**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](#_Toc336329438)

[1.1 Flows 3](#_Toc336329439)

[1.2 Components 4](#_Toc336329440)

[1.2.1 Dependency Injection 7](#_Toc336329441)

[1.3 Transactions 10](#_Toc336329442)

[1.3.1 Transaction Annotation NOT\_SUPPORTED 11](#_Toc336329443)

[1.3.2 Transaction Annotation REQUIRED 12](#_Toc336329444)

[1.3.3 Transaction Annotation REQUIRES\_NEW 12](#_Toc336329445)

[2 Configuration Management 12](#_Toc336329446)

[2.1 Database Configuration Table – T\_ENV 13](#_Toc336329447)

[2.2 Web Service Configuration 17](#_Toc336329448)

[2.3 REST and Web Configuration 17](#_Toc336329449)

[3 EJBs 17](#_Toc336329450)

[3.1 Operational Repository EJBs 17](#_Toc336329451)

[3.2 Reference and Configuration Repository EJBs 18](#_Toc336329452)

[3.3 Targeting and Delivery EJBs 18](#_Toc336329453)

[3.4 Pipeline EJBs 19](#_Toc336329454)

[3.5 REST EJBs 19](#_Toc336329455)

[3.6 Daemon EJBs 19](#_Toc336329456)

[4 Database 20](#_Toc336329457)

[4.1 Event 21](#_Toc336329458)

[4.2 User 21](#_Toc336329459)

[4.3 TimerEntity 21](#_Toc336329460)

[4.4 DeliveryLog 22](#_Toc336329461)

# Overview

Hermes is a system which condenses and distributes information about discussion notes. Data is submitted to Hermes, Hermes stores it, condenses it and distrubutes to users according to their preferences.

One of the main design assumptions in Hermes is that generated content is never stored. Stored on the other hand are submitted events, in a raw format corresponding to the form in which they arrive. Whenever generated content is needed, it is created on demand and delivered right after generation without intermediate storage. Thus, Hermes stores raw data only and delivers dynamically generated content just in time when delivery is needed.

Users may choose the level of granularity of the delivered information. They may choose a finely grained information corresponding 1-1 to submitted events. They may also choose a more coarsely grained information grouped by time periods like day, week, or month. Finally, they can tailor information so that only events in which they participated, or in which their direct clients or contracts are involved, will be delivered to them.

All delivered content is individual, i.e., user specific. Delivery content is a juxtaposition of a subset of events and a user. This juxtaposition is created on demand and never persistently stored. A set of events contributing to the delivered content depends on the chosen timeframe and on the chosen interest (e.g., only events in which I participated, only events in which my clients or contacts are involved).

All delivered content is created in the same way; subset of stored events and a user determine content. For example, daily and weekly digests for a given user are conceptually the same, only input sets of events are different.

## 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.





### Dependency Injection

Hermes components are glued together into an application using a JEE dependency injection (CDI). They are typically defined as Stateless Session Beans SLSB using the @Stateless annotation. They are injected using the following annotation:

@EJB(beanInterface=IEventRepository.class)

private IEventRepository eventRepository;

The injection flavour used is the one based on the implemented interface. This has proven to be the most reliable way of injecting beans, working in both Websphere and Glassfish.

Some beans are defined as singletons, using the following annotation:

@Singleton(name="DigestCreator")

@Startup

To singletons belong beans from the following groups:

* Periodically invoked daemons.
* Digest creator.
* Gatekeeper – bean implementing cluster-aware singleton functionality.
* Verificator of event submitter credentials (ActivityCredentialsVerificator).
* User information repository.
* Targeter.

Singletons are beans for which we do not want to be instantiated more than once. Daemons must be singletons as we do not want multiple instances of timers to be scheduling deliveries.

Digest creator is a singleton as we do not want to have multiple targeting operations being performed at the same time. By default, a JEE6 singleton is using the container managed concurrency, with all methods locked by a WRITE lock. This makes sure that a methods behaves like having a "synchronized" keyword in from of it. Thus, only one thread can execute each method at a time. This is exactly the behavior we want for digest creator.

Gatekeeper could theroretically work having multiple instances, yet in order to conserve database locking, we need to minimize concurrent database accesses it would be making in such case.

Activity credentials verificator reads credentials information from the configuration table upon startup. Since multiple instances being initialized at different times could initialize themselves with different credentials, we make the verificator a singleton, to block such possibility.

User information repository is a singleton for the following reasons:

* Consistency – multiple instances could be initialized at different times containing inconsistent data.
* Memory conservation – multiple instances containing user data would consume too much memory.
* Time conservation – loading multiple instances with data would take too much time.
* Loading parallelism – loading multiple instaces with data in paralled would created database access conflicts.

Targeter is a singleton for the same reasons digest creator is a singleton. We do not want targeter methods to be executed in parallel.

To stateless session beans (non-singletons) belong beans from the following groups:

* Data repositories.
* REST providers.
* Pipeline proper and pipeline processors.
* Mail creators and deliverers.

Stateless session beans are beans for which we allow a certain degree of parallelism. Especially REST providers are driven by web requests and need to have the ability to scale up when the number of parallel requests increases.

