Technical specifications

NyzoSy and Iyomisc

nyzo

Version 1.1

Introduction

Nyzo main strengths

- Diversity – Fairer than PoW and PoS. Time and ipv4 as scarce resources.
- Fast finality – not just 7s block time: once a block is frozen, it's guaranteed to be final.
- Cycle fund with full control by > 50% of the cycle signatures
- Resources efficient

Nytro: Do not tamper with efficient core working, but build on top

- Second layer protocol, only to be processed by applications having an interest.

New features:

- ERC-20 like token
- With Nyzo strengths
- With more safeguards than ETH tokens
- Capable enough to fulfill many use cases.

A second layer protocol

No change to the core layer:

- The verifiers only deal with first layer, regular Nyzo transactions
- They have no knowledge of the layers above
- They will not interpret the second layer data, nor process additional rules.
- They do not need any change to code, consensus or anything
- Verifiers continue to freeze transactions with their associated data, like they already do.

Protocol rules:

- The protocol adds rules to the sender, recipient and data fields of the regular Nyzo transactions.
- The protocol rules define what is a layer-2 valid or invalid transaction.
- The state of the layer-2 objects can be rebuilt anytime from the historical layer-1 transactions, with no extra data.
- Every valid transaction updates the state of the layer-2 objects

Protocol implementations:

- Like with verifiers, there can exist several implementations
- Test vectors are provided to test the implementations with various rules
- In case of conflict, the reference implementation is to be trusted, test vectors added and rules made more clear if needed.

Some insights

Human readable data:

- The protocol was defined so the data can be read by a human. Ascii data, no encoding. One can look through the transactions and read what they do.

Helpers:

- The transactions are however intended to be created through helpers, that make sure you don't send invalid data. We provide online helpers as well as a generic python helper class anyone can use.

Fees:

- Transfer and Ownership transfer operations use the token recipient as Nyzo recipient. This is required due to Nyzo small data field. No other fee than the regular tx fee can then be collected in a direct manner. These operations are then basically for free.
- Token issuance, Mints and Burns are sent to the cycle address. Technically, these 3 can require any Nyzo amount to be paid. At start, Mint and Burn will not require a fee.
- Since required fees are at protocol level, they can change if needed. The cycle could vote on them for instance.

Common tokens properties

Token Name:

- Only one token of a given name can exist. No possible confusion.
- Token name has to follow strict rules, charset and case. No trick with utf8 charsets.

Token Units:

- Every token has a maximum number of decimals, defined at creation time.
- Max possible number of decimals is 18, to be on par with ETH.
- Tokens can be defined with no decimal. Integer, indivisible tokens.

Fees:

- Transfer and Ownership transfer of a token currently have minimal fees: 0.000001 Nyzos
- Token issuance has a 100 Nyzos fee. Objective: Limit spam, limit cybersquatting, reward cycle.
- Every Mint of mint-able tokens could have a fee. Objective: reward the cycle. Current Fees: minimal fees 0.000001 Nyzos

Anatomy of a token transaction 1/2

Regular Nyzo transaction

A Regular Nyzo transaction, from user point of view, consists of:

- A Sender (Nyzo address)
- A Recipient (Nyzo address)
- A Nyzo amount (Nyzos, with micro Nyzo precision)
- A Sender data field of 32 bytes.

Token transaction:

A token transaction is a regular Nyzo transaction making use of the Recipient and Sender data fields to convey additional meaning.

- Nyzo verifiers handle that transaction as a regular transaction, only storing the Sender data.
- Nytro Protocol aware clients leverage the extra data to infer tokens operations and related state changes.

Anatomy of a token transaction 2/2

Use of the Recipient address

With token transactions, Recipient is either the recipient of the token operation (ex: token transfer) or the cycle address (ex: issue, mint, burn).

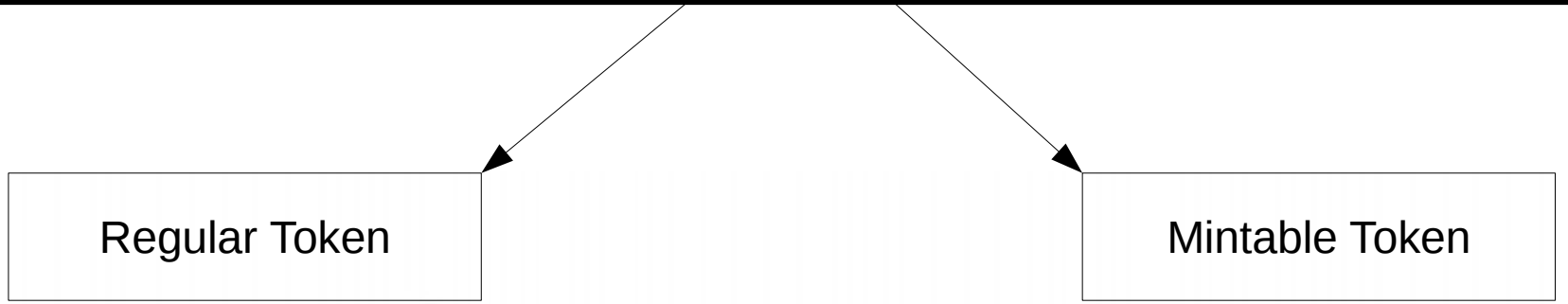
Using the cycle address as Recipient allows to pay fees that can then be used by the cycle fund.

Sender data format

Ex: “TT:TEST:2”

- Sender data is specified as Ascii text only. It can be read by a human no matter the local encoding.
- “:” char is used as a separator
- First segment denotes the token operation, here : “TT” for Token Transfer. First char is always “T” for Token protocol. This is a compressed namespace.
- Following segments are the parameters of that transfer operation: token name and token amount.

Two kinds of tokens



- Fixed supply
- Fixed decimals

Supports:

- TI: Token Issuance
  Initial definition of the token and emission of the full supply to the issuer.
- TT: Token Transfer
  Transfer of token units from sender to recipient.

- Elastic supply
- Fixed decimals

- TI: Token Issuance
  Initial supply is null.
- TT: Token Transfer
- TM: Token Mint
  Current owner can mint any number of token units, adds to supply.
- TB: Token Burn
  Destroy token units, remove from supply.
- TO: Token Ownership change
  Transfers ownership (mint rights) to the recipient.

