dPoS a small descentralized DIY IoT PoS with integration with Nano and Bitcoin

0.1.0

Generated by Doxygen 1.8.13

Contents

Chapter 1

Overview

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

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

API features

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

To add this API in your project you must first:

1. Download the latest version.

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

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

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

Initialize API

Function	Description
f_random_attach()	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 **214** 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 **218** 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 **226** of file **f_nano_crypto_util.h**.

4.2.2.3 link

```
uint8_t link[32]
```

link or destination account

Definition at line **228** of file **f_nano_crypto_util.h**.

4.2.2.4 preamble

```
uint8_t preamble[32]
```

Block preamble.

Definition at line **216** of file **f_nano_crypto_util.h**.

4.2.2.5 prefixes

```
uint8_t prefixes
```

Internal use for this API.

Definition at line **232** of file **f_nano_crypto_util.h**.

4.2.2.6 previous

```
uint8_t previous[32]
```

Previous block.

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

4.2.2.7 representative

```
uint8_t representative[32]
```

Representative for current account.

Definition at line **222** of file **f_nano_crypto_util.h**.

4.2.2.8 signature

```
uint8_t signature[64]
```

Signature of the block.

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

4.2.2.9 work

```
uint64_t work
```

Internal use for this API.

Definition at line **234** 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_ecdsa_key_pair_t Struct Reference

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

Data Fields

- size_t **public_key_sz**
- size_t **private_key_sz**
- mbedtls_ecdsa_context * **ctx**
- mbedtls_ecp_group_id **gid**
- unsigned char **public_key** [MBEDTLS_ECDSA_MAX_LEN]
- unsigned char **private_key** [MBEDTLS_ECDSA_MAX_LEN]

4.3.1 Detailed Description

Definition at line **241** of file **f_util.h**.

4.3.2 Field Documentation

4.3.2.1 ctx

```
mbedtls_ecdsa_context* ctx
```

Definition at line **244** of file **f_util.h**.

4.3.2.2 gid

```
mbedtls_ecp_group_id gid
```

Definition at line **245** of file **f_util.h**.

4.3.2.3 private_key

```
unsigned char private_key[MBEDTLS_ECDSA_MAX_LEN]
```

Definition at line **247** of file **f_util.h**.

4.3.2.4 private_key_sz

```
size_t private_key_sz
```

Definition at line **243** of file **f_util.h**.

4.3.2.5 public_key

```
unsigned char public_key[MBEDTLS_ECDSA_MAX_LEN]
```

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

4.3.2.6 public_key_sz

```
size_t public_key_sz
```

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

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

- **f_util.h**

4.4 f_file_info_err_t Struct Reference

`#include <f_nano_crypto_util.h>`

4.4.1 Detailed Description

Error enumerator for info file functions.

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

- **f_nano_crypto_util.h**

4.5 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**

4.5.1 Detailed Description

struct of the block of encrypted file to store Nano SEED

Definition at line **348** of file **f_nano_crypto_util.h**.

4.5.2 Field Documentation

4.5.2.1 description

`uint8_t description[ F_DESC_SZ]`

File description.

Definition at line **354** of file **f_nano_crypto_util.h**.

4.5.2.2 iv

```
uint8_t iv[16]
```

Initial vector of first encryption layer.

Definition at line **358** of file **f_nano_crypto_util.h**.

4.5.2.3 nano_hdr

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

Header of the file.

Definition at line **350** of file **f_nano_crypto_util.h**.

4.5.2.4 salt

```
uint8_t salt[32]
```

Salt of the first encryption layer.

Definition at line **356** of file **f_nano_crypto_util.h**.

4.5.2.5 seed_block

```
F_ENCRYPTED_BLOCK seed_block
```

Second encrypted block for Nano SEED.

Definition at line **360** of file **f_nano_crypto_util.h**.

4.5.2.6 ver

```
uint32_t ver
```

Version of the file.

Definition at line **352** 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_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]

4.6.1 Detailed Description

struct of the block of encrypted file to store Nano SEED

Definition at line **320** of file **f_nano_crypto_util.h**.

4.6.2 Field Documentation

4.6.2.1 hash_sk_unencrypted

`uint8_t hash_sk_unencrypted[32]`

hash of Nano SEED when unencrypted

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

4.6.2.2 iv

`uint8_t iv[16]`

Initial sub vector.

Definition at line **324** of file **f_nano_crypto_util.h**.

4.6.2.3 reserved

`uint8_t reserved[16]`

Reserved (not used)

Definition at line **326** of file **f_nano_crypto_util.h**.

4.6.2.4 sk_encrypted

```
uint8_t sk_encrypted[32]
```

Secret.

SEED encrypted (second layer)

Definition at line **330** of file **f_nano_crypto_util.h**.

4.6.2.5 sub_salt

```
uint8_t sub_salt[32]
```

Salt of the sub block to be stored.

Definition at line **322** 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_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]

4.7.1 Detailed Description

struct of the body block of the info file

Definition at line **432** of file **f_nano_crypto_util.h**.

4.7.2 Field Documentation

4.7.2.1 last_used_wallet_number

```
uint32_t last_used_wallet_number
```

Last used wallet number.

Definition at line **436** of file **f_nano_crypto_util.h**.

4.7.2.2 max_fee

```
char max_fee[ F_RAW_STR_MAX_SZ]
```

Custom preferred max fee of Proof of Work.

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

4.7.2.3 reserved

```
uint8_t reserved[44]
```

Reserved.

Definition at line **442** of file **f_nano_crypto_util.h**.

4.7.2.4 wallet_prefix

```
uint8_t wallet_prefix
```

Wallet prefix: 0 for NANO; 1 for XRB.

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

4.7.2.5 wallet_representative

```
char wallet_representative[ MAX_STR_NANO_CHAR]
```

Wallet representative.

Definition at line **438** of file **f_nano_crypto_util.h**.

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

- **f_nano_crypto_util.h**

4.8 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**

4.8.1 Detailed Description

struct of the body block of the info file

Definition at line **464** of file **f_nano_crypto_util.h**.

4.8.2 Field Documentation

4.8.2.1 body

`F_NANO_WALLET_INFO_BODY body`

Body of the file info.

Definition at line **476** of file **f_nano_crypto_util.h**.

4.8.2.2 desc

`char desc[ F_NANO_DESC_SZ]`

Description.

Definition at line **470** of file **f_nano_crypto_util.h**.

4.8.2.3 file_info_integrity

```
uint8_t file_info_integrity[32]
```

File info integrity of the body block.

Definition at line **474** of file **f_nano_crypto_util.h**.

4.8.2.4 header

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

Header magic.

Definition at line **466** of file **f_nano_crypto_util.h**.

4.8.2.5 nanoseed_hash

```
uint8_t nanoseed_hash[32]
```

Nano SEED hash file.

Definition at line **472** of file **f_nano_crypto_util.h**.

4.8.2.6 version

```
uint16_t version
```

Version.

Definition at line **468** of file **f_nano_crypto_util.h**.

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

- **f_nano_crypto_util.h**

4.9 QRCode Struct Reference

```
#include <qrcode.h>
```

Data Fields

- uint8_t **version**
- uint8_t **size**
- uint8_t **ecc**
- uint8_t **mode**
- uint8_t **mask**
- uint8_t * **modules**

4.9.1 Detailed Description

Definition at line **69** of file **qrcode.h**.

4.9.2 Field Documentation

4.9.2.1 ecc

```
uint8_t ecc
```

Definition at line **72** of file **qrcode.h**.

4.9.2.2 mask

```
uint8_t mask
```

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

4.9.2.3 mode

```
uint8_t mode
```

Definition at line **73** of file **qrcode.h**.

4.9.2.4 modules

```
uint8_t* modules
```

Definition at line **75** of file **qrcode.h**.

4.9.2.5 size

```
uint8_t size
```

Definition at line **71** of file **qrcode.h**.

4.9.2.6 version

```
uint8_t version
```

Definition at line **70** of file **qrcode.h**.

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

- **qrcode.h**

4.10 upos_register_events_t Struct Reference

```
#include <upos_events.h>
```

Data Fields

- int **err**
- const char * **tag**
- esp_event_base_t **event_base**
- int32_t **event_id**
- esp_event_handler_t **event_handler**
- void * **arg**
- **fn_evt ev_cb**

4.10.1 Detailed Description

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

4.10.2 Field Documentation

4.10.2.1 arg

```
void* arg
```

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

4.10.2.2 err

```
int err
```

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

4.10.2.3 ev_cb

```
fn_evt ev_cb
```

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

4.10.2.4 event_base

```
esp_event_base_t event_base
```

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

4.10.2.5 event_handler

```
esp_event_handler_t event_handler
```

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

4.10.2.6 event_id

```
int32_t event_id
```

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

4.10.2.7 tag

```
const char* tag
```

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

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

- **upos_events.h**

4.11 upos_wifi_cb_ctx_t Struct Reference

`#include <upos_wifi.h>`

Data Fields

- void * **on_connect_ctx**
- **upos_wifi_cb on_connect**
- void * **on_disconnect_ctx**
- **upos_wifi_cb on_disconnect**
- void * **on_ipv4_ctx**
- **upos_wifi_cb on_ipv4**
- void * **on_ipv6_ctx**
- **upos_wifi_cb on_ipv6**

4.11.1 Detailed Description

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

4.11.2 Field Documentation

4.11.2.1 on_connect

upos_wifi_cb on_connect

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

4.11.2.2 on_connect_ctx

void* on_connect_ctx

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

4.11.2.3 on_disconnect

upos_wifi_cb on_disconnect

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

4.11.2.4 on_disconnect_ctx

```
void* on_disconnect_ctx
```

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

4.11.2.5 on_ipv4

```
upos_wifi_cb on_ipv4
```

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

4.11.2.6 on_ipv4_ctx

```
void* on_ipv4_ctx
```

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

4.11.2.7 on_ipv6

```
upos_wifi_cb on_ipv6
```

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

4.11.2.8 on_ipv6_ctx

```
void* on_ipv6_ctx
```

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

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

- **upos_wifi.h**

4.12 upos_wifi_event_cb_ctx_t Struct Reference

```
#include <upos_wifi.h>
```

Data Fields

- int **err**
- const char * **event_name**
- uint32_t **event_type**
- void * **initial_ctx**

4.12.1 Detailed Description

Definition at line **38** of file **upos_wifi.h**.

4.12.2 Field Documentation

4.12.2.1 err

```
int err
```

Definition at line **39** of file **upos_wifi.h**.

4.12.2.2 event_name

```
const char* event_name
```

Definition at line **40** of file **upos_wifi.h**.

4.12.2.3 event_type

```
uint32_t event_type
```

Definition at line **41** of file **upos_wifi.h**.

4.12.2.4 initial_ctx

```
void* initial_ctx
```

Definition at line **42** of file **upos_wifi.h**.

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

- **upos_wifi.h**

4.13 upos_wifi_status_t Struct Reference

`#include <upos_wifi.h>`

Data Fields

- int **err**
- const char * **tag**

4.13.1 Detailed Description

Definition at line **10** of file **upos_wifi.h**.

4.13.2 Field Documentation

4.13.2.1 err

`int err`

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

4.13.2.2 tag

`const char* tag`

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

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

- **upos_wifi.h**

Chapter 5

File Documentation

5.1 f_add_bn_288_le.h File Reference

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

Typedefs

- typedef uint8_t **F_ADD_288**[36]

Functions

- void **f_add_bn_288_le** (**F_ADD_288**, **F_ADD_288**, **F_ADD_288**, int *, int)
- void **f_sl_elv_add_le** (**F_ADD_288**, int)

5.1.1 Detailed Description

Low level implementation of Nano Cryptocurrency C library.

Definition in file **f_add_bn_288_le.h**.

5.1.2 Typedef Documentation

5.1.2.1 F_ADD_288

`F_ADD_288`

288 bit big number

Definition at line **20** of file **f_add_bn_288_le.h**.

5.1.3 Function Documentation

5.1.3.1 f_add_bn_288_le()

```
f_add_bn_288_le (
              F_ADD_288 X,
              F_ADD_288 Y,
              F_ADD_288 RES,
             int * carry_out,
             int carry_in )
```

Adds two big numbers of size 288 bits.

This function is implemented in low level for API use. It performs RES = X + Y + carry_in

Parameters

in	*X*	Big number 288 bit X value
in	*Y*	Big number 288 bit Y value
out	*RES*	Big number 288 bit result RES value
out	*carry_out*	Carry out. It can be NULL if you want to omit *carry_out*
in	*carry_in*	Carry in (borrow) of last sum. Parse 0 to omit.

5.1.3.2 f_sl_elv_add_le()

```
void f_sl_elv_add_le (
              F_ADD_288 ,
             int  )
```

5.2 f_add_bn_288_le.h

```
00001 /*
00002     AUTHOR: Fábio Pereira da Silva
00003     YEAR: 2019-20
00004     LICENSE: MIT
00005     EMAIL: fabioegel@gmail.com or fabioegel@protonmail.com
00006 */
00007
00008 #include <config.h>
00009 #include <stdint.h>
00010
00020 typedef uint8_t F_ADD_288[36];
00021
00022
00023 #ifndef F_DOC_SKIP
00024
00034  void f_add_bn_288_le(F_ADD_288, F_ADD_288, F_ADD_288, int *, int);
00035  void f_sl_elv_add_le(F_ADD_288, int);
00036
00037 #endif
00038
```

5.3 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

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)

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]

5.3.1 Macro Definition Documentation

5.3.1.1 DERIVE_XPRIV_XPUB_DYN_OUT_BASE58

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

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

5.3.1.2 DERIVE_XPRIV_XPUB_DYN_OUT_XPRIV

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

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

5.3.1.3 DERIVE_XPRIV_XPUB_DYN_OUT_XPUB

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

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

5.3.1.4 F_BITCOIN_BUF_SZ

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

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

5.3.1.5 F_BITCOIN_P2PKH

```
#define F_BITCOIN_P2PKH (uint8_t)0x00
```

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

5.3.1.6 F_BITCOIN_SEED_GENERATOR

```
#define F_BITCOIN_SEED_GENERATOR "Bitcoin seed"
```

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

5.3.1.7 F_BITCOIN_T2PKH

```
#define F_BITCOIN_T2PKH (uint8_t)0x6F
```

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

5.3.1.8 F_BITCOIN_WIF_MAINNET

```
#define F_BITCOIN_WIF_MAINNET (uint8_t)0x80
```

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

5.3.1.9 F_BITCOIN_WIF_TESTNET

```
#define F_BITCOIN_WIF_TESTNET (uint8_t)0xEF
```

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

5.3.1.10 F_GET_XKEY_IS_BASE58

```
#define F_GET_XKEY_IS_BASE58 (int)0x00008000
```

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

5.3.1.11 F_MAX_BASE58_LENGTH

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

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

5.3.1.12 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**.

