**EPSMI Documentation**

**(Version 0.0.4** – 02.01.2021**)**

**Proprietary Notes**

EPSMI is a product of MHM Software, license to be determined.

**Document Information**

**Document Details**

|  |  |
| --- | --- |
| Author | Milosz Muszynski |
| Contributors |  |
| Issued on | 02.01.2021 |
| Last Update on | 02.01.2021 |

**Related Documents**

|  |  |
| --- | --- |
| **Name** | **Document File** |
|  |  |
|  |  |

**Open Issues**

|  |  |  |
| --- | --- | --- |
| **Description** | **Resp.** | **Date** |
|  |  |  |
|  |  |  |
|  |  |  |

**Table of Contents**

[1 Overview 3](#_Toc60664439)

[1.1 What is present in EPS and missing in EPSMI 3](#_Toc60664440)

[1.2 Features of EPSMI going beyond EPS 3](#_Toc60664441)

[2 API 3](#_Toc60664442)

[2.1 Miscellaneous Queries by Electrum Wallet 3](#_Toc60664443)

[2.1.1 blockchain.scripthash.get\_balance 4](#_Toc60664444)

[2.2 Block-related Queries by Electrum Wallet 5](#_Toc60664445)

[2.2.1 blockchain.block.header 5](#_Toc60664446)

[2.2.2 blockchain.block.get\_header 6](#_Toc60664447)

[2.2.3 blockchain.block.headers 6](#_Toc60664448)

[2.2.4 blockchain.block.get\_chunk 6](#_Toc60664449)

[2.3 Components 7](#_Toc60664450)

[3 Configuration Management 10](#_Toc60664451)

[3.1 Database Configuration Table – T\_ENV 10](#_Toc60664452)

[3.2 Web Service Configuration 13](#_Toc60664453)

# Overview

EPSMI is the Scala implementation of the original Electrum Personal Server written in Python by Chris Belcher.

Intention of EPSMI is to provide alternative implementation, which could be more accessible and readable to Scala and Java developers, as well as subjectively more accessible for people who find Scala structure and type system more readable than Python.

Another intention is to ease the maintenance burden, again, given Scala-inclined developers are in charge, and last but not least, to ease the effort of adding new features.

## What is present in EPS and missing in EPSMI

* TOR
* Rescan is a command line parameter in EPS, while configuration setting in EPSMI

## Features of EPSMI going beyond EPS

* TBD ☺

# API

## Miscellaneous Queries by Electrum Wallet

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| Name | Description | Input Parameters | Output Parameters | Errors and Exceptions |
| server.ping |  | None | None | TODO |
| server.banner | Provides information about server, blockchain and Bitcoin network that is visible in the console tab of Electrum app. | None | String with banner text. | Calls networkInfo and blockchainInfo – may throw exception (TODO – see how these exceptions are handled). |
| server  .donation\_address | Provides donation address which will be used by Electrum app in the Help/Donate to server function | None | String with Bitcoin address that can be used as Electrum payment destination. | - |
| Mempool  .get\_fee\_histogram | The histogramisan array of [fee, vsize] pairs, where vsizen is the cumulative virtual size of mempool transactions with a fee rate in the interval [feen-1, feen], and feen-1 > feen.  (Deprecated? – check it out, this API is resource intensive and is currently switched off – TODO – turn it on and test**)** | None | Array[Array[Int]] – array of pairs of integers for fee and vsize |  |
| server.version | Note that returned server version must be 1.4, changing it has serious implications as some logic triggers on the client side, change it from 1.4 only if you know what you are doing | Two strings representing Electrum client version, currently ignored | Array of strings, consisting of server name and version |  |
| blockchain.estimatefee |  | waitBlocks – integer indicating urgency for the fee requested – in how many blocks confirmation is expected (must be in range 1-1008 | BigDecimal – fee rate | Error when waitBlocks parameter is not in the range 1-1008 |
| blockchain.scripthash  .get\_balance | Confirmed and unconfirmed balances of a scripthash | String containing scripthash as hex, e.g.: 5be022609383  d23e2d545b3  b359446466c  269686c1e697  b60355424ed3  0490d2 | {  "confirmed": "1.03873966",  unconfirmed": "0.236844"  } | **TODO – futurize it** |

### blockchain.scripthash.get\_balance

For a given scripthash sh, we retrieve history elements from transaction monitor state transaction history. For each history element, we obtain txhash, from txhash we obtain decoded raw transaction (via 2 calls to bitcoin rpc). In a raw transaction we have access to vout, we filter vouts whose scriptpubkey is our sh, and collect amounts and confirmation counts. We also collect from vin utxos txhashes and delete them if thay happen to already be in our collection – this is to avoid counting output of one transaction is consumed by another for the same scripthash. At the end we sum separately amounts with zero confirmations and with 1 or more confirmations.