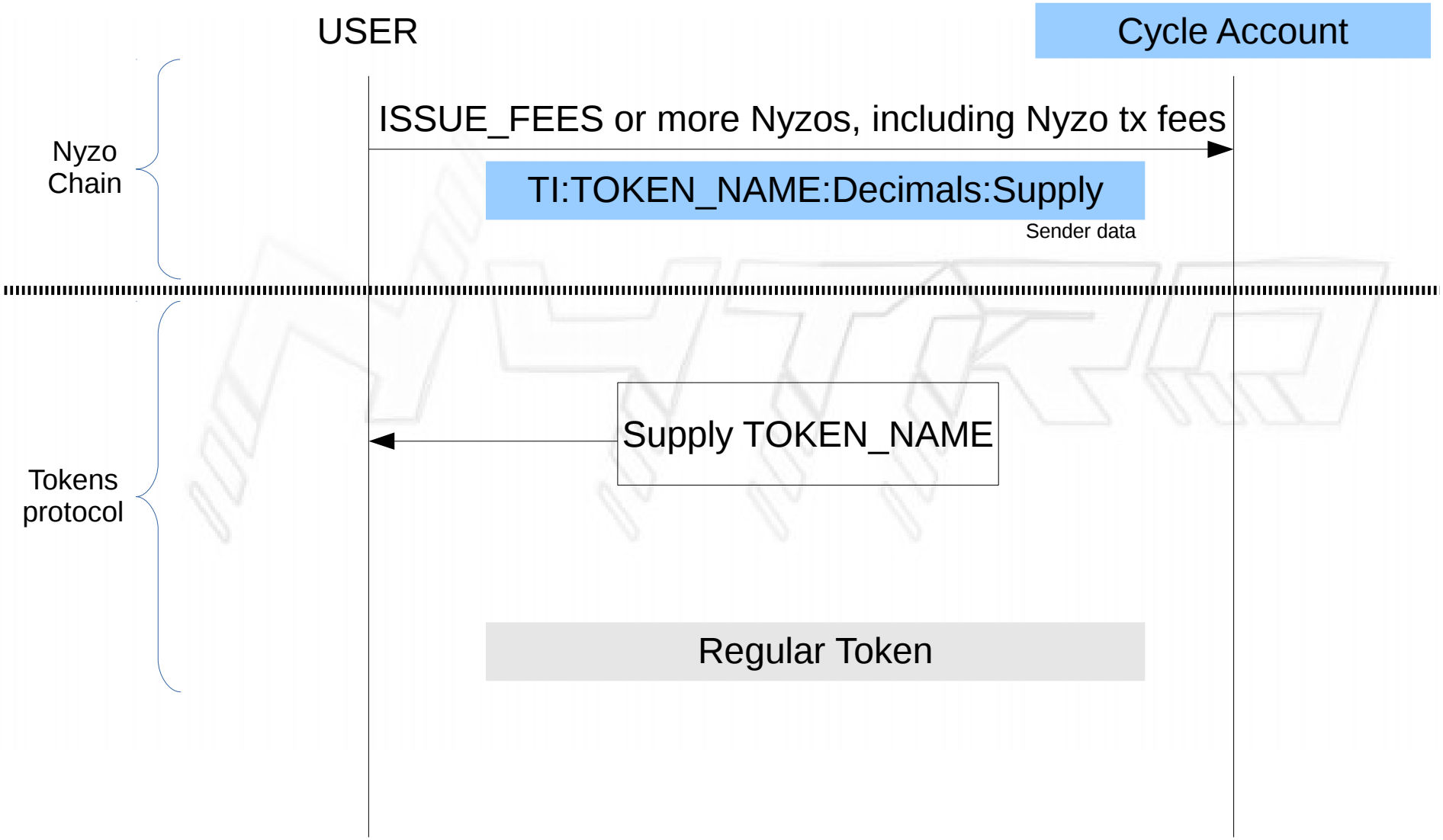
TOKENS OPERATIONS DETAILS

NyzoSy and Iyomisc

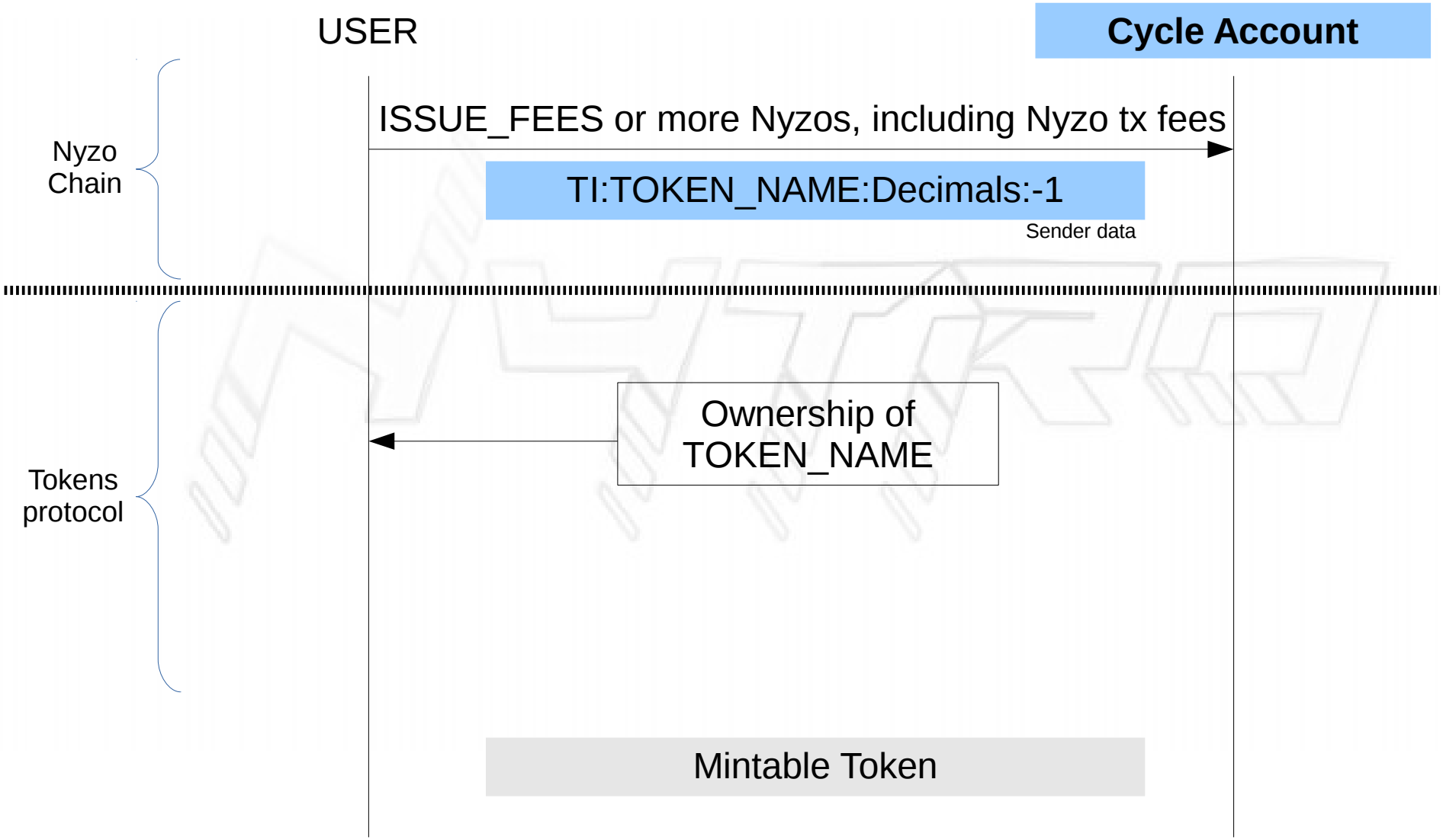
nyzo

Version 1.1

ISSUE – TI:


USER
Cycle Account
Nyzo Chain
ISSUE_FEES or more Nyzos, including Nyzo tx fees
TI:TOKEN_NAME:Decimals:Supply
Sender data
Supply TOKEN_NAME
Tokens protocol
Regular Token

ISSUE – TI:


USER
Cycle Account
Nyzo Chain
ISSUE_FEES or more Nyzos, including Nyzo tx fees
TI:TOKEN_NAME:Decimals:-1
Sender data
Ownership of TOKEN_NAME
Tokens protocol
Mintable Token

TI: Rules

- TOKEN_NAME – The unique name of the token

  A..Z, 0..9, _
  Min 3 chars – **Max 10 chars**
  Valid: TEST, MY_TOKEN, TOK123
  Invalid: Test, MY-TOKEN, TOK@12

- Decimals – The number of possible decimals for this token amounts (max decimals = 18)

  dnn, with n=0..9 – nn <= 18
  Valid: d00, d18
  Invalid: 0, d20, dff
  Note: any transaction related to this token, with an amount specifying more decimals than the token allows will be invalid.

- Supply – The initial Supply of the token, or mintable token

  Integer or float as base 10 decimal, with dot as decimal separator and no thousand separator
  Supply is strictly greater than 0 or equal to -1
  If supply is "-1", this denotes a mintable token with elastic supply that can then be minted and burnt.
  Valid: 1, 10, 1000000, -1, 3.14
  Invalid: 0, 1 000, 000, "1,000,000"
  Note: Supply can be a decimal but is also bound to the maximum number of decimals of the token.

- Whole rules – Apply on the whole TI: transaction

  Max total length is 32 chars, including "TI:" and the ":" separators
  "TI:" transaction has to be sent to the cycle account "xxxxxx" to be valid
