herajs Documentation

aergo team and contributors

Mar 24, 2020

User Documentation

CHAPTER 1

Getting started

This package includes everything you need to build a wallet application on Aergo. It is useful for other kinds of dapps, too.

In contrary to @herajs/client, which includes just the API client and useful classes and helpers, @herajs/wallet includes managers that can keep, coordinate, and optionally persist state.

1.1 Features

- Manage keys
- Easy to use interface for sending transactions
- Track accounts for new transactions (using a full node or other external APIs)
- Sign and verify
- Integrates with storages (supported out of the box: IndexedDB (for browsers), LevelDB (for Node.js), and in-memory storage)

1.2 Overview

All calls go through an instance of Wallet. Wallet keeps references to the different managers and offers shortcut functions.

```
import { Wallet } from '@herajs/wallet';
const wallet = new Wallet();

// wallet.accountManager
// wallet.keyManager
// wallet.transactionManager
```

(continues on next page)

(continued from previous page)

```
// Configure chain
wallet.useChain({
    chainId: 'testnet.localhost',
    nodeUrl: '127.0.0.1:7845'
});

// Set up account and key
const account = await wallet.accountManager.createAccount();

// Build tx
const tx = {
    from: account.address,
    to: account.address,
    amount: '1 aergo'
};

// Send
let txTracker = await wallet.sendTransaction(account, tx);

// Wait for block confirmation
txTracker.on('block', (tx) => {
    console.log(tx);
});
```

CHAPTER 2

Wallet

class Wallet(*config*)
exported from wallet

Extends:

- middleware.MiddlewareConsumer()

Wallet is the central object that keeps configuration and references to the different managers.

Arguments

- **config**(*Partial<wallet.WalletConfig>*) –

Wallet.**accountManager**
type: managers/account-manager.AccountManager

Used to access account manager instance

Wallet.**datastore**
type: storages/storage.Storage

Wallet.**defaultChainId**
type: string

Wallet.**defaultLimit**
type: undefinedlnumber

Wallet.**keyManager**
type: managers/key-manager.KeyManager

Used to access key manager instance

Wallet.**keystore**
type: storages/storage.Storage

Wallet.**transactionManager**
type: managers/transaction-manager.TransactionManager

Used to access transaction manager instance

Wallet.**unlocked**
type: boolean

Wallet.**close**()
Closes storages.

Returns **Promise<void>** –

Wallet.**deleteAllData**()
Clears all storages

Returns **Promise<void>** –

Wallet.**getClient**(*chainId*)
Get AergoClient for chainId. If called the first time, create AergoClient instance.

Arguments

- **chainId** (*undefined|string*) – optional, uses default chainId when undefined

Returns **AergoClient** –

Wallet.**isSetup**()
Shortcut for keyManager.isSetup()

Returns **Promise<boolean>** –

Wallet.**lock**()
Shortcut for keyManager.lock()

Wallet.**prepareTransaction**(*account*, *transaction*)
Prepare a transaction from given account specified by simple TxBody. Completes missing information (chainIdHash, nonce) and signs tx using key of account.

Arguments

- **account** (*models/account.Account|models/account.AccountSpec*) – account object or specification
- **transaction** (*Partial<models/transaction.TxBody>*) –

Returns **Promise<models/transaction.SignedTransaction>** – prepared and signed transaction

Wallet.**sendTransaction**(*account*, *transaction*)
Send a transaction to the network using the specified account. Calls prepareTransaction() if not already prepared.

Arguments

- **account** (*models/account.Account|models/account.AccountSpec*) –
- **transaction** (*Partial<models/transaction.TxBody>|models/transaction.SignedTransaction*) –

Returns **Promise<managers/transaction-manager.TransactionTracker>** –

Wallet.**set**(*configKey*, *value*)
Sets a configuration value

Arguments

- **configKey** (*keyof:wallet.WalletConfig*) –
- **value** –

Wallet.**setDefaultChain**(*chainId*)
Set the default chain for subsequent actions.

Arguments

- **chainId** (*string*) –

Wallet.**setDefaultLimit**(*limit*)
Set the default gas limit for subsequent transactions.

Arguments

- **limit** (*number*) –

Wallet.**setupAndUnlock**(*passphrase*)
Shortcut for keyManager.setupAndUnlock()

Arguments

- **passphrase** (*string*) –

Returns **Promise<void>** –

Wallet.**unlock**(*passphrase*)
Shortcut for keyManager.unlock()

Arguments

- **passphrase** (*string*) –

Returns **Promise<void>** –

Wallet.**useChain**(*chainConfig*)
Add a chain configuration. Sets new chain as default if first to be added and default chain was unchanged.

