**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

1.1 What is present in EPS and missing in EPSMI 3

1.2 Features of EPSMI going beyond EPS 3

2 API 3

2.1 Queries by Electrum Wallet 3

2.2 Flows 4

2.3 Components 4

2.3.1 7

3 Configuration Management 7

3.1 Database Configuration Table – T\_ENV 7

3.2 Web Service Configuration 10

3.3 REST and Web Configuration 11

4 EJBs 11

4.1 Operational Repository EJBs 11

4.2 Reference and Configuration Repository EJBs 11

4.3 Targeting and Delivery EJBs 12

4.4 Pipeline EJBs 12

4.5 REST EJBs 13

4.6 Daemon EJBs 13

5 Database 14

5.1 Event 15

5.2 User 15

5.3 TimerEntity 15

5.4 DeliveryLog 15

# 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

## 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 |  |  |  |  |

## Flows

There are two main flows of information in Hermes:

1. activity submission -> database -> targeter-> deliverer
2. database -> digester -> targeter -> deliverer

Flow 1 is triggered by the Web Service listener.

Flow 2 is triggered by the application server timer (scheduler).

Flow 1 may result in one or many deliveries of type AP (Always Push).

Flow 2 may result in one or many deliveries of type DD, WD, and MD.

Flow 2 depends on database content created in previous executions of flow 1. Flow 1 does not have any dependency on flow 2.



## 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/ ".

## REST and Web Configuration

REST resources for providing user preferences information will be automatically configured upon deployment to Websphere.

The following context roots for web modules are used:

|  |  |
| --- | --- |
| Module | Context Root |
| sr-hermes-web | /webapp/cl3 |
| sr-hermes-services-v1 | /webapp/cl3/services/1 |
| sr-hermes-resources-v1 | /webapp/cl3/resources/1 |

# EJBs

## Operational Repository EJBs

Operational (Read Write) Repository Stateless EJBs:

|  |  |
| --- | --- |
| DeliveryLogRepository | One entry for each delivery. |
| EventRepository | Raw event information stored in 2 database tables. |
| TimerEntityRepository | Cluster arbitrage entries – needed for mechanism which enforces cluster wide uniqueness. |
| UserRepository | Repository of users and their preferences. |

## Reference and Configuration Repository EJBs

Reference (Read Only) Repository Stateless EJBs:

|  |  |
| --- | --- |
| CorePropertyTypeRepository | Constants for storing event property types. |
| CountryRepository | Mapping between countries and preferred delivery times. |
| EnvRepository | Hermes configuration. |
| SubscriptionFrequencyRepository | Constants for subscription frequencies. |

Human Resources Reference (Read Only) Repository Stateless EJBs:

|  |  |
| --- | --- |
| JobRepository | Repository of user contacts. |
| PartnerHierarchyRepository | Currently not used. |
| PartnerRepository | Repository of users. |
| UserFirewallRepository | Repository of user firewalls. |
| UserPartnersRepository | Repository of user contacts and partners. |

## Targeting and Delivery EJBs

Targeting Stateless EJBs:

|  |  |
| --- | --- |
| TargetingProcessor | Targets Always Push deliveries. Holds deliverer and uses generic targeter, to whom it passes the deliverer. |
| GenericTargeter | Performs targeting and delivery via a passed deliverer. |

Delivery Stateless EJBs:

|  |  |
| --- | --- |
| DeliveryProcessor | Delivers Always Push deliveries. |
| EmailDeliverer | Deliverer implementation passed to targeter or to delivery processor. Used for all email deliveries. |

## Pipeline EJBs

Pipeline Stateless EJBs:

|  |  |
| --- | --- |
| SimplePipeline | Pipeline accepting submitted events. Stores events and after making sure event is stored, returns control and invokes the asynchronous pipeline. |
| SimpleAsyncPipeline | Takes care of the Always Push delivery of submitted events. |

## REST EJBs

REST Stateless EJBs:

|  |  |
| --- | --- |
| UserPreferencesResource | Produces user preferences in a JSON format. It is used by user preferences web interface. |
| EventFeed | Produces RSS feed. |