  "TI:" transaction has to send at least ISSUE_FEES Nyzos to the cycle, with ISSUE_FEES currently being XXX Nyzos, subject to change after cycle vote.

- **New since v1.1:** Max token name length is now 10 chars. This allows for 37^10 = 4.8E15 combinations.
  This restriction could be lighten in the future.

Edited 1.1

TI: Validation flow and rules

- Trigger: Data begins with “TI:” - case sensitive.
- Split by “:” delimiter
- TI_R0: consists of exactly 4 segments “segment0:segment1:segment2:segment3”
- TI_R1a: segment1 follows TOKEN_NAME rules, **including 10 chars max len** constraint
- TI_R1b: segment1 is not an already issued TOKEN_NAME
- TI_R2a: segment2 obeys the decimal “dnn” rules. Regexp: /^d[0-9]{2}$/
- Extract “decimals” as integer.
- TI_R2b: decimals <= 18 and >= 0
- TI_R3a: segment3 obeys supply format. Regexp: /^[0-9.]+$/
- TI_R3b: segment3 has no more than “decimals” decimals
- Extract “Supply” as integer by converting from string*.
- TI_R3c: Supply is > 0 or equals -1
- TI_R4a: recipient is cycle address
- TI_R4b: amount of Nyzos sent – including native Nyzo tx fees - is equal or higher than current ISSUE_FEES
- Note: The order in which the rules are evaluated is not important since any broken rule renders the token transaction invalid. In practice and for performance reasons low costs rules may be tested first. A canonical test order will be provided for test suites.

Edited 1.1

TI: Example

I want to create a new Nyzo token, named “EXAMPLE”. That name follows the rules and does not exist yet.

I don't need decimals and want a fixed supply of one million tokens.

I send a 100 Nyzos (ISSUE_FEES) transaction to the cycle address “**id__8000000000000000000000000000000000000002kmiC2yjF**” with the data properly encoded.

