MOTECH Documentation

Release 0.24-SNAPSHOT

Grameen Foundation

August 19, 2015

Contents

The topics below will give you an introduction to MOTECH, an open source mHealth platform developed by the Grameen Foundation. Some of these topics are quite technical in nature (i.e. aimed at software developers), while others are more accessible to people who aren't familiar with software code.

CHAPTER 1

About MOTECH

Mobile Technology for Community Health (MOTECH) is a modular, extensible open source software project originally designed for mobile health (mHealth), which can also be used outside of the health domain. It allows organizations to use mobile technology to communicate information to patients, collect data about patients, alert caregivers to patients' status, and schedule caregivers' work. The modular system allows organizations to choose among multiple mHealth technologies and to enable data sharing for users of the system.

1.1 MOTECH Architecture Overview

MOTECH consists of the core platform and several modules, each providing use of a technology such as SMS or email, or access to an external system such as CommCare. Organizations can choose to install one or more modules, and developers can extend MOTECH by writing new modules. MOTECH is written in Java. It depends on open source systems including Apache Tomcat, Apache ActiveMQ, and Quartz. For more information about MOTECH architecture, see *Core Architecture* and *Modules Architecture*.

1.2 What can MOTECH do?

The MOTECH Platform can be used for setting appointments, tracking any scheduled activity, and managing workers deployed in the field. Its initial implementations have been for mHealth projects that improve health by sending messages to patients and caregivers based on an evaluation of the recommended schedule of care compared to the patient's health-related actions. Some features of typical MOTECH-based applications are:

Communicate information to patients via voice or SMS according to a schedule of care defined for the patient's condition, e.g.:

- Reminders for ANC appointments, lab visits, etc.
- Reminders to take medication on schedule, e.g., DOTS, ART, etc.
- Reminder notices to take children for scheduled immunization services

Collect data from patients or caregivers, e.g.:

- Patients report their symptoms prior to treatment or during treatment (adverse events)
- Patients give feedback on service delivery
- Caregivers report what service was delivered to a patient and on what date

Alert caregivers of the status of their patients, e.g.:

- Notify Community Health Worker when patient has not taken ART, DOTS or other drugs

- Notify nurse when patient has not kept a scheduled appointment (e.g., ANC visit)

Facilitate communication between patients, caregivers, and/or health administrators, e.g.:

- Establish secure peer networks for patients who share similar health concerns
- Initiate conversations between patients and caregivers in a way that allows the caregiver to manage the workload effectively

CHAPTER 2

Getting Started Implementers

We recognize that the MOTECH Platform Server requires deep programmatic understanding to implement. The MOTECH Platform Server was built to be generic, allowing for complex customization through the user interface. This section is intended for you, as an implementer of MOTECH, to help get started with installation and begin creating custom solutions to meet your needs. After the installation, you will have to configure the implementation and begin building applications within the platform. The topics below cover some of the common features that need to be configured for a MOTECH app.

As the platform matures, most of the features below will be usable by implementers without developing any software code. As of now, a number of these features do require some coding in order to use (the topics below provide sample code where appropriate). Feel free to reach out to our *MOTECH-dev group <https://groups.google.com/forum/#!forum/motech-dev>* if you get stuck or can't decide the best path forward.

2.1 Installing MOTECH

This installation guide is used for demo systems based on Ubuntu 14.04 LTS. It assumes you are running on the same and is not suitable for a full production install which requires additional steps to ensure security of the system. For example, it's not advisable to use the root username in MySQL to setup the databases. MOTECH will run on other operating systems but the commands required and external packages will change. The changes shouldn't be drastic but they aren't documented here.

2.1.1 Install and Configure Dependencies

MOTECH depends on the following software:

Tomcat7
Java7 (OpenJDK or OracleJDK) *OpenJDK installs as a dependency of Tomcat7*
ActiveMQ
MySQL

A full install script is available here.

1. Install Tomcat7, ActiveMQ and MySQL

```
sudo apt-get install -y tomcat7
sudo apt-get install -y activemq mysql-server
```

> **Note:** MySQL is going to ask you for a root password. You have to input it twice. Remember it because we'll use this later.

2. Change the ownership of the tomcat7 directories

```
sudo chown -R tomcat7:tomcat7 /var/lib/tomcat7/ /usr/share/tomcat7/
sudo service tomcat7 restart
```

3. Configure ActiveMQ

   ActiveMQ needs an enabled instance to run. Use the following command to create a symbolic link from instances-available to instances-enabled.

```
sudo ln -s /etc/activemq/instances-available/main /etc/activemq/instances-enabled/main
sudo sed -e 's/<broker /<broker schedulerSupport="true" /' -i /etc/activemq/instances-en

#Then start ActiveMQ
sudo service activemq restart
```

2.1.2 Deploy MOTECH .war file

We have to deploy the latest MOTECH release from our Nexus repository. The following code uses curl to download motech-platform-server v 0.25.1 to the tomcat webapps folder so it will automatically deploy.

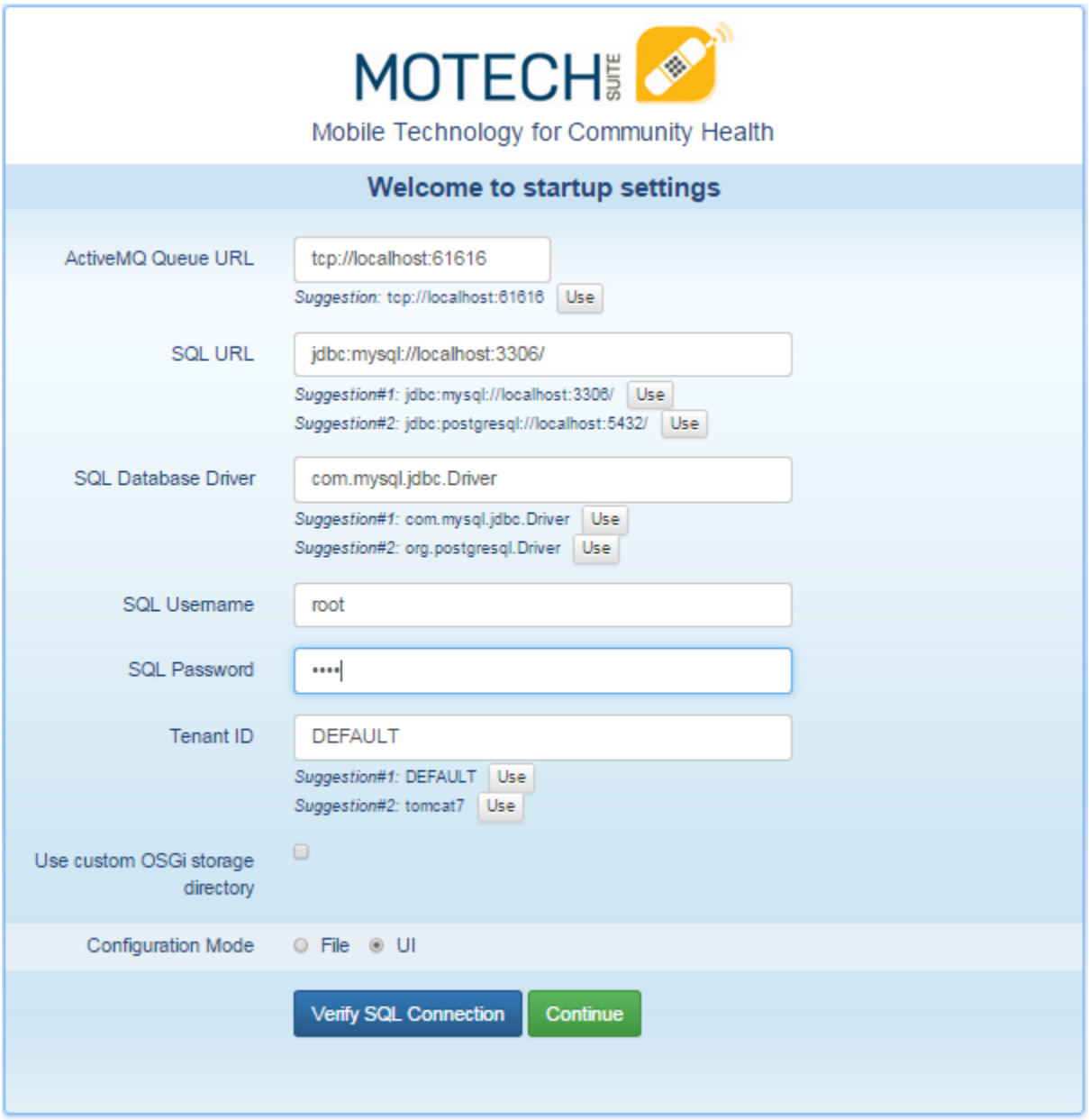
> This tutorial was written for v.0.26 which hasn't been released. Please check the Nexus Server for more recent releases.

```
sudo curl -L http://nexus.motechproject.org/service/local/artifact/maven/redirect?r=releases&g=org.mo
```

Navigate to http://localhost:8080/motech-platform-server

2.1.3 Complete the Bootstrap Form

You will be redirected to the bootstrap form the first time. Complete the form by clicking the 'use' button under each field. The MySQL username is 'root' and password is what you entered during the MySQL installation.



Once complete, test the MySQL connection by clicking the 'Verify SQL Connection' button. Then, click 'Continue'

2.1.4 Complete Startup Settings

The MOTECH startup settings screen asks you to choose a language and select a login mode. Choose 'Repository' to create a new admin username and password.

You will be redirected to the MOTECH login screen where you enter the admin username and password you just created and your installation is complete.

2.2 Configuration System

This document describes the MOTECH configuration system.

2.2.1 Step 1: Specify Config Locations

Default Behavior

By default, all configuration files are loaded from one of the following locations:

```
....${user.home}/.motech/config/
```

For example, if Motech runs under the user motech in Linux, configuration files will be searched in **"/home/motech/.motech/config"** folder.

Or:

```
..../etc/motech/
```

If the configuration files are not found in the above mentioned location, files will be searched in **"/etc/motech"** folder.

Overriding Default Behavior

If you want to override the location where configuration files are loaded from, you have to add config-locations.properties to **${user.home}/.motech** or to **${user.home}/** directory. Otherwise the property file in classpath

will be picked up. Default config-locations.properties file:

```
....config.location= ${sys:user.home}/.motech/config/,/etc/motech/.
```

where **"${sys:user.home}"** is used to specify the home directory.

Note that, multiple locations can be specified for config.location property, and motech-settings.properties (which contains platform core config) config file will be searched starting from the first location and then falling back to next specified location, if it is not found. The directory in which it is found, is considered as the current config location and all other config files will be looked only in that particular location. For e.g., in the above sample config, if you have *motech-settings.properties* in **/etc/motech/**, then all config files will be searched in **/etc/motech/** location only. You cannot have files in different locations.

2.2.2 Step 2: Bootstrap Configuration

There are certain properties which are essential for the system to start up. These properties can be defined either in bootstrap.properties file or by environment variables. The properties are:

- db.url – The database connection url, e.g.: localhost:5984
- db.username – The user who has access to the database.
- db.password – Password to connect to the database.
- tenant.id – Optional. Default value is default.
- config.source – Optional. The source from which MOTECH core configuration and all module configurations should be read. Valid config values can be either one of FILE or UI. Default value is UI.

During startup, the system will look for these configurations in the following locations, falling back in order (that is when the properties are found in any of the following locations, search look up will stop):

1. Config Directory Environment Variable: If MOTECH_CONFIG_DIR environment variable is defined, then the system will load properties from ${MOTECH_CONFIG_DIR}/bootstrap.properties.
2. **Config Environment Variables: Config is loaded from one or more environment variables:**
    - MOTECH_DB_URL – specifies value for db.url
    - MOTECH_DB_USERNAME - specifies value for db.username
    - MOTECH_DB_PASSWORD – specifies value for db.password
    - MOTECH_TENANT_ID – specifies value for tenant.id
    - MOTECH_CONFIG_SOURCE – specifies value for config.source
3. Location from Property file: bootstrap.properties file is loaded from any one of the locations specified in config-locations.properties file described above.

