NAME

nano − Tcl bindings for Nano

SYNOPSIS

nano::

address::

toPublicKey *address* ?**-hex**|**-binary**? ?**-verify**|**-no-verify**?
fromPublicKey *pubKey* ?**-xrb**|**-nano**?
fromPrivateKey *privateKey* ?**-xrb**|**-nano**?

key::

newSeed ?**-hex**|**-binary**?
newKey ?**-hex**|**-binary**?
fromSeed *seed* ?*index*? ?**-hex**|**-binary**?
publicKeyFromPrivateKey *privateKey* ?**-hex**|**-binary**?

block::

json::toBlock *blockJSON*
json::fromDict *blockDict*
json::fromBlock *blockData* ?**-xrb**|**-nano**? ? **-type=***blockType* ? ? **-signKey=***privateKey* ?
json::sign *blockJSON privateKey* ?**-update**|**-signature** ?**-hex**|**binary**??
json::verifySignature *blockJSON*
json::work *blockJSON* ?**-update**|**-work** ?**-hex**|**-binary**??
json::validateWork *blockJSON*

dict::toBlock *blockDict*
dict::fromJSON *blockJSON*
dict::fromBlock *blockData* ?**-xrb**|**-nano**? ? **-type=***blockType* ? ? **-signKey=***privateKey* ?
dict::sign *blockDict privateKey* ?**-update**|**-signature** ?**-hex**|**binary**??
dict::verifySignature *blockDict*
dict::work *blockDict* ?**-update**|**-work** ?**-hex**|**-binary**??
dict::validateWork *blockDict*

hash *blockData* ?**-hex**|**-binary**?
signBlockHash *blockHash privateKey* ?**-hex**|**-binary**?
sign *blockData privateKey* ?**-hex**|**-binary**?
verifyBlockHash *blockHash signature publicKey*
verify *blockData signature publicKey*

create::send *args*
create::receive *args*
create::setRepresentative *args*

work::

fromWorkData *blockHashOrPublicKey*
fromBlock *blockData*
validate *workData work*

account::

setFrontier *account frontierHash balance representative*
getFrontier *account*
getFrontier *account* ?**frontierHash**|**balance**|**representative**?
addPending *account blockHash amount*
getPending *account* ?*blockHash*?
clearPending *account* ?*blockHash*?

receive *account blockHash privateKey*
receiveAllPending *account privateKey*
send *fromAccount toAccount amount privateKey*
setRepresentative *account representative privateKey*

INTRODUCTION

Nano is a low-latency payment platform that requires minimal resources, relying on a peer-to-peer network to distribute "blocks", which are cryptographically signed transactions. This package provides bindings for interacting with the Nano network from *Tcl*.

Nano uses Ed25519 with Blake2b as the cryptographic hashing primitive for digital signatures, rather than the common construction of Ed25519 with the SHA2-512 cryptographic hashing function.

Nano implements a "blockchain", which is a cryptographic linked-list, by identifying every "block" by its crytographic hash and providing a pointer from every block to its predecessor in the "chain" as part of the hashed data.

This predecessors is referred to here as the "previous" block. In Nano, each account has its own blockchain and they reference each other using a data structure referred to as "block lattice", where the individual chains contain blocks that reference blocks in other chains to tie them together. The field within blocks that reference other blocks on a different blockchain is referred to as either the "link" field or "source block hash".

Each Nano block also encapsulates the full state of the account, containing, at a minimum, a tuple of (*account*, *balance*, *representative*, *previous*).

Since Nano blocks are signed by independent actors, who may, for their own gain, generate multiple valid blocks referring to the same predecessor (*previous*) block, an arbitration mechanism is employed by the Nano network to decide which blocks are valid within a given chain. This arbitration mechanism operates on the principles of consensus. Each account holder has a stake in the network operating nominally, otherwise the balance represented by an account is not useful for a transfer of value. In Nano the stake an account has in the network is equal to the account's balance. The larger the stake an account has the more incentivized the account-holder is to ensure the network is operating nominally and not accepting multiple blocks that reference the same predecessor.

Nano utilizes a mechanism called *voting* to determine which blocks are valid and which blocks are not valid. Each stakeholder votes their stake upon seeing a new subordinate block (*i.e.*, a block with a unique *previous* value). Since voting is an active and on-going process that occurs on the Nano peer-to-peer network, participants must be online to vote their stake. As this is often inconvienent or impossible, stakeholders may select another stakeholder to vote their share of the network. This delegate is referred to as a *representative*.

Representatives should be chosen carefully by stakeholders since malicious representatives may attempt to gather voting power and destablize the Nano network by altering decisions made by consensus previously.