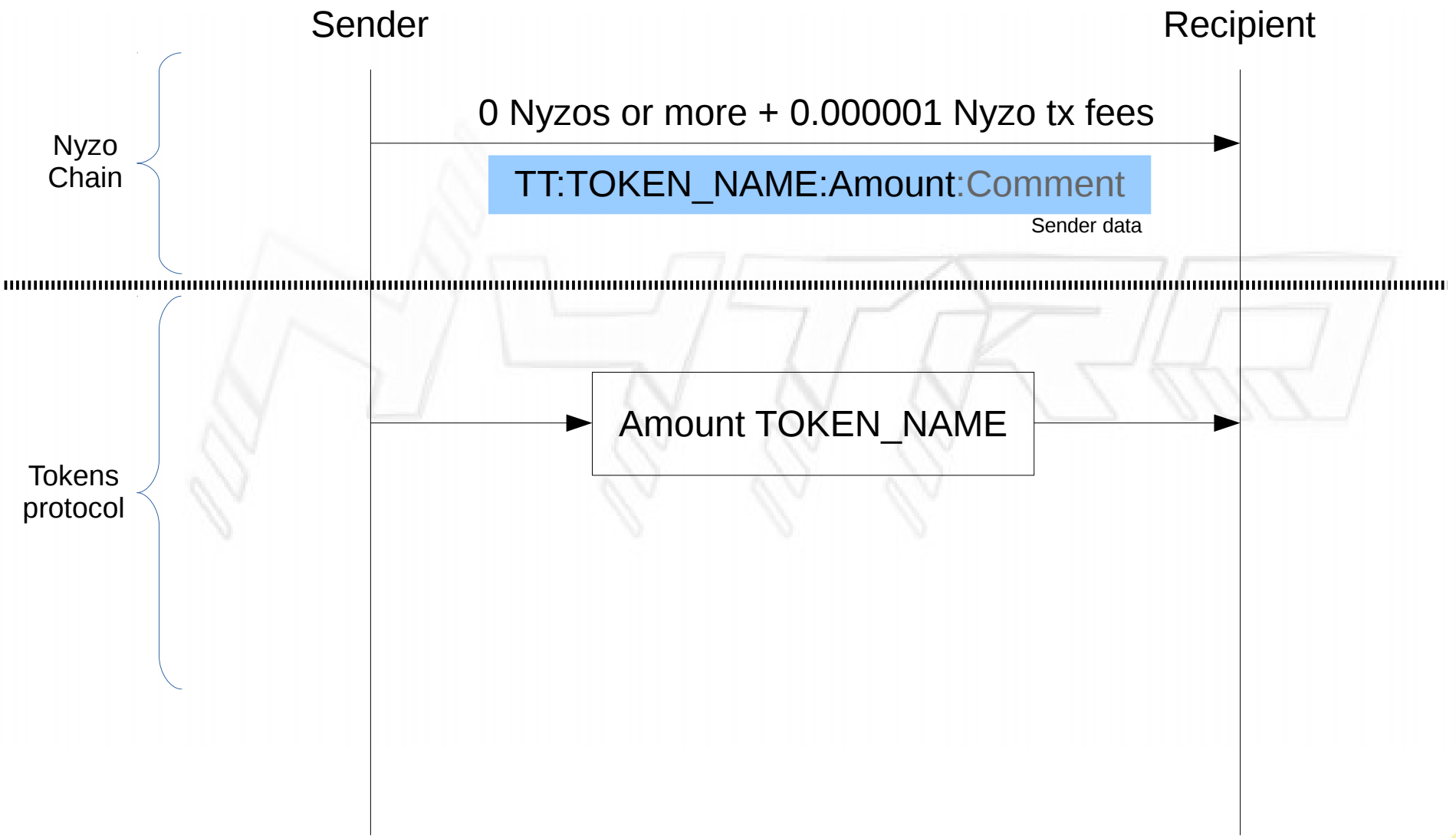
Sender data would then be “**TI:EXAMPLE:d00:1000000**”

I don't need to remember all the rules and risk errors that would invalidate my operation.
To make sure, I can use an online helper, like the one at https://tokens.nyzo.today/helpers/issue or a wallet that supports tokens operations.

Once my transaction is frozen by the cycle, the token explorer will show my newly created token and balance, one million “EXAMPLE” to my address.

I can now send my tokens to other users, via “TT” operation.

TRANSFER – TT:



Edited 1.1

TT: RULES

- TOKEN_NAME – The token type to send

  This token has to exist
  TOKEN_NAME has to obey the general TOKEN_NAME rules about charset, uppercase and size.

- Amount

  Integer or float as base 10 decimal, with dot as decimal separator and no thousand separator
  Amount is strictly greater than 0
  The sender needs a balance of – at least – that amount of this token
  Note: Amount can be a decimal but is also bound to the maximum number of decimals of the token.

- **Comment**

  String, Optional.
  a..z, A..Z, 0..9, _ (63 different chars)

- Whole rules – Apply on the whole TT: transaction

  (Regular Nyzo tx rule)
  Max total length is 32 chars, including “TT:” and the “:” separators
  Can't send to yourself

Edited 1.1

TT: Validation flow and rules

- Trigger: Data begins with “TT:” - case sensitive.
- Split by “:” delimiter
- TT_R0: consists of 3 or 4 segments “segment0:segment1:segment2” or “segment0:segment1:segment2:segment3”
- TT_R1a: segment1 follows TOKEN_NAME rules
- Retrieve TOKEN_NAME decimals and balance for sender address
- TT_R2a: segment2 obeys the decimal format. Regexp: /^[0-9.]+$/
- TT_R2b: segment2 has no more decimals than the token allows
- Convert “amount” string to integer*
- TT_R2c: amount is > 0 and lower than sender balance
- The following rules only apply if segment3 is defined
- **TT_R3**: segment3 obeys “comment” rules

- Note: The order in which the rules are evaluated is not important since any broken rule renders the token transaction invalid. In practice and for performance reasons low costs rules may be tested first. A canonical test order will be provided for test suites.

Edited 1.1

TT: Example

I want to send 10 of my “EXAMPLE” token to “id__8cdasPC2QVZ13iG42RWhp47gow9SsIXZgp0Aga59oEITG2X-M7Ur”.

This is a token I created, I still have 1 million in my balance.

I send a 1 micronyzo – that is 0.000001 Nyzos, the minimum allowed amount - transaction to the recipient address “**id__8cdasPC2QVZ13iG42RWhp47gow9SsIXZgp0Aga59oEITG2X-M7Ur**” with the data properly encoded.

Sender data in that case would be “**TT:EXAMPLE:10**”