5.3.1.13 F_XPRIV_BASE58

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

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

5.3.1.14 F_XPUB_BASE58

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

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

5.3.1.15 MAINNET_PRIVATE

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

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

5.3.1.16 MAINNET_PUBLIC

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

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

5.3.1.17 TESTNET_PRIVATE

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

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

5.3.1.18 TESTNET_PUBLIC

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

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

5.3.2 Function Documentation

5.3.2.1 __attribute__()

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

5.3.2.2 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  )
```

5.3.2.3 f_bitcoin_valid_bip32()

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

5.3.2.4 f_decode_b58_util()

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

5.3.2.5 f_derive_xkey_dynamic()

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

5.3.2.6 f_derive_xpriv_or_xpub_dynamic()

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

5.3.2.7 f_encode_b58()

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

5.3.2.8 f_fingerprint()

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

5.3.2.9 f_generate_master_key()

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

5.3.2.10 f_get_xkey_type()

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

5.3.2.11 f_private_key_to_wif()

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

5.3.2.12 f_public_key_to_address()

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

5.3.2.13 f_uncompress_elliptic_curve()

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

5.3.2.14 f_wif_to_private_key()

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

5.3.2.15 f_xpriv2xpub()

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

5.3.2.16 load_master_private_key()

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

5.3.3 Variable Documentation

5.3.3.1 chain_code

```
uint8_t chain_code[32]
```

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

5.3.3.2 child_number

```
uint8_t child_number[4]
```

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

5.3.3.3 chksum

```
uint8_t chksum[4]
```

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

5.3.3.4 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**.

5.3.3.5 finger_print

```
uint8_t finger_print[4]
```

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

5.3.3.6 master_node

```
uint8_t master_node
```

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

5.3.3.7 sk_or_pk_data

```
uint8_t sk_or_pk_data[33]
```

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

5.3.3.8 version_bytes

```
uint8_t version_bytes[4]
```

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

5.4 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
00067
```

5.5 f_nano_crypto_util.h File Reference

```
#include <stdint.h>
#include <f_util.h>
#include <f_bitcoin.h>
#include "esp_system.h"
#include "sodium/crypto_generichash.h"
#include "sodium/crypto_sign.h"
#include "sodium.h"
#include "sodium/private/curve25519_ref10.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 **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 **BIP39_DICTIONARY** "/spiffs/dictionary.dic"
- #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 **EXPORT_KEY_TO_CHAR_SZ** (size_t)sizeof(**NANO_SEED**)+1
- #define **F_BLOCK_TRANSFER_SIZE** (size_t)sizeof(F_BLOCK_TRANSFER)
- #define **F_P2POW_BLOCK_TRANSFER_SIZE** 2∗ **F_BLOCK_TRANSFER_SIZE**
- #define **F_BLOCK_TRANSFER_SIGNABLE_SZ** (size_t)(sizeof(F_BLOCK_TRANSFER)-64-sizeof(uint64↩ _t)-sizeof(uint8_t))
- #define **READ_SEED_FROM_STREAM** (int)1
- #define **READ_SEED_FROM_FILE** (int)2
- #define **WRITE_SEED_TO_STREAM** (int)4
- #define **WRITE_SEED_TO_FILE** (int)8
- #define **PARSE_JSON_READ_SEED_GENERIC** (int)16
- #define **F_STREAM_DATA_FILE_VERSION** (uint32_t)((1<<16)|0)
- #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 !!!"
- #define **F_DESC_SZ** (size_t) (160-sizeof(uint32_t))
- #define **REP_XRB** (uint8_t)0x4
- #define **SENDER_XRB** (uint8_t)0x02
- #define **DEST_XRB** (uint8_t)0x01
- #define **F_RAW_TO_STR_UINT128** (int)1
- #define **F_RAW_TO_STR_STRING** (int)2
- #define **F_RAW_STR_MAX_SZ** (size_t)41
- #define **F_MAX_STR_RAW_BALANCE_MAX** (size_t)40
- #define **F_NANO_EMPTY_BALANCE** "0.0"

- #define **F_NANO_WALLET_INFO_DESC** "Nano file descriptor used for fast custom access. BUY BITCOIN AND NANO."
- #define **F_NANO_WALLET_INFO_VERSION** (uint16_t)((1<<8)|1)
- #define **F_NANO_DESC_SZ** (size_t)78
- #define **F_NANO_ADD_A_B** (uint32_t)(1<<0)
- #define **F_NANO_SUB_A_B** (uint32_t)(1<<1)
- #define **F_NANO_A_RAW_128** (uint32_t)(1<<2)
- #define **F_NANO_A_RAW_STRING** (uint32_t)(1<<3)
- #define **F_NANO_A_REAL_STRING** (uint32_t)(1<<4)
- #define **F_NANO_B_RAW_128** (uint32_t)(1<<5)
- #define **F_NANO_B_RAW_STRING** (uint32_t)(1<<6)
- #define **F_NANO_B_REAL_STRING** (uint32_t)(1<<7)
- #define **F_NANO_RES_RAW_128** (uint32_t)(1<<8)
- #define **F_NANO_RES_RAW_STRING** (uint32_t)(1<<9)
- #define **F_NANO_RES_REAL_STRING** (uint32_t)(1<<10)
- #define **F_NANO_C_RAW_128** (uint32_t)(**F_NANO_B_RAW_128**<<16)
- #define **F_NANO_C_RAW_STRING** (uint32_t)(**F_NANO_B_RAW_STRING**<<16)
- #define **F_NANO_C_REAL_STRING** (uint32_t)(**F_NANO_B_REAL_STRING**<<16)
- #define **F_NANO_COMPARE_EQ** (uint32_t)(1<<16)
- #define **F_NANO_COMPARE_LT** (uint32_t)(1<<17)
- #define **F_NANO_COMPARE_LEQ** (**F_NANO_COMPARE_LT**| **F_NANO_COMPARE_EQ**)
- #define **F_NANO_COMPARE_GT** (uint32_t)(1<<18)
- #define **F_NANO_COMPARE_GEQ** (**F_NANO_COMPARE_GT**| **F_NANO_COMPARE_EQ**)
- #define **DEFAULT_MAX_FEE** "0.001"
- #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) 0xfffffffc000000000
- #define **F_VERIFY_SIG_NANO_WALLET** (uint32_t)1
- #define **F_VERIFY_SIG_RAW_HEX** (uint32_t)2
- #define **F_VERIFY_SIG_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)

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_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** }

- enum **f_nano_p2pow_block_dyn_err_t** {
NANO_P2POW_CREATE_BLOCK_OK = 0, **NANO_P2POW_CREATE_BLOCK_INVALID_USER_BLOCK** = 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
- **_Static_assert** ((sizeof(F_NANO_CRYPTOWALLET)&0x1F)==0, "Error 1")
- **_Static_assert** ((sizeof(F_ENCRYPTED_BLOCK)&0x1F)==0, "Error 2")
- **_Static_assert** ((sizeof(F_NANO_WALLET_INFO_BODY)&0x1F)==0, "Error F_NANO_WALLET_INFO_BODY is not byte aligned")
- **_Static_assert** ((sizeof(F_NANO_WALLET_INFO)&0x1F)==0, "Error F_NANO_WALLET_INFO is not byte aligned")
- 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_BLOCK_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 *)

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]
- static const uint8_t **NANO_WALLET_MAGIC** [ ] = {'_', 'n', 'a', 'n', 'o', 'w', 'a', 'l', 'l', 'e', 't', 'f', 'i', 'l', 'e', '_'}
- 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**]
- static const uint8_t **F_NANO_WALLET_INFO_MAGIC** [ ] = {'_', 'n', 'a', 'n', 'o', 'w', 'a', 'l', 'l', 'e', 't', '_', 'n', 'f', 'o', '_'}
- 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**

5.5.1 Detailed Description

This API Integrates Nano Cryptocurrency to low computational devices.

Definition in file **f_nano_crypto_util.h**.

5.5.2 Macro Definition Documentation

5.5.2.1 BIP39_DICTIONARY

```
#define BIP39_DICTIONARY "/spiffs/dictionary.dic"
```

Path to Bip39 dictionary file.

File containing dictionary words must be 16 bytes aligned ! Default name: "dictionary.dic"

Definition at line **132** of file **f_nano_crypto_util.h**.

5.5.2.2 DEFAULT_MAX_FEE

```
#define DEFAULT_MAX_FEE "0.001"
```

Definition at line **547** of file **f_nano_crypto_util.h**.

5.5.2.3 DEST_XRB

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

Definition at line **386** of file **f_nano_crypto_util.h**.

5.5.2.4 EXPORT_KEY_TO_CHAR_SZ

```
#define EXPORT_KEY_TO_CHAR_SZ (size_t)sizeof( NANO_SEED)+1
```

Definition at line **179** of file **f_nano_crypto_util.h**.

5.5.2.5 F_BALANCE_RAW_128

```
#define F_BALANCE_RAW_128  F_NANO_A_RAW_128
```

Definition at line **1391** of file **f_nano_crypto_util.h**.

5.5.2.6 F_BALANCE_RAW_STRING

```
#define F_BALANCE_RAW_STRING  F_NANO_A_RAW_STRING
```

Definition at line **1393** of file **f_nano_crypto_util.h**.

5.5.2.7 F_BALANCE_REAL_STRING

```
#define F_BALANCE_REAL_STRING  F_NANO_A_REAL_STRING
```

Definition at line **1392** of file **f_nano_crypto_util.h**.

5.5.2.8 F_BLOCK_TRANSFER_SIGNABLE_SZ

```
#define F_BLOCK_TRANSFER_SIGNABLE_SZ (size_t)(sizeof(F_BLOCK_TRANSFER)-64-sizeof(uint64_↩
t)-sizeof(uint8_t))
```

Definition at line **241** of file **f_nano_crypto_util.h**.

5.5.2.9 F_BLOCK_TRANSFER_SIZE

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

Definition at line **237** of file **f_nano_crypto_util.h**.

5.5.2.10 F_BRAIN_WALLET_BAD

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

[bad].

Crack within one day

Definition at line **1152** of file **f_nano_crypto_util.h**.

5.5.2.11 F_BRAIN_WALLET_GOOD

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

[good].

Crack within one thousand year

Definition at line **1183** of file **f_nano_crypto_util.h**.

5.5.2.12 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 **1176** of file **f_nano_crypto_util.h**.

5.5.2.13 F_BRAIN_WALLET_NICE

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

[very nice].

Crack withing one hundred thousand year

Definition at line **1195** of file **f_nano_crypto_util.h**.

5.5.2.14 F_BRAIN_WALLET_PERFECT

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

[Perfect!] 3.34x10^53 Years to crack

Definition at line **1201** of file **f_nano_crypto_util.h**.

5.5.2.15 F_BRAIN_WALLET_POOR

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

[poor].

Crack within minutes

Definition at line **1140** of file **f_nano_crypto_util.h**.

5.5.2.16 F_BRAIN_WALLET_STILL_WEAK

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

[still weak].

Crack within one year

Definition at line **1170** of file **f_nano_crypto_util.h**.

5.5.2.17 F_BRAIN_WALLET_VERY_BAD

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

[very bad].

Crack within one hour

Definition at line **1146** of file **f_nano_crypto_util.h**.

5.5.2.18 F_BRAIN_WALLET_VERY_GOOD

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

[very good].

Crack within ten thousand year

Definition at line **1189** of file **f_nano_crypto_util.h**.

5.5.2.19 F_BRAIN_WALLET_VERY_POOR

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

[very poor].

Crack within seconds or less

Definition at line **1134** of file **f_nano_crypto_util.h**.

5.5.2.20 F_BRAIN_WALLET_VERY_WEAK

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

[very weak].

Crack within one week

Definition at line **1158** of file **f_nano_crypto_util.h**.

5.5.2.21 F_BRAIN_WALLET_WEAK

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

[weak].

Crack within one month

Definition at line **1164** of file **f_nano_crypto_util.h**.

5.5.2.22 F_DEFAULT_THRESHOLD

```
#define F_DEFAULT_THRESHOLD (uint64_t) 0xfffffffc000000000
```

Default Nano Proof of Work Threshold.

Definition at line **1304** of file **f_nano_crypto_util.h**.

5.5.2.23 F_DESC_SZ

```
#define F_DESC_SZ (size_t) (160-sizeof(uint32_t))
```

Definition at line **337** of file **f_nano_crypto_util.h**.

5.5.2.24 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 **1291** of file **f_nano_crypto_util.h**.

5.5.2.25 F_MAX_STR_RAW_BALANCE_MAX

```
#define F_MAX_STR_RAW_BALANCE_MAX (size_t)40
```

Definition at line **420** of file **f_nano_crypto_util.h**.

5.5.2.26 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 **1298** of file **f_nano_crypto_util.h**.

5.5.2.27 F_NANO_A_RAW_128

```
#define F_NANO_A_RAW_128 (uint32_t)(1<<2)
```

Definition at line **529** of file **f_nano_crypto_util.h**.

5.5.2.28 F_NANO_A_RAW_STRING

```
#define F_NANO_A_RAW_STRING (uint32_t)(1<<3)
```

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

5.5.2.29 F_NANO_A_REAL_STRING

```
#define F_NANO_A_REAL_STRING (uint32_t)(1<<4)
```

Definition at line **531** of file **f_nano_crypto_util.h**.

5.5.2.30 F_NANO_ADD_A_B

```
#define F_NANO_ADD_A_B (uint32_t)(1<<0)
```

Definition at line **527** of file **f_nano_crypto_util.h**.

5.5.2.31 F_NANO_B_RAW_128

```
#define F_NANO_B_RAW_128 (uint32_t)(1<<5)
```

Definition at line **532** of file **f_nano_crypto_util.h**.

5.5.2.32 F_NANO_B_RAW_STRING

```
#define F_NANO_B_RAW_STRING (uint32_t)(1<<6)
```

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

5.5.2.33 F_NANO_B_REAL_STRING

```
#define F_NANO_B_REAL_STRING (uint32_t)(1<<7)
```

Definition at line **534** of file **f_nano_crypto_util.h**.

5.5.2.34 F_NANO_C_RAW_128

```
#define F_NANO_C_RAW_128 (uint32_t)( F_NANO_B_RAW_128<<16)
```

Definition at line **538** of file **f_nano_crypto_util.h**.

5.5.2.35 F_NANO_C_RAW_STRING

```
#define F_NANO_C_RAW_STRING (uint32_t)( F_NANO_B_RAW_STRING<<16)
```

Definition at line **539** of file **f_nano_crypto_util.h**.

5.5.2.36 F_NANO_C_REAL_STRING

```
#define F_NANO_C_REAL_STRING (uint32_t)( F_NANO_B_REAL_STRING<<16)
```

Definition at line **540** of file **f_nano_crypto_util.h**.

5.5.2.37 F_NANO_COMPARE_EQ

```
#define F_NANO_COMPARE_EQ (uint32_t)(1<<16)
```

Definition at line **542** of file **f_nano_crypto_util.h**.

5.5.2.38 F_NANO_COMPARE_GEQ

```
#define F_NANO_COMPARE_GEQ ( F_NANO_COMPARE_GT| F_NANO_COMPARE_EQ)
```

Definition at line **546** of file **f_nano_crypto_util.h**.

5.5.2.39 F_NANO_COMPARE_GT

```
#define F_NANO_COMPARE_GT (uint32_t)(1<<18)
```

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

5.5.2.40 F_NANO_COMPARE_LEQ

```
#define F_NANO_COMPARE_LEQ ( F_NANO_COMPARE_LT| F_NANO_COMPARE_EQ)
```

Definition at line **544** of file **f_nano_crypto_util.h**.

5.5.2.41 F_NANO_COMPARE_LT

```
#define F_NANO_COMPARE_LT (uint32_t)(1<<17)
```

Definition at line **543** of file **f_nano_crypto_util.h**.

5.5.2.42 F_NANO_DESC_SZ

```
#define F_NANO_DESC_SZ (size_t)78
```

Definition at line **453** of file **f_nano_crypto_util.h**.

5.5.2.43 F_NANO_EMPTY_BALANCE

```
#define F_NANO_EMPTY_BALANCE "0.0"
```

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

5.5.2.44 F_NANO_FILE_DESC

```
#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 !!!"
```

Definition at line **336** of file **f_nano_crypto_util.h**.

5.5.2.45 F_NANO_RES_RAW_128

```
#define F_NANO_RES_RAW_128 (uint32_t)(1<<8)
```

Definition at line **535** of file **f_nano_crypto_util.h**.

5.5.2.46 F_NANO_RES_RAW_STRING

```
#define F_NANO_RES_RAW_STRING (uint32_t)(1<<9)
```

Definition at line **536** of file **f_nano_crypto_util.h**.

5.5.2.47 F_NANO_RES_REAL_STRING