## Block-related Queries by Electrum Wallet

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| Name | Description | Input Parameters | Output Parameters | Errors and Exceptions |
| blockchain.block  .header | Hex representation of block’s header information | Block height (integer) | Block header hex (string) |  |
| blockchain.block  .get\_header | Dictionary representation of block’s header information | Block height (integer) | Header result dictionary HeaderResult |  |
| blockchain.block  .headers | Concatenated hex representation of multiple block’s header information | Block start height and count | Result dictionary BlockHeadersResult |  |
| blockchain.block  .get\_chunk | Concatenated hex representation of multiple block’s header information | Desired 2016 blocks chunk number – first 2016 blocks when 0, second 2016 blocks when 1, etc. | Concatenated block header hexes for all blocks in a chunk (of size 2016 or less) |  |

### blockchain.block.header

Block height is converted to block hash via RPC getBlockHash, then block header structure is obtained via RPC getBlockHeader. Information from the header structure is packed according to the format:

<i32s32sIII

Which translates to:

little endian int | byte[32] | byte[32] | unsigned int | unsigned int | unsigned int

The following information is inserted:

blockHeader.version

blockHeader.previousblockhash

blockHeader.merkleroot

blockHeader.time

blockHeader.bits

blockHeader.nonce

Returned string length is 160 which is hex representation of the 80 bytes filled out as above (4+32+32+4+4+4).

The information is the same as bitcoin-cli getblockheader with verbose = false, e.g.:

***bitcoin-cli getblockheader 00000000c937983704a73af28acdec37b049d214adbda81d7e2a3dd146f6ed09 false***

Note that you need block hash as input for the above call, which you can obtain from block height via a call to getblockhash, e.g.:

***bitcoin-cli getblockhash 600000***

### blockchain.block.get\_header

Similar to blockchain.block.header, only information is returned not as array of bytes in hex, but rather as a dictionary with the following fields:

* Block height
* Previous block hash
* Timestamp
* Merkle root
* Version
* Nonce
* Bits

The information is the same as bitcoin-cli getblockheader with verbose = true, e.g.:

***bitcoin-cli getblockheader 00000000c937983704a73af28acdec37b049d214adbda81d7e2a3dd146f6ed09 true***

### blockchain.block.headers

Similar to blockchain.block.header, only returned hex contains concatenated byte arrays of many blocks. Returned BlockHeadersResult structure contains the following fields:

* Hex – hex string being a result of concatenation of 80-byte arrays (160 character hex strings), each in format as in the result of blockchain.block.header
* Count – effective count, maybe smaller than given count if highest block has been reached
* Max – maximum number of blocks that can be processed, it is a constant set to 2016

Note that implementation utilizes the fact that get RPC getblockheader returns hashes to the previous and next blocks. Hence, we only need to call RPC getblockhash, converting block height to block hash, once, and for subsequent blocks we can only call RPC getblockheader, always using the nextblockhash field from GetBlockHeaderResult. Note that GetBlockHeaderResult is a structure returned by RPC getblockheader.

### blockchain.block.get\_chunk

Similar to blockchain.block.headers, only does not return a structure but rather a single concatenated string. Input parameter is a chunk number, so it is 0, result will contain headers for the first 2016 blocks. If it is 1, result will contain headers for the second 2016 blocks, etc. If chunk exceeds the highest block, fewer than 2016 headers will be returned.

Result is a hex encoded string whose length is a multiple of 160, typically the length will be 2016\*160 (322560) or less if the chunk borders with the highest block.

## Transaction-related Queries by Electrum Wallet

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| Name | Description | Input Parameters | Output Parameters | Errors and Exceptions |
| blockchain.transaction  .get | Provides hex representation of transaction data | Transaction id (string) | Hex representation of transaction (string) |  |
| blockchain.transaction  .id\_from\_pos | Provides n-th transaction id in a given block | Block height h, position within block pos, include merkle information (Boolean) | If merkle is true, pos-th transaction id within block of height h  If merkle is false, returns a struct with 2 fields: tx\_hash with pos-th transaction id and merkle with electrum merkle proof (list of strings) |  |
|  |  |  |  |  |
|  |  |  |  |  |

### blockchain.transaction.get

Accepts transaction id and returns transaction in hexadecimal form.