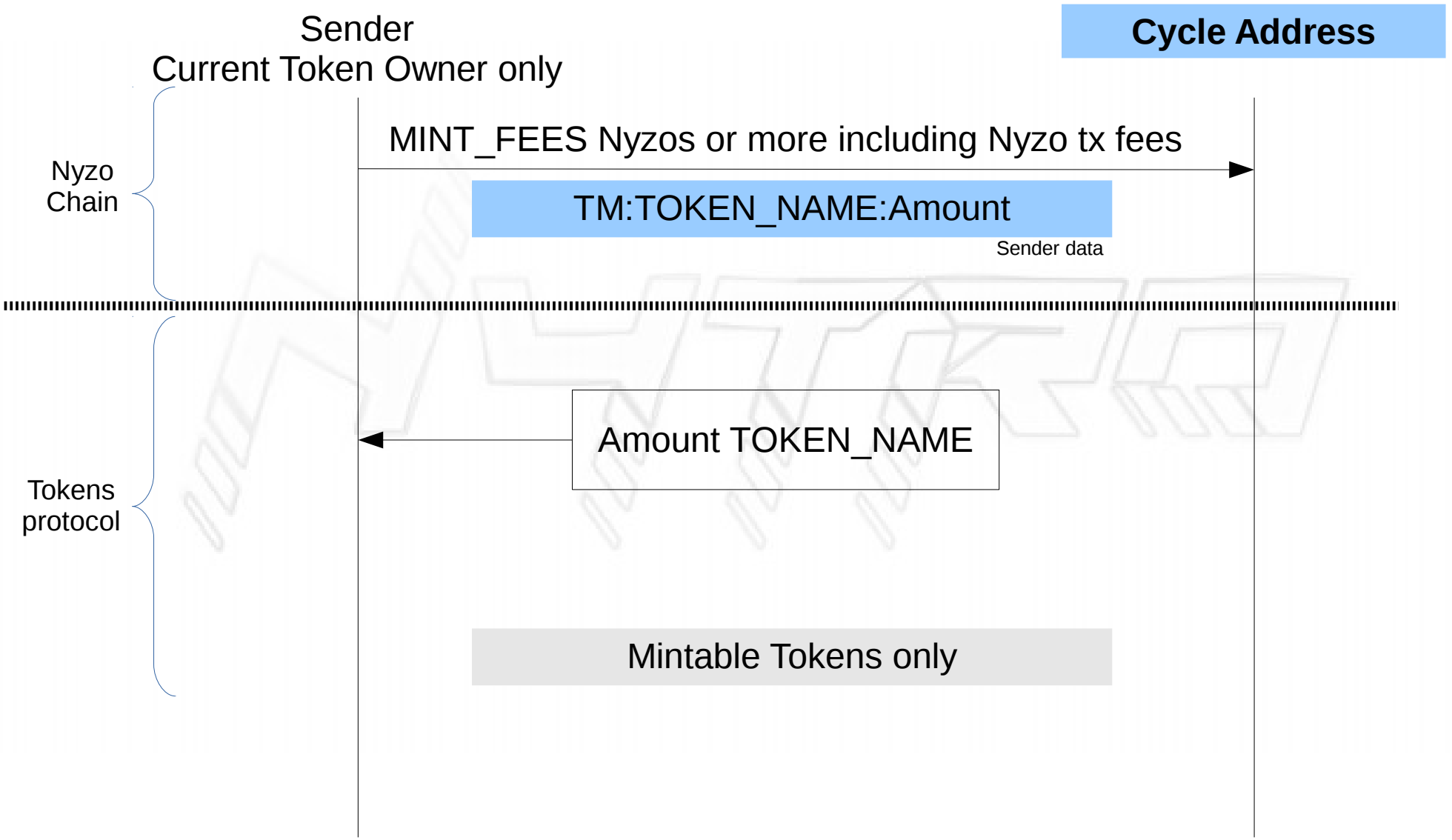
If I'd like to add a comment - for instance an order reference - It would just be added after, like “**TT:EXAMPLE:10:REF123**”

I don't need to remember all the rules and risk errors that would invalidate my operation.
To make sure, I can use an online helper, like the one at https://tokens.nyzo/today/helpers/transfer/ or a wallet that supports tokens operations.

Once my transaction is frozen by the cycle, the token explorer will show 10 “EXAMPLE” tokens to my recipient, and 999990 remaining in my own balance.

Edited 1.1

MINT – TM:


Sender
Current Token Owner only
Cycle Address
Nyzo
Chain
MINT_FEES Nyzos or more including Nyzo tx fees
TM:TOKEN_NAME:Amount
Sender data
Amount TOKEN_NAME
Tokens
protocol
Mintable Tokens only

TM: RULES

- TOKEN_NAME – The token type to mint

  This token has to exist and be of “Mintable” type
  TOKEN_NAME has to obey the general TOKEN_NAME rules about charset, uppercase and size.

  The sender has to be the current token rights owner (see TO: command)

- Amount

  Integer or float as base 10 decimal, with dot as decimal separator and no thousand separator
  Amount is strictly greater than 0
  Note: Amount can be a decimal but is also bound to the maximum number of decimals of the token.

- Amount TOKEN_NAME will be credited to the sender balance

- Recipient is the cycle address

- Whole rules – Apply on the whole TM: transaction

  (Regular Nyzo tx rule)
  Max total length is 32 chars, including “TM:” and the “:” separators