```
#define F_NANO_RES_REAL_STRING (uint32_t)(1<<10)
```

Definition at line **537** of file **f_nano_crypto_util.h**.

5.5.2.48 F_NANO_SUB_A_B

```
#define F_NANO_SUB_A_B (uint32_t)(1<<1)
```

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

5.5.2.49 F_NANO_WALLET_INFO_DESC

```
#define F_NANO_WALLET_INFO_DESC "Nano file descriptor used for fast custom access.  BUY BITC↩
OIN AND NANO."
```

Definition at line **449** of file **f_nano_crypto_util.h**.

5.5.2.50 F_NANO_WALLET_INFO_VERSION

```
#define F_NANO_WALLET_INFO_VERSION (uint16_t)((1<<8)|1)
```

Definition at line **450** of file **f_nano_crypto_util.h**.

5.5.2.51 F_P2POW_BLOCK_TRANSFER_SIZE

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

Definition at line **238** of file **f_nano_crypto_util.h**.

5.5.2.52 F_RAW_STR_MAX_SZ

```
#define F_RAW_STR_MAX_SZ (size_t)41
```

Definition at line **419** of file **f_nano_crypto_util.h**.

5.5.2.53 F_RAW_TO_STR_STRING

```
#define F_RAW_TO_STR_STRING (int)2
```

Definition at line **418** of file **f_nano_crypto_util.h**.

5.5.2.54 F_RAW_TO_STR_UINT128

```
#define F_RAW_TO_STR_UINT128 (int)1
```

Definition at line **417** of file **f_nano_crypto_util.h**.

5.5.2.55 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 **1284** of file **f_nano_crypto_util.h**.

5.5.2.56 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 **1263** of file **f_nano_crypto_util.h**.

5.5.2.57 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 **1270** of file **f_nano_crypto_util.h**.

5.5.2.58 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 **1277** of file **f_nano_crypto_util.h**.

5.5.2.59 F_SIGNATURE_RAW

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

Signature is raw data.

See also

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

Definition at line **1249** of file **f_nano_crypto_util.h**.

5.5.2.60 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 **1256** of file **f_nano_crypto_util.h**.

5.5.2.61 F_STREAM_DATA_FILE_VERSION

```
#define F_STREAM_DATA_FILE_VERSION (uint32_t)((1<<16)|0)
```

Definition at line **309** of file **f_nano_crypto_util.h**.

5.5.2.62 F_VALUE_SEND_RECEIVE_RAW_128

```
#define F_VALUE_SEND_RECEIVE_RAW_128  F_NANO_B_RAW_128
```

Definition at line **1394** of file **f_nano_crypto_util.h**.

5.5.2.63 F_VALUE_SEND_RECEIVE_RAW_STRING

```
#define F_VALUE_SEND_RECEIVE_RAW_STRING  F_NANO_B_RAW_STRING
```

Definition at line **1396** of file **f_nano_crypto_util.h**.

5.5.2.64 F_VALUE_SEND_RECEIVE_REAL_STRING

```
#define F_VALUE_SEND_RECEIVE_REAL_STRING  F_NANO_B_REAL_STRING
```

Definition at line **1395** of file **f_nano_crypto_util.h**.

5.5.2.65 F_VALUE_TO_RECEIVE

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

Definition at line **1398** of file **f_nano_crypto_util.h**.

5.5.2.66 F_VALUE_TO_SEND

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

Definition at line **1397** of file **f_nano_crypto_util.h**.

5.5.2.67 F_VERIFY_SIG_ASCII_HEX

```
#define F_VERIFY_SIG_ASCII_HEX (uint32_t)4
```

Public key is a hex ASCII encoded string.

See also

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

Definition at line **1356** of file **f_nano_crypto_util.h**.

5.5.2.68 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 **1342** of file **f_nano_crypto_util.h**.

5.5.2.69 F_VERIFY_SIG_RAW_HEX

```
#define F_VERIFY_SIG_RAW_HEX (uint32_t)2
```

Public key raw 32 bytes data.

See also

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

Definition at line **1349** of file **f_nano_crypto_util.h**.

5.5.2.70 MAX_STR_NANO_CHAR

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

Defines a max size of Nano char (70 bytes)

Definition at line **107** of file **f_nano_crypto_util.h**.

5.5.2.71 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 **140** of file **f_nano_crypto_util.h**.

5.5.2.72 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 **158** of file **f_nano_crypto_util.h**.

5.5.2.73 NANO_PASSWD_MAX_LEN

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

Password max length.

Definition at line **146** of file **f_nano_crypto_util.h**.

5.5.2.74 NANO_PREFIX

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

Nano prefix.

Definition at line **119** of file **f_nano_crypto_util.h**.

5.5.2.75 PARSE_JSON_READ_SEED_GENERIC

```
#define PARSE_JSON_READ_SEED_GENERIC (int)16
```

Definition at line **308** of file **f_nano_crypto_util.h**.

5.5.2.76 PUB_KEY_EXTENDED_MAX_LEN

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

Max size of public key (extended)

Definition at line **113** of file **f_nano_crypto_util.h**.

5.5.2.77 READ_SEED_FROM_FILE

```
#define READ_SEED_FROM_FILE (int)2
```

Definition at line **305** of file **f_nano_crypto_util.h**.

5.5.2.78 READ_SEED_FROM_STREAM

```
#define READ_SEED_FROM_STREAM (int)1
```

Definition at line **304** of file **f_nano_crypto_util.h**.

5.5.2.79 REP_XRB

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

Representative XRB flag.

Destination XRB flag.

Sender XRB flag.

5.5.2.80 SENDER_XRB

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

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

5.5.2.81 STR_NANO_SZ

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

String size of Nano encoded Base32 including NULL char.

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

5.5.2.82 WRITE_SEED_TO_FILE

```
#define WRITE_SEED_TO_FILE (int)8
```

Definition at line **307** of file **f_nano_crypto_util.h**.

5.5.2.83 WRITE_SEED_TO_STREAM

```
#define WRITE_SEED_TO_STREAM (int)4
```

Definition at line **306** of file **f_nano_crypto_util.h**.

5.5.2.84 XRB_PREFIX

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

XRB (old Raiblocks) prefix.

Definition at line **125** of file **f_nano_crypto_util.h**.

5.5.3 Typedef Documentation

5.5.3.1 F_FILE_INFO_ERR

F_FILE_INFO_ERR

Typedef Error enumerator for info file functions.

5.5.3.2 F_NANO_CREATE_BLOCK_DYN_ERR

```
typedef enum f_nano_create_block_dyn_err_t F_NANO_CREATE_BLOCK_DYN_ERR
```

5.5.3.3 f_nano_err

f_nano_err

Error function enumerator.

See also

f_nano_err_t (p. **??**)

5.5.3.4 F_NANO_P2POW_BLOCK_DYN_ERR

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

5.5.3.5 F_TOKEN

typedef uint8_t F_TOKEN[16]

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

5.5.3.6 f_uint128_t

f_uint128_t

128 bit big number of Nano balance

Definition at line **176** of file **f_nano_crypto_util.h**.

5.5.3.7 f_write_seed_err

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

5.5.3.8 NANO_PRIVATE_KEY

NANO_PRIVATE_KEY

Size of Nano Private Key.

Definition at line **186** of file **f_nano_crypto_util.h**.

5.5.3.9 NANO_PRIVATE_KEY_EXTENDED

`NANO_PRIVATE_KEY_EXTENDED`

Size of Nano Private Key extended.

Definition at line **192** of file **f_nano_crypto_util.h**.

5.5.3.10 NANO_PUBLIC_KEY

`NANO_PUBLIC_KEY`

Size of Nano Public Key.

Definition at line **198** of file **f_nano_crypto_util.h**.

5.5.3.11 NANO_PUBLIC_KEY_EXTENDED

`NANO_PUBLIC_KEY_EXTENDED`

Size of Public Key Extended.

Definition at line **204** of file **f_nano_crypto_util.h**.

5.5.3.12 NANO_SEED

`NANO_SEED`

Size of Nano SEED.

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

5.5.4 Enumeration Type Documentation

5.5.4.1 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_NOT_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_ENCRYPTED_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 **492** of file **f_nano_crypto_util.h**.

5.5.4.2 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 **551** of file **f_nano_crypto_util.h**.

5.5.4.3 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 **251** of file **f_nano_crypto_util.h**.

5.5.4.4 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 **574** of file **f_nano_crypto_util.h**.

5.5.4.5 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 **388** of file **f_nano_crypto_util.h**.

5.5.5 Function Documentation

5.5.5.1 __attribute__()

```
struct f_block_transfer_t __attribute__ (
            (packed) )
```

5.5.5.2 _Static_assert() [1/4]

```
_Static_assert (
            (sizeof(F_NANO_CRYPTOWALLET)&0x1F) == 0,
            "Error 1" )
```

5.5.5.3 _Static_assert() [2/4]

```
_Static_assert (
             (sizeof(F_ENCRYPTED_BLOCK)&0x1F)  = =0,
             "Error 2"  )
```

5.5.5.4 _Static_assert() [3/4]

```
_Static_assert (
             (sizeof(F_NANO_WALLET_INFO_BODY)&0x1F)  = =0,
             "Error F_NANO_WALLET_INFO_BODY is not byte aligned"  )
```

5.5.5.5 _Static_assert() [4/4]

```
_Static_assert (
             (sizeof(F_NANO_WALLET_INFO)&0x1F)  = =0,
             "Error F_NANO_WALLET_INFO is not byte aligned"  )
```

5.5.5.6 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. **??**)

5.5.5.7 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

5.5.5.8 f_extract_seed_from_brainwallet()

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

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

Parameters

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

Parameters

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

Return values

0	If success, otherwise error.

See also

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

5.5.5.9 f_generate_nano_seed()

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

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

Parameters

out	*seed*	SEED generated in system PRNG or TRNG

Parameters

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

Return values

0	On Success, otherwise Error

5.5.5.10 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. **??**)

5.5.5.11 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. **??**)

5.5.5.12 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

5.5.5.13 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.

5.5.5.14 f_nano_add_sub()

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

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

Parameters

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

Return values

NANO_ERR_OK	If Success, otherwise f_nano_err_t enum type error

See also

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

5.5.5.15 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. **??**)

5.5.5.16 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

5.5.5.17 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

5.5.5.18 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

5.5.5.19 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

5.5.5.20 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

5.5.5.21 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

5.5.5.22 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

5.5.5.23 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

5.5.5.24 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. **??**)

5.5.5.25 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. **??**)

5.5.5.26 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. **??**)

5.5.5.27 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

5.5.5.28 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. **??**)

5.5.5.29 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

5.5.5.30 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)

Parameters

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

5.5.5.31 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*

Parameters

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. **??**)

5.5.5.32 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. **??**)

5.5.5.33 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

5.5.5.34 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_DICTIONARY* is used

See also

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

5.5.5.35 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

5.5.5.36 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. **??**)

5.5.5.37 f_verify_signed_block()

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

5.5.5.38 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_VERIFY_SIG_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. **??**)

5.5.5.39 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. **??**)

5.5.5.40 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()

5.5.5.41 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. **??**)

5.5.5.42 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

5.5.5.43 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

5.5.5.44 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

5.5.5.45 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. **??**)

5.5.5.46 nano_create_block_dynamic()

```
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  )
```

5.5.5.47 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  )
```

5.5.5.48 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. **??**)

5.5.5.49 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

5.5.5.50 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

5.5.5.51 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

5.5.6 Variable Documentation

5.5.6.1 account

```
uint8_t account[32]
```

Account in raw binary data.

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

5.5.6.2 balance

```
f_uint128_t balance
```

Big number 128 bit raw balance.

See also

f_uint128_t (p. **??**)

Definition at line **216** of file **f_nano_crypto_util.h**.

5.5.6.3 body

```
F_NANO_WALLET_INFO_BODY body
```

Body of the file info.

Definition at line **463** of file **f_nano_crypto_util.h**.

5.5.6.4 desc

```
char desc[ F_NANO_DESC_SZ]
```

Description.

Definition at line **457** of file **f_nano_crypto_util.h**.

5.5.6.5 description

```
uint8_t description[ F_DESC_SZ]
```

File description.

Definition at line **341** of file **f_nano_crypto_util.h**.

5.5.6.6 F_NANO_WALLET_INFO_MAGIC

```
const uint8_t F_NANO_WALLET_INFO_MAGIC[] = {'_', 'n', 'a', 'n', 'o', 'w', 'a', 'l', 'l', 'e',
't', '_', 'n', 'f', 'o', '_'}  [static]
```

Definition at line **451** of file **f_nano_crypto_util.h**.

5.5.6.7 file_info_integrity

```
uint8_t file_info_integrity[32]
```

File info integrity of the body block.

Definition at line **461** of file **f_nano_crypto_util.h**.

5.5.6.8 hash_sk_unencrypted

```
uint8_t hash_sk_unencrypted[32]
```

hash of Nano SEED when unencrypted

Definition at line **212** of file **f_nano_crypto_util.h**.

5.5.6.9 header

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

Header magic.

Definition at line **453** of file **f_nano_crypto_util.h**.

5.5.6.10 iv

```
uint8_t iv
```

Initial sub vector.

Initial vector of first encryption layer.

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

5.5.6.11 last_used_wallet_number

```
uint32_t last_used_wallet_number
```

Last used wallet number.

Definition at line **370** of file **f_nano_crypto_util.h**.

5.5.6.12 link

```
uint8_t link[32]
```

link or destination account

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

5.5.6.13 max_fee

```
char max_fee[ F_RAW_STR_MAX_SZ]
```

Custom preferred max fee of Proof of Work.

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

5.5.6.14 nano_hdr

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

Header of the file.

Definition at line **337** of file **f_nano_crypto_util.h**.

5.5.6.15 NANO_WALLET_MAGIC

```
const uint8_t NANO_WALLET_MAGIC[ ] = {'_', 'n', 'a', 'n', 'o', 'w', 'a', 'l', 'l', 'e', 't',
'f', 'i', 'l', 'e', '_'}  [static]
```

Definition at line **335** of file **f_nano_crypto_util.h**.

5.5.6.16 nanoseed_hash

```
uint8_t nanoseed_hash[32]
```

Nano SEED hash file.

Definition at line **459** of file **f_nano_crypto_util.h**.

5.5.6.17 preamble

```
uint8_t preamble[32]
```

Block preamble.

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

5.5.6.18 prefixes

```
uint8_t prefixes
```

Internal use for this API.

Definition at line **222** of file **f_nano_crypto_util.h**.

5.5.6.19 previous

```
uint8_t previous[32]
```

Previous block.

Definition at line **210** of file **f_nano_crypto_util.h**.

5.5.6.20 representative

```
uint8_t representative[32]
```

Representative for current account.

Definition at line **212** of file **f_nano_crypto_util.h**.

5.5.6.21 reserved

```
uint8_t reserved
```

Reserved (not used)

Reserved.

Definition at line **210** of file **f_nano_crypto_util.h**.

5.5.6.22 salt

```
uint8_t salt[32]
```

Salt of the first encryption layer.

Definition at line **343** of file **f_nano_crypto_util.h**.

5.5.6.23 seed_block

```
F_ENCRYPTED_BLOCK seed_block
```

Second encrypted block for Nano SEED.

Definition at line **347** of file **f_nano_crypto_util.h**.

5.5.6.24 signature

```
uint8_t signature[64]
```

Signature of the block.

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

5.5.6.25 sk_encrypted

```
uint8_t sk_encrypted[32]
```

Secret.

SEED encrypted (second layer)

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

5.5.6.26 sub_salt

```
uint8_t sub_salt[32]
```

Salt of the sub block to be stored.

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

5.5.6.27 ver

```
uint32_t ver
```

Version of the file.

Definition at line **339** of file **f_nano_crypto_util.h**.

5.5.6.28 version

```
uint16_t version
```

Version.

Definition at line **455** of file **f_nano_crypto_util.h**.