Arguments

- **chainConfig** (*wallet.ChainConfig*) –

Wallet.**useDataStorage**(*classOrInstance*)
Sets storage to use for datastore

Arguments

- **classOrInstance** (*T|utils.Constructor<T|extends|storages/storage.Storage>*) –

Returns **Promise<storages/storage.Storage>** –

Wallet.**useKeyStorage**(*classOrInstance*)
Sets storage to use for keystore

Arguments

- **classOrInstance** (*T|utils.Constructor<T|extends|storages/storage.Storage>*) –

Returns **Promise<storages/storage.Storage>** –

Wallet.**useStorage**(*classOrInstance*)
Sets storage to use for both keystore and datastore

Arguments

- **classOrInstance** (*T|utils.Constructor<T|extends|storages/storage.Storage>*) –

Returns **Promise<[storages/storage.Storage,storages/storage.Storage]>** –

CHAPTER 3

Account Manager

class AccountManager(*wallet*)

exported from managers/account-manager

Extends:

- utils.PausableTypedEventEmitter

AccountManager manages and tracks single accounts

Arguments

- **wallet** (*wallet.Wallet*) –

AccountManager.**wallet**

type: wallet.Wallet

AccountManager.**addAccount**(*accountSpec*)

Adds account to manager and datastore.

Arguments

- **accountSpec** (*models/account.AccountSpec*) –

Returns Promise<models/account.Account> –

AccountManager.**clearAccounts**()

Remove all accounts from manager and datastore. Does not delete keys, call keyManager.clearKeys() for that.

Returns Promise<void> –

AccountManager.**createAccount**(*chainId*)

Generates a new account and private key.

Arguments

- **chainId** (*undefined|string*) – optional, uses default chainId if undefined

Returns Promise<models/account.Account> –

AccountManager.**getAccounts**()
Returns list of all accounts. Loads data persisted in datastore.

Returns **Promise<models/account.Account[]>** –

AccountManager.**getChainIdHashForAccount**(*account*)

Arguments

- **account** (*models/account.Account*) –

Returns **Promise<string>** –

AccountManager.**getCompleteAccountSpec**(*accountSpec*)
Completes accountSpec with chainId in case chainId is undefined.

Arguments

- **accountSpec** (*models/account.AccountSpec*) – completed account spec

Returns **models/account.CompleteAccountSpec** –

AccountManager.**getNonceForAccount**(*account*)
Returns next usable nonce for account. This uses the Aergo client to determine the nonce from the server.

Arguments

- **account** (*models/account.Account*) –

Returns **Promise<number>** –

AccountManager.**getOrAddAccount**(*accountSpec*)
Gets an account and adds it to the manager if not existing.

Arguments

- **accountSpec** (*models/account.CompleteAccountSpec|models/account.AccountSpec*) –

Returns **Promise<models/account.Account>** –

AccountManager.**loadAccount**(*accountSpec*)
Initializes account from datastore or with initial values.

Arguments

- **accountSpec** (*models/account.CompleteAccountSpec*) –

Returns **Promise<models/account.Account>** –

AccountManager.**pause**()
Pause all existing account trackers.

AccountManager.**prepareTransaction**(*account*, *tx*)
Calculates nonce and converts transaction body into tx ready for signing

Arguments

- **account** (*models/account.Account*) –
- **tx** (*Partial<models/transaction.TxBody>*) –

Returns **Promise<models/transaction.Transaction>** –

AccountManager.**removeAccount**(*accountSpec*)
Removes account from manager and datastore. Also removes account's key from keystore if no other account uses it.

Arguments

- **accountSpec** (*models/account.AccountSpec*) –

Returns Promise<void> –

AccountManager.**resume**()
Resume all existing account trackers.

AccountManager.**trackAccount**(*accountOrSpec*)
Returns an account tracker.

Arguments

- **accountOrSpec** (*models/account.AccountSpec|models/account.Account*) –

Returns Promise<managers/account-manager.AccountTracker> –

class AccountTracker(*manager*, *account*)

Extends:

- utils.PausableTypedEventEmitter

Arguments

- **manager** (*managers/account-manager.AccountManager*) –
- **account** (*models/account.Account*) –

AccountTracker.**load**()

Returns Promise<models/account.Account> –

AccountTracker.**pause**()

AccountTracker.**resume**()

CHAPTER 4

Key Manager

class KeyManager(*wallet*)
exported from managers/key-manager

Extends:

- TypedEventEmitter

KeyManager manages keys for accounts.

Arguments

- **wallet** (*wallet.Wallet*) –

KeyManager.**unlocked**
type: boolean

True if keystore is currently unlocked, i.e. a master passphrase is saved in memory.