Nano accounts are referred to by address. A Nano address starts with the prefix "**nano_**" or "**xrb_**". A Nano address is actually the public portion of a private/public keypair, plus the prefix, and a checksum to ensure that no digits are mistyped by users when communicating them. Nano public keys are 256-bit keys in the Ed25519 algorithm.

A user may have many accounts. To simplify the process of maintaining the private/public keypairs for all the accounts, Nano supports the concept of a *wallet*. A*wallet* is a conceptual entity that is used to refer to a *seed*, which is a random 256-bit number that can be used to derive mulitple private/public keypairs from.

Balances in Nano are stored in a 128-bit integer value. There are various units for representing the balance, the smallest and base unit is called "*raw*". The most common unit for users to use is called "*Nano*", one of which is equal to 1e30 raw.

PROCEDURES

Addresses

::nano::address::toPublicKey

address ?**-hex**|**-binary**? ?**-verify**|**-no-verify**? **->** *publicKey*

Converts a Nano address to a public key. The**-hex** option indicates that the public k ey should be returned in hexidecimal form. The option indicates that the public key should be returned in binary form. The **-verify** option verifies the checksum embedded in the Nano address before returning. The**-no-v erify** option inhibits verifying the checksum embedded in the Nano address.

::nano::address::fromPublicKey

pubKey ?**-xrb**|**-nano**? **->** *address*

Converts a public key to a Nano address. The option specifies that the returned address should be prefixed with the old-style "xrb_" prefix, where the **-nano** option specifies that the returned address should be prefixed with the new-style "nano_" prefix.

::nano::address::fromPrivateKey

privateKey ?**-xrb**|**-nano**? **->** *address*

Converts a private key to a Nano address. It accepts the same arguments as **fromPublicKey**.

Key Management

::nano::key::newSeed

?**-hex**|**-binary**? *->seed*

Generates a new seed. A seed is a 256-bit bitfield which, along with a 32-bit index, is used to derive enumerated keys from a single point of entropy. See the**fr omSeed** procedure. The **-hex** and **-binary** options determine the formatting of the result.

::nano::key::newKey

?**-hex**|**-binary**? *->privateK ey*

Generates a new private key. A private key can be used to sign transactions, which can then be verified with its cooresponding public key (see **publicKeyFromPrivateKey**). This procedure is normally not used, but rather private keys are derived from a *seed* and *index* pair using the **fromSeed** procedure. The **-hex** and **-binary** options determine the formatting of the result.

::nano::key::fromSeed

seed ?*index*? ?**-hex**|**-binary**? *->privateK ey*

Derive a private key from the seed specified as *seed* and the *index* indicated. This procedure is deterministic (i.e., the same *seed* and *index* will always give you the same private key). This procedure is used to derive many keypairs from a single user-managed piece of data, so the user does not have to manage multiple private keys. If the*inde x* is not specified it defaults to **0**. The**-hex** and **-binary** options determine the formatting of the result.

::nano::key::publicKeyFromPrivateKey
privateKey ?**-hex**|**-binary**? *->publicK ey*

Converts a private key into its corresponding public key. Normally Ed25519 private keys are a concatencation of the private and public keys, however in this package they are each treated separately. The**-hex** and **-binary** options determine the formatting of the result.

Low-level Block

::nano::block::*representation***::toBlock**
blockRepresentation -> blockData

Converts from one of the internal representations (either Tcl dictionary or JSON) to a Nano block. The *representation* portion of the command name may be one of **dict** or **json**.

::nano::block::json::fromDict
blockDict -> blockJSON

Converts from a Tcl dictionary representation to a JSON representation of a block.

::nano::block::dict::fromJSON
blockJSON -> blockDict

Converts from a JSON object representation to a Tcl dictionary representation of a block.

::nano::block::*representation***::fromBlock**
blockData ?**-xrb**|**-nano**? ? **-type=***blockType* ? ? **-signKey=***privateKey* ? *-> blockRepresentation*

Parses a Nano block and returns either a Tcl dictionary or a JSON object. The **-xrb** option causes all parsed addresses to be prefixed with the old-style "xrb_" address prefix, while the **-nano** option causes them to be prefixed with the new-style "nano_prefix". The *representation* portion of the command name may be one of **dict** or **json**.

::nano::block::*representation***::sign**
blockRepresentation privateKey ?**-update**|**-signature** ?**-hex**|**binary**?? *->signatur e|blockJSON*

Sign a block, in either Tcl dictionary or JSON representation, with the specified *privateKey*. If the **-update** option is used, return the object with the updated attribute. If the**-signatur e** option is used, return just the signature. The **-hex** and **-binary** options determine the formatting of the result. Ther *epresentation* portion of the command name may be one of **dict** or **json**.