5.5.6.29 wallet_prefix

```
uint8_t wallet_prefix
```

Wallet prefix: 0 for NANO; 1 for XRB.

Definition at line **368** of file **f_nano_crypto_util.h**.

5.5.6.30 wallet_representative

```
char wallet_representative[ MAX_STR_NANO_CHAR]
```

Wallet representative.

Definition at line **372** of file **f_nano_crypto_util.h**.

5.5.6.31 work

```
uint64_t work
```

Internal use for this API.

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

5.6 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 #include <stdint.h>
00008 #include <f_util.h>
00009 #include <f_bitcoin.h>
00010 #include "esp_system.h"
00011 #include "sodium/crypto_generichash.h"
00012 #include "sodium/crypto_sign.h"
00013 #include "sodium.h"
00014 #include "sodium/private/curve25519_ref10.h"
00015
00016
00099 #ifdef __cplusplus
00100 extern "C" {
00101 #endif
00102
00107 #define MAX_STR_NANO_CHAR (size_t)70 //5+56+8+1
00108
00113 #define PUB_KEY_EXTENDED_MAX_LEN (size_t)40
00114
00119 #define NANO_PREFIX "nano_"
00120
00125 #define XRB_PREFIX "xrb_"
00126
00132 #define BIP39_DICTIONARY "/spiffs/dictionary.dic"
00133
00140 #define NANO_ENCRYPTED_SEED_FILE "/spiffs/secure/nano.nse"
00141
00146 #define NANO_PASSWD_MAX_LEN (size_t)80
00147
00152 #define STR_NANO_SZ (size_t)66// 65+1 Null included
00153
00158 #define NANO_FILE_WALLETS_INFO "/spiffs/secure/walletsinfo.i"
00159
00164 typedef uint8_t F_TOKEN[16];
00165
00170 typedef uint8_t NANO_SEED[crypto_sign_SEEDBYTES];
00171
00176 typedef uint8_t f_uint128_t[16];
00177
00178 #ifndef F_DOC_SKIP
00179  #define EXPORT_KEY_TO_CHAR_SZ (size_t)sizeof(NANO_SEED)+1
00180 #endif
00181
00186 typedef uint8_t NANO_PRIVATE_KEY[sizeof(NANO_SEED)];
00187
00192 typedef uint8_t NANO_PRIVATE_KEY_EXTENDED[crypto_sign_ed25519_SECRETKEYBYTES];
00193
00198 typedef uint8_t NANO_PUBLIC_KEY[crypto_sign_ed25519_PUBLICKEYBYTES];
00199
00204 typedef uint8_t NANO_PUBLIC_KEY_EXTENDED[PUB_KEY_EXTENDED_MAX_LEN];
00205
00214 typedef struct f_block_transfer_t {
00216    uint8_t preamble[32];
00218    uint8_t account[32];
00220    uint8_t previous[32];
00222    uint8_t representative[32];
00226    f_uint128_t balance;
00228    uint8_t link[32];
00230    uint8_t signature[64];
00232    uint8_t prefixes;
00234    uint64_t work;
00235 } __attribute__((packed)) F_BLOCK_TRANSFER;
00236
00237 #define F_BLOCK_TRANSFER_SIZE (size_t)sizeof(F_BLOCK_TRANSFER)
00238 #define F_P2POW_BLOCK_TRANSFER_SIZE 2*F_BLOCK_TRANSFER_SIZE
00239
00240 #ifndef F_DOC_SKIP
00241  #define F_BLOCK_TRANSFER_SIGNABLE_SZ
       (size_t)(sizeof(F_BLOCK_TRANSFER)-64-sizeof(uint64_t)-sizeof(uint8_t))
00242 #endif
00243
00251 typedef enum f_nano_err_t {
00253    NANO_ERR_OK=0,
00255    NANO_ERR_CANT_PARSE_BN_STR=5151,
00257    NANO_ERR_MALLOC,
00259    NANO_ERR_CANT_PARSE_FACTOR,
00261    NANO_ERR_MPI_MULT,
00263    NANO_ERR_CANT_PARSE_TO_BLK_TRANSFER,
00265    NANO_ERR_EMPTY_STR,
```

```
00267    NANO_ERR_CANT_PARSE_VALUE,
00269    NANO_ERR_PARSE_MPI_TO_STR,
00271    NANO_ERR_CANT_COMPLETE_NULL_CHAR,
00273    NANO_ERR_CANT_PARSE_TO_MPI,
00275    NANO_ERR_INSUFICIENT_FUNDS,
00277    NANO_ERR_SUB_MPI,
00279    NANO_ERR_ADD_MPI,
00281    NANO_ERR_NO_SENSE_VALUE_TO_SEND_NEGATIVE,
00283    NANO_ERR_NO_SENSE_VALUE_TO_SEND_ZERO,
00285    NANO_ERR_NO_SENSE_BALANCE_NEGATIVE,
00287    NANO_ERR_VAL_A_INVALID_MODE,
00289    NANO_ERR_CANT_PARSE_TO_TEMP_UINT128_T,
00291    NANO_ERR_VAL_B_INVALID_MODE,
00293    NANO_ERR_CANT_PARSE_RAW_A_TO_MPI,
00295    NANO_ERR_CANT_PARSE_RAW_B_TO_MPI,
00297    NANO_ERR_UNKNOWN_ADD_SUB_MODE,
00299    NANO_ERR_INVALID_RES_OUTPUT
00300 } f_nano_err;
00301
00302 #ifndef F_DOC_SKIP
00303
00304  #define READ_SEED_FROM_STREAM (int)1
00305  #define READ_SEED_FROM_FILE (int)2
00306  #define WRITE_SEED_TO_STREAM (int)4
00307  #define WRITE_SEED_TO_FILE (int)8
00308  #define PARSE_JSON_READ_SEED_GENERIC (int)16
00309  #define F_STREAM_DATA_FILE_VERSION (uint32_t)((1<<16)|0)
00310
00311 #endif
00312
00320 typedef struct f_nano_encrypted_wallet_t {
00322    uint8_t sub_salt[32];
00324    uint8_t iv[16];
00326    uint8_t reserved[16];
00328    uint8_t hash_sk_unencrypted[32];
00330    uint8_t sk_encrypted[32];
00331 } __attribute__ ((packed)) F_ENCRYPTED_BLOCK;
00332
00333 #ifndef F_DOC_SKIP
00334
00335  static const uint8_t NANO_WALLET_MAGIC[] = {'_', 'n', 'a', 'n', 'o', 'w', 'a', 'l', 'l', 'e', 't', 'f',
      'i', 'l', 'e', '_'};
00336  #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 !!!"
00337  #define F_DESC_SZ (size_t) (160-sizeof(uint32_t))
00338
00339 #endif
00340
00348 typedef struct f_nano_crypto_wallet_t {
00350    uint8_t nano_hdr[sizeof(NANO_WALLET_MAGIC)];
00352    uint32_t ver;
00354    uint8_t description[F_DESC_SZ];
00356    uint8_t salt[32];
00358    uint8_t iv[16];
00360    F_ENCRYPTED_BLOCK seed_block;
00361 } __attribute__ ((packed)) F_NANO_CRYPTOWALLET;
00362
00363 #ifndef F_DOC_SKIP
00364
00365 _Static_assert((sizeof(F_NANO_CRYPTOWALLET)&0x1F)==0, "Error 1");
00366 _Static_assert((sizeof(F_ENCRYPTED_BLOCK)&0x1F)==0, "Error 2");
00367
00368 #endif
00369
00374 #define REP_XRB (uint8_t)0x4
00375
00380 #define SENDER_XRB (uint8_t)0x02
00381
00386 #define DEST_XRB (uint8_t)0x01
00387
00388 typedef enum f_write_seed_err_t {
00390    WRITE_ERR_OK=0,
00392    WRITE_ERR_NULL_PASSWORD=7180,
00394    WRITE_ERR_EMPTY_STRING,
00396    WRITE_ERR_MALLOC,
00398    WRITE_ERR_ENCRYPT_PRIV_KEY,
00400    WRITE_ERR_GEN_SUB_PRIV_KEY,
00402    WRITE_ERR_GEN_MAIN_PRIV_KEY,
00404    WRITE_ERR_ENCRYPT_SUB_BLOCK,
00406    WRITE_ERR_UNKNOWN_OPTION,
00408    WRITE_ERR_FILE_ALREDY_EXISTS,
00410    WRITE_ERR_CREATING_FILE,
00412    WRITE_ERR_WRITING_FILE
00413 } f_write_seed_err;
00414
00415 #ifndef F_DOC_SKIP
00416
```

```
00417  #define F_RAW_TO_STR_UINT128 (int)1
00418  #define F_RAW_TO_STR_STRING (int)2
00419  #define F_RAW_STR_MAX_SZ (size_t)41 // 39 + '\0' + '.' -> 39 = log10(2^128)
00420  #define F_MAX_STR_RAW_BALANCE_MAX (size_t)40 //39+'\0'
00421  #define F_NANO_EMPTY_BALANCE "0.0"
00422
00423 #endif
00424
00432 typedef struct f_nano_wallet_info_bdy_t {
00434    uint8_t wallet_prefix; // 0 for NANO; 1 for XRB
00436    uint32_t last_used_wallet_number;
00438    char wallet_representative[MAX_STR_NANO_CHAR];
00440    char max_fee[F_RAW_STR_MAX_SZ];
00442    uint8_t reserved[44];
00443 } __attribute__((packed)) F_NANO_WALLET_INFO_BODY;
00444
00445 #ifndef F_DOC_SKIP
00446
00447  _Static_assert((sizeof(F_NANO_WALLET_INFO_BODY)&0x1F)==0, "Error F_NANO_WALLET_INFO_BODY is not byte
       aligned");
00448
00449  #define F_NANO_WALLET_INFO_DESC "Nano file descriptor used for fast custom access. BUY BITCOIN AND NANO."
00450  #define F_NANO_WALLET_INFO_VERSION (uint16_t)((1<<8)|1)
00451  static const uint8_t F_NANO_WALLET_INFO_MAGIC[] = {'_', 'n', 'a', 'n', 'o', 'w', 'a', 'l', 'l', 'e', 't',
      '_', 'n', 'f', 'o', '_'};
00452
00453  #define F_NANO_DESC_SZ (size_t)78
00454
00455 #endif
00456
00464 typedef struct f_nano_wallet_info_t {
00466    uint8_t header[sizeof(F_NANO_WALLET_INFO_MAGIC)];
00468    uint16_t version;
00470    char desc[F_NANO_DESC_SZ];
00472    uint8_t nanoseed_hash[32];
00474    uint8_t file_info_integrity[32];
00476    F_NANO_WALLET_INFO_BODY body;
00477 } __attribute__((packed)) F_NANO_WALLET_INFO;
00478
00479 #ifndef F_DOC_SKIP
00480
00481  _Static_assert((sizeof(F_NANO_WALLET_INFO)&0x1F)==0, "Error F_NANO_WALLET_INFO is not byte aligned");
00482
00483 #endif
00484
00492 typedef enum f_file_info_err_t {
00494    F_FILE_INFO_ERR_OK=0,
00496    F_FILE_INFO_ERR_CANT_OPEN_INFO_FILE=7001,
00498    F_FILE_INFO_ERR_NANO_SEED_ENCRYPTED_FILE_NOT_FOUND,
00500    F_FILE_INFO_ERR_CANT_DELETE_NANO_INFO_FILE,
00502    F_FILE_INFO_ERR_MALLOC,
00504    F_FILE_INFO_ERR_CANT_READ_NANO_SEED_ENCRYPTED_FILE,
00506    F_FILE_INFO_ERR_CANT_READ_INFO_FILE,
00508    F_FILE_INFO_INVALID_HEADER_FILE,
00510    F_FILE_INFO_ERR_INVALID_SHA256_INFO_FILE,
00512    F_FILE_INFO_ERR_NANO_SEED_HASH_FAIL,
00514    F_FILE_INFO_ERR_NANO_INVALID_REPRESENTATIVE,
00516    F_FILE_INFO_ERR_NANO_INVALID_MAX_FEE_VALUE,
00518    F_FILE_INFO_ERR_OPEN_FOR_WRITE_INFO,
00520    F_FILE_INFO_ERR_EXISTING_FILE,
00522    F_FILE_INFO_ERR_CANT_WRITE_FILE_INFO
00523 } F_FILE_INFO_ERR;
00524
00525 #ifndef F_DOC_SKIP
00526
00527  #define F_NANO_ADD_A_B (uint32_t)(1<<0)
00528  #define F_NANO_SUB_A_B (uint32_t)(1<<1)
00529  #define F_NANO_A_RAW_128 (uint32_t)(1<<2)
00530  #define F_NANO_A_RAW_STRING (uint32_t)(1<<3)
00531  #define F_NANO_A_REAL_STRING (uint32_t)(1<<4)
00532  #define F_NANO_B_RAW_128 (uint32_t)(1<<5)
00533  #define F_NANO_B_RAW_STRING (uint32_t)(1<<6)
00534  #define F_NANO_B_REAL_STRING (uint32_t)(1<<7)
00535  #define F_NANO_RES_RAW_128 (uint32_t)(1<<8)
00536  #define F_NANO_RES_RAW_STRING (uint32_t)(1<<9)
00537  #define F_NANO_RES_REAL_STRING (uint32_t)(1<<10)
00538  #define F_NANO_C_RAW_128 (uint32_t)(F_NANO_B_RAW_128<<16)
00539  #define F_NANO_C_RAW_STRING (uint32_t)(F_NANO_B_RAW_STRING<<16)
00540  #define F_NANO_C_REAL_STRING (uint32_t)(F_NANO_B_REAL_STRING<<16)
00541
00542  #define F_NANO_COMPARE_EQ (uint32_t)(1<<16) //Equal
00543  #define F_NANO_COMPARE_LT (uint32_t)(1<<17) // Lesser than
00544  #define F_NANO_COMPARE_LEQ (F_NANO_COMPARE_LT|F_NANO_COMPARE_EQ) // Less or equal
00545  #define F_NANO_COMPARE_GT (uint32_t)(1<<18) // Greater
00546  #define F_NANO_COMPARE_GEQ (F_NANO_COMPARE_GT|F_NANO_COMPARE_EQ) // Greater or equal
00547  #define DEFAULT_MAX_FEE "0.001"
00548
```

