AIM

Application Functional Design

EETDB Data Access Services

Programme: UNIDO EETDB

Author: Nikolay Komissarenko

Creation Date: 16 April 2013

Last Updated: 23 April 2013

Version: 1.1

1. **Title, Subject, Last Updated Date, Reference Number**, **and** **Version** are marked by a Word Bookmark so that they can be easily reproduced in the header and footer of documents. When you change any of these values, be careful not to accidentally delete the bookmark. **You can make bookmarks visible by selecting Tools->Options…View and checking the Bookmarks option in the Show region.**

**Approvals:**

|  |  |
| --- | --- |
| TBD |  |
| TBD |  |
| TBD |  |

1. To add additional approval lines, press [Tab] from the last cell in the table above.
2. You can delete any elements of this cover page that you do not need for your document. For example, Copy Number is only required if this is a controlled document and you need to track each copy that you distribute.

## Document Control

Change Record

| Date | Author | Version | Change Reference |
| --- | --- | --- | --- |
|  |  |  |  |
| 16 April 2013 | Nikolay Komissarenko | 1.0 | draft |
| 23 April 2013 | Nikolay Komissarenko | 1.1 | Technical details added |

Reviewers

| Name | Position |
| --- | --- |
|  |  |
|  | Approver |
|  | Reviewer |
|  | Approver |

1. The copy numbers referenced above should be written into the **Copy Number** space on the cover of each distributed copy. If the document is not controlled, you can delete this table, the Note To Holders, and the **Copy Number** label from the cover page.

References

| Document Title | Description | Owner | Location |
| --- | --- | --- | --- |
|  |  |  |  |
| EETDB DB specification | Defines EETDB database structure and code |  |  |
|  |  |  |  |
|  |  |  |  |
|  |  |  |  |

### Contents

Document Control  
 ii

Contents iv

Overview v

Definitions v

Assumptions v

Basic Needs vi

Current state vi

Common requirements vi

EETDB Data Access Services requirements vi

EETDB Data Access Services architecture vii

EETDB API ix

Data Access Layer x

Configuration xi

Open and Closed Issues for this Deliverable xiii

Open Issues xiii

Closed Issues xiii

Appendix A xiv

1. To update the table of contents, put the cursor anywhere in the table and press [F9]. To change the number of levels displayed, select the menu option Insert‑>Index and Tables, make sure the Table of Contents tab is active, and change the Number of Levels to a new value.

## Overview

### Definitions

EETDB – Energy Efficient Technologies Data Bank

UNIDO – United Nations Industrial Development Organization

DAS – Data Access Services

CRUD – Create, Read, Update and Delete

MVC – Model, View, Controller

REST - Representational State Transfer

### Assumptions

1. EETDB is not a standalone web-enabled system, it’s part of the UNIDO web site <http://energy.unido.ru/>
2. If you use a user-friendly name for this customization as the replacement for <Subject>, the following paragraphs will default nicely.

## Basic Needs

### Current state

At present it is rather difficult to investigate and to describe in detail the whole situation with the energy efficiency management system of Russia on the whole and IEE technology online database systems in particular.

Characterized by a very large geographical distribution, the information on energy efficiency is still scattered and lacks good structuring. Despite the abundance of web sites the databases covering the technology aspect of energy efficiency are rather limited in number. There is apparent lack of structured databanks particularly focusing on energy efficiency/renewable energy technologies, innovations and equipment, i.e. those offering comprehensive industry-specific information which can be utilized by real practitioners – various industrial enterprises planning to introduce energy efficiency improvements.

### Common requirements

EETDB is recommended to opt for the energy efficiency portal with comprehensive and all-inclusive information on energy efficiency rather than a pure technology database which will target only a narrow segment of users.

Content which should be tailored to the information needs of the target audience and encompass different areas of energy efficiency which might be potentially interesting specifically for industries and SMEs. Apart from technology/equipment databases the portal must also include information on legislation/regulations, energy management systems and standards, registers of energy audit and energy service companies, profiles of typical projects implemented in the industrial or SME sector, and other relevant information.