We are working on a feature in which, if bootstrap.properties is not found in any of the above mentioned locations, a UI will be presented to user after startup, prompting for bootstrap properties.

2.2.3 Step 3: MOTECH Core Config

There are some system configurations and activemq configurations which are needed to get MOTECH up and running.

- **System configurations:**
    - system.language - Can take en(English), pl(Polski), es(Spanish), fr(French), it(Italian), sw(Swahili) as values(although only English and Polski are implemented as of now). Optional, default value is en.

  - statusmsg.timeout - Represents the expiration time(in seconds) of messages and notifications in admin UI. Optional, default value is 60.
  - login.mode - Can be repository or openId (case insensitive).
  - provider.name - OpenId? provider name, mandatory in case login mode is openId.
  - provider.url - OpenId? provider url, mandatory in case login mode is openId.
- **Security configurations(For more details you should read the *security configuration section*):**
  - security.required.email - Indicates whether you must provide an email address when creating the user.
  - security.failure.login.limit - The permissible number of incorrect login attempts, default value is 0. After this limit is reached the user is blocked. After a successful login counter is reset. If the value is 0 then blocking is inactive.
  - security.session.timeout - The session timeout in seconds, default 30 minutes. After this time session will be closed.
  - security.password.minlength - The minimum length of the password, default 0. if the value is 0 then length checking is disabled.
  - security.password.validator - The password validator, it specify password rules e.g. 1 number, 1 special character. Can take none, lower_upper(at least 1 uppercase and lowercase), lower_upper_digit(at least 1 uppercase, lowercase and digit), lower_upper_digit_special(at least 1 uppercase, lowercase, digit and special character) as values, default none validator is used.
- **Activemq configurations:**
  - jms.queue.for.events - Queue name to hold motech event messages. Optional, default value is QueueForEvents.
  - jms.topic.for.events - Topic name to hold motech event messages. Optional, default value is TopicForEvents.
  - jms.broker.url - JMS broker URL. Can take failover URLs also. Sample values: tcp://localhost:61616, failover:(tcp://192.168.32.1:61616,tcp://192.168.32.2:61616)?randomize=false
  - jms.maximumRedeliveries - Maximum number of redeliveries in case of any exceptions and a message consumption fails. Optional, default value is 0.
  - jms.redeliveryDelayInMillis - Delay(in seconds) between successive re-deliveries of messages. If delay=d and first exception was raised at time=t, then successive redelivery times are calculated using exponential backoff . i.e. t+d, t+(d*2), t+(d*4), t+(d*8), t+(d*16) and so on, till maximum redelivery count is reached. Optional, default value is 2000.
  - jms.concurrentConsumers - Optional, default value is 1.
  - jms.maxConcurrentConsumers - Optional, default value is 10.
  - jms.session.cache.size - Optional, default value is 10.
  - jms.cache.producers - Optional, default value is false.

Case 1: When ConfigSource is FILE

Define motech-settings.properties file in any one of the locations defined in config-locations.properties with the above mentioned properties.

Case 2: When ConfigSource is UI

After server startup, if core settings are not configured already, you will be presented with a startup page which asks for System Language, Queue URL for events, Login Mode and user setup based on login mode. Other activemq settings can be changed in Settings tab after logging in.

2.2.4 Step 4: Module Configurations

Case 1: When ConfigSource is FILE

Module specific property files can be added to:

```
....<config-location-dir>/<module-symbolic-name>/ directory
```

and any JSON templates/configurations to:

```
....<config-location-dir>/<module-symbolic-name>/raw/ directory.
```

A typical example of a motech's module symbolic name:

```
....<module-name>
```

prefixed with "org.motechproject.motech-".

All these files are monitored for changes. So, any change to these config files at runtime would be detected and saved in DB. Restart the module if required using Manage Modules tab in UI. We are enhancing the config monitor to raise an event in case of config change. This event can be listened by interested modules and take appropriate actions.

Case 2: When ConfigSource is UI

After server startup, you can find each module having settings UI associated with it in the Manage Modules tab, where you can edit the properties for the module. Also, restart the module if required. We are enhancing the config monitor to raise an event in case of config change.

2.3 Modeling Data with MOTECH Data Services

Table of Contents

2.3.1 Introduction

MOTECH Data Services (MDS) is the data layer for the MOTECH Platform. MDS allows defining the data model both through code (using annotations or the exposed API) and the Schema Editor UI. It is capable of exposing generic services which allow executing CRUD operations on the defined model. It also is capable of exposing a fully functional REST API for the defined entities on the fly. Entities defined through means of code can always be extended or get their settings modified through the MDS Schema Editor or its underlying API.

The benefits of MDS include:

- Generated user interface for data entry
- UI-based schema editor, and the ability to enhance developer-defined entities
- Generated OSGi services and Java APIs for accessing data objects
- Generated REST APIs for external data access
- Generated CRUD events for MDS entities (and exposure of these events via the Tasks module)
- Ability to register actions that execute on instances based on CRUD triggers
- Bulk import/export of data
- Change tracking (auditing) of data
- Object-level security

MDS uses DataNucleus underneath for persistence in a relational data store. Currently MDS officially supports two RDBMS engines MySQL and PostgreSQL. Javassist is used for code generation and OSGi mechanics such as bytecode weaving are used for replacing the code at runtime.

MDS generated entities bundle

All classes generated by MDS live in the mds-entities OSGi bundle, which gets generated at runtime and installed in the directory ~/.motech/bundles. The bundle is always regenerated when changes are made to the MDS schema. This generated bundle can also be downloaded using the following url: *http://<motech_url>/modulemds/jar*.

2.3.2 MDS Entities

MDS defines an Entity concept. An MDS entity maps directly to a POJO class and a table in the relational database. Entities consist of fields which are directly mapped to the object fields and columns in the database table. MDS supports multiple *field types*.

MDS integrates itself with the `Tasks module`, so a user can create a working application with a minimal amount of code. Entities generate task data providers which allow access to the data within MDS. Entities can also be configured to publish MOTECH events which are fired after CRUD operations are completed in MDS. These CRUD events, are exposed as task triggers in a dynamically generated task channel. CRUD actions are also exposed as actions within the task module, allowing users to create database manipulating logic through the tasks module.

We can group entities into three categories:

EUDE - End User Defined Entities. The entities created using the UI by the end user. These classes do not exist at compile time, but only after they are generated by MDS. Adding the bundle generated by MDS to the classpath will allow compile time access however. EUDE entities can also be defined using the MDS API through the **EntityService**. Users can view and create instances of the entities through the MDS Data Browser.

DDE - Developer Defined Entities. Developers can use annotations to mark their POJO classes as MDS Entities. These will be treated in the same way as EUDE entities, instances of the DDEs will also be accessible through the

data browser. Users can still view the schema for these entities through the Schema Editor, add fields and modify settings(although they can't remove fields declared by the developer in the java class).

MEDE - MDS Enhanced Developer Defined Entity. These are DDEs that were enhanced with additional fields added either through the UI or the Entity API. This are the same as DDE, but with additional fields added at runtime. Those fields can be accessed at compile time using Java Reflection API.

Automatically added fields

All entities in MDS will be enhanced with the following fields automatically:

Name	Type	Description
id	Long	The id field of the entity, used to uniquely identify the instance.
owner	String	The username of the owner of the instance. This field can be used with security settings for the entity in order to filter access to only instance owners.
creator	String	The username of the creator of the instance. Automatically set to username of the MOTECH user that created the instance. Note that security can be set up to limit instance access to only creators of those instances.
modifiedBy	String	The username of the user that last modifier of the instance. Automatically set to the username of the user that last edited the entity.
creationDate	DateTime	The datetime on which this entity was created. Filled automatically.
modificationDate	DateTime	The datetime on which this entity was last modified. Updated automatically.

Access to these fields can be done through reflections, through re-declaring them in the DDE class or by inheriting the **MDSEntity** class.

2.3.3 MDS Lookups

Lookups allow easily defining and executing queries on MDS entities. A lookup allows querying for a single or multiple fields. A lookup field is always corresponding to a single field in the entity. It can be also configured to either return a single or multiple results.

Note: If more then one instance matches the criteria of a single return lookup, the lookup will fail.

Lookups at this moment can only use AND logic for doing a query for multiple fields. For OR(or move complex) logic *JDO queries* have to be used. Lookups also allow comparing fields against provided parameters using a custom operator or using a range or set of values, defining such lookups is not supported through the UI at the moment though.

For each lookup two additional versions of the method will be generated. The first one is the same as the lookup, but with an additional parameter at the end - org.motechproject.mds.query.QueryParams. This class contains pagination directives - page number and page size, it also contains information about ordering the results - an org.motechproject.mds.util.Order object containing the sort direction and sort column. This version of the lookup is useful for operating on large data sets and providing ordered views to the user. The third version is the same as the basic lookup, but it returns a number (long) - the total count of the entity in the database. The name of the count method consists of *count* and the capitalized original lookup method name. For example for a lookup with a method name *byName* the count method will be called *countByName*.

Note: When defining a DDE, it doesn't matter which version of the lookup you define, all three methods will be generated. For compile access to them however, they have to be explicitly defined in your service. More info on defining lookups in DDEs can be found in the section about defining *DDE Data Services*

2.3.4 Data Services

All access to entities in MDS is done through Data Services. These are services implementing the **org.motechproject.mds.service.MotechDataService** interface. They are exposed as OSGi service that can be retrieved from the OSGi BundleContext. All data access exposed by MDS, either the REST API, the UI data browser, Csv Import/Export etc. is done through these services. The class of the service is generated at runtime and it extends the base **DefaultMotechDataService** class. *Developers can extend the **MotechDataService** interface* in order to add their own lookups to the interface simply by declaring the method signatures and annotating them properly.

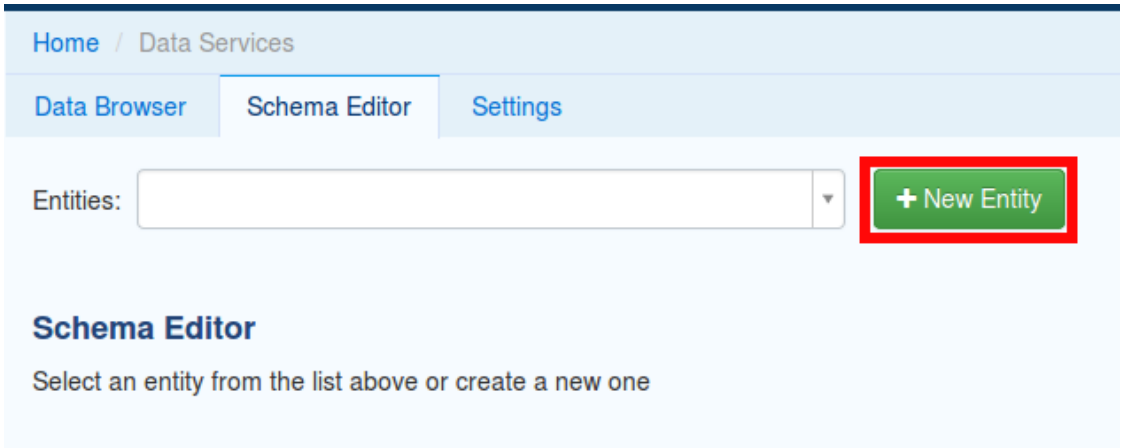
2.3.5 EUDE - End User Defined Entities

These entities are created by end users, either through the UI or using the exposed API. No programming knowledge is required in order to define an EUDE using the first method. Although these entities are not known at compile time(unless the jar generated by MDS is added to the classpath) programmatic access to these entities is still possible using Java Reflection API and some handy helper classes exposed by MDS - mainly the *MdsLookupService*.