```
00549 #endif
00550
00551 typedef enum f_nano_create_block_dyn_err_t {
00552    NANO_CREATE_BLK_DYN_OK = 0,
00553    NANO_CREATE_BLK_DYN_BLOCK_NULL = 8000,
00554    NANO_CREATE_BLK_DYN_ACCOUNT_NULL,
00555 //   NANO_CREATE_BLK_DYN_PREV_NULL,
00556    NANO_CREATE_BLK_DYN_COMPARE_BALANCE,
00557    NANO_CREATE_BLK_DYN_GENESIS_WITH_NON_EMPTY_BALANCE,
00558    NANO_CREATE_BLK_DYN_CANT_SEND_IN_GENESIS_BLOCK,
00559    NANO_CREATE_BLK_DYN_REP_NULL,
00560    NANO_CREATE_BLK_DYN_BALANCE_NULL,
00561    NANO_CREATE_BLK_DYN_SEND_RECEIVE_NULL,
00562    NANO_CREATE_BLK_DYN_LINK_NULL,
00563    NANO_CREATE_BLK_DYN_BUF_MALLOC,
00564    NANO_CREATE_BLK_DYN_MALLOC,
00565    NANO_CREATE_BLK_DYN_WRONG_PREVIOUS_SZ,
00566    NANO_CREATE_BLK_DYN_WRONG_PREVIOUS_STR_SZ,
00567    NANO_CREATE_BLK_DYN_PARSE_STR_HEX_ERR,
00568    NANO_CREATE_BLK_DYN_FORBIDDEN_AMOUNT_TYPE,
00569    NANO_CREATE_BLK_DYN_COMPARE,
00570    NANO_CREATE_BLK_DYN_EMPTY_VAL_TO_SEND_OR_REC,
00571    NANO_CREATE_BLK_DYN_INVALID_DIRECTION_OPTION
00572 } F_NANO_CREATE_BLOCK_DYN_ERR;
00573
00574 typedef enum f_nano_p2pow_block_dyn_err_t {
00575    NANO_P2POW_CREATE_BLOCK_OK = 0,
00576    NANO_P2POW_CREATE_BLOCK_INVALID_USER_BLOCK = 8400,
00577    NANO_P2POW_CREATE_BLOCK_MALLOC,
00578    NANO_P2POW_CREATE_BLOCK_NULL,
00579    NANO_P2POW_CREATE_OUTPUT,
00580    NANO_P2POW_CREATE_OUTPUT_MALLOC
00581 } F_NANO_P2POW_BLOCK_DYN_ERR;
00582
00594 double to_multiplier(uint64_t, uint64_t);
00595
00607 uint64_t from_multiplier(double, uint64_t);
00608
00618 void f_set_dictionary_path(const char *);
00619
00627 char *f_get_dictionary_path(void);
00628
00641 int f_generate_token(F_TOKEN, void *, size_t, const char *);
00642
00655 int f_verify_token(F_TOKEN, void *, size_t, const char *);
00656
00679 int f_cloud_crypto_wallet_nano_create_seed(size_t, char *, char *);
00680
00693 int f_generate_nano_seed(NANO_SEED, uint32_t);
00694
00709 int pk_to_wallet(char *, char *, NANO_PUBLIC_KEY_EXTENDED);
00710
00728 int f_seed_to_nano_wallet(NANO_PRIVATE_KEY, NANO_PUBLIC_KEY, NANO_SEED, uint32_t);
00729
00739 int f_nano_is_valid_block(F_BLOCK_TRANSFER *);
00740
00753 int f_nano_block_to_json(char *, size_t *, size_t, F_BLOCK_TRANSFER *);
00754
00765 int f_nano_get_block_hash(uint8_t *, F_BLOCK_TRANSFER *);
00766
00778 int f_nano_get_p2pow_block_hash(uint8_t *, uint8_t *, F_BLOCK_TRANSFER *);
00779
00792 int f_nano_p2pow_to_JSON(char *, size_t *, size_t, F_BLOCK_TRANSFER *);
00793
00803 char *f_nano_key_to_str(char *, unsigned char *);
00804
00823 int f_nano_seed_to_bip39(char *, size_t, size_t *, NANO_SEED, char *);
00824
00839 int f_bip39_to_nano_seed(uint8_t *, char *, char *);
00840
00862 int f_parse_nano_seed_and_bip39_to_JSON(char *, size_t, size_t *, void *, int, const char *);
00863
00881 int f_read_seed(uint8_t *, const char *, void *, int, int);
00882
00897 int f_nano_raw_to_string(char *, size_t *, size_t, void *, int);
00898
00907 int f_nano_valid_nano_str_value(const char *);
00908
00916 int valid_nano_wallet(const char *);
00917
00927 int nano_base_32_2_hex(uint8_t *, char *);
00928
00943 int f_nano_transaction_to_JSON(char *, size_t, size_t *, NANO_PRIVATE_KEY_EXTENDED, F_BLOCK_TRANSFER *);
00944
00952 int valid_raw_balance(const char *);
00953
00961 int is_null_hash(uint8_t *);
```

```
00962
00974 int is_nano_prefix(const char *, const char *);
00975
00984 F_FILE_INFO_ERR f_get_nano_file_info(F_NANO_WALLET_INFO *);
00985
00995 F_FILE_INFO_ERR f_set_nano_file_info(F_NANO_WALLET_INFO *, int);
00996
01018 f_nano_err f_nano_value_compare_value(void *, void *, uint32_t *);
01019
01040 f_nano_err f_nano_verify_nano_funds(void *, void *, void *, uint32_t);
01041
01051 f_nano_err f_nano_parse_raw_str_to_raw128_t(uint8_t *, const char *);
01052
01062 f_nano_err f_nano_parse_real_str_to_raw128_t(uint8_t *, const char *);
01063
01086 f_nano_err f_nano_add_sub(void *, void *, void *, uint32_t);
01087
01098 int f_nano_sign_block(F_BLOCK_TRANSFER *, F_BLOCK_TRANSFER *, NANO_PRIVATE_KEY_EXTENDED);
01099
01113 f_write_seed_err f_write_seed(void *, int, uint8_t *, char *);
01114
01127 f_nano_err f_nano_balance_to_str(char *, size_t, size_t *, f_uint128_t);
01128
01129
01134 #define F_BRAIN_WALLET_VERY_POOR (uint32_t)0
01135
01140 #define F_BRAIN_WALLET_POOR (uint32_t)1
01141
01146 #define F_BRAIN_WALLET_VERY_BAD (uint32_t)2
01147
01152 #define F_BRAIN_WALLET_BAD (uint32_t)3
01153
01158 #define F_BRAIN_WALLET_VERY_WEAK (uint32_t)4
01159
01164 #define F_BRAIN_WALLET_WEAK (uint32_t)5
01165
01170 #define F_BRAIN_WALLET_STILL_WEAK (uint32_t)6
01171
01176 #define F_BRAIN_WALLET_MAYBE_GOOD (uint32_t)7
01177
01178
01183 #define F_BRAIN_WALLET_GOOD (uint32_t)8
01184
01189 #define F_BRAIN_WALLET_VERY_GOOD (uint32_t)9
01190
01195 #define F_BRAIN_WALLET_NICE (uint32_t)10
01196
01201 #define F_BRAIN_WALLET_PERFECT (uint32_t)11
01202
01229 int f_extract_seed_from_brainwallet(uint8_t *, char **, uint32_t, const char *, const char *);
01230
01242 int f_verify_work(uint64_t *, const unsigned char *, uint64_t *, uint64_t);
01243
01249 #define F_SIGNATURE_RAW (uint32_t)1
01250
01256 #define F_SIGNATURE_STRING (uint32_t)2
01257
01263 #define F_SIGNATURE_OUTPUT_RAW_PK (uint32_t)4
01264
01270 #define F_SIGNATURE_OUTPUT_STRING_PK (uint32_t)8
01271
01277 #define F_SIGNATURE_OUTPUT_XRB_PK (uint32_t)16
01278
01284 #define F_SIGNATURE_OUTPUT_NANO_PK (uint32_t)32
01285
01291 #define F_IS_SIGNATURE_RAW_HEX_STRING (uint32_t)64
01292
01298 #define F_MESSAGE_IS_HASH_STRING (uint32_t)128
01299
01304 #define F_DEFAULT_THRESHOLD (uint64_t) 0xfffffffc000000000
01305
01329 int f_sign_data(
01330    unsigned char *signature,
01331    void *out_public_key,
01332    uint32_t ouput_type,
01333    const unsigned char *message,
01334    size_t msg_len,
01335    const unsigned char *private_key);
01336
01342 #define F_VERIFY_SIG_NANO_WALLET (uint32_t)1
01343
01349 #define F_VERIFY_SIG_RAW_HEX (uint32_t)2
01350
01356 #define F_VERIFY_SIG_ASCII_HEX (uint32_t)4
01357
01378 int f_verify_signed_data( const unsigned char *, const unsigned char *, size_t, const void *, uint32_t);
01379
```

```
01389 int f_is_valid_nano_seed_encrypted(void *, size_t, int);
01390
01391 #define F_BALANCE_RAW_128 F_NANO_A_RAW_128
01392 #define F_BALANCE_REAL_STRING F_NANO_A_REAL_STRING
01393 #define F_BALANCE_RAW_STRING F_NANO_A_RAW_STRING
01394 #define F_VALUE_SEND_RECEIVE_RAW_128 F_NANO_B_RAW_128
01395 #define F_VALUE_SEND_RECEIVE_REAL_STRING F_NANO_B_REAL_STRING
01396 #define F_VALUE_SEND_RECEIVE_RAW_STRING F_NANO_B_RAW_STRING
01397 #define F_VALUE_TO_SEND (int)(1<<0)
01398 #define F_VALUE_TO_RECEIVE (int)(1<<1)
01399
01400 int nano_create_block_dynamic(
01401    F_BLOCK_TRANSFER **,
01402    const void *,
01403    size_t,
01404    const void *,
01405    size_t,
01406    const void *,
01407    size_t,
01408    const void *,
01409    const void *,
01410    uint32_t,
01411    const void *,
01412    size_t,
01413    int
01414 );
01415
01416 int nano_create_p2pow_block_dynamic(
01417    F_BLOCK_TRANSFER **,
01418    F_BLOCK_TRANSFER *,
01419    const void *,
01420    size_t,
01421    const void *,
01422    uint32_t,
01423    const void *,
01424    size_t
01425 );
01426
01427 int f_verify_signed_block(F_BLOCK_TRANSFER *);
01428
01429 #ifdef __cplusplus
01430 }
01431 #endif
01432
```

5.7 f_util.h File Reference

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

Data Structures

- struct **f_ecdsa_key_pair_t**

Macros

- #define **F_LOG_MAX** 8∗256
- #define **LICENSE** "MIT License\n\n\Copyright (c) 2019 Fábio Pereira da Silva\n\n\Permission is hereby granted, free of charge, to any person obtaining a copy\n\of this software and associated documentation files (the \"Software\"), to deal\n\in the Software without restriction, including without limitation the rights\n\to use, copy, modify, merge, publish, distribute, sublicense, and/or sell\n\copies of the Software, and to permit persons to whom the Software is\n\furnished to do so, subject to the following conditions:\n\n\The above copyright notice and this permission notice shall be included in all\n\copies or substantial portions of the Software.\n\n\THE SOFTWARE IS PROVIDED \"AS IS\", WITHOUT WARRANTY OF ANY KIND, EXPRESS

OR\n\IMPLIED, INCLUDING BUT NOT LIMITED TO THE WARRANTIES OF MERCHANTABILITY,\n\FITN↩ ESS FOR A PARTICULAR PURPOSE AND NONINFRINGEMENT. IN NO EVENT SHALL THE\n\AUTHORS OR COPYRIGHT HOLDERS BE LIABLE FOR ANY CLAIM, DAMAGES OR OTHER\n\LIABILITY, WHETHER IN AN ACTION OF CONTRACT, TORT OR OTHERWISE, ARISING FROM,\n\OUT OF OR IN CONNECTION WITH THE SOFTWARE OR THE USE OR OTHER DEALINGS IN THE\n\SOFTWARE.\n\n\n"

- #define **F_WDT_MAX_ENTROPY_TIME** 2∗120
- #define **F_WDT_PANIC true**
- #define **F_WDT_MIN_TIME** 20
- #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_PBKDF2_ITER_SZ** 2∗4096

Typedefs

- typedef enum **f_pbkdf2_err_t f_pbkdf2_err**
- typedef enum **f_aes_err f_aes_err**
- typedef enum **f_md_hmac_sha512_t f_md_hmac_sha512**
- typedef enum **f_ecdsa_key_pair_err_t f_ecdsa_key_pair_err**
- typedef struct **f_ecdsa_key_pair_t f_ecdsa_key_pair**
- typedef int(∗ **fn_det**) (void ∗, unsigned char ∗, size_t)

Enumerations

- enum **f_pbkdf2_err_t** { **F_PBKDF2_RESULT_OK** =0, **F_PBKDF2_ERR_CTX** =95, **F_PBKDF2_ERR_↩ PKCS5**, **F_PBKDF2_ERR_INFO_SHA** }
- enum **f_aes_err** {
  F_AES_RESULT_OK =0, **F_AES_ERR_ENCKEY** =30, **F_AES_ERR_DECKEY**, **F_AES_ERR_MALLOC**, **F_AES_UNKNOW_DIRECTION**, **F_ERR_ENC_DECRYPT_FAILED** }
- enum **f_md_hmac_sha512_t** {
  F_HMAC_SHA512_OK = 0, **F_HMAC_SHA512_MALLOC** = 304, **F_HMAC_SHA512_ERR_INFO**, **F_H↩ MAC_SHA512_ERR_SETUP**,
  F_HMAC_SHA512_DIGEST_ERROR }
- enum **f_ecdsa_key_pair_err_t** { **F_ECDSA_KEY_PAIR_OK** = 0, **F_ECDSA_KEY_PAIR_NULL** = 330, **F_ECDSA_KEY_PAIR_MALLOC** }

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_verify_system_entropy_begin** ()
- void **f_verify_system_entropy_finish** ()
- int **f_file_exists** (char *)
- int **f_find_str** (size_t *, char *, size_t, char *)
- int **f_find_replace** (char *, size_t *, size_t, char *, size_t, char *, char *)
- int **f_is_integer** (char *, size_t)
- int **is_filled_with_value** (uint8_t *, size_t, uint8_t)
- char * **fhex2strv2** (char *, const void *, size_t, int)
- int **f_sha256_digest** (void **, int, uint8_t *, size_t)
- **f_pbkdf2_err** **f_pbkdf2_hmac** (unsigned char *, size_t, unsigned char *, size_t, uint8_t *)
- **f_aes_err** **f_aes256cipher** (uint8_t *, uint8_t *, void *, size_t, void *, 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 *)
- 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)
- int **f_ecdsa_secret_key_valid** (mbedtls_ecp_group_id, unsigned char *, size_t)
- int **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)

5.7.1 Detailed Description

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

Definition in file **f_util.h**.

5.7.2 Macro Definition Documentation

5.7.2.1 ENTROPY_BEGIN

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

Begins and prepares a entropy function.

See also

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

Definition at line **150** of file **f_util.h**.

5.7.2.2 ENTROPY_END

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

Ends a entropy function.

See also

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

Definition at line **157** of file **f_util.h**.

5.7.2.3 F_ENTROPY_TYPE_EXCELENT

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

Type of the excelent entropy used for verifier.

Slow

Definition at line **122** of file **f_util.h**.

5.7.2.4 F_ENTROPY_TYPE_GOOD

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

Type of the good entropy used for verifier.

Not so slow

Definition at line **129** of file **f_util.h**.

5.7.2.5 F_ENTROPY_TYPE_NOT_ENOUGH

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

Type of the moderate entropy used for verifier.

Fast

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

5.7.2.6 F_ENTROPY_TYPE_NOT_RECOMENDED

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

Type of the not recommended entropy used for verifier.

Very fast

Definition at line **143** of file **f_util.h**.

5.7.2.7 F_ENTROPY_TYPE_PARANOIC

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

Type of the very excelent entropy used for verifier.

Very slow

Definition at line **115** of file **f_util.h**.

5.7.2.8 F_LOG_MAX

```
#define F_LOG_MAX 8*256
```

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

5.7.2.9 F_PASS_IS_OUT_OVF

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

Password is overflow and cannot be stored.

Definition at line **205** of file **f_util.h**.

5.7.2.10 F_PASS_IS_TOO_LONG

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

Password is too long.

Definition at line **193** of file **f_util.h**.

5.7.2.11 F_PASS_IS_TOO_SHORT

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

Password is too short.

Definition at line **199** of file **f_util.h**.

5.7.2.12 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 **163** of file **f_util.h**.

5.7.2.13 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 **187** of file **f_util.h**.

5.7.2.14 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 **169** of file **f_util.h**.

5.7.2.15 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 **175** of file **f_util.h**.

5.7.2.16 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 **181** of file **f_util.h**.

5.7.2.17 F_PBKDF2_ITER_SZ

```
#define F_PBKDF2_ITER_SZ 2*4096
```

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

5.7.2.18 F_WDT_MAX_ENTROPY_TIME

```
#define F_WDT_MAX_ENTROPY_TIME 2*120
```

Definition at line **46** of file **f_util.h**.

5.7.2.19 F_WDT_MIN_TIME