Cross-linkages with other web sites and databases will provide users with access to global resources on energy efficiency (industry data, research papers, articles, documents, case studies and success stories, best practice examples, equipment and technologies).

### EETDB Data Access Services requirements

Data Access Services is part of EETDB solution and is aimed to provide access to the physical data storage isolating DB structure and specifics from higher layers. DAS provides an easy to use API for:

* data search, filtering and paging
* CRUD operations for EETDB data entities

DAS is to be used by EETDB UI layer for accessing and manipulating EETDB data: both Data Entry tools and User UI.

DAS is operating in context of EETDB Domain Model isolating physical DB entities.

### Tools and technologies

Development tools:

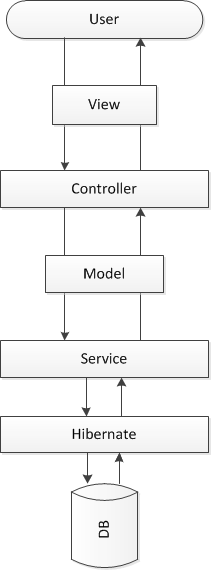
* Java Eclipse
* MySQL Workbench

Platform and technologies:

* Java 1.6
* Spring Framework
* Hibernate ORM Framework
* RDBMS: MySQL

## EETDB Data Access Services architecture

DAS architecture follows the de-facto system design standard for data services – RESTful MVC service:



**Controller** provides access to the application behavior that is defined through a service interface. Controller interprets user input and transforms it into a **Model** that is represented to the user by the **View**. Every method defined in the Controller is mapped to HTTP URL to follow REST notation, for example:

GET:<http://hostname/EetdbServices/entity/1> returns JSON view for a DB entity with id=1.

**Service** provides access to the DB storage for Controller. It is responsible for actual executing operations defined via Controller. The physical operations against the DB as well as transactional context are provided by Hibernate ORM framework.

**Model** represents the EETDB Domain Model.

A typical request handling is depicted on the following sequence diagram:



## EETDB API

EETDB DAS exposes its API via HTPP requests following REST notation as described below.

#### Entity Controller

Entity Controller provides methods for dealing with EETDB Entities (equipment, articles, catalogues etc, everything that is subject of persisting to the data bank)

Controller method: Entity createEntity(@RequestBody Entity entity)

Description: creates a new Entity.

URL pattern: /entity

HTTP method: *POST*

Controller method: Entity updateEntity(@RequestBody Entity entity)

Description: updates Entity passed to the method.

URL pattern: /entity

HTTP method: *PUT*

Controller method: String deleteEntity(@RequestBody Entity entity)

Description: deletes Entity passed to the method from DB.

URL pattern: /entity

HTTP method: *DELETE*

Controller method:

Entity getEntity(@PathVariable Long id, @MatrixVariable(required=**true**) **int** skip\_childs)

Description: reads an Entity by ID, skip\_childs controls if the method should load child entities.

URL pattern: /entity/{id}

HTTP method: *GET*

Controller method: Set<Entity> getEntitiesByTemplateCode(@PathVariable String code)

Description: returns all Entities of some class (template). Usually used for getting Catalogue entities (fuel types, equipment types, vendors etc). But can be used for getting, for example, all boilers (class BOILER) registered in EETDB. For list of entity classes see DB Functional specification.

URL pattern: /entities-by-code/{code}

HTTP method: *GET*

Controller method: Set<Entity> getEntitiesByTopic(@PathVariable Long topicId)

Description: returns all Entities referenced by the Topic, for example – all Entities of

URL pattern: /entities-by-topic/{topicId}

HTTP method: *GET*

#### Topic Controller

Controller method: Topic createTopic(@RequestBody Topic topic)

Description: creates a new Topic.

URL pattern: /topic

HTTP method: *POST*

Controller method: Topic updateTopic(@RequestBody Topic topic)

Description: updates Topic passed to the method.

URL pattern: /topic

HTTP method: *PUT*

Controller method: String deleteTopic(@RequestBody Topic topic)

Description: deletes Topic passed to the method from DB. The method doesn't delete all child Topics, they should be deleted by they own.