Calls RPC gettransaction which provides in-wallet transaction, e.g.:

***bitcoin-cli gettransaction "22667c482f0f69daefabdf0969be53b8d539e1d2abbfc1c7a193ae38ec0d3e31"***

Among fields returned by this call is a field hex, which contains raw data for the transaction.

If transaction is not in-wallet, the API uses another RPC, getrawtransaction, e.g.:

***bitcoin-cli getrawtransaction "b850bd9f727888019ddd5481124b83c17b9dd263fe4c7c007a0a6c0f4c0f1573"***

The second call returns only raw data for the transaction.

### blockchain.transaction.id\_from\_pos

Third parameter merkle decides if merkle proof information is required.

If merkle is false, the call boils down to calling RPC getblockhash to obtain blockhash from block height, then calling RPC getblock to obtain GetBlockResult, from which pos-th transaction id is taken from the field tx.

If merke is true, additional call is being made to gettxoutproof, which provides MerkleBlock, which is then converted to Electrum merkle proof format, which is ElectrumMerkleProof. The conversion disassembles raw data provided by gettxoutproof.

RPC gettxoutproof returns a serialized, hex-encoded proof that a given transaction is included in the block.



## Components

Hermes flows are realized by components.

Flow 1 is implemented by a pipeline component which in turn uses repository, targeter and deliverer components.

Flow 2 is implemented by a daemon component which in turn reads the repository, and passes data to targeter and deliverer.

Capturing and enrichment components are currently noop processors, as their functionality (for purely technical reasons) has been moved to the web service component.





### 

# Configuration Management

As Hermes runs as a JEE6 application inside the Websphere Application Server, its direct configuration is done via Websphere resources. The following Websphere resources need to be configured and made available to Hermes:

* Database data source (JNDI path: "jdbc/CL3")
* Mail session object (JNDI path: "mail/SRMail")
* Web service
* REST URLs
* HTTP URLs

Hermes bootstraps itself via the data source resource and then reads all the remaining configuration properties from a database. One particular database table is devoted to storing configuration items – T\_ENV.

## Database Configuration Table – T\_ENV

The following entries in T\_ENV are relevant for Hermes function:

|  |  |  |
| --- | --- | --- |
| Entry Name | Entry Default Value | Description |
| hermes.env | Empty string | Used to distinguish between DEV and non-DEV environments: if set to DEV, prohibits exceptions when activity submitters' credentials are incorrect. |
| hermes.technical.user | "\_\_\_default\_\_\_" | Checks this user against activity submitter principal. Should be set to the technical user of the party which submits activities to Hermes via web services. |
| hermes.from.address | "hermes@swissre.com" | Used by Hermes email sending module, will be seen by Hermes' email recipients as "from" address. |
| cerebro.activity.url | Severe exception logged when not set. | Shown as Cerebro link in the digest and in the RSS feed. Contains placeholder ({activityId}) which will be replaced in runtime. |
| cerebro.activity.sso.url | Severe exception logged when not set. | Used as prefix when creating Cerebro link. Both cerebro.activity.url and cerebro.activity.sso.url are part of the same Cerebro link which is used in the digest and in the RSS feed. |
| subscription.link | "-na-" | Used for "Manage Subscription" link in the digest. |
| cerebro.link | "http://swissre.com" | Hardcoded, configuration item will be overridden by default – needs to be fixed (TODO) |
| arbiter.maxage.days | 1 | Clustering arbiter entries maximum age in days. Entries older than the given number of days will be removed by the arbiter clean-up daemon. |
| arbiter.cleanup.hours | 1 | Time of day at which arbiter clean up daemon will perform its job. It runs once daily at a given time, default is 1AM. |
| arbiter.cleanup.minutes | 28 | Minutes past the hour at which arbiter clean up daemon will perform its job. Default is 28 minutes past the hour. |
| deliverylog.maxage.months | 6 | Delivery log entries maximum age in months. Entries older than the given number of months will be removed by the delivery log clean up daemon. |
| deliverylog.cleanup.hours | 2 | Time of day at which delivery log clean up daemon will perform its job. It runs once daily at a given time, default is 2AM. |
| deliverylog.cleanup.minutes | 17 | Minutes past the hour at which delivery log clean up daemon will perform its job. Default is 17 minutes past the hour. |
| oldevents.maxage.months | 4 | Events maximum age in months. Events older than the given number of months will be removed by the old events clean up daemon. |
| oldevents.cleanup.hours | 3 | Time of day at which old events clean up daemon will perform its job. It runs once daily at a given time, default is 3AM. |
| oldevents.cleanup.minutes | 23 | Minutes past the hour at which old events clean up daemon will perform its job. Default is 23 minutes past the hour. |
| reload.schedule.hours | \* | User information reloading daemon activation time. By default the daemon runs every hour. Concrete hour, like 3, will limit it to run daily. |
| reload.schedule.minutes | 10,25,40,55 | Minutes past the hour at which reloading daemon will run. By default it will run four times past every hour. |
| filteringdigest.schedule.hours | 2,6,12,14,23 | Daily, weekly and monthly digest daemon activation time. By default it will run at the following hours: 2,6,12,14,23. Recommended setting is hours as present in the T\_COUNTRY table. |
| filteringdigest.schedule.minutes | 0 | Minutes past the hour at which daily digest daemon will be run. By default it runs 0 minutes past the hours specified in the hours setting. Directly after daily digest the weekly digest is executed if the day of the week is as configured for daily digests. Directly after the weekly digest the monthly digest is executed, if the day of the month is as configured for monthly digests. |
| filteringdigest.schedule.dayofweek | 2 | Weekly digest daemon activation time. By default it will run every week at specified day of week, default is 2 meaning Monday, 3 would indicate Wednesday, and so on (1 for Sunday). |
| filteringdigest.schedule.dayofmonth | 1 | Monthly digest daemon activation time. By default it will run every month at the first day of a month. |
| default.am.delivery.zh | 6:00 | Default delivery time used when no country specific delivery time information is found. |

Note that all time values accept quartz-like expressions. For example, when day of week is required, "\*" will also be accepted, meaning "every day of the week", as well as "Mon,Tue,Wed" can be entered as well. Similarly, with hours settings, "\*" means every hour. Wherever minutes settings is needed, "\*/NN" will mean "every NN minutes". A list of minutes is also valid, like "10, 25, 40, 55".

As example, current production settings are as follows:

|  |  |
| --- | --- |
| Entry Name | Entry Value |
| cerebro.activity.sso.url | http://sappc2ci.swissre.com:8080/sap/zcrm\_sso?crmparam= |
| cerebro.activity.url | http://sappc2ci.swissre.com:8080/sap/crm\_logon/default.htm?sap-client=110&sap-syscmd=nocookie&crm-object-type=BT126\_APPT&crm-object-action=B&crm-object-value={activityId}&crm-object-keyname=OBJECT\_ID |
| hermes.env | PROD |
| hermes.technical.user | TECCL3P1 |
| hermes.from.address | hermes@swissre.com |
| default.am.delivery.zh | 6:00 |
| arbiter.maxage.days | 1 |
| arbiter.cleanup.hours | 1 |
| arbiter.cleanup.minutes | 10 |
| deliverylog.maxage.months | 3 |
| deliverylog.cleanup.hours | 1 |
| deliverylog.cleanup.minutes | 12 |
| oldevents.maxage.months | 4 |
| oldevents.cleanup.hours | 1 |
| oldevents.cleanup.minutes | 14 |
| reload.schedule.hours | \* |
| reload.schedule.minutes | 18 |
| dailydigest.schedule.hours | 2,6,12,14,23 |
| dailydigest.schedule.minutes | 0 |
| weeklydigest.schedule.dayofweek | Mon |
| weeklydigest.schedule.hours | 2,6,12,14,23 |
| weeklydigest.schedule.minutes | 7 |
| monthlydigest.schedule.dayofmonth | 1 |
| monthlydigest.schedule.hours | 2,6,12,14,23 |
| monthlydigest.schedule.minutes | 23 |
| subscription.link | http://web.swissre.com/webapp/cl3 |

## Web Service Configuration

Web service for accepting event submissions from Cerebro will be automatically configured upon deployment to Websphere.

Service name is: "ActivityService".

Service target namespace is: "http://cl3.swissre.com/services/activity/1/ ".