AIM

Application Functional Design

EETDB Mass Upload Tool and Extended Search capabilities

Programme: UNIDO EETDB

Author: Nikolay Komissarenko

Creation Date: 2 August 2013

Last Updated: 20 August 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 |
| --- | --- | --- | --- |
|  |  |  |  |
| 2 August 2013 | Nikolay Komissarenko | 1.0 | draft |
| 20 August 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

Contents

[Document Control ii](#_Toc367658691)

[Contents iv](#_Toc367658692)

[Overview v](#_Toc367658693)

[Definitions v](#_Toc367658694)

[Assumptions v](#_Toc367658695)

[Basic Needs vi](#_Toc367658696)

[Current state vi](#_Toc367658697)

[Common requirements vi](#_Toc367658698)

[EETDB Mass Upload Tool requirements vi](#_Toc367658699)

[Tools and technologies vi](#_Toc367658700)

[EETDB Mass Upload Tool architecture vii](#_Toc367658701)

[Mass Upload Tool technical specification x](#_Toc367658702)

[Configuration xiv](#_Toc367658703)

[Deployment xvi](#_Toc367658704)

[Open and Closed Issues for this Deliverable xvii](#_Toc367658705)

[Open Issues xvii](#_Toc367658706)

[Closed Issues xvii](#_Toc367658707)

[Appendix A xviii](#_Toc367658708)

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 EETDB Admin Tools allow a system user to manage and maintain EETDB data by creating, deleting or updating the database entities. The Admin Tools UI provides a convenient and simple way to update or create a single entity. However it’s not suitable for managing or creating multiple entities. In some cases it’s required to add a set of new entities to the system, for example, equipment catalogue that can contain hundreds of records.

### Common requirements

The system should provide a convenient and easy to use way to add/update multiple EETDB entities.

### EETDB Mass Upload Tool requirements

The Mass Upload Tool is part of EETDB solution and is aimed to provide an ability to upload data files to the system in the following file formats:

* Comma Separated Values (\*.csv)
* Microsoft Excel (\*.xls, \*.xlsx)

The data files should comply with the EETDB Data Format Specification.

The Mass Upload Tool should be easily extendable to support other file formats if required (for example (XML).

The Mass Upload Tool should support different transport protocols for pushing data files to the system, the minimal set is:

* By sending a data file in email attachment (SMTP)
* By uploading via FTP
* By uploading on the EETDB web pages (HTTP upload)

The Mass Upload Tool should validate the data file to make sure it complies with the Data Format Specification. When parsing the file and adding data to the EETDB database all errors should be logged and then displayed in a report on the web pages. All successfully parsed records should be added to the database.

### Tools and technologies

Development tools:

* Java Eclipse
* MySQL Workbench

Platform and technologies:

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

## EETDB Mass Upload Tool architecture

The Mass Upload Tool is a Java-based multithread service. The main thread is a container for hosting services responsible for handling incoming data files. The Service Host is configurable to run any number of services implementing Service interface. The basic implementation involves running two main services:

* POP3 Daemon which is responsible for monitoring email inbox
* File System Daemon which is monitoring a file system folder for incoming data files.



When receiving a data file the transport layer handlers use Parsers to extract EETDB data from the file and then pass the extracted data to the DB Helper to persist the data.

POP3Daemon is monitoring a configurable email inbox and in case there is an incoming email it tries to save mail attachments to the folder monitored by the File System Daemon which does rest of the work.



File System Daemon is monitoring a system folder (configurable) for incoming data files. If any files appear in the folder the service checks if the files are of the supported file format. Files of the supported formats get parsed by an appropriate parser and then persisted to the DB. Non supported files get moved to the Garbage folder for further manual inspection.

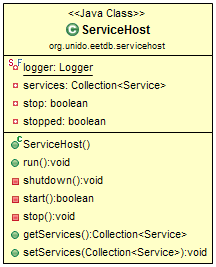


There is no dedicated service for FTP transport. This requirement is fulfilled by configuring FTP server so the incoming folder is the one monitored by the File System Daemon.

## Mass Upload Tool technical specification

#### Service Host

ServiceHost is the main execution thread of the Mass Upload Tool. This manages an internal list of services to run and host.