URL pattern: /topic

HTTP method: *DELETE*

Controller method: Topic getTopic(@PathVariable Long id)

Description: reads a Topic by ID.

URL pattern: /topic/{id}

HTTP method: *GET*

Controller method: Set<Topic> getRootTopics()

Description: returns root Topics i.e. Topics that do not have parent topics and, in contrary, are the root Topics of some categorization hierarchy. For list of predefined category trees see EETDB Functional Specification.

URL pattern: /topics

HTTP method: *GET*

#### Search Controller

Controller method: List<Topic> searchForTopics(@RequestParam("param") String param)

Description: returns all Topics that satisfy search criteria.

URL pattern: /search-for-topics

HTTP method: *GET*

Controller method: List<EntitySearchResult> searchForEntities(@RequestParam("param") String param)

Description: returns all Entities that satisfy search criteria. The method does search in Entity and Entity properties tables.

URL pattern: /search-for-entities

HTTP method: *GET*

#### ValueBlob Controller

Controller method: String getValueBlob(@PathVariable Long id, HttpServletResponse response)

Description: returns binary data for some Entity property, for example and image or a document.

URL pattern: /blob/{id}

HTTP method: *GET*

Controller method: String saveValueBlob(@RequestBody ValueBlob valueBlob, @RequestParam("blob") MultipartFile blobData)

Description: uploads binary data to the database.

URL pattern: /blob

HTTP method: *POST*

#### Template Controller

Controller method: Set<ValueType> getValueTypes()

Description: returns a list of available Value types (Integer, String, Image etc).

URL pattern: /value-types

HTTP method: *GET*

Controller method: EntityTemplate getTemplate(@PathVariable Long id, @MatrixVariable(required=**true**) **int** skip\_childs)

Description: returns a requested Entity Template with its properties if skip\_childs=1.

URL pattern: /template/{id}

HTTP method: *GET*

Controller method: EntityTemplate createTemplate(@RequestBody EntityTemplate template)

Description: creates a new Entity Template.

URL pattern: /template

HTTP method: *POST*

Controller method: String deleteTemplate(@RequestBody EntityTemplate template)

Description: deletes an Entity Template.

URL pattern: /template

HTTP method: *DELETE*

Controller method: EntityTemplate updateTemplate(@RequestBody EntityTemplate template)

Description: updates an Entity Template.

URL pattern: /template

HTTP method: *PUT*

Controller method: Set<EntityTemplate> getTemplates()

Description: returns all Entity Templates available within the system.

URL pattern: /templates

HTTP method: *GET*

## Data Access Layer

Data access interface provides CRUD and data search methods as follows:

Entity related operations:

**public** Entity getEntity(Long entityId, **boolean** skipChilds);

**public** Entity createEntity(Entity entity);

**public** Entity updateEntity(Entity entity);

**public** **void** deleteEntity(Entity entity);

**public** Set<Entity> getEntitiesByTemplateCode(String code);

**public** Set<Entity> getEntitiesByTopic(Long topicId);

Topic related operations:

**public** Topic getTopic(Long topicId);

**public** Topic createTopic(Topic topic);

**public** Topic updateTopic(Topic topic);

**public** **void** deleteTopic(Topic topic);

**public** Set<Topic> getRootTopics();

Entity Template related operations:

**public** EntityTemplate getEntityTemplate(Long templateId, **boolean** skipChilds);

**public** Set<EntityTemplate> getEntityTemplates();

**public** EntityTemplate createEntityTemplate(EntityTemplate template);

**public** EntityTemplate updateEntityTemplate(EntityTemplate template);

**public** **void** deleteEntityTemplate(EntityTemplate template);

Value Type related operations:

**public** Set<ValueType> getValueTypes();

Search related operations:

**public** List<Topic> searchForTopics(String param);

**public** List<EntitySearchResult> searchForEntities(String param);

BLOB operations:

**public** ValueBlob getValueBlob(Long blobId);

**public** **void** saveValueBlob(ValueBlob valueBlob, **byte**[] blobData);

#### ORM Mapping

Entity

<hibernate-mapping package=*"org.unido.eetdb.common.model"*>

<class name=*"Entity"* table=*"UNIDO\_ENTITY"* lazy=*"false"*>

<id name=*"id"* column=*"ENTITY\_ID"*>

<generator class=*"org.unido.eetdb.service.IdGenerator"* />

</id>

<version name=*"version"* column=*"VERSION"* />

<property name=*"name"* column=*"ENTITY\_NAME"* />

<property name=*"lastUpdatedBy"* column=*"UPDATED\_BY"* />

<many-to-one name=*"entityTemplate"* fetch=*"join"*

column=*"ENTITY\_TEMPLATE\_ID"* unique=*"true"* />

<set name=*"properties"* table=*"UNIDO\_ENTITY\_PROPERTY"* fetch=*"join"*

inverse=*"true"* cascade=*"all-delete-orphan"*>

<key column=*"ENTITY\_ID"* />

<one-to-many class=*"EntityProperty"* />

</set>

<set name=*"childEntities"* table=*"UNIDO\_ENTITY\_LINK"* lazy=*"true"*

cascade=*"none"*>

<key column=*"PARENT\_ENTITY\_ID"* />

<many-to-many column=*"LINKED\_ENTITY\_ID"* class=*"Entity"* />

</set>

<set name=*"parentTopics"* table=*"UNIDO\_ENTITY\_REFERENCE"*

cascade=*"none"*>

<key column=*"ENTITY\_ID"* />

<many-to-many column=*"TOPIC\_ID"* class=*"Topic"* />

</set>

</class>

</hibernate-mapping>

Entity property

<hibernate-mapping package=*"org.unido.eetdb.common.model"*>

<class name=*"EntityProperty"* table=*"UNIDO\_ENTITY\_PROPERTY"* lazy=*"false"*>

<id name=*"id"* column=*"ENTITY\_PROPERTY\_ID"*>

<generator class=*"identity"* />

</id>

<version name=*"version"* column=*"VERSION"* />

<property name=*"value"* column=*"VALUE"* />

<property name=*"lastUpdatedBy"* column=*"UPDATED\_BY"* />

<many-to-one name=*"templateProperty"* fetch=*"join"* column=*"TEMPLATE\_PROPERTY\_ID"*

unique=*"true"* />

<many-to-one name=*"parentEntity"* column=*"ENTITY\_ID"* not-null=*"true"* />

</class>

</hibernate-mapping>

Entity search result

<hibernate-mapping package=*"org.unido.eetdb.common.model"*>

<class name=*"EntitySearchResult"* mutable=*"false"*>

<id name=*"entityId"* column=*"ENTITY\_ID"* />

<property name=*"entityName"* column=*"ENTITY\_NAME"* />

<property name=*"entityDescription"* column=*"DESCRIPTION"* />

</class>

<sql-query name=*"searchForEntities"*>

<return class=*"EntitySearchResult"* />

<![CDATA[CALL search(:param)]]>

</sql-query>

</hibernate-mapping>

Entity template

<hibernate-mapping package=*"org.unido.eetdb.common.model"*>

<class name=*"EntityTemplate"* table=*"UNIDO\_ENTITY\_TEMPLATE"* lazy=*"false"*>

<id name=*"id"* column=*"ENTITY\_TEMPLATE\_ID"*>

<generator class=*"identity"* />

</id>

<version name=*"version"* column=*"VERSION"* />

<property name=*"name"* column=*"TEMPLATE\_NAME"* />

<property name=*"code"* column=*"TEMPLATE\_CODE"* />

<property name=*"lastUpdatedBy"* column=*"UPDATED\_BY"* />

<set name=*"properties"* table=*"UNIDO\_ENTITY\_TEMPLATE\_PROPERTY"* inverse=*"true"*

cascade=*"all-delete-orphan"*>

<key column=*"ENTITY\_TEMPLATE\_ID"* not-null=*"true"* />

<one-to-many class=*"EntityTemplateProperty"* />

</set>

</class>

</hibernate-mapping>

Entity template property

<hibernate-mapping package=*"org.unido.eetdb.common.model"*>

<class name=*"EntityTemplateProperty"* table=*"UNIDO\_ENTITY\_TEMPLATE\_PROPERTY"* lazy=*"false"*>