## Daemon EJBs

Singleton EJBs:

|  |  |
| --- | --- |
| ActivityCredentialsVerificator | Verifies credentials of event submitter. |
| CleanUpArbiterDaemon | Periodically activated daemon cleaning up database entries of clustering arbiter. |
| CleanUpDeliveryLogDaemon | Periodically activated daemon cleaning up database entries in the delivery log table. |
| CleanUpOldEventsDaemon | Periodically activated daemon cleaning up events stored in the database. |
| DailyDigestDaemon | Periodically activated daemon generating and delivering individual daily digests. |
| DeliveryTimeConverter | Based on the database countries table provides delivery time. |
| DigestCreator | Holds event repository and deliverer, based on a set of events creates digests by calling targeter and passing a deliverer to it. |
| Gatekeeper | Used by modules which require that action is performed uniquely in a cluster scope. |
| MonthlyDigestDaemon | Periodically activated daemon generating and delivering individual monthly digests. |
| ReloadDaemon | Periodically activated daemon reloading user information. User preferences are always read from the database and do not need to be reloaded. |
| WeeklyDigestDaemon | Periodically activated daemon generating and delivering individual weekly digests. |
| ReferenceDataRepository | Keeps data about all users including lists of their clients and contacts. |

# Database

We can distinguish the following types of data tables used by Hermes:

* Operational data tables.
* Logging tables.
* Configuration tables.
* Constant tables.
* Human resources reference data tables.

|  |  |
| --- | --- |
| Operational Data Tables | T\_EVENT  T\_PROPERTY  T\_TIMER  T\_USER |
| Logging Tables | T\_DELIVERY\_LOG |
| Configuration Tables | T\_COUNTRY  T\_ENV |
| Constant Tables | T\_PROPERTY\_TYPE (S,B,C,I2,I4,L,D,DT)  T\_SUBSCRIPTION\_FREQUENCY (NO,AP,DD,WD,MD) |
| Human Resources Reference Data Tables | T\_HIERARCHY  T\_JOB  T\_PARTNER  T\_WORKFORCE\_FIREWALL  T\_WORKFORCE\_RELATIONSHIP  V\_USER\_PARTNERS |

The following domain model classes are related to the following data tables:

|  |  |
| --- | --- |
| Event, DomainEvent | T\_EVENT  T\_PROPERTY |
| CoreProperty | T\_PROPERTY |
| User, DomainUser | T\_USER + reference tables |
| TimerEntity | T\_TIMER |
| DeliveryLog | T\_DELIVERY\_LOG |
| CorePropertyType | T\_PROPERTY\_TYPE |
| SubscriptionFrequencyEntity | T\_SUBSCRIPTION\_FREQUENCY |
| Country | T\_COUNTRY |
| Env | T\_ENV |
| Job, JobPK | T\_JOB |
| Partner, SkinnyPartner | T\_PARTNER |
| UserFirewall, UserFirewallPK | T\_WORKFORCE\_FIREWALL |
| UserPartners, UserPartnersPK | V\_USER\_PARTNERS |
| PartnerHierarchy – NOT USED | T\_PARTNER\_HIERARCHY – NOT USED - TODO |

The following tables are not used directly by Hermes (yet they are used indirectly via V\_USER\_PARTNERS):

|  |  |
| --- | --- |
| T\_WORKFORCE\_RELATIONSHIP |  |

Tables not used at all (not even via V\_USER\_PARTNERS):

|  |  |
| --- | --- |
| T\_PARTNER\_HIERARCHY | TODO – remove or explain – Stephane DellAcqua |

## Event

Event T\_EVENT is represented as a collection of properties (T\_PROPERTY). Some properties may be chained, ordered, secondary chained, have text type and language code. All this complexity is hidden from the programmatic user and provided as an easy to use bean-like class named DomainEvent. DomainEvent is a decorator class for the underlying class Event. DomainEvent does not hold any other state than an instance of class Event.

Class CoreProperty is used internally as a class representing an entry in table T\_PROPERTY.

## User

User instances correspond to rows in table T\_USER. DomainUser holds data gathered from human resource reference tables and is not a simply decorator of User. User represents user preferences, while DomainUser represents full user information, including firewall, user clients, user contacts, user partner id, user aband, and more.

## TimerEntity

Class TimerEntity and the underlying table T\_TABLE is responsible for arbitrage between cluster participants. Entries are written by all servers every five minutes and for the next interval it is decided which cluster participant is the master. Master performs cluster-wide unique tasks such as deliveries.

## DeliveryLog

Class DeliveryLog and the underlying table T\_DELIVERY\_LOG is used to log each performed delivery. Information such as: the addressee, which events are part of the delivery, type of subscription (AP, DD, WD, MD), etc.