The list of services to host is configurable via Mass Upload Tool configuration file:

<bean id=*"serviceHost"* class=*"org.unido.eetdb.servicehost.ServiceHost"*>

<property name=*"services"*>

<list>

<ref bean=*"fileDaemon"* />

<ref bean=*"pop3Daemon"* />

</list>

</property>

</bean>

The ServiceHost implements Runnable interface and has only one public method run() that is the main function of the execution thread.

Private methods:

private boolean start() – starts all hosted services. It also registers a shutdown hook to stop the ServiceHost.

private void stop() – stops all hosted services.

private void shutdown() – waits until all hosted services stopped and then stops the ServiceHost thread. This method is registered as a system shutdown hook in start() method.

#### Services

All services hosted within the ServiceHost should be inherited from the Service interface. The Service interface defines basic operations required to run and manage state of a service.

Private members:

id:int – unique service ID that is used for accessing the service instance in the Service Host

status:ServiceStatus – current status. Possible values are:

* Stopping
* Stopped
* Running
* Starting
* Broken

Public methods:

public abstract boolean start() – starts the service instance, returns true if the service started successfully, otherwise - false.

public abstract void stop() – stops the service instance.

**POP3Daemon**

Parameters of the service are configurable via the Mass Upload Tool configuration file.

Private members:

private String pathToStorage – full path to the folder monitored by the FileSystemDaemon;

private String pop3Host – IP address of the POP3 server;

private int port = 110 – defaulted to POP3 port;

private String uid – user ID for accessing to the POP3 inbox;

private String pwd – user password;

private Timer refreshTimer – frequency of checking the inbox for new emails;

private void checkForMail() is the main thread function that performs all job. It’s scheduled to run every n seconds defined in refreshTimer parameter.

Public methods are inherited from the Service interface and implement service initialization and shutting down logic.

Configuration section for POP3Daemon:

<bean id=*"pop3Daemon"* class=*"org.unido.eetdb.daemon.Pop3Daemon"*>

<property name=*"pathToStorage"* value=*"C:\\Temp"* />

<property name=*"pop3Host"* value=*""* />

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

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

</bean>

**FileSystemDaemon**

private String pathToMonitor – full path to the folder monitored by the FileSystemDaemon;

private String pathToStore – full path to the folder to store processed files;

private Timer refreshTimer – defines frequency the monitored folder is checked for new files;

private Map<String, Parser> supportedFormats – map between supported files formats and corresponding parsers;

private DbHelper dbHelper – instance of DB Helper class;

private void checkFolderToMonitor() is the main thread function that performs all job. It’s scheduled to run every n seconds defined in refreshTimer parameter.

private static String getUniqueName(File file) – is a static helper function for generating unique file names for processed and failed files moved to special folders.

Public methods are inherited from the Service interface and implement service initialization and shutting down logic.

Configuration section for FileSystemDaemon:

<bean id=*"fileDaemon"* class=*"org.unido.eetdb.daemon.FileSystemDaemon"*>

<property name=*"pathToMonitor"* value=*"C:\\Temp\Monitor"* />

<property name=*"pathToStore"* value=*"C:\\Temp\Storage"* />

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

<property name=*"supportedFormats"*>

<map>

<entry key=*"csv"* value-ref=*"csvParser"* />

<entry key=*"xls"* value-ref=*"excelParser"* />

<entry key=*"xlsx"* value-ref=*"excelParser"* />

</map>

</property>

</bean>

#### 

#### Parsers

The initial implementation of the Mass Upload Tool supports Comma Separated Values (CSV) and Microsoft Excel file formats. However the architecture allows easy extending the tool by creating parsers for other formats.

All parsers are inherited from the Parser interface that defines a single function:

public List<Entity> parse(File file) throws Exception.

The function takes a file to parse as the input and returns a list of EETDB entities parsed from the file.