KeyManager.**wallet**
type: wallet.Wallet

KeyManager.**addKey**(*account*, *privateKey*)
Adds a private key for an account to the keystore

Arguments

- **account** (*models/account.Account*) –
- **privateKey** (*Uint8Array|number[]*) –

Returns **models/key.Key** –

KeyManager.**clearKeys**()
Removes all keys stored in keystore.

Returns **Promise<void>** –

KeyManager.**getKey**(*account*)

Arguments

- **account** (*models/account.Account*) –

Returns Promise<models/key.Key> –

KeyManager.**getUnlockedKey**(*account*)

Arguments

- **account** (*models/account.Account*) –

Returns Promise<models/key.Key> –

KeyManager.**importKey**(*importSpec*)
Imports a raw or encrypted private key and add it to the keystore.

Arguments

- **importSpec** (*managers/key-manager.ImportSpec*) –

Returns Promise<models/key.Key> –

KeyManager.**isSetup**()
Checks if wallet is setup with a master passphrase

Returns Promise<boolean> –

KeyManager.**lock**()
Locks keystore by removing master passphrase from memory.

KeyManager.**removeKey**(*address*)
Removes key for address from keystore

Arguments

- **address** (*string*) –

Returns Promise<void> –

KeyManager.**setupAndUnlock**(*appId*, *passphrase*)
Sets up keystore passphrase for the first time.

Arguments

- **appId** (*string*) – string to be saved encrypted with passphrase for later validity check
- **passphrase** (*string*) –

Returns Promise<void> –

KeyManager.**signMessage**(*account*, *message*, *enc*)
Signs a message using key saved for account

Arguments

- **account** (*models/account.Account*) –
- **message** (*Buffer*) –
- **enc** (*managers/key-manager.Encoding*) –

Returns Promise<string> –

KeyManager.**signTransaction**(*account*, *transaction*)
Signs a transaction using key saved for account

Arguments

- **account** (*models/account.Account*) –

- **transaction** (*models/transaction.Transaction*) –

Returns Promise<models/transaction.SignedTransaction> –

KeyManager.**unlock**(*passphrase*)
Unlocks keystore by saving passphrase in memory.

Arguments

- **passphrase** (*string*) –

Returns Promise<void> –

class ImportSpec()
interface

Specification for importing private keys

ImportSpec.**account**
type: models/account.Account

ImportSpec.**b58encrypted**
type: undefinedlstring

ImportSpec.**password**
type: undefinedlstring

ImportSpec.**privateKey**
type: Buffer

CHAPTER 5

Transaction Manager

class TransactionManager(*wallet*)

exported from managers/transaction-manager

Extends:

- utils.PausableTypedEventEmitter

TransactionManager manages and tracks single transactions

Arguments

- **wallet** (*wallet.Wallet*) –

TransactionManager.**wallet**

type: wallet.Wallet

TransactionManager.**addTransaction**(*transaction*)

Arguments

- **transaction** (*models/transaction.SignedTransaction*) –

Returns Promise<void> –

TransactionManager.**fetchAccountTransactions**(*account*)

Arguments

- **account** (*models/account.Account*) –

Returns Promise<models/transaction.SignedTransaction[]> –

TransactionManager.**fetchAccountTransactionsAfter**(*account*, *blockno*, *limit*)

Arguments

- **account** (*models/account.Account*) –
- **blockno** (*number*) –
- **limit** (*undefined|number*) –

Returns **Promise<models/transaction.SignedTransaction[]>** –

TransactionManager.**fetchAccountTransactionsBefore**(*account*, *blockno*, *limit*)

Arguments

- **account** (*models/account.Account*) –
- **blockno** (*number*) –
- **limit** (*undefined|number*) –

Returns **Promise<models/transaction.SignedTransaction[]>** –

TransactionManager.**getAccountTransactions**(*accountOrSpec*)

Returns transactions stored for an account

Arguments

- **accountOrSpec** (*models/account.AccountSpec|models/account.Account*) –

Returns **Promise<models/transaction.Transaction[]>** –

TransactionManager.**pause**()

TransactionManager.**resume**()

TransactionManager.**sendTransaction**(*transaction*)

Sends prepared and signed transaction to the Aergo client. Adds transaction to the manager and starts a tracker.