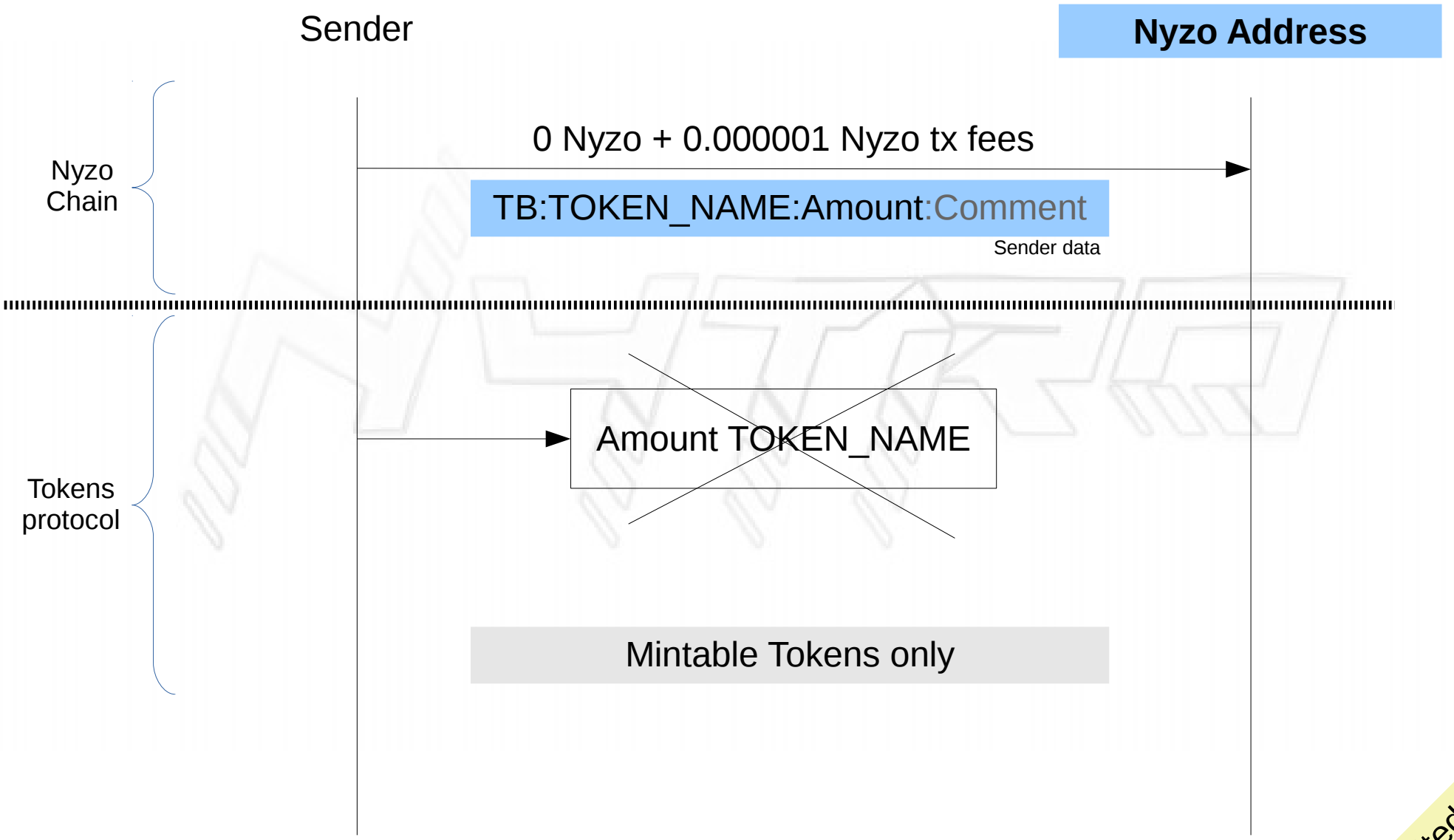
Fixed 1.1

TM: Validation flow and rules

- Trigger: Data begins with “TM:” - case sensitive.
- Split by “:” delimiter
- TM_R0: consists of exactly 3 segments “segment0:segment1:segment2”
- TM_R1a: segment1 follows TOKEN_NAME rules
- Retrieve TOKEN_NAME decimals and issuer
- TM_R1b: sender address is the current token owner
- TM_R1c: token is a mintable token - supply defined as “-1”
- TM_R2a: segment2 obeys the decimal format. Regexp: /^[0-9.]+$/
- TM_R2b: segment2 has no more decimals than the token allows
- TM_R2c: segment2 (amount) is > 0
- TM_R3a: recipient address is the cycle account
- TM_R3b: amount of Nyzos sent – including native Nyzo tx fees - is >= MINT_FEES

- Note: The order in which the rules are evaluated is not important since any broken rule renders the token transaction invalid. In practice and for performance reasons low costs rules may be tested first. A canonical test order will be provided for test suites.

BURN – TB:



Edited 1.1

TB: RULES

- TOKEN_NAME – The token type to burn

  This token has to exist and be of “Mintable” type
  TOKEN_NAME has to obey the general TOKEN_NAME rules about charset, uppercase and size.

- Amount

  Integer or float as base 10 decimal, with dot as decimal separator and no thousand separator
  Amount is strictly greater than 0
  The sender needs a balance of – at least – that amount of this token
  Note: Amount can be a decimal but is also bound to the maximum number of decimals of the token.

- Recipient is **any** Nyzo address

- **Comment**

  String, Optional.
  a..z, A..Z, 0..9, _ (63 different chars)

- Whole rules – Apply on the whole TB: transaction

  (Regular Nyzo tx rule)
  Max total length is 32 chars, including “TB:” and the “:” separators