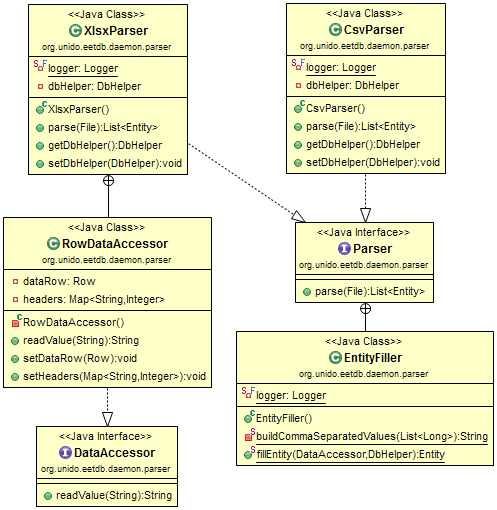
The Parser interface also provides a static helper class EntityFiller that encapsulates logic that is common for all parsers for creating EETDB Entity from the file data.



The EntityFiller class provides a single public function

public static Entity fillEntity(DataAccessor dataAccessor, DbHelper dbHelper) that takes an object implementing the DataAccessor interface that isolates format specific logic for accessing the data and fills the Entity with all data parsed from the file.

CsvParser and XlsxParser both implement their own DataAccessor objects to provide the format specific access to the data.



Configuration section for Parsers:

<bean id=*"csvParser"* class=*"org.unido.eetdb.daemon.parser.CsvParser"*>

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

</bean>

<bean id=*"excelParser"* class=*"org.unido.eetdb.daemon.parser.XlsxParser"*>

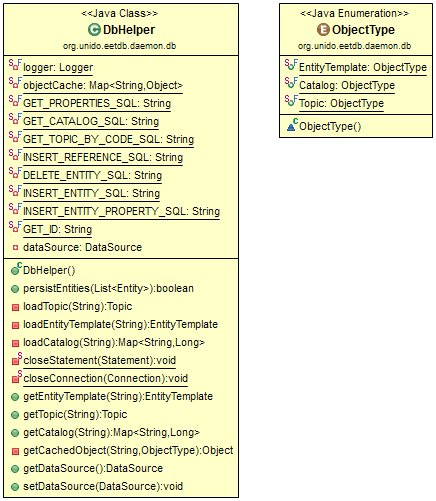
<property name=*"dbHelper"* ref=*"dbHelper"* />

#### </bean>

#### DB Helper

DB Helper class provides public methods for accessing EETDB data and for persisting data extracted from the data files.

DBHelper maintains an internal cache which optimizes work of Parsers and minimizes DB load.



Configuration section for DBHelper:

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

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

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

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

<property name=*"url"*

value=*"jdbc:mysql://50.56.189.15:3306/eetdb?characterEncoding=UTF-8&amp;noAccessToProcedureBodies=true"* />

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

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

</bean>

<bean id=*"dbHelper"* class=*"org.unido.eetdb.daemon.db.DbHelper"*>

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

</bean>

## Configuration

The Mass Upload Tool configuration file allows to configure all services and parsers via dependency injection. The typical configuration looks like the bellow:

<?xml version=*"1.0"* encoding=*"UTF-8"*?>

<beans

xmlns=*"http://www.springframework.org/schema/beans"*

xmlns:xsi=*"http://www.w3.org/2001/XMLSchema-instance"*

xmlns:context=*"http://www.springframework.org/schema/context"*

xsi:schemaLocation=*"*

*http://www.springframework.org/schema/beans*

*http://www.springframework.org/schema/beans/spring-beans-3.0.xsd"*>

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

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

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

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

<property name=*"url"* value=*"jdbc:mysql://50.56.189.15:3306/eetdb?characterEncoding=UTF-8&amp;noAccessToProcedureBodies=true"* />