```
#define F_WDT_MIN_TIME 20
```

Definition at line **48** of file **f_util.h**.

5.7.2.20 F_WDT_PANIC

```
#define F_WDT_PANIC  true
```

Definition at line **47** of file **f_util.h**.

5.7.2.21 LICENSE

```
#define LICENSE "MIT License\n\n\Copyright (c) 2019 Fábio Pereira da Silva\n\n\Permission is
hereby granted, free of charge, to any person obtaining a copy\n\of this software and associated
documentation files (the \"Software\"), to deal\n\in the Software without restriction, including
without limitation the rights\n\to use, copy, modify, merge, publish, distribute, sublicense,
and/or sell\n\copies of the Software, and to permit persons to whom the Software is\n\furnished
to do so, subject to the following conditions:\n\n\The above copyright notice and this permission
notice shall be included in all\n\copies or substantial portions of the Software.\n\n\THE S↩
OFTWARE IS PROVIDED \"AS IS\", WITHOUT WARRANTY OF ANY KIND, EXPRESS OR\n\IMPLIED, INCLUDING
BUT NOT LIMITED TO THE WARRANTIES OF MERCHANTABILITY,\n\FITNESS FOR A PARTICULAR PURPOSE AND
NONINFRINGEMENT. IN NO EVENT SHALL THE\n\AUTHORS OR COPYRIGHT HOLDERS BE LIABLE FOR ANY CLA↩
IM, DAMAGES OR OTHER\n\LIABILITY, WHETHER IN AN ACTION OF CONTRACT, TORT OR OTHERWISE, ARISING
FROM,\n\OUT OF OR IN CONNECTION WITH THE SOFTWARE OR THE USE OR OTHER DEALINGS IN THE\n\SOFT↩
WARE.\n\n\n"
```

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

5.7.3 Typedef Documentation

5.7.3.1 f_aes_err

```
typedef enum f_aes_err f_aes_err
```

5.7.3.2 f_ecdsa_key_pair

```
typedef struct f_ecdsa_key_pair_t f_ecdsa_key_pair
```

5.7.3.3 f_ecdsa_key_pair_err

```
typedef enum f_ecdsa_key_pair_err_t f_ecdsa_key_pair_err
```

5.7.3.4 f_md_hmac_sha512

```
typedef enum f_md_hmac_sha512_t f_md_hmac_sha512
```

5.7.3.5 f_pbkdf2_err

```
typedef enum  f_pbkdf2_err_t  f_pbkdf2_err
```

5.7.3.6 fn_det

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

Definition at line **452** of file **f_util.h**.

5.7.4 Enumeration Type Documentation

5.7.4.1 f_aes_err

```
enum  f_aes_err
```

Enumerator

F_AES_RESULT_OK	
F_AES_ERR_ENCKEY	
F_AES_ERR_DECKEY	
F_AES_ERR_MALLOC	
F_AES_UNKNOW_DIRECTION	
F_ERR_ENC_DECRYPT_FAILED	

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

5.7.4.2 f_ecdsa_key_pair_err_t

```
enum  f_ecdsa_key_pair_err_t
```

Enumerator

F_ECDSA_KEY_PAIR_OK	
F_ECDSA_KEY_PAIR_NULL	
F_ECDSA_KEY_PAIR_MALLOC	

Definition at line **235** of file **f_util.h**.

5.7.4.3 f_md_hmac_sha512_t

enum **f_md_hmac_sha512_t**

Enumerator

F_HMAC_SHA512_OK	
F_HMAC_SHA512_MALLOC	
F_HMAC_SHA512_ERR_INFO	
F_HMAC_SHA512_ERR_SETUP	
F_HMAC_SHA512_DIGEST_ERROR	

Definition at line **227** of file **f_util.h**.

5.7.4.4 f_pbkdf2_err_t

enum **f_pbkdf2_err_t**

Enumerator

F_PBKDF2_RESULT_OK	
F_PBKDF2_ERR_CTX	
F_PBKDF2_ERR_PKCS5	
F_PBKDF2_ERR_INFO_SHA	

Definition at line **211** of file **f_util.h**.

5.7.5 Function Documentation

5.7.5.1 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

5.7.5.2 f_aes256cipher()

```
f_aes_err f_aes256cipher (
            uint8_t * ,
            uint8_t * ,
            void * ,
            size_t ,
            void * ,
            int  )
```

5.7.5.3 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

5.7.5.4 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. **??**)

5.7.5.5 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. **??**)

5.7.5.6 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. **??**)

5.7.5.7 f_convert_to_long_int_std()

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

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

Parameters

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

Return values

0	On Success, Otherwise error

See also

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

5.7.5.8 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. **??**)

5.7.5.9 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. **??**)

5.7.5.10 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. **??**)

5.7.5.11 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. **??**)

5.7.5.12 f_ecdsa_public_key_valid()

```
int f_ecdsa_public_key_valid (
            mbedtls_ecp_group_id ,
            unsigned char * ,
            size_t  )
```

5.7.5.13 f_ecdsa_secret_key_valid()

```
int f_ecdsa_secret_key_valid (
            mbedtls_ecp_group_id ,
            unsigned char * ,
            size_t  )
```

5.7.5.14 f_file_exists()

```
int f_file_exists (
            char *  )
```

5.7.5.15 f_find_replace()

```
int f_find_replace (
            char * ,
            size_t * ,
            size_t ,
            char * ,
            size_t ,
            char * ,
            char *  )
```

5.7.5.16 f_find_str()

```
int f_find_str (
            size_t * ,
            char * ,
            size_t ,
            char *  )
```

5.7.5.17 f_gen_ecdsa_key_pair()

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

5.7.5.18 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

5.7.5.19 f_hmac_sha512()

```
f_md_hmac_sha512 f_hmac_sha512 (
            unsigned char * ,
            const unsigned char * ,
            size_t ,
            const unsigned char * ,
            size_t  )
```

5.7.5.20 f_is_integer()

```
int f_is_integer (
            char * ,
            size_t  )
```

5.7.5.21 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()

5.7.5.22 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)

5.7.5.23 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

5.7.5.24 f_pbkdf2_hmac()

```
f_pbkdf2_err f_pbkdf2_hmac (
            unsigned char * ,
            size_t ,
            unsigned char * ,
            size_t ,
            uint8_t *  )
```

5.7.5.25 f_random_detach()

```
void f_random_detach ( )
```

Detaches system random numeber genarator from myNanoEmbedded API.

See also

f_random_attach()

5.7.5.26 f_reverse()

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

5.7.5.27 f_ripemd160()

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

5.7.5.28 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*

5.7.5.29 f_sha256_digest()

```
int f_sha256_digest (
             void ** ,
             int ,
             uint8_t * ,
             size_t  )
```

5.7.5.30 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

5.7.5.31 f_uncompress_elliptic_curve()

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

5.7.5.32 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

5.7.5.33 f_verify_system_entropy_begin()

```
int f_verify_system_entropy_begin ( )
```

5.7.5.34 f_verify_system_entropy_finish()

```
void f_verify_system_entropy_finish ( )
```

5.7.5.35 fhex2strv2()

```
char* fhex2strv2 (
            char * ,
            const void * ,
            size_t ,
            int  )
```

5.7.5.36 is_filled_with_value()

```
int is_filled_with_value (
            uint8_t * ,
            size_t ,
            uint8_t  )
```

5.8 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
00018 #ifdef __cplusplus
00019 extern "C" {
00020 #endif
00021
00022 #ifndef F_DOC_SKIP
00023
00024  #define F_LOG_MAX 8*256
00025  #define LICENSE \
00026 "MIT License\n\n\
00027 Copyright (c) 2019 Fábio Pereira da Silva\n\n\
00028 Permission is hereby granted, free of charge, to any person obtaining a copy\n\
00029 of this software and associated documentation files (the \"Software\"), to deal\n\
00030 in the Software without restriction, including without limitation the rights\n\
00031 to use, copy, modify, merge, publish, distribute, sublicense, and/or sell\n\
00032 copies of the Software, and to permit persons to whom the Software is\n\
00033 furnished to do so, subject to the following conditions:\n\n\
00034 The above copyright notice and this permission notice shall be included in all\n\
00035 copies or substantial portions of the Software.\n\n\
00036 THE SOFTWARE IS PROVIDED \"AS IS\", WITHOUT WARRANTY OF ANY KIND, EXPRESS OR\n\
00037 IMPLIED, INCLUDING BUT NOT LIMITED TO THE WARRANTIES OF MERCHANTABILITY,\n\
00038 FITNESS FOR A PARTICULAR PURPOSE AND NONINFRINGEMENT. IN NO EVENT SHALL THE\n\
00039 AUTHORS OR COPYRIGHT HOLDERS BE LIABLE FOR ANY CLAIM, DAMAGES OR OTHER\n\
00040 LIABILITY, WHETHER IN AN ACTION OF CONTRACT, TORT OR OTHERWISE, ARISING FROM,\n\
00041 OUT OF OR IN CONNECTION WITH THE SOFTWARE OR THE USE OR OTHER DEALINGS IN THE\n\
00042 SOFTWARE.\n\n\n"
00043
00044 #endif
00045
00046 #define F_WDT_MAX_ENTROPY_TIME 2*120
00047 #define F_WDT_PANIC true
00048 #define F_WDT_MIN_TIME 20//4
00049
00050
00068 int f_verify_system_entropy(uint32_t, void *, size_t, int);
00069
00096 int f_pass_must_have_at_least(char *, size_t, size_t, size_t, int);
00097
00098 #ifndef F_DOC_SKIP
00099
00100 int f_verify_system_entropy_begin();
```

```
00101 void f_verify_system_entropy_finish();
00102 int f_file_exists(char *);
00103 int f_find_str(size_t *, char *, size_t, char *);
00104 int f_find_replace(char *, size_t *, size_t, char *, size_t, char *, char *);
00105 int f_is_integer(char *, size_t);
00106 int is_filled_with_value(uint8_t *, size_t, uint8_t);
00107
00108 #endif
00109
00110 //#define F_ENTROPY_TYPE_PARANOIC (uint32_t)1476682819
00115 #define F_ENTROPY_TYPE_PARANOIC (uint32_t)1477682819
00116
00117 //#define F_ENTROPY_TYPE_EXCELENT (uint32_t)1475885281
00122 #define F_ENTROPY_TYPE_EXCELENT (uint32_t)1476885281
00123
00124 //#define F_ENTROPY_TYPE_GOOD (uint32_t)1471531015
00129 #define F_ENTROPY_TYPE_GOOD (uint32_t)1472531015
00130
00131 //#define F_ENTROPY_TYPE_NOT_ENOUGH (uint32_t)1470001808
00136 #define F_ENTROPY_TYPE_NOT_ENOUGH (uint32_t)1471001808
00137
00138 //#define F_ENTROPY_TYPE_NOT_RECOMENDED (uint32_t)1469703345
00143 #define F_ENTROPY_TYPE_NOT_RECOMENDED (uint32_t)1470003345
00144
00150 #define ENTROPY_BEGIN f_verify_system_entropy_begin();
00151
00157 #define ENTROPY_END f_verify_system_entropy_finish();
00158
00163 #define F_PASS_MUST_HAVE_AT_LEAST_NONE (int)0
00164
00169 #define F_PASS_MUST_HAVE_AT_LEAST_ONE_NUMBER (int)1
00170
00175 #define F_PASS_MUST_HAVE_AT_LEAST_ONE_SYMBOL (int)2
00176
00181 #define F_PASS_MUST_HAVE_AT_LEAST_ONE_UPPER_CASE (int)4
00182
00187 #define F_PASS_MUST_HAVE_AT_LEAST_ONE_LOWER_CASE (int)8
00188
00193 #define F_PASS_IS_TOO_LONG (int)256
00194
00199 #define F_PASS_IS_TOO_SHORT (int)512
00200
00205 #define F_PASS_IS_OUT_OVF (int)1024//768
00206
00207 #ifndef F_DOC_SKIP
00208
00209  #define F_PBKDF2_ITER_SZ 2*4096
00210
00211 typedef enum f_pbkdf2_err_t {
00212     F_PBKDF2_RESULT_OK=0,
00213     F_PBKDF2_ERR_CTX=95,
00214     F_PBKDF2_ERR_PKCS5,
00215     F_PBKDF2_ERR_INFO_SHA
00216 } f_pbkdf2_err;
00217
00218 typedef enum f_aes_err {
00219     F_AES_RESULT_OK=0,
00220     F_AES_ERR_ENCKEY=30,
00221     F_AES_ERR_DECKEY,
00222     F_AES_ERR_MALLOC,
00223     F_AES_UNKNOW_DIRECTION,
00224     F_ERR_ENC_DECRYPT_FAILED
00225 } f_aes_err;
00226
00227 typedef enum f_md_hmac_sha512_t {
00228    F_HMAC_SHA512_OK = 0,
00229    F_HMAC_SHA512_MALLOC = 304,
00230    F_HMAC_SHA512_ERR_INFO,
00231    F_HMAC_SHA512_ERR_SETUP,
00232    F_HMAC_SHA512_DIGEST_ERROR
00233 } f_md_hmac_sha512;
00235 typedef enum f_ecdsa_key_pair_err_t {
00236    F_ECDSA_KEY_PAIR_OK = 0,
00237    F_ECDSA_KEY_PAIR_NULL = 330,
00238    F_ECDSA_KEY_PAIR_MALLOC
00239 } f_ecdsa_key_pair_err;
00240
00241 typedef struct f_ecdsa_key_pair_t {
00242     size_t public_key_sz;
00243     size_t private_key_sz;
00244     mbedtls_ecdsa_context *ctx;
00245     mbedtls_ecp_group_id gid;
00246     unsigned char public_key[MBEDTLS_ECDSA_MAX_LEN];
00247     unsigned char private_key[MBEDTLS_ECDSA_MAX_LEN];
00248 } f_ecdsa_key_pair;
00249
00250 char *fhex2strv2(char *, const void *, size_t, int);
```

```
00251 //uint8_t *f_sha256_digest(uint8_t *, size_t);
00252 int f_sha256_digest(void **, int, uint8_t *, size_t);
00253 f_pbkdf2_err f_pbkdf2_hmac(unsigned char *, size_t, unsigned char *, size_t, uint8_t *);
00254 f_aes_err f_aes256cipher(uint8_t *, uint8_t *, void *, size_t, void *, int);
00255
00256 #endif
00257
00269 int f_passwd_comp_safe(char *, char *, size_t, size_t, size_t);
00270
00281 char *f_get_entropy_name(uint32_t);
00282
00297 uint32_t f_sel_to_entropy_level(int);
00298
00307 int f_str_to_hex(uint8_t *, char *);
00308
00319 int f_convert_to_long_int(unsigned long int *, char *, size_t);
00320
00321
00332 int f_convert_to_unsigned_int(unsigned int *, char *, size_t);
00333
00344 int f_convert_to_long_int0x(unsigned long int *, char *, size_t);
00345
00356 int f_convert_to_long_int0(unsigned long int *, char *, size_t);
00357
00371 int f_convert_to_long_int_std(unsigned long int *, char *, size_t);
00372
00380 void *f_is_random_attached();
00381
00388 void f_random_detach();
00389
00400 int f_convert_to_unsigned_int0x(unsigned int *val, char *value, size_t value_sz);
00401
00412 int f_convert_to_unsigned_int0(unsigned int *val, char *value, size_t value_sz);
00413
00427 int f_convert_to_unsigned_int_std(unsigned int *val, char *value, size_t value_sz);
00428
00438 int f_convert_to_double(double *, const char *);
00439
00450 uint32_t crc32_init(unsigned char *, size_t, uint32_t);
00451 //
00452 typedef int (*fn_det)(void *, unsigned char *, size_t);
00453 int f_reverse(unsigned char *, size_t);
00454 f_md_hmac_sha512 f_hmac_sha512(unsigned char *, const unsigned char *, size_t, const unsigned char *,
      size_t);
00455 int f_ecdsa_secret_key_valid(mbedtls_ecp_group_id, unsigned char *, size_t);
00456 int f_ecdsa_public_key_valid(mbedtls_ecp_group_id, unsigned char *, size_t);
00457 f_ecdsa_key_pair_err f_gen_ecdsa_key_pair(f_ecdsa_key_pair *, int, fn_det, void *);
00458 int f_uncompress_elliptic_curve(uint8_t *, size_t, size_t *, mbedtls_ecp_group_id, uint8_t *, size_t);
00459 uint8_t *f_ripemd160(const uint8_t *, size_t);
00460
00461 #ifdef __cplusplus
00462 }
00463 #endif
```

5.9 qrcode.h File Reference

`#include <stdint.h>`

Data Structures

- struct **QRCode**

Macros

- #define **MODE_NUMERIC** 0
- #define **MODE_ALPHANUMERIC** 1
- #define **MODE_BYTE** 2
- #define **ECC_LOW** 0
- #define **ECC_MEDIUM** 1
- #define **ECC_QUARTILE** 2
- #define **ECC_HIGH** 3
- #define **LOCK_VERSION** 0

Typedefs

- typedef unsigned char **bool**
- typedef struct **QRCode** **QRCode**

Functions

- uint16_t **qrcode_getBufferSize** (uint8_t **version**)
- int8_t **qrcode_initText** (**QRCode** *qrcode, uint8_t *modules, uint8_t **version**, uint8_t ecc, const char *data)
- int8_t **qrcode_initBytes** (**QRCode** *qrcode, uint8_t *modules, uint8_t **version**, uint8_t ecc, uint8_t *data, uint16_t length)
- **bool** **qrcode_getModule** (**QRCode** *qrcode, uint8_t x, uint8_t y)

Variables

- static const **bool** **false** = 0
- static const **bool** **true** = 1

5.9.1 Macro Definition Documentation

5.9.1.1 ECC_HIGH

```
#define ECC_HIGH 3
```

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

5.9.1.2 ECC_LOW

```
#define ECC_LOW 0
```

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

5.9.1.3 ECC_MEDIUM

```
#define ECC_MEDIUM 1
```

Definition at line **57** of file **qrcode.h**.