Edited 1.1

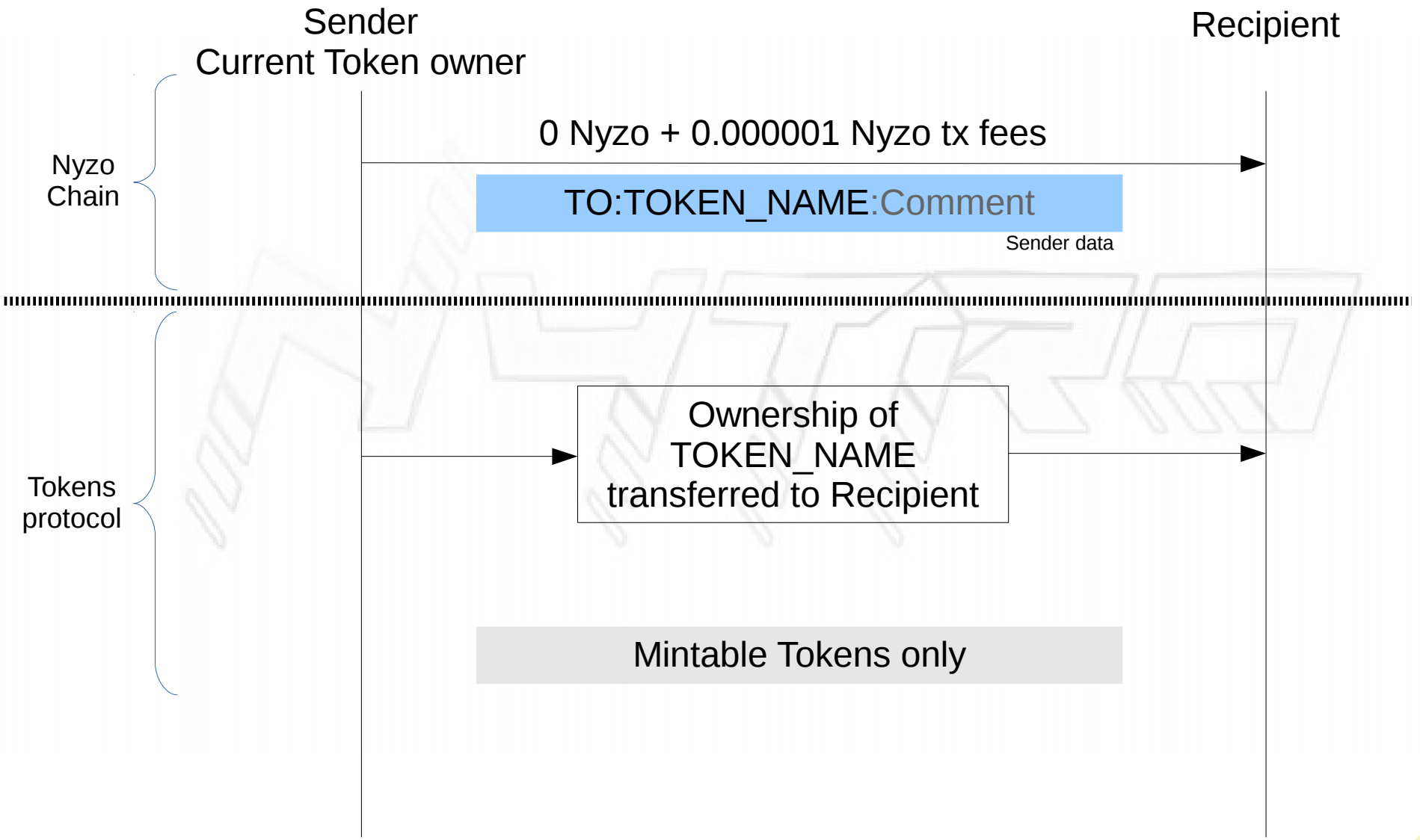
TB: Validation flow and rules

- Trigger: Data begins with “TB:” - case sensitive.
- Split by “:” delimiter
- TB_R0: consists of 3 or 4 segments “segment0:segment1:segment2” or “segment0:segment1:segment2:segment3”
- TB_R1a: segment1 follows TOKEN_NAME rules
- TB_R1b: TOKEN_NAME is a mintable token
- Retrieve TOKEN_NAME decimals and balance for sender address
- TB_R2a: segment2 obeys the decimal format. Regexp: /^[0-9.]+$/
- TB_R2b: segment2 has no more decimals than the token allows
- Convert “amount” string to integer*
- TB_R2c: amount is > 0 and lower than sender balance
- The following rules only apply if segment3 is defined
- **TB_R4**: segment3 obeys “comment” rules

- Note: The order in which the rules are evaluated is not important since any broken rule renders the token transaction invalid. In practice and for performance reasons low costs rules may be tested first. A canonical test order will be provided for test suites.

Edited 1.1

OWNERSHIP TRANSFER – TO:



Edited 1.1

TO: RULES

- TOKEN_NAME – The token to transfer ownership

  This token has to exist and be of “Mintable” type
  TOKEN_NAME has to obey the general TOKEN_NAME rules about charset, uppercase and size.

- Sender

  Sender has to be the current owner of the token.
  The first owner is the issuer.
  This is an ownership transfer: a token only has a single owner at a given time.

- Recipient is the new owner

- **Comment**

  String, Optional.
  a..z, A..Z, 0..9, _ (63 different chars)

- Whole rules – Apply on the whole TO: transaction

  (Regular Nyzo tx rule)
  Max total length is 32 chars, including “TO:” and the “:” separators

Note: Ownership is not meant as balance. To own a token in that context is having the exclusive right to mint it.

Edited 1.1

TO: Validation flow and rules

- Trigger: Data begins with “TO:” - case sensitive.
- Split by “:” delimiter
- TO_R0: consists of 2 or 3 segments “segment0:segment1” or “segment0:segment1:segment2”
- TO_R1a: segment1 follows TOKEN_NAME rules
- TO_R1b: TOKEN_NAME is a mintable token
- Retrieve TOKEN_NAME current owner
- TO_R2: sender address is the current token owner
- Update TOKEN_NAME owner
- The following rules only apply if segment2 is defined
- **TO_R3**: segment2 obeys “comment” rules

- Note: The order in which the rules are evaluated is not important since any broken rule renders the token transaction invalid. In practice and for performance reasons low costs rules may be tested first. A canonical test order will be provided for test suites.

Edited 1.1

CHANGELOG

NyzoSy and Iyomisc

nyzo

Version 1.1

Changelog

- V1.0 – 2020-12-23 - Initial Release
- V1.1 – 2021-06-30 - Limit token names to 10 chars, add optional comment field to some commands, allow token burn to any Nyzo address

Edited 1.1

DEVELOPERS ANNEXES

NyzoSy and Iyomisc

nyzo

Version 1.1

Tools: String <> Integer conversion

- This is to be used for **all** amounts in all tokens operations
- Amounts are specified as string in the “data” field of native Nyzo transactions
- In order to avoid rounding and precision errors due to floats and different architectures, **all computation on amounts and balances have to be applied to integers only**.
- To convert from string to integer:
  - Split the amount on “.”: segment0.segment1
  - Segment 1 is the decimal part
  - Make sure the length of the decimal part is lower or equal to the allowed decimals for that token
  - Pad the decimal part with 0 to the right, until length is exactly the allowed decimals
  - Concatenate segment0 and padded segment1
  - The result is a safe to use integer representation of the amount
  - That integer can be huge (more than 64 bits)
- To convert back from integer to string:
  - Cast the integer to a string
  - Pad the string with 0 to the left until the length of the string is > allowed decimals
  - Insert a “.” char at “decimals” places from the end of the string

NOTES: Balances computations

- All mathematical operations are to take place on the integer versions of the balances and amounts only. See String <> Integer conversion, no exception.
- The balance of an address 'A' for a token 'T' is
  - The sum of all tokens amounts from valid 'T' tokens transactions where 'A' is the recipient
  - Minus the sum of all tokens amounts from valid 'T' tokens transactions where 'A' is the sender
- It can be useful for performances reasons to keep the current state of all balances in cache and update the related balances on new valid tokens transaction only.
- Use a proper DB format to store amounts if you plan to compute balances via SQL for instance.
- Integer representation of amounts and balances can be huge and overflow some fixed integer types.
- DECIMAL column format, used by MySql for instance can be appropriate to represent such large integers.
- Implementation details will depend on storage engine and language used.
- Test vectors will only deal with string versions of fixed precision decimals amounts, including for balances.