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

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

</bean>

<bean id=*"dbHelper"* class=*"org.unido.eetdb.daemon.db.DbHelper"*>

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

</bean>

<bean id=*"csvParser"* class=*"org.unido.eetdb.daemon.parser.CsvParser"*>

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

</bean>

<bean id=*"excelParser"* class=*"org.unido.eetdb.daemon.parser.XlsxParser"*>

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

</bean>

<bean id=*"fileDaemon"* class=*"org.unido.eetdb.daemon.FileSystemDaemon"*>

<property name=*"pathToMonitor"* value=*"C:\\Temp\Monitor"* />

<property name=*"pathToStore"* value=*"C:\\Temp\Storage"* />

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

<property name=*"supportedFormats"*>

<map>

<entry key=*"csv"* value-ref=*"csvParser"* />

<entry key=*"xls"* value-ref=*"excelParser"* />

<entry key=*"xlsx"* value-ref=*"excelParser"* />

</map>

</property>

</bean>

<bean id=*"pop3Daemon"* class=*"org.unido.eetdb.daemon.Pop3Daemon"*>

<property name=*"pathToStorage"* value=*"C:\\Temp"* />

<property name=*"pop3Host"* value=*""* />

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

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

</bean>

<bean id=*"serviceHost"* class=*"org.unido.eetdb.servicehost.ServiceHost"*>

<property name=*"services"*>

<list>

<ref bean=*"fileDaemon"* />

</list>

</property>

</bean>

</beans>

#### Logging

Logging in provided via Log4j and configured via log4j-config.xml file located in the application folder. Typical configuration looks like the below:

<?xml version=*"1.0"* encoding=*"UTF-8"* ?>

<!DOCTYPE log4j:configuration SYSTEM "log4j.dtd">

<log4j:configuration xmlns:log4j=*"http://jakarta.apache.org/log4j/"*>

<appender name=*"console"* class=*"org.apache.log4j.ConsoleAppender"*>

<param name=*"Target"* value=*"System.out"*/>

<layout class=*"org.apache.log4j.EnhancedPatternLayout"*>

<param name=*"ConversionPattern"* value=*"[%d{dd-MM-yyyy HH:mm:ss}] %-5p %c{1} - %m%n%throwable"*/>

</layout>

</appender>

<!--<appender name="file" class="org.apache.log4j.RollingFileAppender">

<param name="File" value="/var/log/kiwi/service-host.log"/>

<param name="MaxBackupIndex" value="1"/>

<param name="MaxFileSize" value="1MB"/>

<layout class="org.apache.log4j.EnhancedPatternLayout">

<param name="ConversionPattern" value="%-5p %c{1} - %m%n%throwable"/>

</layout>

</appender>-->

<root>

<priority value =*"debug"* />

<appender-ref ref=*"console"* />

<!--<appender-ref ref="file" />-->

</root>

</log4j:configuration>

## Deployment

Mass Upload Tools is a Java console application and doesn’t require installation. This is required to configure the application to start at system startup.

## Extended Search Capabilities

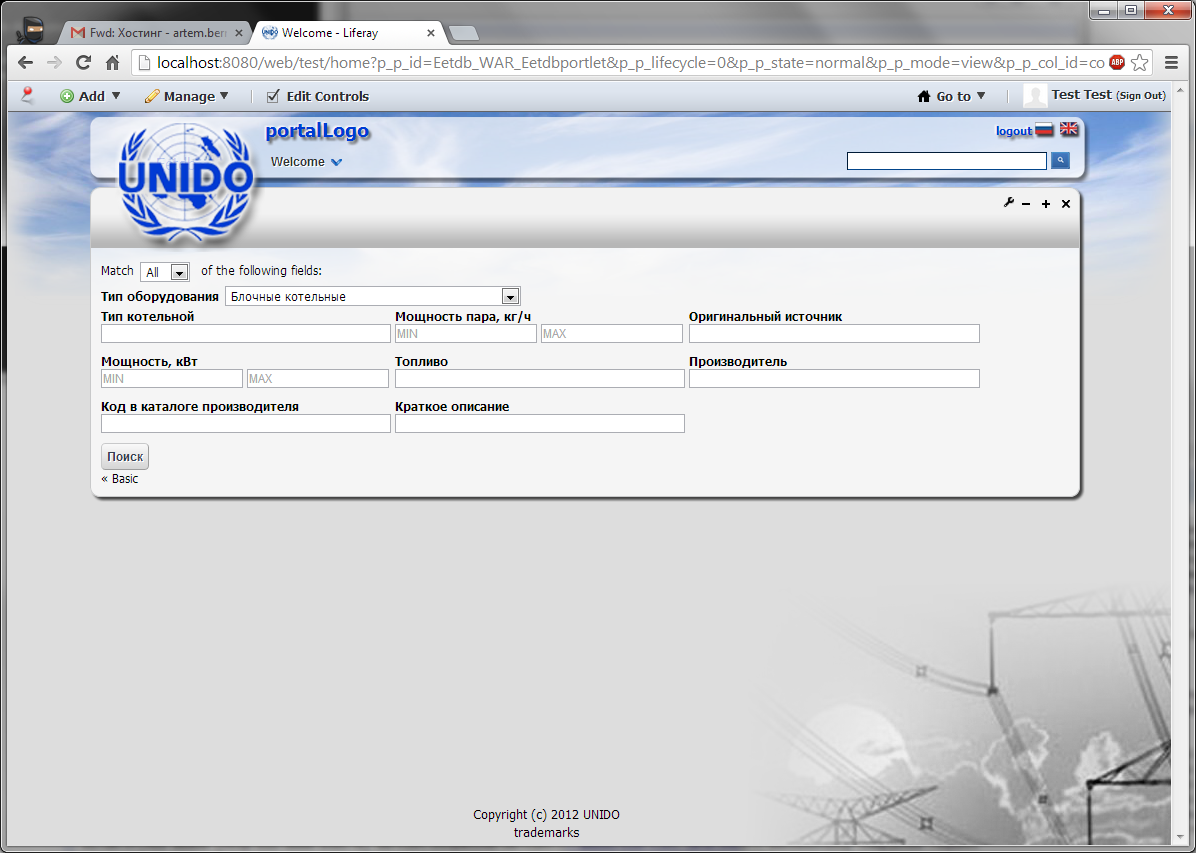
The EETDB UI interface currently provides only a simple full text search against the EETDB data which is not always effective and provides too many results sometimes which are difficult to narrow down.