5.9.1.4 ECC_QUARTILE

```
#define ECC_QUARTILE 2
```

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

5.9.1.5 LOCK_VERSION

```
#define LOCK_VERSION 0
```

Definition at line **65** of file **qrcode.h**.

5.9.1.6 MODE_ALPHANUMERIC

```
#define MODE_ALPHANUMERIC 1
```

Definition at line **51** of file **qrcode.h**.

5.9.1.7 MODE_BYTE

```
#define MODE_BYTE 2
```

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

5.9.1.8 MODE_NUMERIC

```
#define MODE_NUMERIC 0
```

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

5.9.2 Typedef Documentation

5.9.2.1 bool

```
typedef unsigned char  bool
```

The MIT License (MIT)

This library is written and maintained by Richard Moore. Major parts were derived from Project Nayuki's library.

Copyright (c) 2017 Richard Moore (https://github.com/ricmoo/QRCode) Copyright (c) 2017 Project Nayuki (https://www.nayuki.io/page/qr-code-generator-library)

Permission is hereby granted, free of charge, to any person obtaining a copy of this software and associated documentation files (the "Software"), to deal in the Software without restriction, including without limitation the rights to use, copy, modify, merge, publish, distribute, sublicense, and/or sell copies of the Software, and to permit persons to whom the Software is furnished to do so, subject to the following conditions:

The above copyright notice and this permission notice shall be included in all copies or substantial portions of the Software.

THE SOFTWARE IS PROVIDED "AS IS", WITHOUT WARRANTY OF ANY KIND, EXPRESS OR IMPLIED, IN↩ CLUDING BUT NOT LIMITED TO THE WARRANTIES OF MERCHANTABILITY, FITNESS FOR A PARTICULAR PURPOSE AND NONINFRINGEMENT. IN NO EVENT SHALL THE AUTHORS OR COPYRIGHT HOLDERS BE LIABLE FOR ANY CLAIM, DAMAGES OR OTHER LIABILITY, WHETHER IN AN ACTION OF CONTRACT, TORT OR OTHERWISE, ARISING FROM, OUT OF OR IN CONNECTION WITH THE SOFTWARE OR THE USE OR OTHER DEALINGS IN THE SOFTWARE. Special thanks to Nayuki (https://www.nayuki.io/) from which this library was heavily inspired and compared against.

See: https://github.com/nayuki/QR-Code-generator/tree/master/cpp

Definition at line **41** of file **qrcode.h**.

5.9.2.2 QRCode

```
typedef struct  QRCode  QRCode
```

5.9.3 Function Documentation

5.9.3.1 qrcode_getBufferSize()

```
uint16_t qrcode_getBufferSize (
             uint8_t version )
```

5.9.3.2 qrcode_getModule()

```
bool qrcode_getModule (
              QRCode * qrcode,
             uint8_t x,
             uint8_t y )
```

5.9.3.3 qrcode_initBytes()

```
int8_t qrcode_initBytes (
              QRCode * qrcode,
             uint8_t * modules,
             uint8_t version,
             uint8_t ecc,
             uint8_t * data,
             uint16_t length )
```

5.9.3.4 qrcode_initText()

```
int8_t qrcode_initText (
              QRCode * qrcode,
             uint8_t * modules,
             uint8_t version,
             uint8_t ecc,
             const char * data )
```

5.9.4 Variable Documentation

5.9.4.1 false

```
const bool false = 0 [static]
```

Definition at line **42** of file **qrcode.h**.

5.9.4.2 true

```
const bool true = 1 [static]
```

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

5.10 qrcode.h

```
00001
00037 #ifndef __QRCODE_H_
00038 #define __QRCODE_H_
00039
00040 #ifndef __cplusplus
00041 typedef unsigned char bool;
00042 static const bool false = 0;
00043 static const bool true = 1;
00044 #endif
00045
00046 #include <stdint.h>
00047
00048
00049 // QR Code Format Encoding
00050 #define MODE_NUMERIC        0
00051 #define MODE_ALPHANUMERIC   1
00052 #define MODE_BYTE           2
00053
00054
00055 // Error Correction Code Levels
00056 #define ECC_LOW             0
00057 #define ECC_MEDIUM          1
00058 #define ECC_QUARTILE        2
00059 #define ECC_HIGH            3
00060
00061
00062 // If set to non-zero, this library can ONLY produce QR codes at that version
00063 // This saves a lot of dynamic memory, as the codeword tables are skipped
00064 #ifndef LOCK_VERSION
00065 #define LOCK_VERSION        0
00066 #endif
00067
00068
00069 typedef struct QRCode {
00070     uint8_t version;
00071     uint8_t size;
00072     uint8_t ecc;
00073     uint8_t mode;
00074     uint8_t mask;
00075     uint8_t *modules;
00076 } QRCode;
00077
00078
00079 #ifdef __cplusplus
00080 extern "C"{
00081 #endif  /* __cplusplus */
00082
00083
00084
00085 uint16_t qrcode_getBufferSize(uint8_t version);
00086
00087 int8_t qrcode_initText(QRCode *qrcode, uint8_t *modules, uint8_t version, uint8_t
      ecc, const char *data);
00088 int8_t qrcode_initBytes(QRCode *qrcode, uint8_t *modules, uint8_t version, uint8_t ecc, uint8_t *data,
      uint16_t length);
00089
00090 bool qrcode_getModule(QRCode *qrcode, uint8_t x, uint8_t y);
00091
00092
00093
00094 #ifdef __cplusplus
00095 }
00096 #endif  /* __cplusplus */
00097
00098
00099 #endif  /* __QRCODE_H_ */
```

5.11 upos_conf.h File Reference

Macros

- #define **UPOS_MONITORE_STACK_SIZE** (size_t)8*1024

5.11.1 Macro Definition Documentation

5.11.1.1 UPOS_MONITORE_STACK_SIZE

```
#define UPOS_MONITORE_STACK_SIZE (size_t)8*1024
```

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

5.12 upos_conf.h

```
00001 //dom set 27 12:27:11 -03 2020
00002
00003 #define UPOS_MONITORE_STACK_SIZE (size_t)8*1024
00004
```

5.13 upos_events.h File Reference

```
#include "esp_event.h"
```

Data Structures

- struct **upos_register_events_t**

Macros

- #define **UPOS_EVENT_TAG_STRING_MAX** (size_t)96
- #define **UPOS_TIME_EVENT_SECONDS**(time) time∗1000/portTICK_RATE_MS
- #define **UPOS_TIME_EVENT_MILLISECONDS**(time) time/portTICK_RATE_MS
- #define **UPOS_TIME_EVENT_MICROSECONDS**(time) pdMS_TO_TICKS(time)
- #define **UPOS_TIME_WAIT_FOR_EVER** portMAX_DELAY
- #define **UPOS_MIN_EVENT_TO_WAIT_US** 200
- #define **UPOS_TIME_EVENT_EPOCH_MINUTES**(time) time∗60
- #define **UPOS_TIME_EVENT_EPOCH_HOURS**(time) 60∗ **UPOS_TIME_EVENT_EPOCH_MINUTES**(time)
- #define **UPOS_TIME_EVENT_EPOCH_DAY**(time) 24∗ **UPOS_TIME_EVENT_EPOCH_HOURS**(time)
- #define **UPOS_TIME_EVENT_EPOCH_WEEK**(time) 7∗ **UPOS_TIME_EVENT_EPOCH_DAY**(time)
- #define **UPOS_TIME_EVENT_EPOCH_YEAR**(time) 365∗ **UPOS_TIME_EVENT_EPOCH_DAY**(time)

Typedefs

- typedef enum **upos_events_err_t** **UPOS_EVENTS_ERR**
- typedef enum **upos_register_event_err_t** **UPOS_REGISTER_EVENTS_ERR**
- typedef void(∗ **fn_evt**) (void ∗)
- typedef struct **upos_register_events_t** **upos_register_events**

Enumerations

- enum **upos_events_err_t** {
  UPOS_EVENTS_OK = 0, **UPOS_EVENTS_FAIL** = 30000, **UPOS_EVENT_ALREADY_EXISTS**, **UPOS_EVENT_INIT_MALLOC**,
  UPOS_EVENTS_INIT_FAIL }
- enum **upos_register_event_err_t** { **UPOS_REGISTER_EVT_OK** = 0, **UPOS_REGISTER_EVT_MISSING_FUNCTION_NAME** = 30100, **UPOS_REGISTER_MISSING_FUNCTION**, **UPOS_REGISTER_EVT_CREATE_TASK** }

Functions

- int **upos_init_events** ()
- void * **upos_get_global_events_group** ()
- void **upos_unregister_event** (void *fn_handle)
- int **upos_register_event** (void **, **fn_evt**, void *, const char *, uint32_t)
- void **UPOS_WAIT** (TickType_t)

5.13.1 Macro Definition Documentation

5.13.1.1 UPOS_EVENT_TAG_STRING_MAX

```
#define UPOS_EVENT_TAG_STRING_MAX (size_t)96
```

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

5.13.1.2 UPOS_MIN_EVENT_TO_WAIT_US

```
#define UPOS_MIN_EVENT_TO_WAIT_US 200
```

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

5.13.1.3 UPOS_TIME_EVENT_EPOCH_DAY

```
#define UPOS_TIME_EVENT_EPOCH_DAY(
              time ) 24* UPOS_TIME_EVENT_EPOCH_HOURS(time)
```

Definition at line **46** of file **upos_events.h**.

5.13.1.4 UPOS_TIME_EVENT_EPOCH_HOURS

```
#define UPOS_TIME_EVENT_EPOCH_HOURS(
              time ) 60* UPOS_TIME_EVENT_EPOCH_MINUTES(time)
```

Definition at line **45** of file **upos_events.h**.

5.13.1.5 UPOS_TIME_EVENT_EPOCH_MINUTES

```
#define UPOS_TIME_EVENT_EPOCH_MINUTES(
              time ) time*60
```

Definition at line **44** of file **upos_events.h**.

5.13.1.6 UPOS_TIME_EVENT_EPOCH_WEEK

```
#define UPOS_TIME_EVENT_EPOCH_WEEK(
              time ) 7* UPOS_TIME_EVENT_EPOCH_DAY(time)
```

Definition at line **47** of file **upos_events.h**.

5.13.1.7 UPOS_TIME_EVENT_EPOCH_YEAR

```
#define UPOS_TIME_EVENT_EPOCH_YEAR(
              time ) 365* UPOS_TIME_EVENT_EPOCH_DAY(time)
```

Definition at line **48** of file **upos_events.h**.

5.13.1.8 UPOS_TIME_EVENT_MICROSECONDS

```
#define UPOS_TIME_EVENT_MICROSECONDS(
              time ) pdMS_TO_TICKS(time)
```

Definition at line **41** of file **upos_events.h**.

5.13.1.9 UPOS_TIME_EVENT_MILLISECONDS

```
#define UPOS_TIME_EVENT_MILLISECONDS(
              time ) time/portTICK_RATE_MS
```

Definition at line **40** of file **upos_events.h**.

5.13.1.10 UPOS_TIME_EVENT_SECONDS

```
#define UPOS_TIME_EVENT_SECONDS(
              time ) time*1000/portTICK_RATE_MS
```

Definition at line **39** of file **upos_events.h**.

5.13.1.11 UPOS_TIME_WAIT_FOR_EVER

```
#define UPOS_TIME_WAIT_FOR_EVER portMAX_DELAY
```

Definition at line **42** of file **upos_events.h**.