<id name=*"id"* column=*"TEMPLATE\_PROPERTY\_ID"*>

<generator class=*"identity"* />

</id>

<version name=*"version"* column=*"VERSION"* />

<property name=*"name"* column=*"PROPERTY\_NAME"* />

<property name=*"code"* column=*"PROPERTY\_CODE"* />

<property name=*"displayInGrid"* type=*"numeric\_boolean"* column=*"DISPLAY\_IN\_GRID"* />

<property name=*"mandatory"* type=*"numeric\_boolean"* column=*"MANDATORY"* />

<property name=*"searchable"* type=*"numeric\_boolean"* column=*"SEARCHABLE"* />

<property name=*"unitOfMeasure"* column=*"UNIT\_OF\_MEASURE"* />

<property name=*"lastUpdatedBy"* column=*"UPDATED\_BY"* />

<many-to-one name=*"parentTemplate"* column=*"ENTITY\_TEMPLATE\_ID"* not-null=*"true"* />

<many-to-one name=*"valueType"* fetch=*"join"* column=*"VALUE\_TYPE\_ID"* unique=*"true"* />

</class>

</hibernate-mapping>

Topic

<hibernate-mapping package=*"org.unido.eetdb.common.model"*>

<class name=*"Topic"* table=*"UNIDO\_TOPIC"* lazy=*"false"*>

<id name=*"id"* column=*"TOPIC\_ID"*>

<generator class=*"identity"* />

</id>

<version name=*"version"* column=*"VERSION"* />

<property name=*"name"* column=*"TOPIC\_NAME"* />

<property name=*"description"* />

<property name=*"lastUpdatedBy"* column=*"UPDATED\_BY"* />

<set name=*"childTopics"* table=*"UNIDO\_TOPIC\_LINK"*>

<key column=*"PARENT\_TOPIC\_ID"* />

<many-to-many column=*"LINKED\_TOPIC\_ID"* class=*"Topic"* />

</set>

</class>

<sql-query name=*"searchForTopics"*>

<return class=*"Topic"* />

<![CDATA[CALL search\_topic(:param)]]>

</sql-query>

</hibernate-mapping>

Value BLOB

<hibernate-mapping package=*"org.unido.eetdb.common.model"*>

<class name=*"ValueBlob"* table=*"UNIDO\_BLOB"* lazy=*"false"*>

<id name=*"id"* column=*"BLOB\_ID"*>

<generator class=*"identity"* />

</id>

<version name=*"version"* column=*"VERSION"* />

<property name=*"mimeType"* column=*"DATA\_TYPE"* />

<property name=*"content"* column=*"CONTENT"* />

<property name=*"name"* column=*"FILE\_NAME"* />

<property name=*"lastUpdatedBy"* column=*"UPDATED\_BY"* />

</class>

</hibernate-mapping>

Value Type

<hibernate-mapping package=*"org.unido.eetdb.common.model"*>

<class name=*"ValueType"* table=*"UNIDO\_VALUE\_TYPE"* lazy=*"false"*>

<id name=*"id"* column=*"VALUE\_TYPE\_ID"*>

<generator class=*"identity"* />

</id>

<version name=*"version"* column=*"VERSION"* />

<property name=*"format"* />

<property name=*"type"* />

<property name=*"lastUpdatedBy"* column=*"UPDATED\_BY"* />

</class>

</hibernate-mapping>

## Configuration

There are three main configuration files for DAS:

* web.xml: provides basic Spring MVC configuration parameters and normally shouldn't be changed;
* rest-servlet.xml: provides DAS servlet specific configuration parameters. The only parameter that can be the matter of interest for the support personnel is the following:

<bean id=*"multipartResolver"* class=*"org.springframework.web.multipart.commons.CommonsMultipartResolver"*>

<property name=*"maxUploadSize"* value=*"10000000"* />

</bean>

*maxUploadSize* defines the maximum allowed size for uploaded binary data like images, documents etc.

* rest-context.xml: provides custom DAS parameters like DB connection details etc.