NOTE: in Hermes we use dependency injection "by interface" almost exclusively. The only other type of injection used is as follows:

@Inject @Named(value="CapturingProcessor") PipelineProcessor capturingProcessor;

Where @Named is used as follows:

@Named(value="CapturingProcessor")

public class CapturingProcessor implements PipelineProcessor {

In CDI, @Named annotation is a way to give a name to an implementation. Since we have multiple implementations of interface PipelineProcessor in Hermes, we need to name them in order to give CDI a chance to distinguish them when injecting. In addition, some processors are not EJBs, so we could not use injection via @EJB.

#### Annotation @Startup

Annotation @Startup is used for singletons which need to be initialized when server starts up. To this group belong:

* Daemons – to initialize timers.
* DigestCreator – to be available to daemons.
* Gatekeeper – to be available to daemons.
* DeliveryTimeConverter – to load country information from the database.
* LoggingInitializer – to initialize logging levels.
* ReferenceDataRepository – to load full user data information.

Annotation @Startup is not working properly if modules are not initialized in a proper order, what can be achieved using the following Maven ear plugin configuration:

<plugin>

<groupId>org.apache.maven.plugins</groupId>

<artifactId>maven-ear-plugin</artifactId>

<version>2.7</version>

<configuration>

<version>6</version>

<skinnyWars>true</skinnyWars>

<defaultLibBundleDir>lib</defaultLibBundleDir>

<displayName>CLIB EAR Application</displayName>

<generateApplicationXml>true</generateApplicationXml>

<initializeInOrder>true</initializeInOrder>

<modules>

<ejbModule>

<groupId>com.swissre.cl3</groupId>

<artifactId>sr-hermes-targeting</artifactId>

</ejbModule>

<ejbModule>

<groupId>com.swissre.cl3</groupId>

<artifactId>sr-hermes-repository</artifactId>

</ejbModule>

<ejbModule>

<groupId>com.swissre.cl3</groupId>

<artifactId>sr-hermes-delivery</artifactId>

</ejbModule>

<ejbModule>

<groupId>com.swissre.cl3</groupId>

<artifactId>sr-hermes-pipeline</artifactId>

</ejbModule>

<webModule>

<groupId>com.swissre.cl3</groupId>

<artifactId>sr-hermes-web</artifactId>

<contextRoot>/webapp/cl3</contextRoot>

</webModule>

<webModule>

<groupId>com.swissre.cl3</groupId>

<artifactId>sr-hermes-services-v1</artifactId>

<contextRoot>/webapp/cl3/services/1</contextRoot>

</webModule>

<webModule>

<groupId>com.swissre.cl3</groupId>

<artifactId>sr-hermes-resources-v1</artifactId>

<contextRoot>/webapp/cl3/resources/1</contextRoot>

</webModule>

</modules>

Adding @Startup annotation may cause initialization failure if module ordering is not set up. Without @Startup annotation application may work fine, and then, after adding @Startup it will stop working, unless the ordering of module is enforced.

## Transactions

There are two main cases of transaction usage in Hermes:

1. Activity (event) submission via web service is transactional and the transaction scope stretches between the web service and the database repository. Web service caller's transaction is committed if and only if the submitted activity has been persisted by Hermes.
2. Reading data for digest creation is transactional, one event at a time. Event is either fully read from the database or not read at all.



### Transaction Annotation NOT\_SUPPORTED

Annotation NOT\_SUPPORTED is used by EJBs performing long database read operations where transaction timeout is likely to happen.

The following beans have been marked with TransactionAttributeType.NOT\_SUPPORTED for this reason:

* FilteringDigestDaemon – digest creation
* ReloadDeamon – reloading full user information
* DigestCreator
* ReferenceDataRepository – methods performing load of full user information
* UserRepository – methods performing load of full user information
* DigestResource – invoking digest creation from REST (DigestResource is not enabled in production)
* GenericTargeter – main method "target" performing bulk targeting

### Transaction Annotation REQUIRED

Annotation REQUIRED is used by EJBs performing database operations which need to be transactional, i.e., operations which should be rolled back in the presence of runtime exceptions. To this group belong SimplePipeline and SimpleAsyncPipeline.

### Transaction Annotation REQUIRES\_NEW

Annotation REQUIRES\_NEW is used limit the scope of rollback so that it does not cross boundaries of the annotated EJB.

The following beans have bean marked with TransactionAttributeType.REQUIRES\_NEW annotation for this reason:

* EventRepositoryPojo – method deleteAllBefore(Date) – this method is used by delete all events daemon, since the daemon is transactional, we do not want the deletion to be performed multiple times even in case of errors.
* RepositoryProcessor – we do not want the database rollback to propagate back to the calling web service, as we want to communicate errorsto the web service using the SOAP fault rather than by an uncontrolled exception.
* UserRepository
  + Method createDomainUser – we want to have a separate transaction for domain user creation in order to avoid timeouts if multiple users are loaded in bulk.
  + Method loadSkinnyPartnerMap – we want to create transaction as this method is being called from code which does not support transactions, in order to avoid transaction timeouts.
* GenericTargeter – on a method which is called by code which does not support transactions, in order to avoid transaction timeouts

# 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.