Arguments

- **transaction** (*models/transaction.SignedTransaction*) –

Returns **Promise<managers/transaction-manager.TransactionTracker>** – transaction tracker

TransactionManager.**trackAccount**(*account*)

Track transactions for account. There is no default implementation for this. The only generally available method would be to scan the entire blockchain which is highly inefficient. If you want that, use your own full node and add the data source using wallet.use(NodeTransactionScanner);

Arguments

- **account** (*models/account.Account*) –

Returns **managers/transaction-manager.AccountTransactionTracker** –

TransactionManager.**trackTransaction**(*transaction*)

Arguments

- **transaction** (*models/transaction.SignedTransaction*) –

Returns **Promise<managers/transaction-manager.TransactionTracker>** –

class TransactionTracker(*manager*, *transaction*)

exported from managers/transaction-manager

Extends:

- utils.PausableTypedEventEmitter

Arguments

- **manager** (*managers/transaction-manager.TransactionManager*) –

- **transaction** (*models/transaction.SignedTransaction*) –

TransactionTracker.**hash**
type: string

TransactionTracker.**transaction**
type: models/transaction.SignedTransaction

TransactionTracker.**cancel**()
Cancels transaction tracker. This does not cancel sending the transaction.

TransactionTracker.**getReceipt**()
Returns a promise that resolves when the transaction is confirmed and the receipt is available and rejects when an error or timeout occurs.

Returns **Promise<managers/transaction-manager.GetReceiptResult>** –

TransactionTracker.**load**()
Attempt to retrieve transaction data from node. Emits events according to changed status.

Returns **Promise<void>** –

TransactionTracker.**retryLoad**()
Repeatedly tries loading transaction status. This uses an exponential backoff in case the transaction is not confirmed yet.

class AccountTransactionTracker(*manager*, *account*)

Extends:

- utils.PausableTypedEventEmitter

Arguments

- **manager** (*managers/transaction-manager.TransactionManager*) –
- **account** (*models/account.Account*) –

AccountTransactionTracker.**load**()

Returns **Promise<models/account.Account>** –

AccountTransactionTracker.**pause**()

AccountTransactionTracker.**resume**()

CHAPTER 6

Storages

6.1 IndexedDb

class IndexedDbStorage(*name*, *version*)
exported from storages/indexdb

Extends:

- storages/storage.Storage()

IndexedDbStorage uses the browser-native IndexedDb.

Arguments

- **name** (*string*) –
- **version** (*number*) –

IndexedDbStorage.**db**
type: IDBPDatabase<storages/indexdb.IdbSchema>

IndexedDbStorage.**indices**
type: Map<string,storages/indexdb.IDBIndex>

IndexedDbStorage.**name**
type: string

IndexedDbStorage.**version**
type: number

IndexedDbStorage.**close**()

Returns Promise<void> –

IndexedDbStorage.**getIndex**(*name*)

Arguments

- **name** (*storages/indexdb.StoreNames<storages/indexdb.IdbSchema>*) –

Returns storages/indexdb.IDBIndex –

`IndexedDbStorage.`**`open`**`()`

Returns Promise<this> –

6.2 LevelDb

`class LevelDbStorage`(*name*, *version*)
exported from `storages/leveldb`

Extends:

- `storages/storage.Storage()`

LevelDbStorage can be used to get an IndexedDb-like storage in Node.js

Arguments

- **`name`** (*`string`*) –
- **`version`** (*`number`*) –

`LevelDbStorage.`**`db`**
type: LevelUp

`LevelDbStorage.`**`indices`**
type: Map<string,storages/leveldb.LevelDbIndex>

`LevelDbStorage.`**`name`**
type: string

`LevelDbStorage.`**`version`**
type: number

`LevelDbStorage.`**`close`**`()`

Returns Promise<void> –

`LevelDbStorage.`**`getIndex`**(*name*)

Arguments

- **`name`** (*`string`*) –

Returns storages/leveldb.LevelDbIndex –

`LevelDbStorage.`**`open`**`()`

Returns Promise<this> –

6.3 In-memory

`class MemoryStorage`(*name*, *version*)
exported from `storages/memory`

Extends:

- `storages/storage.Storage()`

MemoryStorage is a storage interface compatible with other LevelDB-like storages. It is mostly used for testing. It is not very efficient.

Arguments

- **name** (*string*) –
- **version** (*number*) –

MemoryStorage.**indices**
type: Map<string,storages/memory.MemoryIndex>

MemoryStorage.**name**
type: string

MemoryStorage.**version**
type: number

MemoryStorage.**close**()

Returns **Promise<void>** –

MemoryStorage.**getIndex**(*name*)

Arguments

- **name** (*string*) –

Returns **storages/memory.MemoryIndex** –

MemoryStorage.**open**()

Returns **Promise<this>** –

CHAPTER 7

Models

7.1 Account

class Account(*key*, *data*)
exported from models/account

Extends:

- models/record.Record

Arguments

- **key** (*string*) – database key of record
- **data** (*models/account.AccountData*) – data of record

Account.**address**
type: Address

Account.**balance**
type: Amount

Account.**nonce**
type: number

class AccountSpec()
interface, *exported from* models/account

Unique identifier of an account. ChainId is optional. Managers use default chainId if undefined.