::nano::block::*representation***::verifySignature**
blockRepresentation -> boolean

Verify the signature on a block, in either Tcl dictionary or JSON representation, matches the public key specified in the **account** attribute of that object. This may not work correctly for old-style blocks unless you manually add the **account** attribute. Ther *epresentation* portion of the command name may be one of **dict** or **json**.

::nano::block::*representation***::work**
blockRepresentation ?**-update**|**-work** ?**-hex**|**binary**?? *->work|bloc kRepresentation*

Generate proof-of-work (PoW) required to submit a given block to the network. Nano uses PoW to increase the cost of submitting blocks to the network to cut down on spam. The *work* that is computed is based on the hash of the previous block on this chain, or if there is no previous block on this chain (i.e., because it is the first block on an account) the public key of the account. If the **-update** option is used, return the object with the updated attribute. If the**-w ork** option is used, just return the work. The**-hex** and **-binary** options determine the formatting of the result. The *representation* portion of the command name may be one of **dict** or **json**.

::nano::block::*representation***::validateWork**
blockRepresentation -> boolean

Validate the proof-of-work (PoW) in the object specified as *blockRepresentation* with the attribute **work** is valid for the block passed in. The *representation* portion of the command name may be one of **dict** or **json**.

::nano::block::hash
blockData ?**-hex**|**-binary**? *->bloc kHash*

Compute the cryptographic hash of a block. The cryptographic hashing algorithm used for Nano is Blake2b. Blocks are typically identified by their hash (i.e., content addressable). The**-hex** and **-binary** options determine the formatting of the result.

::nano::block::signBlockHash
blockHash privateKey ?**-hex**|**-binary**? *->signatur e*

Compute an Ed25519-with-Blake2b signature of a given block hash specified as *blockHash* with the private key specified as *privateKey*. In Nano, signed blocks are signed by signing the block's hash thus all that is needed to sign a block is its hash and the private key that corresponds to the account. **NOTE: Ensure that the** *privateKey* **specified matches the account the block belongs to.** The **-hex** and **-binary** options determine the formatting of the result.

::nano::block::sign
blockData privateKey ?**-hex**|**-binary**? *->signatur e*

This is a convienence procedure which computes the hash of a block given as *blockData*, and then calls **signBlockHash**. The**-hex** and **-binary** options determine the formatting of the result.

::nano::block::verifyBlockHash
blockHash signature publicKey -> boolean

Verify that a block hash (*blockHash*) was signed (*signature*) by an account holding the private key that corresponds to the public key specified as *publicKey*.

::nano::block::verify
blockData signature publicKey -> boolean

This is a convienence procedure which computes the hash of a block given as *blockData*, and then calls **verifyBlockHash**.

::nano::block::create::send
from *address* **to** *address* **previous** *blockHash* **representative** *address* **previousBalance** *integer* **amount** *integer* ? **-json** *boolean* ? *-> blockJSON|blockDict*

This is a low-level interface for creating blocks which correspond to sending Nano from one account to another. It constructs a block which sends the**amount** specified from the **fr om** address to the destination (**to**). The previous block's hash must be specified as the *blockHash* following **previous**. Additionally the balance of the account at the previous block must be supplied as the integer argument to **previousBalance**. All balance amounts are in units of**raw**. If the optional **-json** argument is used and specified as true the result is a JSON representation, otherwise a Tcl dict representation is used.

::nano::block::create::receive
to *address* **sourceBlock** *blockHash* **previous** *blockHash* **representative** *address* **previousBalance** *integer* **amount** *integer* ? **-json** *boolean* ? *-> blockJSON|blockDict*

This is a low-level interface for creating blocks which correspond to receiving (pocketing) Nano previously sent from another account to the account specified as the *address* supplied to the **to** argument. It constructs a block which receives the amount of Nano specified as the **amount** argument. The block hash (*blockHash*) of the send block which was used to send the Nano to this account must be specified as the argument to the **sourceBlock** option. The previous block's hash must be specified as the *blockHash* following **previous**. Additionally the balance of the account at the previous block must be supplied as the integer argument to **previousBalance**. All balance amounts are in units of **raw**. If the optional**-json** ar gument is used and specified as true the result is a JSON representation, otherwise a Tcl dict representation is used.