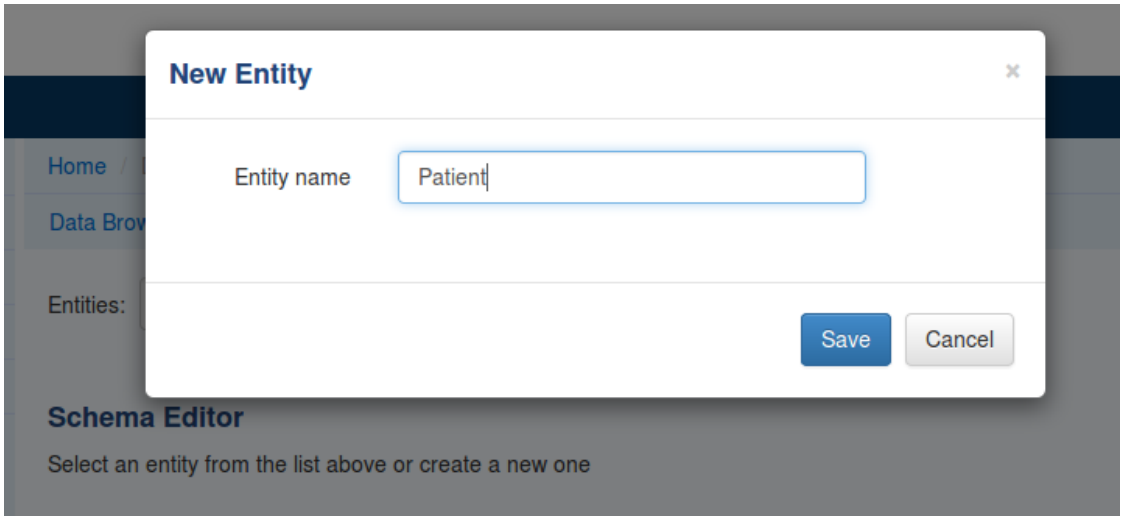
Note: All EUDE classes share the same java package: **org.motechproject.mds.entity**

Creating EUDE through UI

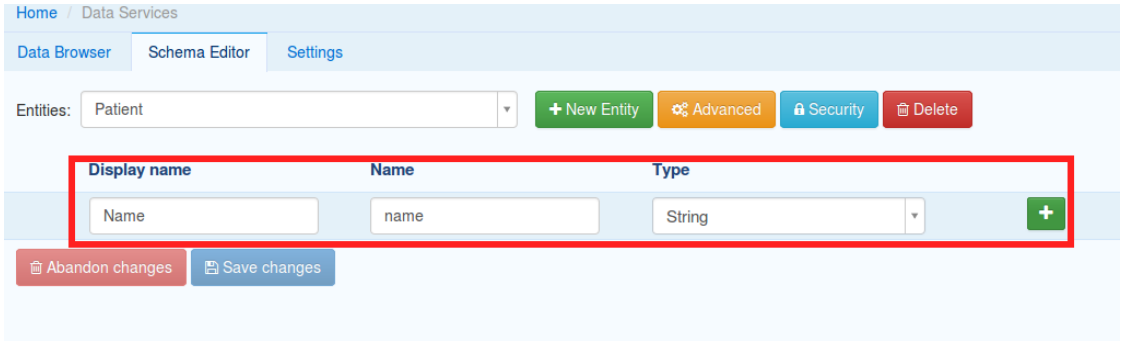
The easiest way to create EUDE entities is to use the MOTECH UI. First select **Data Services** from the left navigation menu(**Modules** menu), then navigate to the **Schema Editor** tab. You will see a dropdown allowing to select an existing entity for modification or deletion. Next to the dropdown menu you will see a New Entity button.



After that the user is asked for the name of the entity. This can be anything that is a legal name of a class in Java.

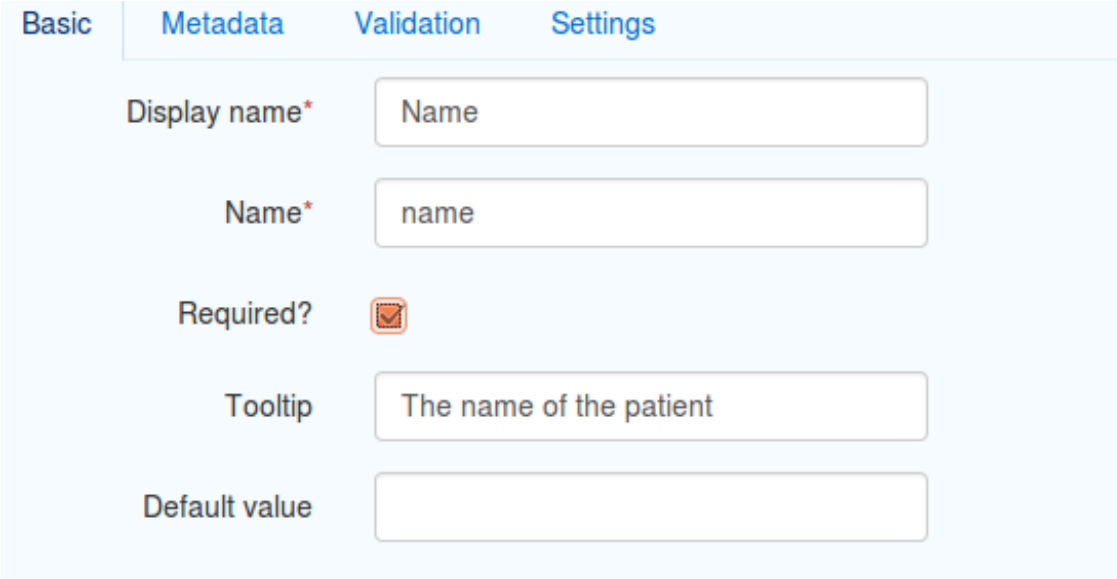


The view for managing entity fields is then displayed to the user. Users can add a field by selecting its type, choosing a name and a display name. 'display name' represents what will be visualised to the users in the MDS Data Browser, task editor etc. 'name' represents the actual name of the field that will be used for class and table creation. After this data is entered, hitting the green plus sign will add the field.

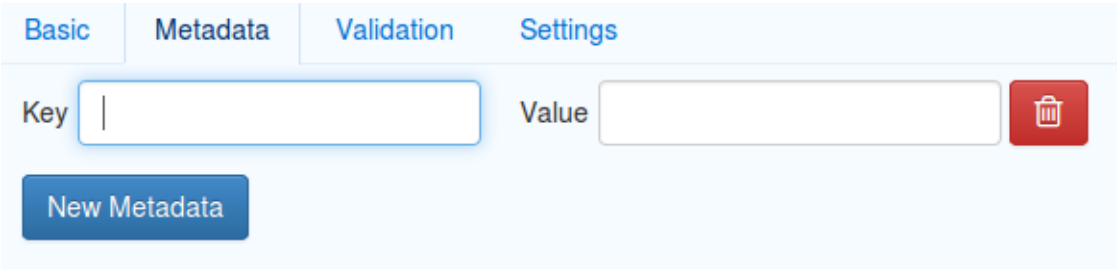


The field is then expanded and the user is presented with options to modify the field settings:

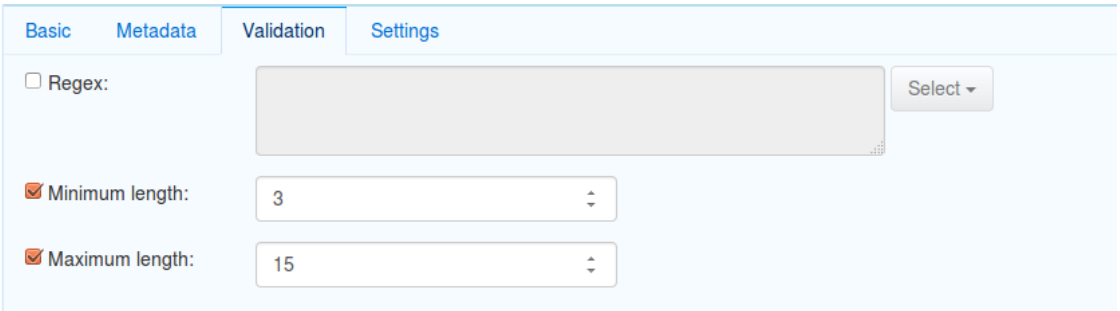
The **Basic** sections allows to change the previously entered name and display name, it also allows marking the field as required, meaning that users will be prevented from creating an instance without any value in this field. A default value for the field can also be entered, as well as a tooltip that will be shown to users creating instances of the entity.



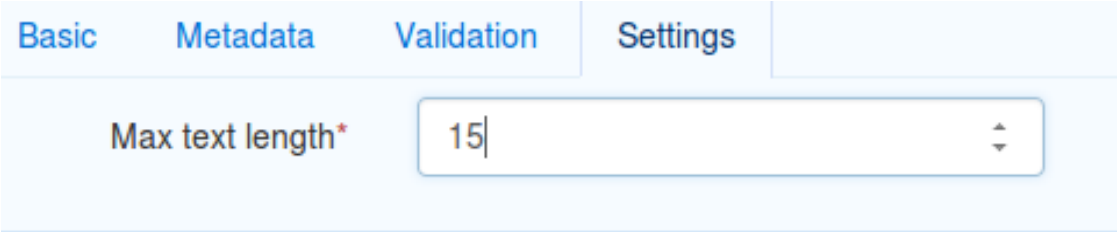
The **Metadata** section allows adding metadata to the field. This used internally by MDS for features such as relationships. End users should not worry about this section, but advanced users can add any values they wish for their own processing needs. Metadata is retrieved with the field schema using the Entity API. An example of using metadata could be a scenario when we are writing a third party export tool, that takes the MDS Schema and imports it into a 3rd party system. The field metadata can be used by that tool in order to recognize some fields as requiring special processing logic.



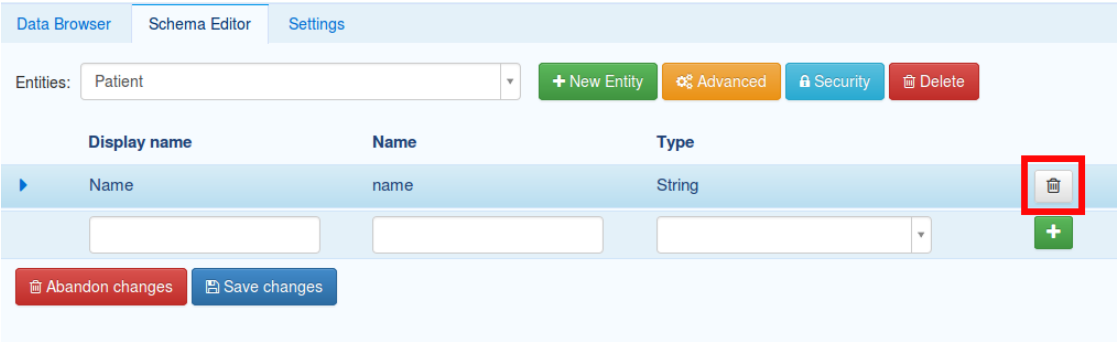
The **Validation** section allows setting specific validation rules for the field. Users will then be constrained by these validations when creating instances of the entity. Validations are type specific.



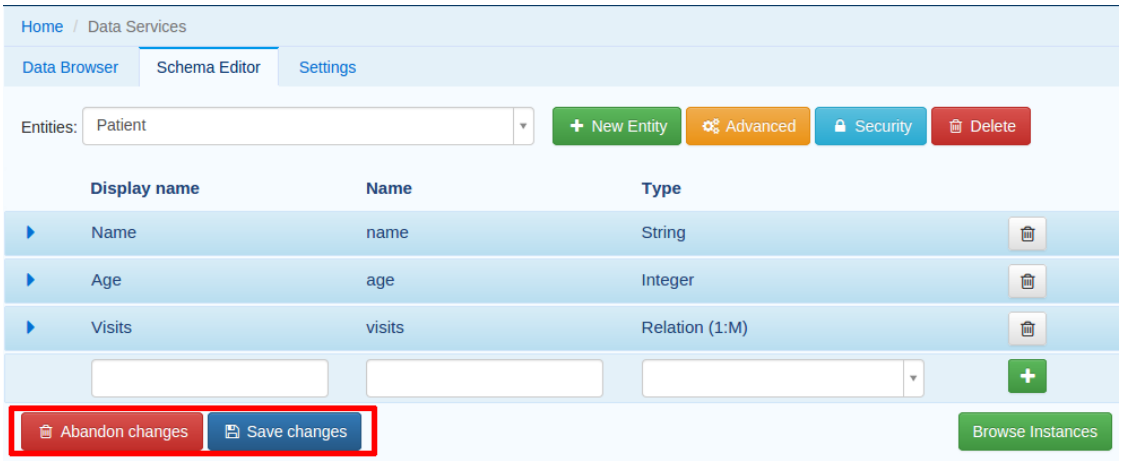
The **Settings** tab allows users to set type specific settings of the field. An example setting is the 'Max text length' of a String field, which indicates the maximum length of the string at the database level.