AccountSpec.**address**
type: string|Address

AccountSpec.**chainId**
type: undefined|string

7.2 Key

class Key(*key*, *data*)
exported from models/key

Extends:

- models/record.Record

Arguments

- **key** (*string*) – database key of record
- **data** (*models/key.KeyData*) – data of record

Key.**keyPair**
type: any

Key.**fromRecord**(*record*)

Arguments

- **record** (*models/record.Record<any>*) –

Returns **models/key.Key** –

Key.**signMessage**(*message*, *enc*)

Arguments

- **message** (*Buffer*) –
- **enc** (*models/key.Encoding*) –

Returns **Promise<string>** –

Key.**signTransaction**(*tx*)

Arguments

- **tx** (*models/transaction.Transaction*) –

Returns **Promise<models/transaction.SignedTransaction>** –

Key.**unlock**(*passphrase*)

Arguments

- **passphrase** (*undefined|string*) –

7.3 Transaction

class Transaction(*key*, *data*, *txBody*)
exported from models/transaction

Extends:

- models/record.Record

Arguments

- **key** (*string*) –
- **data** (*models/transaction.TransactionData*) –

- **txBody** (*models/transaction.CompleteTxBody*) –

Transaction.**amount**
type: Amount

Transaction.**txBody**
type: models/transaction.CompleteTxBody

Transaction.**unsignedHash**
type: string

Transaction.**getUnsignedHash**()
Calculate the hash excluding any signature

Returns **string** –

Transaction.**Status**
type: models/transaction.Status

class SignedTransaction(*key*, *data*, *txBody*, *signature*)
exported from models/transaction

Extends:

- models/transaction.Transaction()

Arguments

- **key** (*string*) –
- **data** (*models/transaction.TransactionData*) –
- **txBody** (*models/transaction.CompleteTxBody*) –
- **signature** (*string*) –

SignedTransaction.**hash**
type: string

SignedTransaction.**isConfirmed**
type: boolean

SignedTransaction.**isPending**
type: boolean

SignedTransaction.**signature**
type: string

SignedTransaction.**status**
type: string

SignedTransaction.**txBody**
type: models/transaction.CompleteTxBody

SignedTransaction.**fromTxBody**(*txBody*, *chainId*)

Arguments

- **txBody** (*models/transaction.CompleteTxBody*) –
- **chainId** (*string*) –

Returns **models/transaction.SignedTransaction** –

SignedTransaction.**getHash**()
Calculate the hash, including all present body

Returns **Promise<string>** –

class TxBody()
interface, *exported from* models/transaction

TxBody.**amount**
type: string|number|Amount

TxBody.**chainIdHash**
type: string|Uint8Array

TxBody.**from**
type: string|Address

TxBody.**hash**
type: undefined|string

TxBody.**limit**
type: undefined|number

TxBody.**nonce**
type: undefined|number

TxBody.**payload**
type: string|Uint8Array

TxBody.**price**
type: string|Amount

TxBody.**sign**
type: undefined|string

TxBody.**to**
type: string|Address|null

TxBody.**type**
type: undefined|number

CHAPTER 8

Utils

8.1 Exponential backoff

backoffIntervalStep(*n*, *multiplier*)

Returns the next interval to use for exponential backoff. This curve yields every value 4 times before doubling in the next step. The function is `multiplier * 2**Math.floor(n/4)`. By default (multiplier = 1s), the intervals reach ca. 1 minute (total time elapsed ca. 4 minutes) after step 24, so it is advised to declare a timeout after a certain number of steps.

Arguments

- **n** (*number*) – step on the interval curve
- **multiplier** (*number*) – multiplier, default 1000 (1s)

Returns **number** –

8.2 AccountSpec serialization

serializeAccountSpec(*accountSpec*)

Serializes accountSpec, e.g. { chainId: 'foo', address: 'bar' } => foo/bar

Arguments

- **accountSpec** (*models/account.AccountSpec*) –

Returns **string** –

deserializeAccountSpec(*serialized*)

Deserializes accountSpec, e.g. foo/bar => { chainId: 'foo', address: 'bar' }. If string has no /, uses whole string as address with empty chainId.

Arguments

- **serialized** (*string*) –

Returns **models/account.AccountSpec –**

CHAPTER 9

Indices and tables

- genindex

Index

K

L

M

S

T

W