5.13.2 Typedef Documentation

5.13.2.1 fn_evt

```
typedef void(* fn_evt) (void *)
```

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

5.13.2.2 UPOS_EVENTS_ERR

```
typedef enum  upos_events_err_t  UPOS_EVENTS_ERR
```

5.13.2.3 upos_register_events

```
typedef struct  upos_register_events_t  upos_register_events
```

5.13.2.4 UPOS_REGISTER_EVENTS_ERR

```
typedef enum  upos_register_event_err_t  UPOS_REGISTER_EVENTS_ERR
```

5.13.3 Enumeration Type Documentation

5.13.3.1 upos_events_err_t

```
enum  upos_events_err_t
```

Enumerator

UPOS_EVENTS_OK	
UPOS_EVENTS_FAIL	
UPOS_EVENT_ALREADY_EXISTS	
UPOS_EVENT_INIT_MALLOC	
UPOS_EVENTS_INIT_FAIL	

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

5.13.3.2 upos_register_event_err_t

```
enum upos_register_event_err_t
```

Enumerator

UPOS_REGISTER_EVT_OK	
UPOS_REGISTER_EVT_MISSING_FUNCTION_NAME	
UPOS_REGISTER_MISSING_FUNCTION	
UPOS_REGISTER_EVT_CREATE_TASK	

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

5.13.4 Function Documentation

5.13.4.1 upos_get_global_events_group()

```
void* upos_get_global_events_group ( )
```

5.13.4.2 upos_init_events()

```
int upos_init_events ( )
```

5.13.4.3 upos_register_event()

```
int upos_register_event (
             void ** ,
              fn_evt ,
             void * ,
             const char * ,
             uint32_t  )
```

5.13.4.4 upos_unregister_event()

```
void upos_unregister_event (
            void * fn_handle )
```

5.13.4.5 UPOS_WAIT()

```
void UPOS_WAIT (
            TickType_t  )
```

5.14 upos_events.h

```
00001 #ifndef ESP_EVENT_H
00002  #define ESP_EVENT_H
00003  #include "esp_event.h"
00004 #endif
00005 #define UPOS_EVENT_TAG_STRING_MAX (size_t)96
00006 typedef enum upos_events_err_t {
00007    UPOS_EVENTS_OK = 0,
00008    UPOS_EVENTS_FAIL = 30000,
00009    UPOS_EVENT_ALREADY_EXISTS,
00010    UPOS_EVENT_INIT_MALLOC,
00011    UPOS_EVENTS_INIT_FAIL
00012 } UPOS_EVENTS_ERR;
00013
00014 typedef enum upos_register_event_err_t {
00015    UPOS_REGISTER_EVT_OK = 0,
00016    UPOS_REGISTER_EVT_MISSING_FUNCTION_NAME = 30100,
00017    UPOS_REGISTER_MISSING_FUNCTION,
00018    UPOS_REGISTER_EVT_CREATE_TASK
00019 } UPOS_REGISTER_EVENTS_ERR;
00020
00021 typedef void (*fn_evt)(void *);
00022
00023 typedef struct upos_register_events_t {
00024    int err;
00025    const char *tag;
00026    esp_event_base_t event_base;
00027    int32_t event_id;
00028    esp_event_handler_t event_handler;
00029    void *arg;
00030    fn_evt ev_cb;
00031 } upos_register_events;
00032
00033 int upos_init_events();
00034 void *upos_get_global_events_group();
00035 void upos_unregister_event(void *fn_handle);
00036 int upos_register_event(void **, fn_evt, void *, const char *, uint32_t);
00037 void UPOS_WAIT(TickType_t);
00038
00039 #define UPOS_TIME_EVENT_SECONDS(time) time*1000/portTICK_RATE_MS
00040 #define UPOS_TIME_EVENT_MILLISECONDS(time) time/portTICK_RATE_MS
00041 #define UPOS_TIME_EVENT_MICROSECONDS(time) pdMS_TO_TICKS(time)
00042 #define UPOS_TIME_WAIT_FOR_EVER portMAX_DELAY
00043 #define UPOS_MIN_EVENT_TO_WAIT_US 200
00044 #define UPOS_TIME_EVENT_EPOCH_MINUTES(time) time*60
00045 #define UPOS_TIME_EVENT_EPOCH_HOURS(time) 60*UPOS_TIME_EVENT_EPOCH_MINUTES(time)
00046 #define UPOS_TIME_EVENT_EPOCH_DAY(time) 24*UPOS_TIME_EVENT_EPOCH_HOURS(time)
00047 #define UPOS_TIME_EVENT_EPOCH_WEEK(time) 7*UPOS_TIME_EVENT_EPOCH_DAY(time)
00048 #define UPOS_TIME_EVENT_EPOCH_YEAR(time) 365*UPOS_TIME_EVENT_EPOCH_DAY(time)
00049
00050
```

5.15 upos_system.h File Reference

```
#include "tcpip_adapter.h"
```

Macros

- #define **upos_init_tcp_ip**() tcpip_adapter_init()
- #define **TAB** "\n\t"
- #define **UPOS_NULL_STRING** ""

Typedefs

- typedef enum **upos_mac_err_t** **UPOS_MAC_ERR**

Enumerations

- enum **upos_mac_err_t** { **UPOS_MAC_OK** = 0, **UPOS_MAC_READ_EFUSE** = 1100, **UPOS_MAC_CR↩ C8_ERROR** }

Functions

- int IRAM_ATTR **upos_init_nvs** ()
- int IRAM_ATTR **upos_set_mac** (uint8_t ∗, int)

5.15.1 Macro Definition Documentation

5.15.1.1 TAB

```
#define TAB "\n\t"
```

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

5.15.1.2 upos_init_tcp_ip

```
#define upos_init_tcp_ip( ) tcpip_adapter_init()
```

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

5.15.1.3 UPOS_NULL_STRING

```
#define UPOS_NULL_STRING ""
```

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

5.15.2 Typedef Documentation

5.15.2.1 UPOS_MAC_ERR

```
typedef enum  upos_mac_err_t  UPOS_MAC_ERR
```

5.15.3 Enumeration Type Documentation

5.15.3.1 upos_mac_err_t

```
enum  upos_mac_err_t
```

Enumerator

UPOS_MAC_OK	
UPOS_MAC_READ_EFUSE	
UPOS_MAC_CRC8_ERROR	

Definition at line **10** of file **upos_system.h**.

5.15.4 Function Documentation

5.15.4.1 upos_init_nvs()

```
int IRAM_ATTR upos_init_nvs ( )
```

5.15.4.2 upos_set_mac()

```
int IRAM_ATTR upos_set_mac (
            uint8_t * ,
            int  )
```

5.16 upos_system.h

```
00001
00002 #include "tcpip_adapter.h"
00003
00004 #define upos_init_tcp_ip() tcpip_adapter_init()
00005 int IRAM_ATTR upos_init_nvs();
00006 int IRAM_ATTR upos_set_mac(uint8_t *, int);
00007 #define TAB "\n\t"
00008 #define UPOS_NULL_STRING ""
00009
00010 typedef enum upos_mac_err_t {
00011    UPOS_MAC_OK = 0,
00012    UPOS_MAC_READ_EFUSE = 1100,
00013    UPOS_MAC_CRC8_ERROR
00014 } UPOS_MAC_ERR;
00015
```

5.17 upos_time.h File Reference

Functions

- int **upos_init_sntp** ()

5.17.1 Function Documentation

5.17.1.1 upos_init_sntp()

```
int upos_init_sntp ( )
```

5.18 upos_time.h

```
00001
00002 int upos_init_sntp();
00003
```

5.19 upos_wifi.h File Reference

```
#include "esp_event.h"
```

Data Structures

- struct **upos_wifi_status_t**
- struct **upos_wifi_cb_ctx_t**
- struct **upos_wifi_event_cb_ctx_t**

Macros

- #define **UPOS_WIFI_EVENT_CALLBACK**(param) ((size_t)offsetof(struct **upos_wifi_cb_ctx_t**, param))
- #define **_UPOS_IP6** (int)0
- #define **_UPOS_IP4** (int)1
- #define **upos_get_ip_string**() **get_ip_string_util**(**_UPOS_IP4**)
- #define **upos_get_ip6_string**() **get_ip_string_util**(**_UPOS_IP6**)
- #define **upos_wifi_disconnect**() **upos_wifi_stop_util**(0)
- #define **upos_wifi_stop**() **upos_wifi_stop_util**(-2)

Typedefs

- typedef enum **upos_wifi_err_t** **UPOS_WIFI_ERR**
- typedef struct **upos_wifi_status_t** **UPOS_WIFI**
- typedef void(∗ **upos_wifi_cb**) (void ∗)
- typedef struct **upos_wifi_cb_ctx_t** **UPOS_WIFI_CB_CTX**
- typedef enum **upos_wifi_event_type_e** **UPOS_WIFI_EVENT_ENUM**
- typedef struct **upos_wifi_event_cb_ctx_t** **UPOS_WIFI_EVENT_CTX**

Enumerations

- enum **upos_wifi_err_t** { **UPOS_WIFI_OK** = 0, **UPOS_WIFI_SSID_TOO_LONG** = 5200, **UPOS_WIFI_↩ PASSWORD_TOO_LONG**, **UPOS_WIFI_SSID_EMPTY_STRING** }
- enum **upos_wifi_event_type_e** {
  WIFI_EV_DISCONNECT = 1, **WIFI_EV_DISCONNECT_ERROR**, **WIFI_EV_RECONNECT**, **WIFI_EV_R↩ ECONNECT_ERROR**,
  WIFI_EV_CONNECT, **WIFI_EV_CONNECT_ERROR**, **WIFI_EV_GOT_IP**, **WIFI_EV_GOT_IPV6_IP** }

Functions

- int **upos_wifi_start** (const char ∗, const char ∗, **UPOS_WIFI_CB_CTX** ∗)
- const char ∗ **get_ip_string** ()
- const char ∗ **get_ip6_string** ()
- int **upos_wait_connect** (TickType_t)
- int **upos_get_wifi_error** ()
- const char ∗ **upos_get_error_message** ()
- const char ∗ **get_ip_string_util** (int)
- int **upos_wifi_stop_util** (int destroy)
- void **upos_wifi_set_event_error_cb** (**upos_wifi_cb**)
- void **upos_wifi_delete_event_error_cb** ()
- const char ∗ **get_ssid** ()
- int **upos_is_wifi_enabled** ()

5.19.1 Macro Definition Documentation

5.19.1.1 _UPOS_IP4

```
#define _UPOS_IP4 (int)1
```

Definition at line **64** of file **upos_wifi.h**.

5.19.1.2 _UPOS_IP6

```
#define _UPOS_IP6 (int)0
```

Definition at line **63** of file **upos_wifi.h**.

5.19.1.3 upos_get_ip6_string

```
#define upos_get_ip6_string( )  get_ip_string_util( _UPOS_IP6)
```

Definition at line **66** of file **upos_wifi.h**.

5.19.1.4 upos_get_ip_string

```
#define upos_get_ip_string( )  get_ip_string_util( _UPOS_IP4)
```

Definition at line **65** of file **upos_wifi.h**.

5.19.1.5 upos_wifi_disconnect

```
int upos_wifi_disconnect( )  upos_wifi_stop_util(0)
```

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

5.19.1.6 UPOS_WIFI_EVENT_CALLBACK

```
#define UPOS_WIFI_EVENT_CALLBACK(
              param ) ((size_t)offsetof(struct  upos_wifi_cb_ctx_t, param))
```

Definition at line **47** of file **upos_wifi.h**.

5.19.1.7 upos_wifi_stop

```
#define upos_wifi_stop( )  upos_wifi_stop_util(-2)
```

Definition at line **69** of file **upos_wifi.h**.

5.19.2 Typedef Documentation

5.19.2.1 UPOS_WIFI

```
typedef struct  upos_wifi_status_t  UPOS_WIFI
```

5.19.2.2 upos_wifi_cb

```
typedef void(* upos_wifi_cb) (void *)
```

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

5.19.2.3 UPOS_WIFI_CB_CTX

```
typedef struct  upos_wifi_cb_ctx_t  UPOS_WIFI_CB_CTX
```

5.19.2.4 UPOS_WIFI_ERR

```
typedef enum  upos_wifi_err_t  UPOS_WIFI_ERR
```

5.19.2.5 UPOS_WIFI_EVENT_CTX

```
typedef struct  upos_wifi_event_cb_ctx_t  UPOS_WIFI_EVENT_CTX
```

5.19.2.6 UPOS_WIFI_EVENT_ENUM

```
typedef enum  upos_wifi_event_type_e  UPOS_WIFI_EVENT_ENUM
```

5.19.3 Enumeration Type Documentation

5.19.3.1 upos_wifi_err_t

```
enum  upos_wifi_err_t
```

Enumerator

UPOS_WIFI_OK	
UPOS_WIFI_SSID_TOO_LONG	
UPOS_WIFI_PASSWORD_TOO_LONG	
UPOS_WIFI_SSID_EMPTY_STRING	

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

5.19.3.2 upos_wifi_event_type_e

```
enum upos_wifi_event_type_e
```

Enumerator

WIFI_EV_DISCONNECT	
WIFI_EV_DISCONNECT_ERROR	
WIFI_EV_RECONNECT	
WIFI_EV_RECONNECT_ERROR	
WIFI_EV_CONNECT	
WIFI_EV_CONNECT_ERROR	
WIFI_EV_GOT_IP	
WIFI_EV_GOT_IPV6_IP	

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

5.19.4 Function Documentation

5.19.4.1 get_ip6_string()

```
const char* get_ip6_string ( )
```

5.19.4.2 get_ip_string()

```
const char* get_ip_string ( )
```

5.19.4.3 get_ip_string_util()

```
const char* get_ip_string_util (
             int  )
```

5.19.4.4 get_ssid()

```
const char* get_ssid ( )
```

5.19.4.5 upos_get_error_message()

```
const char* upos_get_error_message ( )
```

5.19.4.6 upos_get_wifi_error()

```
int upos_get_wifi_error ( )
```

5.19.4.7 upos_is_wifi_enabled()

```
int upos_is_wifi_enabled ( )
```

5.19.4.8 upos_wait_connect()

```
int upos_wait_connect (
             TickType_t  )
```

5.19.4.9 upos_wifi_delete_event_error_cb()

```
void upos_wifi_delete_event_error_cb ( )
```

5.19.4.10 upos_wifi_set_event_error_cb()

```
void upos_wifi_set_event_error_cb (
               upos_wifi_cb  )
```

5.19.4.11 upos_wifi_start()

```
int upos_wifi_start (
              const char * ,
              const char * ,
               UPOS_WIFI_CB_CTX *  )
```

5.19.4.12 upos_wifi_stop_util()

```
int upos_wifi_stop_util (
              int destroy )
```

5.20 upos_wifi.h

```
00001 #include "esp_event.h"
00002
00003 typedef enum upos_wifi_err_t {
00004    UPOS_WIFI_OK = 0,
00005    UPOS_WIFI_SSID_TOO_LONG = 5200,
00006    UPOS_WIFI_PASSWORD_TOO_LONG,
00007    UPOS_WIFI_SSID_EMPTY_STRING
00008 } UPOS_WIFI_ERR;
00009
00010 typedef struct upos_wifi_status_t {
00011    int err;
00012    const char *tag;
00013 } UPOS_WIFI;
00014
00015 typedef void (*upos_wifi_cb)(void *);
00016 typedef struct upos_wifi_cb_ctx_t {
00017    void *on_connect_ctx;
00018    upos_wifi_cb on_connect;
00019    void *on_disconnect_ctx;
00020    upos_wifi_cb on_disconnect;
00021    void *on_ipv4_ctx;
00022    upos_wifi_cb on_ipv4;
00023    void *on_ipv6_ctx;
00024    upos_wifi_cb on_ipv6;
00025 } UPOS_WIFI_CB_CTX;
00026
00027 typedef enum upos_wifi_event_type_e {
00028    WIFI_EV_DISCONNECT = 1,
00029    WIFI_EV_DISCONNECT_ERROR,
00030    WIFI_EV_RECONNECT,
00031    WIFI_EV_RECONNECT_ERROR,
00032    WIFI_EV_CONNECT,
00033    WIFI_EV_CONNECT_ERROR,
00034    WIFI_EV_GOT_IP,
00035    WIFI_EV_GOT_IPV6_IP
00036 } UPOS_WIFI_EVENT_ENUM;
00037
00038 typedef struct upos_wifi_event_cb_ctx_t {
00039    int err;
00040    const char *event_name;
00041    uint32_t event_type;
00042    void *initial_ctx;
00043 //   void *ctx;
```

```
00044 } UPOS_WIFI_EVENT_CTX;
00045
00046
00047 #define UPOS_WIFI_EVENT_CALLBACK(param) ((size_t)offsetof(struct upos_wifi_cb_ctx_t, param))
00048
00049 int upos_wifi_start(const char *, const char *, UPOS_WIFI_CB_CTX *);
00050 int upos_wifi_disconnect();
00051 //int upos_wifi_stop();
00052 const char *get_ip_string();
00053 const char *get_ip6_string();
00054 int upos_wait_connect(TickType_t);
00055 int upos_get_wifi_error();
00056 const char *upos_get_error_message();
00057 const char *get_ip_string_util(int);
00058 int upos_wifi_stop_util(int destroy);
00059 void upos_wifi_set_event_error_cb(upos_wifi_cb);
00060 void upos_wifi_delete_event_error_cb();
00061 const char *get_ssid();
00062 int upos_is_wifi_enabled();
00063 #define _UPOS_IP6 (int)0
00064 #define _UPOS_IP4 (int)1
00065 #define upos_get_ip_string() get_ip_string_util(_UPOS_IP4)
00066 #define upos_get_ip6_string() get_ip_string_util(_UPOS_IP6)
00067
00068 #define upos_wifi_disconnect() upos_wifi_stop_util(0)
00069 #define upos_wifi_stop() upos_wifi_stop_util(-2)
00070
00071
```

Index