Online Helpers

We provide Helpers, using the reference implementation and current block, at https://tokens.nyzo.today/helpers/

For instance:

- TOKEN ISSUER Helper
  - Tells whether the name you entered follows the tokens rules
  - Checks the token does not already exist
  - Select the supply type (fixed or mintable)
  - Provides the pre_ nyzostring to sign and forward to issue that token from your wallet
- TOKEN TRANSFER
  - From your balance page on tokens.nyzo.today
  - Provides a validated pre_ nyzostring to send AMOUNT of tokens to a recipient
- Transaction tester
  - Test a transaction before forwarding to the cycle
  - Enter sender address
  - Enter a pre_ nyzostring or transaction details
  - Tells whether the tokens transaction is valid, and if not: why.

Tokens API

Tokens explorer features are available as an API

API Endpoint: https://tokens.nyzo.today/api
Called by a simple get request with parameters in the URLl, returns Json.

- /tokens_list
  All tokens with their current properties

- /balances/{address}
  All tokens owned by address with their balance

- /richlist/{token_name}
  All current owners of that token and associated balance, sorted by amount desc

- /transactions/{address}
  Last 50 tokens transactions for that address
  Returns last 50 tokens transactions for all addresses if address is empty.

- /check_tx/{sender}/{recipient}/{amount_nyzo}/{data}
  Token transaction tester, tests that tx and returns ok or specific error code.

- /fees
  Fees history: list of { “block height”: {“issue_fees”: micronyzo_fees, “mint_fees”: micronyzo_fees}}

Swagger Doc: WIP

Tokens Python Class

In addition to the specs and workflows, we also release a reference Python implementation of the protocol.

The “Token” class takes Nyzo transactions as inputs and validates them against the Nytro protocol rules.

Valid transactions are stored in a MySql database, and tokens state updated.

All you need to reconstruct tokens state is to feed it the Nyzo history from the first Nytro valid block, 10650000.

The MySql database can then be used to power a Token explorer or any token based service.

This class is the reference implementation and has been used to validate the test vectors.

See Github https://github.com/Open-Nyzo/Project-Nytro

Reserved Tokens

The following tokens have been issued as Mintable tokens:

- BTC
- ETH
- USD

They are meant to avoid cybersquatting and be handed over to a serious actor willing to implement and handle in a trust-able way these Nyzo wrapped crypto-currencies.

- NYTRO was issued as a non mintable, 0 decimals token with 100,000 supply. We'll likely use it later on as a demo use case.
- NYZO was issued as a mintable token, 6 decimals. This is to avoid confusion between Nyzo (the main currency) and NYZO token. Ownership will be transferred after cycle vote if a future use case needs it.

In addition to these, we issued from the NCFP funds the following test tokens:

- TEST (no decimals)
- TEST2 (2 decimals, currency like)
- TEST18 (18 decimals, max allowed, ETH like)

They are defined as mintable tokens as well, and will serve as test tokens for anyone willing to experiment with Nyzo tokens. A faucet will provide them for free.

Rationale behind tokens fees

Token transfer fees:

- As low as possible to favor liquidity and use of the tokens, no friction.

Token issuance fees:

- Low enough to encourage token creation, use in apps, games, currencies...
- But high enough to discourage spam and mass cybersquatting
- Not meant to lower supply nor lock funds to the cycle
- We feel 100 Nyzos is high enough to protect from massive abuse
- As a comparison, an in-cycle verifier yields around 140 Nyzos a month.
- 100 Nyzos is around USD 15 at the moment but could be worth way more, why it's important not to define too high a starting fee.
- Fees could evolve with time, but lower them would mean penalizing early adopters who would have paid their token more. Better start lower and raise if we see this is not sustainable.
- The more the tokens created and used, the more the use cases and Nyzo value. Nyzo value will come from it's use, not its supply.

Other possible fees:

- Mint and Burn fees could be defined later on if needed

A Note about cybersquatting

We suppose and hope tokens on Nyzo will give birth to a thriving ecosystem with many actors and use cases.

It's also likely that some nicely named tokens end up reserved for future resale. This can be part of some economic loop and trigger more interest in Nyzo tokens, in Nyzo itself.

In that regard, we think this is not necessarily a bad thing unless there are obvious abuses and massive lock down of many tokens, that end up abandoned or unused for instance.

In such events there would always be possible actions at protocol level to mitigate the issue. Such actions could be voted upon by the cycle for instance, or by tokens owners.

That's why we don't consider it an immediate nor critical threat and will address it should a problem arise.

A Note about sustainability

Transactions fees are at heart of Nyzo long term sustainability model.
Once the seed transactions are over (see https://nyzo.co/coins), only organic transactions plus account fees will reward the verifiers.

How will a second layer protocol with minimal fees play with that?

The first answer is that Nyzo can absorb a lot of transactions, with high efficiency. There is no mempool for instance. In the past we had several times where some users experimented with huge numbers of 1 µNyzo transactions, filled up blocks, and the cycle continued as usual.

Nothing in current implementation does prevent minimal fee transactions to be sent, this is a real feature of the system.

Then, Nyzo concern at the moment is not the number of transactions and their fees.
It's the number of users, addresses, organic Nyzo transactions, real usage that has to grow for Nyzo to have a future.

We feel attracting new users, developers, use cases, by the way of cheap and blazingly fast – final – tokens transfers is a good way to have Nyzo gain attention and more use.

More users also means more addresses, more maintenance fees.
More tokens means more Nyzos used to issue them, more volume on exchange, and since you already have a Nyzo address and wallet, no more entry barrier to use the real Nyzo as a currency and micro payment solution.

Lastly, if the Nytro project was detrimental to Nyzo because of too much volume and way too little fees, then it would be easy to slightly raise token transfer fees for instance. This could be done via pre-paid account for instance.

References

- Official Github for Nytro Protocol: https://github.com/Open-Nyzo/Project-Nytro

- Current Nytro aware applications and services:
  - First Token Explorer by Iyomisc https://tokens.nyzo.today
  - Tokens API https://tokens.nyzo.today/api
  - Nyzocli command line client, since 0.0.9 https://github.com/EggPool/NyzoCli

Credits

Credits to

- Layer-2 protocols
- Event sourcing
  https://github.com/EggPool/BismuthEvents/blob/master/doc/What_is_event_sourcing.md
- Bismuth abstract transactions and layer-2 protocols
  https://github.com/bismuthfoundation/Hack-with-BIS/blob/master/01-Concepts/Bismuth-Model.md
  https://github.com/bismuthfoundation/Hack-with-BIS/blob/master/01-Concepts/protocols/readme.md
- z0rn for the Nytro graphics