Existing fields can be deleted using the trash bin icon next to their type.

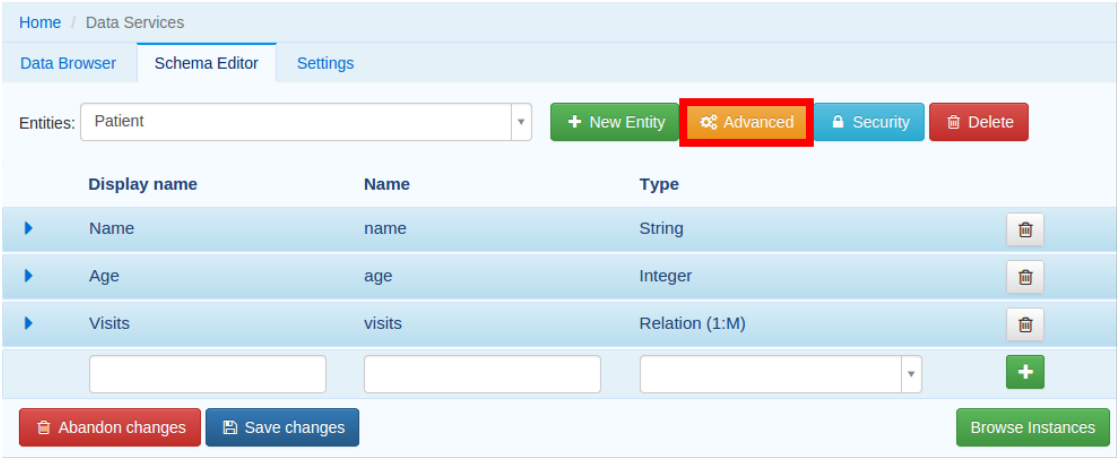


When the user is done modifying the entity, clicking **Save changes** will save the changes to schema and regenerate MDS entities. Clicking **Abandon Changes** will abandon all changes made by the user since the last save.

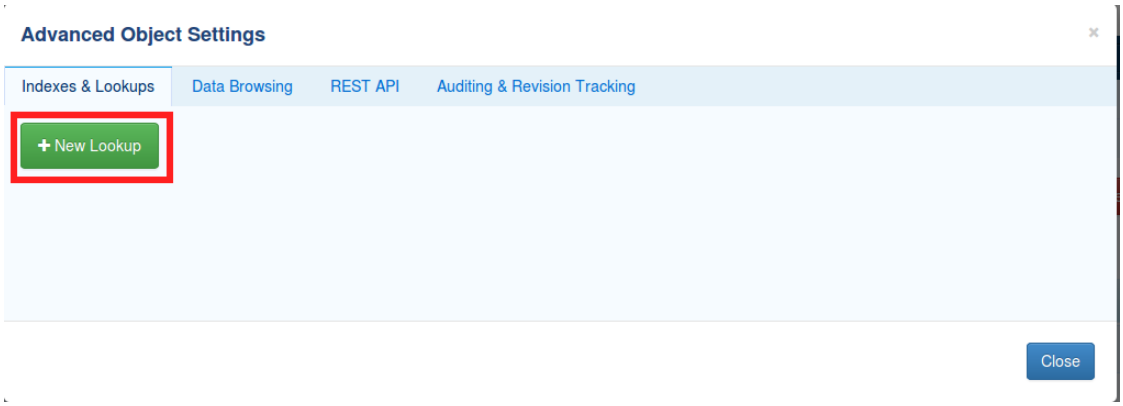


Defining a Lookup through the UI

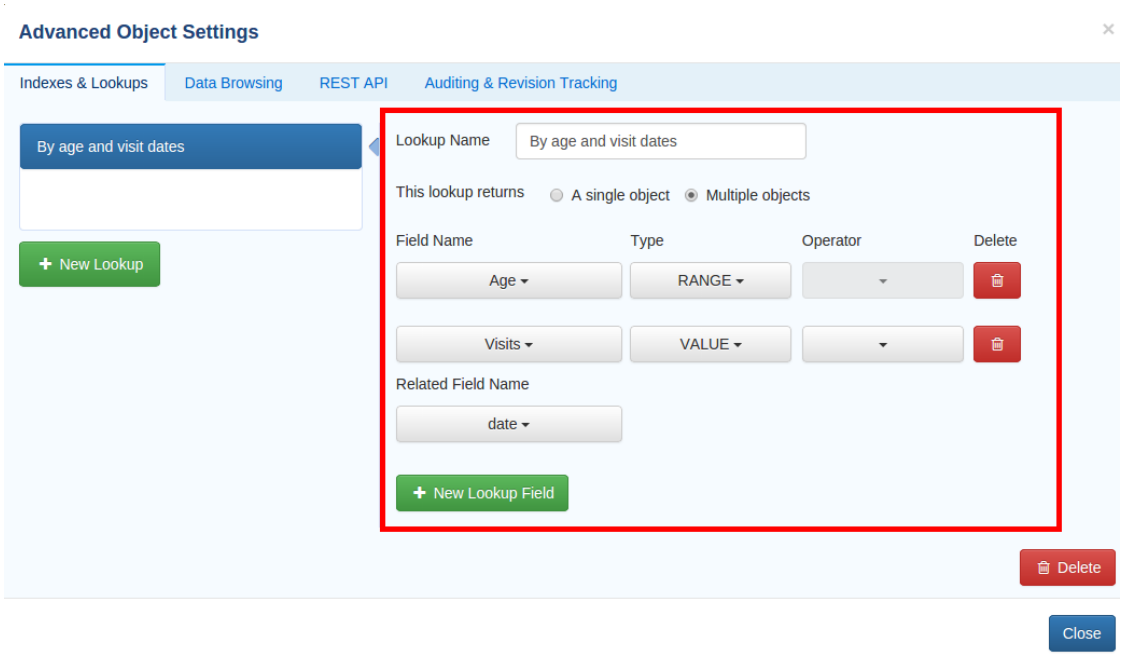
Users can use the UI for adding lookups to an entity. These lookups can then be executed either directly through the data services or using the Data Browser UI. In order to add a new lookup, first open the advanced settings of an entity by clicking the 'Advanced Settings' button.



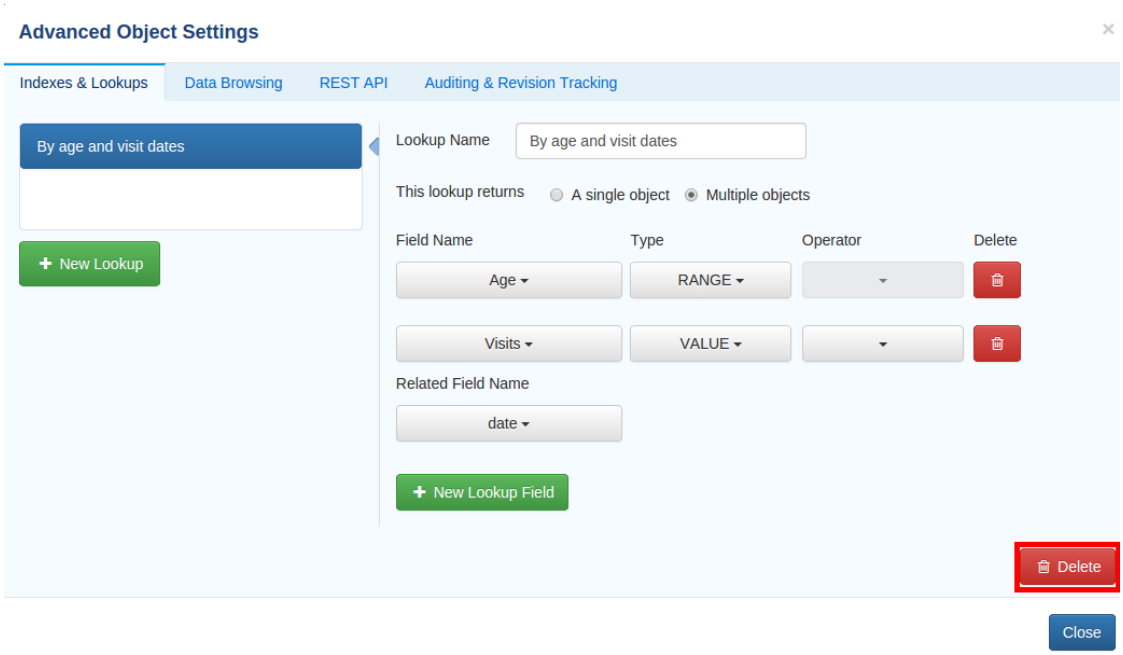
After that users can create lookups by clicking on the 'New Lookup' button.



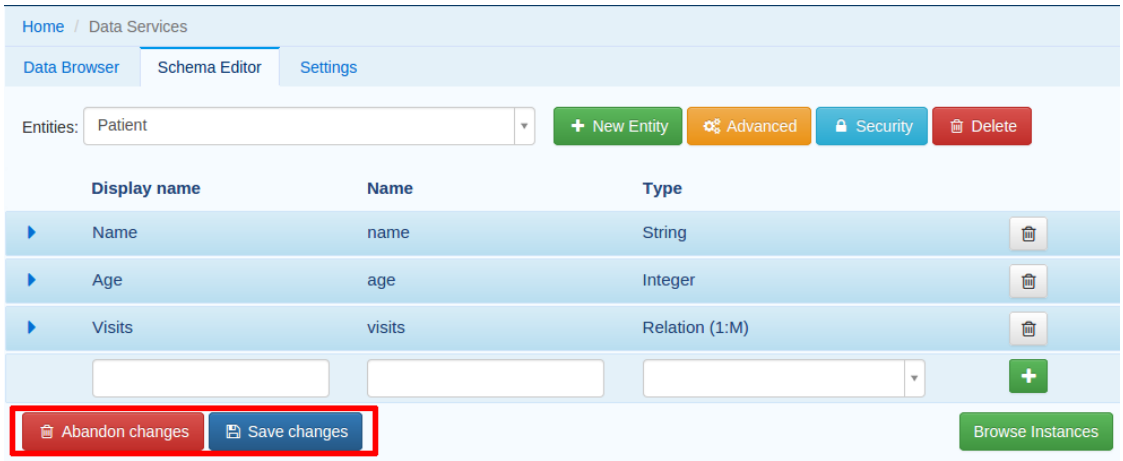
The name fo the lookup can then be modified as well as whether it returns a single or multiple objects. In order to make a lookup useful, it has be executed on a given set of fields, which can be added on the right side of the window by clicking the 'New Lookup Field' button and selecting the right field from the dropdown. They can be deleted using the trash bin button.



In order to remove a lookup, the delete button in the lower right of dialog can be used.



When the user is done adding lookups to an entity, clicking **Save changes** will save the changes and trigger regeneration. Clicking **Abandon Changes** will abandon all changes made by the user since the last save.



Creating EUDE through the Entity API

Creation of entities can be also done using the **org.motechproject.mds.service.EntityService**. This an OSGi service exposed by MDS which allows creation and modification of MDS entities, exposing everything that the UI does. In order to use the service it has to be retrieved from the OSGi context, either directly using the OSGi API or a Blueprint reference can be used to inject a proxy for that service directly as a Spring bean.