As part of the second phase of the project extended search capabilities are provided with the user interface for more precise selection of EETDB data.

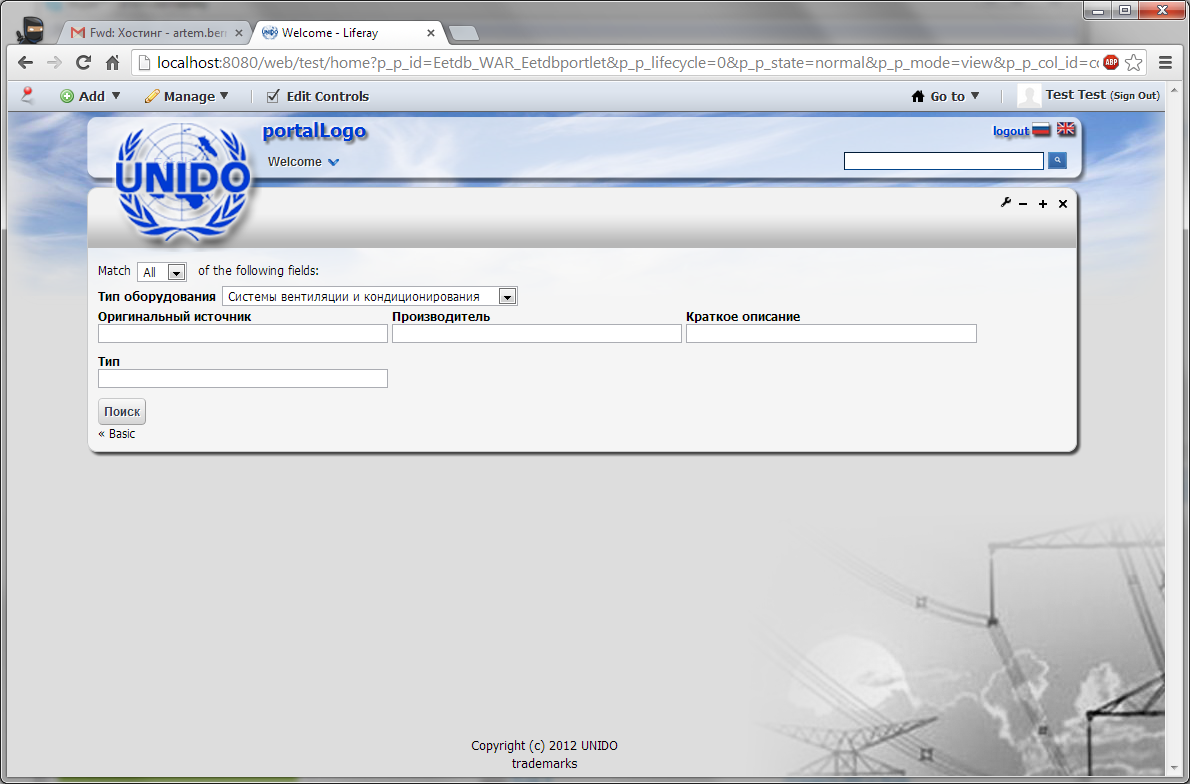
The Extended Search module dynamically creates search forms depending on the search context, i.e. when searching for Boilers users will get a search form containing search parameters only specific to boilers, when searching for Engines – the form will have search criteria to enter specific for engines, etc.

