1()

```
uint16_t getVersionMinor1 ( )
```

Returns

Numeric value of first minor version of *myNanoEmbedded*

5.13.1.6 getVersionMinor2()

```
uint16_t getVersionMinor2 ( )
```

Returns

Numeric value of second minor version of *myNanoEmbedded*

5.13.1.7 releaseDateVersion()

```
char * releaseDateVersion ( )
```

Returns

Displays YYYYMMDDHHMM release date in *myNanoEmbedded*
Displays x.x.x version in *myNanoEmbedded*

5.14 version.h

```
00001
00010 char *getTextInfoVersion();
00011
00019 char *releaseDateVersion();
00020
00028 char *getTextVersion();
00029
00037 uint32_t getVersion();
00038
00046 uint16_t getVersionMinor1();
00047
00055 uint16_t getVersionMinor2();
00056
00064 uint16_t getVersionMajor();
00065
```

Index