Example of retrieving the service manually:

```
import org.motechproject.mds.service.EntityService;
import org.osgi.framework.*;

...

public EntityService getEntityService() {
    // note that if using Spring, the BundleContext can be injected as any other bean
    // which allows skipping this step
    BundleContext bundleContext = FrameworkUtil.getBundle(EntityService.class).getBundleContext();

    // get the service reference from the bundle context
    ServiceReference<EntityService> ref = bundleContext.getServiceReference(EntityService.class);

    // return the service for the reference, or null if there are no references
    // the service should always be available, so a null reference definitely indicates some sort er
    return ref == null ? null : bundleContext.getService(ref);
}
```

and the preferred way using blueprint. Note that thanks to this declaration an EntityService bean becomes available in your Spring context.

```
<?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:osgi="http://www.eclipse.org/gemini/blueprint/schema/blueprint"
    xsi:schemaLocation="http://www.springframework.org/schema/beans
        http://www.springframework.org/schema/beans/spring-beans.xsd
        http://www.eclipse.org/gemini/blueprint/schema/blueprint
        http://www.eclipse.org/gemini/blueprint/schema/blueprint/gemini-blueprint.xsd">

    <osgi:reference id="entityService" interface="org.motechproject.mds.service.EntityService"/>
```

```
</beans>
```

After getting hold of the service the entity can be created using the createEntity method:

```
EntityService entityService = getEntityService();

EntityDto entity = new EntityDto("Patient");

// the EntityDto instance returned will have the id value set
entity = entityService.createEntity(entity);
```

If we want to edit an existing entity, we can retrieve it using the EntityService:

```
// We can use the org.motechproject.mds.util.ClassName utility in order
// to get the EUDE class name given just the name
String className = ClassName.getEntityName("Patient");

// className is org.motechproject.mds.entity.Patient
EntityDto entity = entityService.getEntityByClassName(className);
```

When we have the EntityDto instance, fields can get added to the entity using the service and EntityDto returned:

```
// a simple integer field
FieldDto simpleField = new FieldDto("simpleInt", "Simple integer", TypeDto.INTEGER);

// a required name field
FieldDto nameField = new FieldDto("name", "Patient Name", TypeDto.STRING, true);

// an optional date of birth field, with a tooltip
FieldDto dobField = new FieldDto("dob", "Date of Birth", TypeDto.DATETIME, false, null,
        "Patients date of birth, leave blank if unknown");

// a required Social ID field, defaulting to 0
FieldDto socialIdField = new FieldDto("socialId", "Social ID", TypeDto.LONG, true, 0L);

// add the fields to the entity created earlier
entityService.addFields(entity, simpleField, nameField, dobField, socialIdField);
```

In order to make these changes take effect, *data bundle regeneration must be triggered*.

Creating Lookups through the API

Just as any other edits on the entity schema, lookups can also be created using the EntityService. In a similar fashion to fields, the **addLookups** method can be used for adding lookups to an entity. Given that we have the EntityDto object and the EntityService(), we can create lookups in the following manner:

```
// this lookup will check the name field, during an exact comparison
LookupDto lookupByName = new LookupDto("By name",
        true, // single object return
        true, // expose this lookup through REST
        Arrays.asList(new LookupFieldDto("name", LookupFieldDto.Type.VALUE)
));

// this a complex lookup using multiple fields
LookupDto complexLookup = new LookupDto("Complex lookup",
        false,  // return multiple objects
        false,  // do not expose by REST
    Arrays.asList(
```

```
        // the custom operator matches() will be used for querying on the name field
        new LookupFieldDto("name", LookupFieldDto.Type.VALUE, Constants.Operators.MATCHES),
        // the dob parameter will take a range, with a min and max value
        new LookupFieldDto("dob", LookupFieldDto.Type.RANGE),
        // for the state field, a set of possible values can be supplied
        new LookupFieldDto("state", LookupFieldDto.Type.SET),
        // the search through relationship fields is possible using the dot operator
        new LookupFieldDto("relationshipField.number", LookupFieldDto.Type.VALUE))
);

// add the lookup
entityService.addLookups(entity, lookupByName, complexLookup);
```

In order to make these changes take effect, *data bundle regeneration must be triggered*.

Regenerating the entities bundle

After we are done with modifications to the entity schema, we must trigger regeneration in order for the classes to get updated and made available in OSGi. For this we need to use **org.motechproject.mds.service.JarGeneratorService**, which we can retrieve the same way that we can retrieve the EntityService. Once we have an instance of the service, all we need to do is call the regenerateMdsDataBundle method:

```
JarGeneratorService jarGeneratorService = getJarGeneratorService();

jarGeneratorService.regenerateMdsDataBundle();
```

After the schema gets regenerated and all bundles using MDS get refreshed, the EUDE class should be available for use.

Programmatic access to EUDE entities

EUDE classes can be accessed using java reflections. This is an example of creating an instance using reflections:

```
// first get the interface class name of the name entity
// this helper method will always return org.motechproject.mds.entity.Patient
String interfaceName = ClassName.getInterfaceName("Patient")

// Retrieve the Data Service
MotechDataService service = ServiceUtil.getServiceForInterfaceName(bundleContext, interfaceName);

// Get the Class object for the entity
Class entityClass = service.getClassType();

// create a patient instance and set the name to "John"
Object instance = entityClass.newInstance();
PropertyUtil.setProperty(instance, "name", "John");

// save it using the service
service.create(instance);
```

As you can see the access is done through the Data Service. We can obtain the Class object for the generated class and use it for doing all required operations using reflections.

2.3.6 DDE - Developer Defined Entities

Developers can use annotated POJO classes in order to define the model for their application. Entities defined in this way will be treated in a similar fashion to EUDE entities, they can also be accessed using the MDS Data Browser. New fields can also be added to DDEs - so that they become MEDE.

DDEs are represented by actual Java classes used for defining them. OSGi bytecode weaving is used in order to enhance these classes at runtime and add additional fields for them. Because of this, these classes can be used with ease in code, since they are available during compile time to developers.

Defining entities - the @Entity annotation

In order to define a DDE by using the **org.motechproject.mds.annotations.Entity** annotation. This are the contents of Patient.java, an example fo a DDE entity:

```
package org.motechproject.example;

import org.motechproject.mds.annotations.*;

@Entity
public class Patient {

}
```

When the module containing this entity gets installed MDS will scan it for classes annotated with **@Entity**, and the class above would get picked up for processing. Schema for the entity is then generated and persisted in the database of MDS, the class is also enhanced by DataNucleus. The MDS weaving hook then replaces the bytecode for this class in module ClassLoaders with the DataNucleus/MDS enhanced version, making it available to the modules using it.

Note: The module must export the package of the entity in OSGi, using the Export-Package directive in its manifest.

The **@Entity** annotation has the following parameters:

- `name` - The name of the entity displayed to the user. Defaults to the simple name of the annotated class.
- `module` - The name of the module for this entity. Defaults to the module name of the bundle from which this entity comes from.
- `namespace` - The namespace in which the entity is defined. Optional, defaults to empty.
- `tableName` - The actual name of the table in the database for this entity. Allows users to directly control the name in the data store. The default table name will take the form of: `MDS_<MODULE>_<NAMESPACE>_<ENTITY_NAME>`. If an entity has no namespace or module, those parts will be omitted.
- `recordHistory` - Set to true if MDS should record history for this entity.

DDE entity fields - @Field and @Ignore annotations

An entity does not have much use without any fields. MDS will treat any public field or field with public getter/setter in the class as an MDS field. In the class below, the field **name** will be picked up automatically as a field to be persisted in the database:

```
@Entity
public class Patient {

    private String name;
```

```
    public String getName() {
        return name;
    }

    public void setName(String name) {
        this.name = name;
    }
}
```

The **@Field** annotation can be used for more explicit marking and control over the fields basic properties. In the example below, the **required** parameter of the annotations is used to mark the name field as required, moreover the physical column name in the database is set to "P_NAME":

```
@Entity
public class Patient {

    @Field(name = "P_NAME", required = true)
    private String name;

    public String getName() {
        return name;
    }

    public void setName(String name) {
        this.name = name;
    }
}
```

The @Field annotation could also be placed on the setter or getter methods for the same effect.

Not every public field, or not every field that has a public getter or setter has to be persisted in the database. The **@Ignore** annotation can be used for marking such field as not persistent:

```
@Entity
public class Patient {

    @Ignore
    public String name;
}
```

The name field in the example above will not become a database field and no MDS schema will be generated for it. This field will also not be accessible through the data browser.

DDE relationships

Creating relationships between entities is currently only possible for DDE. The definition of a relationship depends on the type of the relation. MDS supports one-to-one, one-to-many, many-to-many and master-detail relationships, both uni-directional and bi-directional. The way to define relationships for DDEs is presented in the examples below.

- **One-to-one** To create a one to one relationship, one of the related entities, should define a field of class, that represents the second entity. Both classes must of course be valid MDS Entities. The code below, provided that Book is an entity, will create a simple, uni-directional, one-to-one relationship between Author and Book.

```
@Entity
public class Author {
    @Field
    private String name;
```

```
    @Field
    private Book book;

    ...
}
```

- **One-to-many** To create a one to many relationship, one of the entities should define a collection of related entity. Just like in one-to-one relationships, both classes must be valid MDS entities to work. The code below shows an example of a simple, uni-directional, one-to-many relationship between Author and Book (one author is related with many books).

```
@Entity
public class Author {
    @Field
    private String name;

    @Field
    private Set<Book> book;

    ...
}
```

- **Bi-directional relationships** The bi-directional relationship is a model, in which both sides of a relation are aware of the existence of a relationship and can both refer to the other side of a relation.

  To make the relationship bi-directional, two additional steps must be taken:

  - The second entity must also define a relationship to the other entity
  - Exactly one MDS field of a bi-directional relationship must be annotated with the @javax.jdo.annotations.Persistent(mappedBy = "fieldName") annotation. The fieldName should correspond to the field name that is in a relationship, in the another entity.

  Please see the code below, for an example of a one-to-many, bi-directional relationship.

```
@Entity
public class Author {
    @Field
    private String name;

    @Field
    @Persistent(mappedBy = "author")
    private Set<Book> book;

    ...
}

@Entity
public class Book {
    @Field
    private String title;

    @Field
    private Author author;

    ...
}
```

- **Many-to-many** In this type of a relationship, both classes define a collection of related entity instances. The

many to many relationships are bi-directional by definition, which means it's not possible to create a uni-directional version of such relation. The code below shows an example of a many-to-many relationship.

```
@Entity
public class Author {
    @Field
    private String name;

    @Field
    @Persistent(mappedBy = "author")
    private Set<Book> book;

    ...
}

@Entity
public class Book {
    @Field
    private String title;

    @Field
    private Set<Author> author;

    ...
}
```

- **Master-detail** MDS also supports master-detail model, where entity can inherit some fields from another entity. This is achieved by simple class inheritance, using Java keyword **extends**. Naturally, both classes must be valid MDS entities for this to work. The code below shows an example of such master-detail model.

```
@Entity
public abstract class Config {
    @Field
    private String name;

    @Field
    private Map<String, String> properties;

    ...
}

@Entity
public class ModuleConfig extends Config {
    @Field
    private String moduleName;

    @Field
    private String moduleVersion;

    ...
}
```

- **Eager/lazy loading** By default loading an entity with relationship will load its related entities, but that behaviour can be configured through @Persistent(defaultFetchGroup = "true/false") annotation. Please see the code below for an example.

```
@Entity
public class Author {
    @Field
```

```
    private String name;

    @Field
    @Persistent(defaultFetchGroup = "false")
    private Set<Book> book;

    ...
}
```

Using DataNucleus annotations

DataNucleus JDO annotations can be used for enhancing DDEs. These annotations will be taken into consideration by DataNucleus and override the metadata that MDS generates. For example the **@javax.jdo.Unique** annotation can be used in order to mark fields in an entity as unique. Refer to the DataNucleus documentation for more information on using those annotations.