The module builds the search forms basing on the Entity Templates every Entity within EETDB belong to.

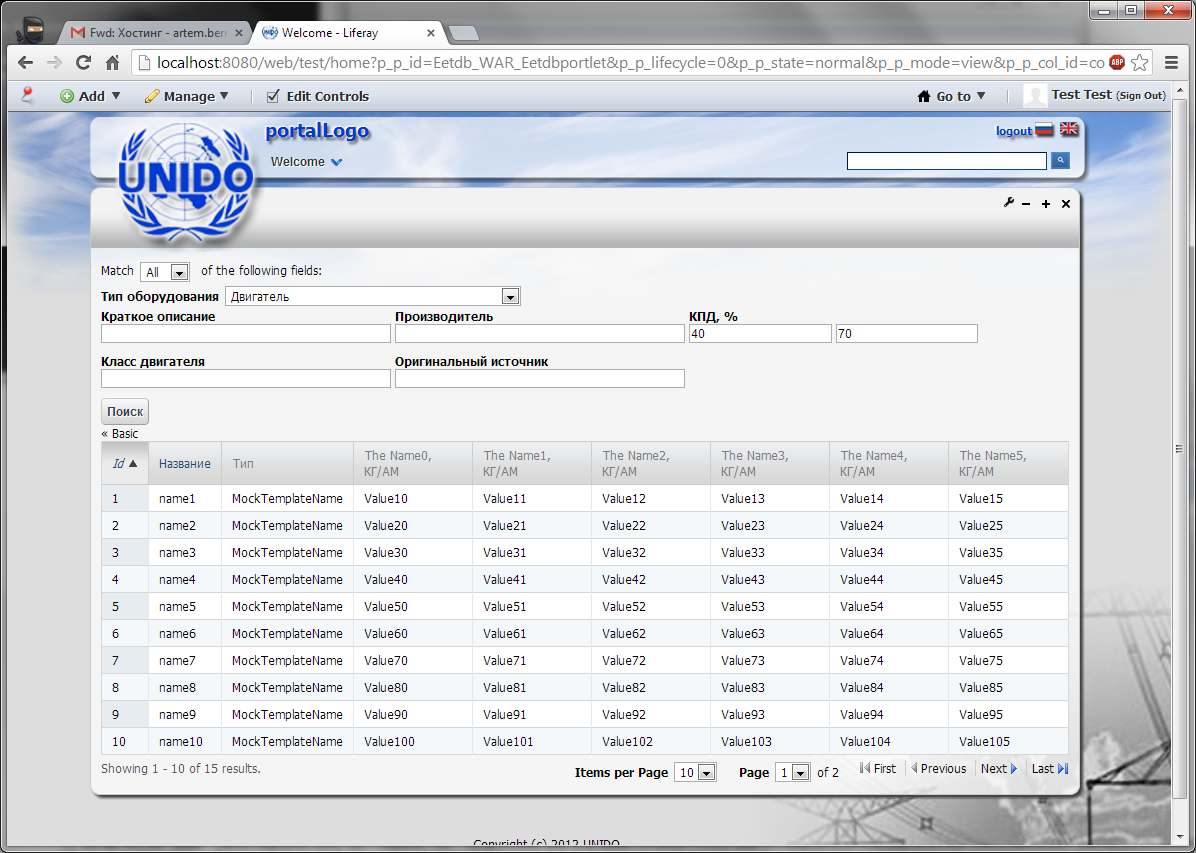
Here are some examples of the dynamically build forms:



*Search form for boilers*



*Search form for air conditioning systems*



*Search result*

## 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**