::nano::block::create::setRepresentative
account *address* **previous** *blockHash* **representative** *address* ? **-json** *boolean* ? *-> blockJSON|blockDict*

This is a low-level interface for creating blocks which correspond to an explicit change of representative. Representatives in Nano are used as part of the Delegated Proof-of-Stake (dPoS) consensus mechanism which is used by the Nano network to determine which block (if any) out of many possible subordinate blocks in a chain are valid. So that every account holder does not have to be online to vote for valid transactions, an account may delegate another account to vote its stake on its behalf. That delegate is called a representative. An account may change its representative at any time by issuing a block with a new representative, such as a send or receive block, or by issuing an explicit change of representative block. This procedure creates an explicit change of representative block for the **account** specified. It changes to the delegate to the **representative** specified. Further, the *blockHash* of the previous block must be specified as the argument to **previous**. If the optional**-json** ar gument is used and specified as true the result is a JSON representation, otherwise a Tcl dict representation is used.

Work Generation

::nano::work::fromWorkData
blockHashOrPublicKey -> work

Create proof-of-work (PoW) from a block hash or public key. Which one is used depends on whether or not there are any other blocks in this account's chain. If this is the first block in this account's chain then the public key of the account is used, otherwise the hash of the blocks predecessor (*previous*) is used. The specific value needed should be accessible from the **_workData** member of a JSON object or Tcl dictionary. Note that this attribute (and all attributes that begin with an underscore) should be discarded when sending the block outside of the Tcl process.

::nano::work::fromBlock
blockData -> work

This is a convienence procedure which computes work data (either a block hash or a public key) for a given block and then calls **fromWorkData**.

::nano::work::validate
workData work -> boolean

This procedure validates that the supplied *work* is valid for the supplied *workData*, which is either a block hash or an account public key. For more information see the description of **fromWorkData**.

High-level Account

:nano::account::setFrontier
account frontierHash balance representative

This procedure is used as part of the High-level Account interface. It sets the*frontier*, which is the block hash (*frontierHash*) and data (*balance*, *representative*) associated with that block that corresponds to the head of an account's chain.

:nano::account::getFrontier
account -> frontierInfo

This procedure is used as part of the High-level Account interface. It gets the Tcl dictionary associated with the frontier most recently set for the specified *account*.

:nano::account::getFrontier
account ?**frontierHash**|**balance**|**representative**? *->frontierHash|balance|representative*

This procedure is used as part of the High-level Account interface. It gets a specific item from Tcl dictionary associated with the frontier most recently set for the specified *account*.

:nano::account::addPending
account blockHash amount

This procedure is used as part of the High-level Account interface. It is used to indicate than a given *account* has a **receive** block that they could create. The block hash of the corresponding **send** block should be supplied as the *blockHash* parameter. The amount of Nano that was sent in the **send** block should be specified as the *amount* parameter (in units of raw).

:nano::account::getPending
account ?*blockHash*? *->dict*

This procedure is used as part of the High-level Account interface. It is used to retrieve information stored by **addPending** for a given *account*. If the*blockHash* parameter is supplied then a Tcl dictionary is returned with a key called **amount** which contains the amount stored previously. If the *blockHash* parameter is not supplied then a Tcl dictionary is returned with keys corresponding to each block hash pending for the specified *account*, and containing a subordinate Tcl dictionary with a key called **amount** as previously described.

::nano::account::clearPending
account ?*blockHash*?

This procedure is used as part of the High-level Account interface. It is used to clear (that is, remove from the conceptual state of "pending") entries created previously with **addPending** for a given *account*. If the*bloc kHash* parameter is supplied then only the entry corresponding to that blockhash is cleared, otherwise all entries for the specified *account* are cleared.

:nano::account::receive
account blockHash privateKey -> blockJSON|blockDict

This procedure is used as part of the High-level Account interface. It is used to generate a receive block. Its interface is subject to change and not considered stable.

:nano::account::receiveAllPending
account privateKey -> listOfBlockJSON|listOfBlockDict

This procedure is used as part of the High-level Account interface. It is used to generate receive blocks for every pending receive on a given *account*. Its interface is subject to change and not considered stable.

:nano::account::send
fromAccount toAccount amount privateKey -> blockJSON|blockDict

This procedure is used as part of the High-level Account interface. It is used to generate a send block. Its interface is subject to change and not considered stable.

::nano::account::setRepresentative
account representative privateKey -> blockJSON|blockDict

This procedure is used as part of the High-level Account interface. It is used to generate a block that changes the representative for the given *account*. Its interface is subject to change and not considered stable.

EXAMPLES

Example 1

```
package require nano 1.0

set seed [::nano::key::newSeed -hex]
puts "Generated seed: $seed"

for {set index 0} {$index < 10} {incr index} {
      set accountPrivateKey [::nano::key::fromSeed $seed $index -hex]
      set accountAddress [::nano::address::fromPrivateKey $accountPrivateKey]
      puts "    - $index: $accountAddress"
}
```

Example 2

Example 3

AUTHOR

Roy Keene <*rkeene@nano.org*>