DDE service interfaces

DDEs can define their own interfaces that extend the default service interface that will be used for generating MDS services. The service will be published under that interface, and thanks to inheritance, it will also expose type safe methods from the base service. Here is an example of defining an interface for a 'Patient' DDE:

```
public interface PatientDataService extends MotechDataService<Patient> {

}
```

Thanks to this declaration type safe access to methods of the interface will be gained, the generic parameter Patient will be inserted for the returned/parameter values.

This way of defining services for DDEs also allows to define additional lookups on the service. These lookups are defined as plain method declarations with annotations and their implementation will be generated at runtime by MDS. The lookup method must be annotated with a **@Lookup** annotation. Method parameters should be marked with @LookupField annotation in order to connect the parameter with the actual entity field.

Note: If the @LookupField annotation is not present, MDS will fall back to an attempt to recognize the method parameter name, take note that this requires debug information at runtime, so you have to compile your classes appropriately.

```
public interface PatientDataService extends MotechDataService<Patient> {

    /*
     * This lookup finds a single patient based on the field 'name'.
     * So invoking this method like this: byName("John") will
     * return the patient with the name "John".
     */
    @Lookup
    Patient byName(@LookupField(name = "name") String name);

    /*
     * The count method. Note that if this method is not defined,
       it will be generated automatically from the lookup above.
     */
    long countByName(String name);

     /*
```

```
     * Same as above, but returns multiple results.
     */
    @Lookup
    List<Patient> byName2(@LookupField(name = "name") String name);

    /*
     * Same as above, but with QueryParams. Note that if this method is not defined,
       it will be generated automatically from the lookup above.
     */
    @Lookup
    List<Patient> byName2(@LookupField(name = "name") String name, QueryParams queryParams);
}
```

The type of the parameter must match the type of the field, unless its one of the two special types:

Range - ranges can be used for looking up values that fall within the given range. An example is a range of dates. Range consist of min and max values, it is possible to provide only one of these values so there will be no boundary on the second end.

```
public interface PatientDataService extends MotechDataService<Patient> {

    /*
     * Looks up patients for which the date of birth falls in the supplied range of
     * values. Example of usage:

        byDateOfBirth(new Range<>(DateTime.now().minusYears(30), DateTime.now().minusYears(10)));

     * this returns patients born between 30 and 10 years ago.
     */
    @Lookup
    List<Patient> byDateOfBirth(@LookupField(name = "dob") Range<DateTime> dobRange);

}
```

Set - Doing lookups by sets is also possible. Instead of providing a single value, you provide a set of values. If an instance field matches one of the values, that is considered a hit(basically this is logical OR matching).

```
public interface PatientDataService extends MotechDataService<Patient> {

    /*
     * Looks up patients which name matches one of the values from the set.
     * Usage example:
     *
     *  byName(new HashSet<>(Arrays.asList("Tom", "John", "Bob")));
     *
     * This will return patients named Tom, John or Bob.
     */
    @Lookup
    List<Patient> byName(@LookupField(name = "name") Set<String> names);

}
```

Lookups can also use custom operators. The operator is inserted between the field name and the lookup parameter in the JDO query generated for the lookup. The default symbol is '=' - the equality sign, however different operators can also be used. Both JDO QL operators and methods can be used for lookups. If an operator like "<" is provided as the custom operator, it will be put between field name and parameter value. If the operator has the form a function like "matches()" it will generate a method call of the form "parameter.matches(value)" - the value is inserted between the brackets. In order to provide a custom operator for a lookup field, the customOperator field of the @LookupField annotation has to be set:

```
public interface PatientDataService extends MotechDataService<Patient> {

    /*
     * Does a matches() lookup on the name field.
     * Because matches() is used, a regex pattern can be passed as the parameter.
     */
    @Lookup
    List<Patient> byName(@LookupField(name = "name", customOperator = "matches()") String name);

}
```

Note: The list of standard JDO operators that can be used in lookups is defined as constants in the class **org.motechproject.mds.util.Constants.Operators**.

Defining editable lookups for DDE entities

One way to define lookups for DDE entities is to include a `mds-lookups.json` file in module resource directory. The file should be a valid array of `EntityLookups` class objects. Every lookup defined in the file will be added only once, so even after user had deleted lookup it won't be recreated during module or MOTECH restart. This gives the user complete control over those lookups without any restrictions. The unique identifier of every lookup is its entity class name and lookup name combination. This is the intended way for modules to define lookups that should be made editable by the end user. Backend code should not depend on these lookups.

Example `mds-lookups.json` file.

```
[
    {
        "entityClassName" : "org.motechproject.tasks.domain.Task",
        "lookups" : [
            {
                "lookupName" : "Find Task by Owner",
                "singleObjectReturn" : false,
                "exposedViaRest" : false,
                "lookupFields" : [
                    {
                        "name" : "owner",
                        "type" : "VALUE",
                        "customOperator" : "\u003d\u003d",
                        "useGenericParam" : false
                    }
                ],
                "readOnly" : false,
                "methodName" : "findTaskByOwner",
                "fieldsOrder" : [
                  "owner"
                ]
            }
        ]
    }
]
```

Including the example json in Tasks module will result in adding lookup for Task entity that will return all tasks that are owned by the specified user.

Programmatic usage of DDE entities

All that has to be done in order to use a DDE is to retrieve the service for its interface. Because of the nature of DDEs, their classes are available during compile time. The service reference can be then retrieved using the standard OSGi facilities:

```
public PatientService getPatientService() {
    BundleContext bundleContext = FrameworkUtil.getBundle(Patient.class).getBundleContext();
    ServiceReference<PatientService> ref = bundleContext.getServiceReference(PatientService.class);
    return ref == null ? null : bundleContext.getService(ref);
}
```

The preferred way however is to use Blueprint OSGi references. The service will be injected as a Spring bean into the Spring application context of the module and can be then used as any other bean(for example it can be @Autowired into other beans).

```
<?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:osgi="http://www.eclipse.org/gemini/blueprint/schema/blueprint"
    xsi:schemaLocation="http://www.springframework.org/schema/beans
        http://www.springframework.org/schema/beans/spring-beans.xsd
        http://www.eclipse.org/gemini/blueprint/schema/blueprint
        http://www.eclipse.org/gemini/blueprint/schema/blueprint/gemini-blueprint.xsd">

    <osgi:reference id="patientDataService" interface="org.motechproject.example.PatientService"/>

</beans>
```

Once the service instance is obtained, the only thing left to do is to just call the right method exposed.

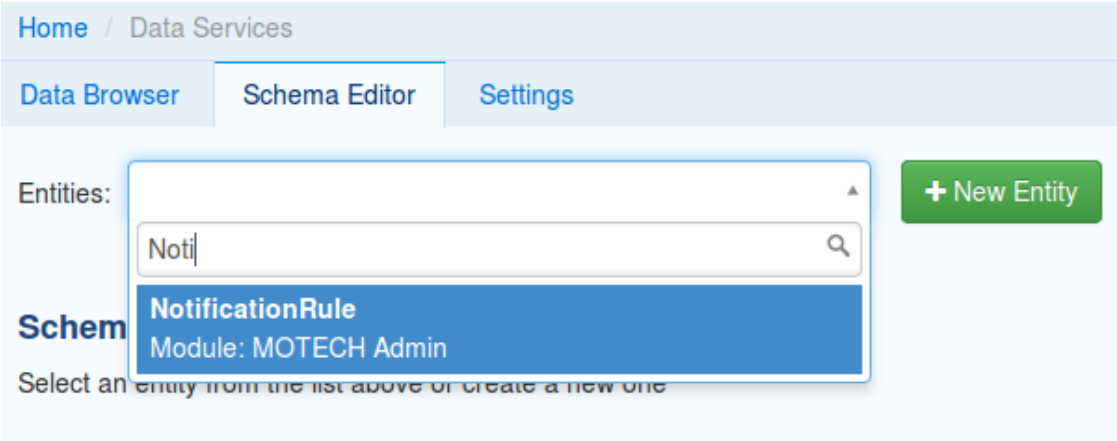
Note: Usually a module should provide a service layer between the end user and the data layer implemented by MDS. It is not required however and left to the implementer.

2.3.7 MEDE - MDS Enhanced Developer Defined Entities

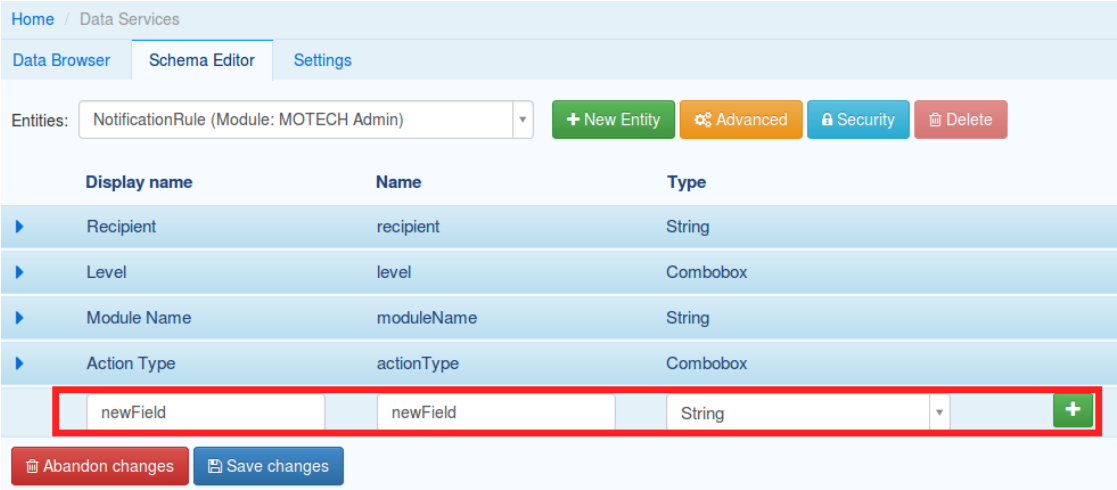
MEDE, MDS Enhanced Developer Defined Entities, are the DDE that were enhanced by users with additional fields at runtime. In practice they are not much different from DDEs. The only difference lies in the additional fields added at runtime. These fields are not part of the class at compile time, so access to these fields has to be done using reflections. They can also be set through the MDS Data Browser, so this is a way for nontechnical users to attach their own schema to the model.