Database connection details are configured in this section:

<bean id=*"dataSource"* class=*"org.apache.commons.dbcp.BasicDataSource"*

destroy-method=*"close"*>

<property name=*"driverClassName"* value=*"com.mysql.jdbc.Driver"* />

<property name=*"url"* value=*"jdbc:mysql://IP/eetdb?characterEncoding=UTF-8&amp;noAccessToProcedureBodies=true"* />

<property name=*"maxActive"* value=*"-1"* />

<property name=*"maxIdle"* value=*"1"* />

<property name=*"username"* value=*""* />

<property name=*"password"* value=*""* />

</bean>

Hibernate specific parameters can be amended in this section:

<bean id=*"sessionFactory"* class=*"org.springframework.orm.hibernate4.LocalSessionFactoryBean"*>

<property name=*"dataSource"* ref=*"dataSource"* />

<property name=*"mappingResources"*>

<list>

<value>Entity.hbm.xml</value>

<value>EntityProperty.hbm.xml</value>

<value>EntityTemplate.hbm.xml</value>

<value>EntityTemplateProperty.hbm.xml</value>

<value>ValueType.hbm.xml</value>

<value>Topic.hbm.xml</value>

<value>EntitySearchResult.hbm.xml</value>

<value>ValueBlob.hbm.xml</value>

</list>

</property>

<property name=*"hibernateProperties"*>

<props>

<prop key=*"hibernate.dialect"*>org.hibernate.dialect.MySQLDialect</prop>

<prop key=*"hibernate.show\_sql"*>true</prop>

<prop key=*"hibernate.format\_sql"*>true</prop>

<prop key=*"hibernate.max\_fetch\_depth"*>1</prop>

<prop key=*"hibernate.cache.region.factory\_class"*>org.hibernate.cache.ehcache.SingletonEhCacheRegionFactory</prop>

<prop key=*"hibernate.cache.use\_query\_cache"*>true</prop>

<prop key=*"hibernate.cache.use\_second\_level\_cache"*>true</prop>

<prop key=*"configurationResourceName"*>ehcache.xml</prop>

<prop key=*"hibernate.connection.charSet"*>utf8</prop>

</props>

</property>

</bean>

#### Logging

Logging in provided via Log4j and configured via log4j.properties file located in WebContent/WEB-INF/classes folder.

## Deployment

DAS is distributed in a web archive (WAR) file. Deployment of DAS is a standard procedure for WARs – the file should be copied in to the Tomcat webapp/ folder and then it will get installed by the server.

After the installation is complete it can be required to update DAS config files (ref:Configuration).

## Open and Closed Issues for this Deliverable

1. **Define Auto Cash Rules:** Dell Prepaid Rule, Dell Standard RuleAdd open issues that you identify while writing or reviewing this document to the open issues section. As you resolve issues, move them to the closed issues section and keep the issue ID the same. Include an explanation of the resolution.  
     
   When this deliverable is complete, any open issues should be transferred to the project- or process-level Risk and Issue Log (PJM.CR.040) and managed using a project level Risk and Issue Form (PJM.CR.040). In addition, the open items should remain in the open issues section of this deliverable, but flagged in the resolution column as being transferred.

### Open Issues

|  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- |
|  |  |  |  |  |  |  |
|  |  |  |  |  |  |  |
|  |  |  |  |  |  |  |
|  |  |  |  |  |  |  |
|  |  |  |  |  |  |  |
|  |  |  |  |  |  |  |
|  |  |  |  |  |  |  |
|  |  |  |  |  |  |  |
|  |  |  |  |  |  |  |

### Closed Issues

| ID | Raised by | Issue | Resolution | Owner | Target Date | Impact Date |
| --- | --- | --- | --- | --- | --- | --- |
|  |  |  |  |  |  |  |
| 1 |  |  |  |  |  |  |
|  |  |  |  |  |  |  |
|  |  |  |  |  |  |  |
|  |  |  |  |  |  |  |
|  |  |  |  |  |  |  |
|  |  |  |  |  |  |  |

## Appendix A

**JSON examples**

**Entity Data Object example**