Extending DDEs through the UI

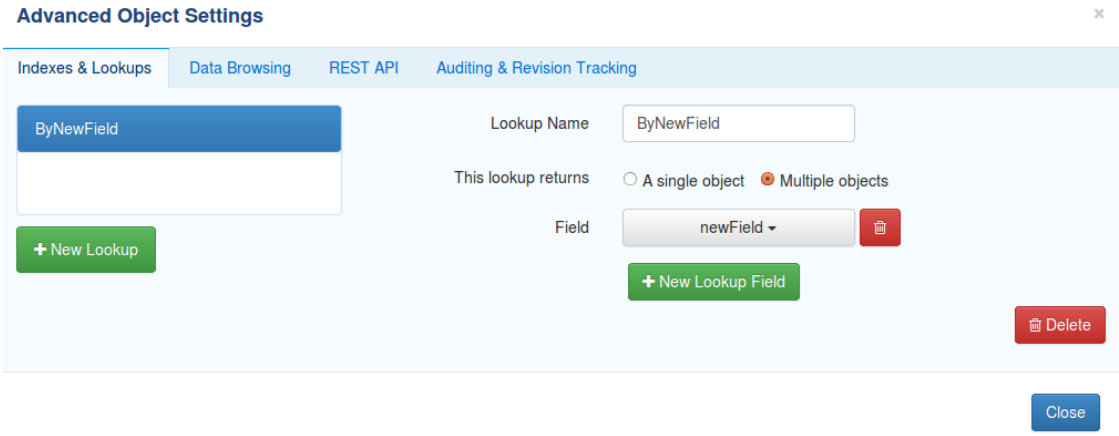
Extending DDEs through the UI is not different from manipulating the schema of EUDE entities. Refer to the documentation section on *creating EUDE entities* for more info. In order to extend a DDE first go the MDS Schema Editor and select the DDE entity you wish to edit:



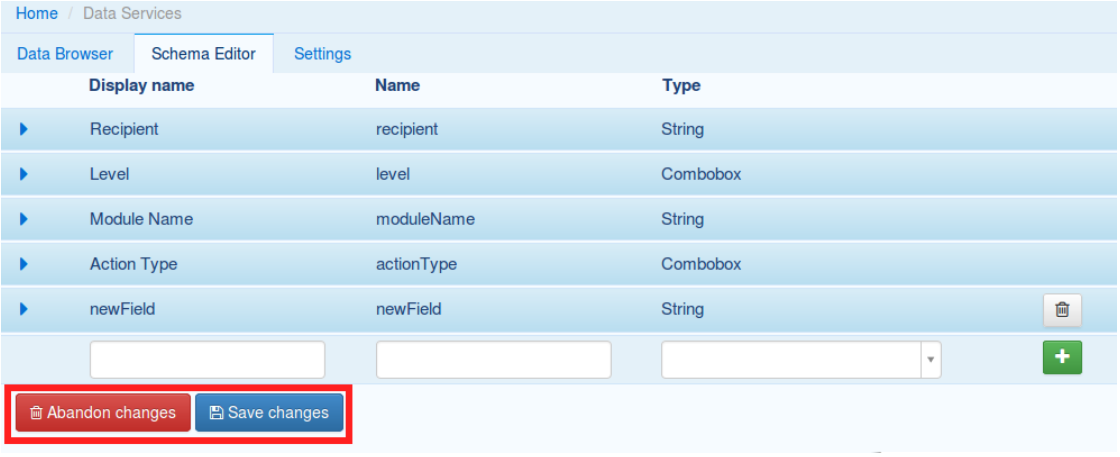
Next add the field you wish to add to the entity:



You can also add lookup to the DDE:



Finally, save your changes to trigger MDS schema regeneration and make your changes take effect(you can also abandon your changes if you wish):



Extending DDEs through code

Extending DDEs through code is no different from extending EUDE entities. The only difference is that the EntityDto for the DDE has to be retrieved by providing its class name. Refer to the documentation on *extending EUDE through code*.

2.3.8 Supported field types

MDS supports multiple types

MDS Type	Java type	MySQL DB type	PostgreSQL DB type	Description
Blob	java.lang.Byte[]	mediumblob	bytea	A huge binary object, used to represent binary objects such as files or images.
Boolean	java.lang.Boolean	bit(1)	boolean	A boolean field, that can take either true or false as value.
Combobox	Based on settings: enum enum collection java.lang.String String collection	separate table separate table varchar separate table	separate table separate table varchar separate table	A combobox showing users a selection of predefined values. It can take single or multiple selections and can be configured to take user defined values.
	java.util.Date	datetime	timestamp with time zone	A type representing the java.util.Date. Only available for DDE.
Date	org.joda.time.LocalDate	date	date	A type representing the LocalDate class from the Joda library. Does not represent time, only date.
DateTime	org.joda.time.DateTime	datetime	timestamp with time zone	A type representing the DateTime class from the Joda library.
Decimal	java.lang.Double	double	double precision	A decimal field number.
Integer	java.lang.Integer	int(11)	integer	An integer number.
Locale	java.util.Locale	varchar	varchar	A type representing locale. Users will be shown a locale selection dropdown for type.
Map	java.util.Map	Separate table	Separate table	A map of key-value pairs.
Period	org.joda.time.Period	varchar	varchar	A type representing the Period class from the Joda library. Represents a period in time, i.e. 3 months.
String	java.lang.String	varchar	varchar	A string of characters. The max length can be configured. For long text fields, consider using TextArea.
TextArea	java.lang.String	mediumtext	text	A string of characters without max length. Suited for long text fields.
Time	org.motechproject.commons.date.model Time	varchar	varchar	A time representation without any date or timezone information.

Map type

You can declare map with keys and values having generic type. MDS supports the following types of generics :

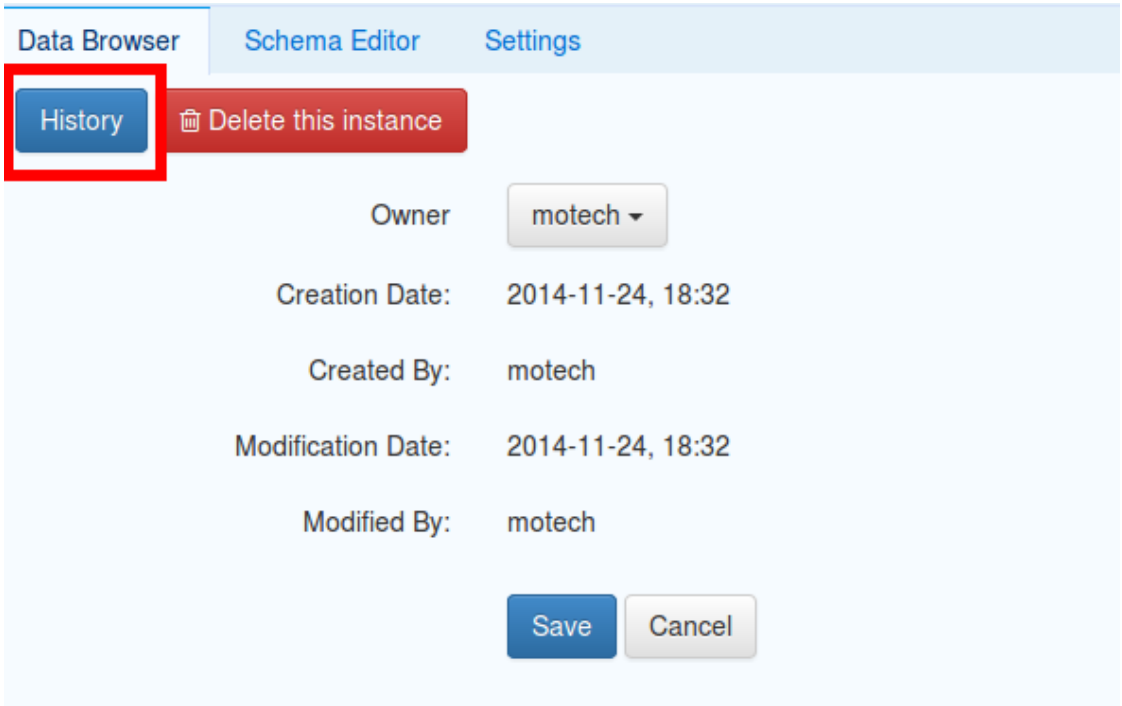
- key types (String, Integer, Long)
- value types (String, Integer, Long)

If you use the supported types, the field will be stored as a separate table in a database. Otherwise the field will be serialized.

Note: In a separate table map keys will be treated as primary keys. By default max key length in InnoDB is 767 bytes. When the innodb_large_prefix configuration option is enabled, this length limit is raised to 3072 bytes, for InnoDB tables that use the DYNAMIC and COMPRESSED row formats. Here you can find more details : http://dev.mysql.com/doc/refman/5.6/en/innodb-parameters.html#sysvar_innodb_large_prefix

2.3.9 History tracking for entities

MDS allows to keep track of any changes made on the instances, as well as reverting the state of an instance to a concrete revision. Both viewing the history of an instance and reverting can be done via the code and UI. This feature will only be available if you explicitly set, that the history tracking for your entity should be enabled. If you want to view the history for your instance via UI, simply go to the detailed view of that instance, and click on the **History** button.



Note: If you introduce any changes to the entity definition (e.g. add or delete a field), you will still be able to view the state of an instance, but you will lose the ability to revert an instance (because of a schema mismatch).

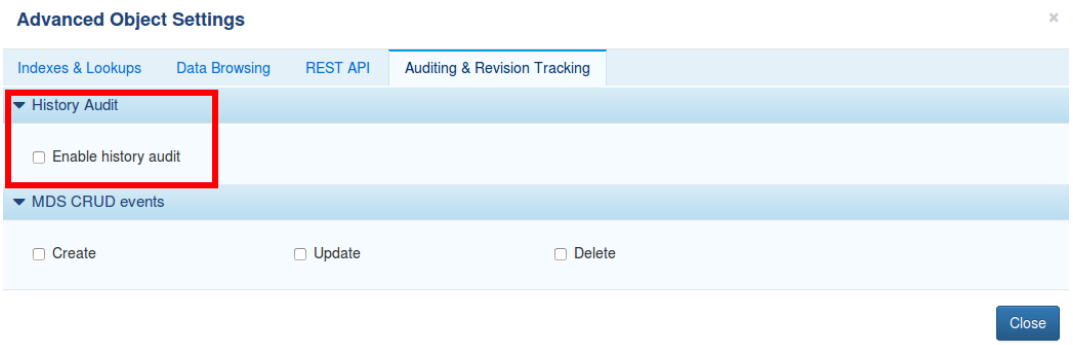
Controlling whether to record history

By default MDS doesn't keep track of the instance revisions. Most of the DDEs that come with MOTECH modules have the tracking of the history disabled as well. To enable history tracking for the...

- Developer Defined Entity (DDE) - You have to set the **recordHistory** parameter of the **@Entity** annotation to true.

```
@Entity(recordHistory = true)
```

- End User Defined Entity (EUDE) - The **Enable history audit** option is available under the **Advanced** window of an entity, in the **Auditing & Revision Tracking** tab



Retrieving history using code

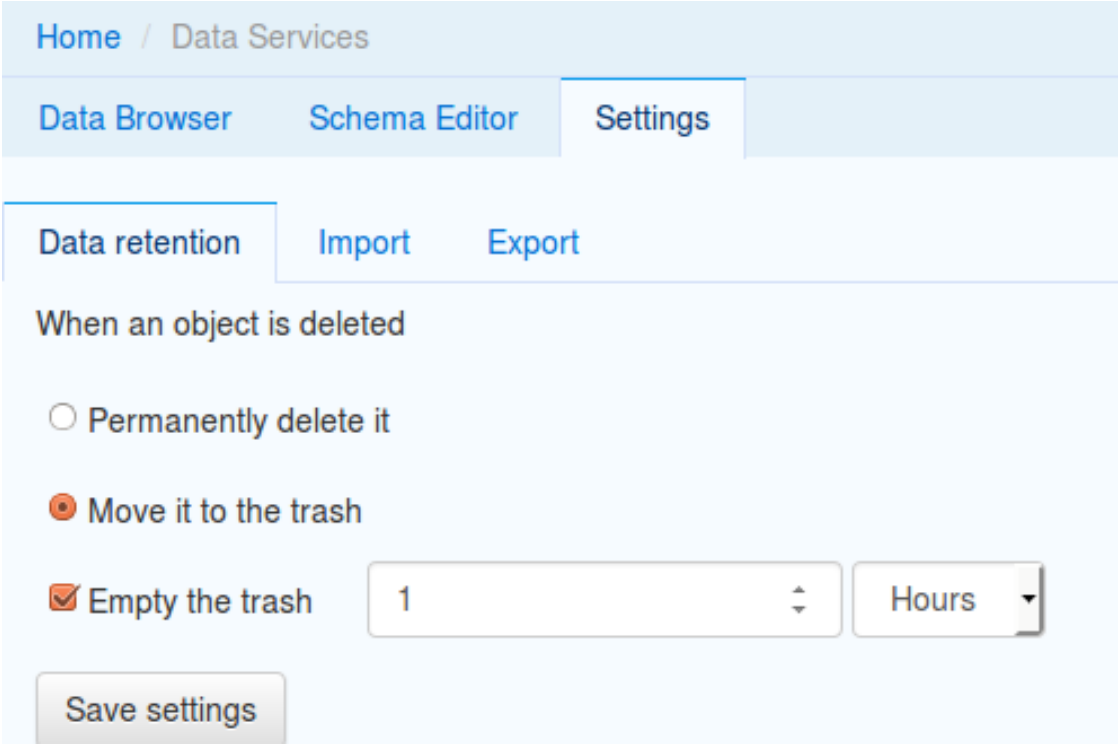
MDS exposes an implementation of the **org.motechproject.mds.service.HistoryService**. To make use of it, you should simply create a reference to that service in your blueprint:

```
<osgi:reference id="historyServiceOSGi" interface="org.motechproject.mds.service.HistoryService" />
```

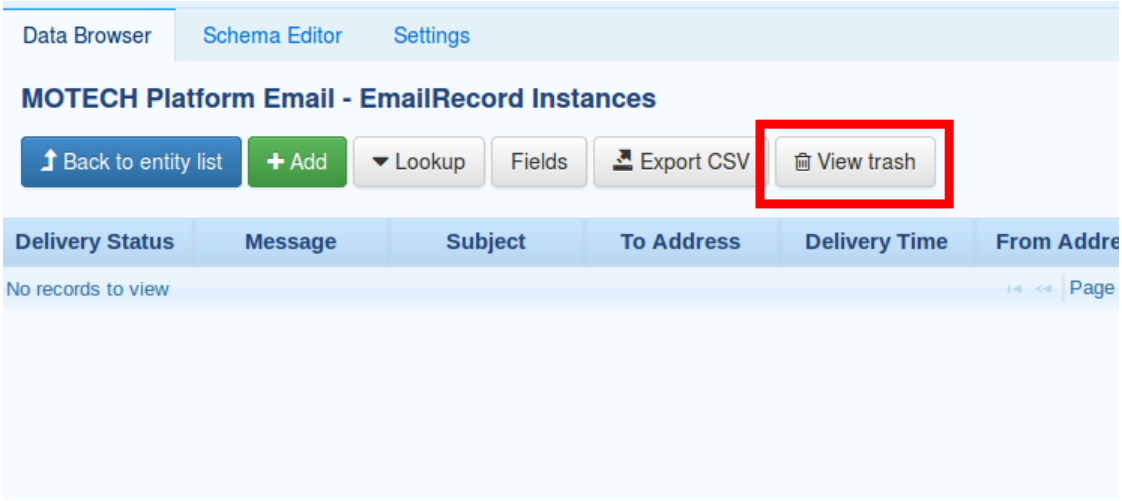
From now on, you will be able to use the history service, just like any other Spring bean, for example, by placing the **@Autowired** annotation on a field of type **org.motechproject.mds.service.HistoryService**. The service allows recording history, deleting the whole history for an instance and retrieving the historical revisions of an instance.

2.3.10 MDS Trash Bin

When an instance is deleted, it can either be removed completely or moved to the trash. In case an instance is moved to the trash, there will be an ability to view all instances that have been deleted, as well as to restore any instance from the trash. Users may also choose to empty the trash from time to time. All the data retention settings are available in the MDS settings tab. If you choose to empty the trash, MDS will use the scheduler to set up a job, that runs every specified period and empties the trash.



To view instances that have been moved to the trash, click the **View trash** button, after selecting an entity in the data browser. To restore any instance from the trash, select that instance and click **Restore** button on the detailed view of the deleted instance.



Note: If you introduce any changes to the entity definition (e.g. add or delete a field), you will lose access to all the deleted instances of the previous schema. That means you will no longer be able to view or restore them anymore.

Using Trash using code

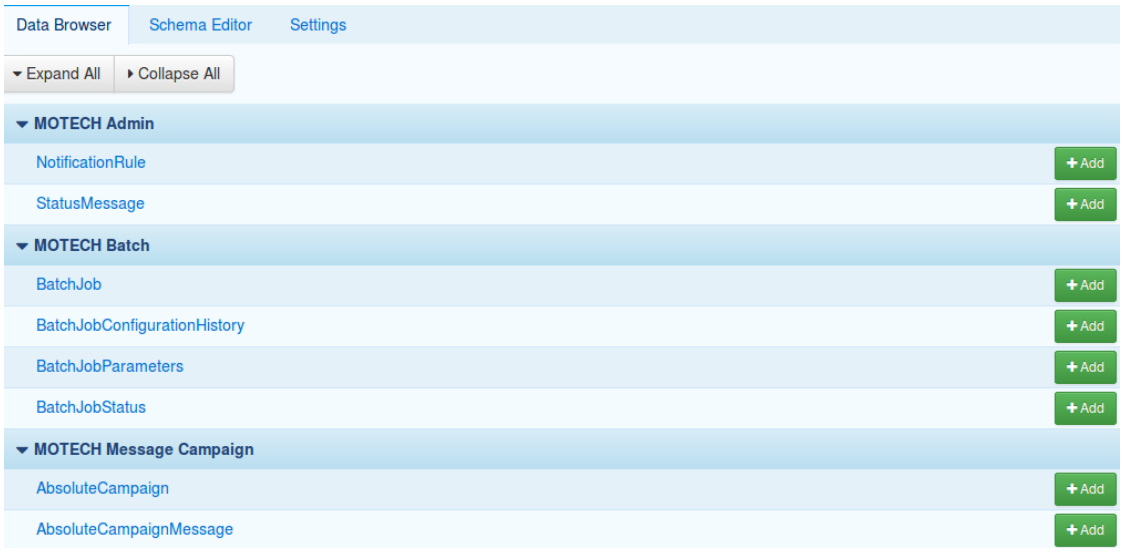
Similar to the HistoryService mentioned above, MDS also exposes the **TrashService** that allows operations on the Trash bin from the code. To use the exposed service, create a reference in your blueprint file:

```
<osgi:reference id="trashServiceOSGi" interface="org.motechproject.mds.service.TrashService" />
```

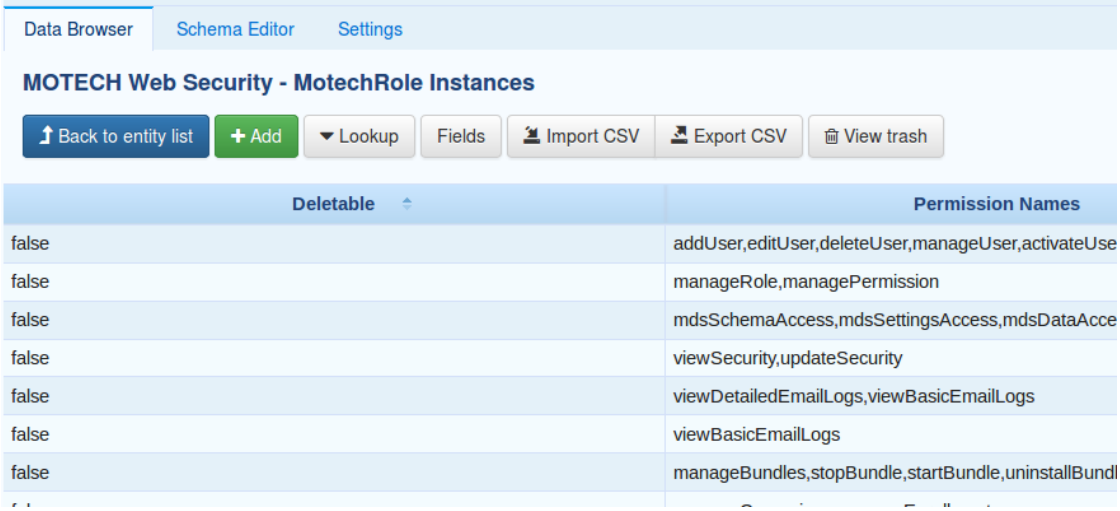
Accessing the service also works the same way as with the HistoryService - treat it as any other Spring bean, for example by placing the **@Autowired** annotation on the field of type **org.motechproject.mds.service.TrashService**. The trash service allows to place instances in trash, retrieve instances from trash, schedule the trash purging, empty the trash and check current data retention settings.

2.3.11 The MDS Data Browser

The data browser is a place, where you can perform CRUD operations on the instances of an entity. The main window of the data browser shows a list of all entities, grouped by modules to which they belong. From this point, you can choose to view instances of a certain entity by clicking on the name of that entity, or add an instance of an entity by pressing the **Add** button, next to the entity name.

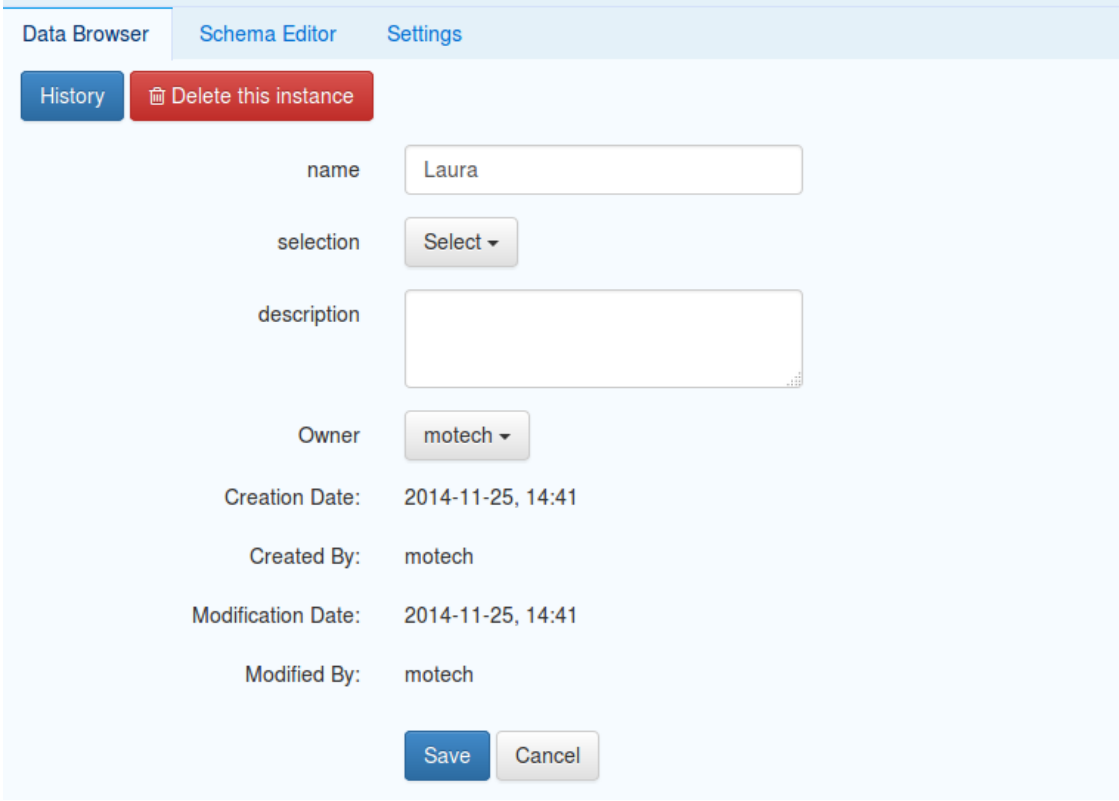


If you pick one of the entities, you will be brought to the view, showing the instances of that entity. From this view, you can perform several operations on the instances.



Button	Role
Back to entity list	Brings you back to the main data browser view, listing entities
Add	Brings you to the Add instance dialog, where you can add an instance of an entity
Lookup	Allows you to view only instances that match certain criteria. The definition of these criteria are set in the Advanced dialog on the Schema Editor
Fields	Allows you to display only certain fields in the browser. Useful when your entity has got a lot of fields, and you are only interested in few of them
Import CSV	This option allows the import of instances from a CSV file. If there is an instance with the same id present both in the database and the file, it will get updated with the values from the file
Export CSV	This option allows the export of all instances of the selected entity to the CSV file
View trash	Allows to view all instances that have been moved to the trash, on the current entity schema

If you click on any instance, a detailed view for that instance will be shown. Depending on the entity definition, necessary input fields will be presented, where you can set the values for these fields. You may also choose to delete that instance or view the revision history (if history tracking is enabled for that entity). When you are done editing an instance, click the **Save** button. To abandon changes, click **Cancel**.



2.3.12 Data browsing settings

The data browsing settings allow to control several data browser UI options for an entity. Available options are:

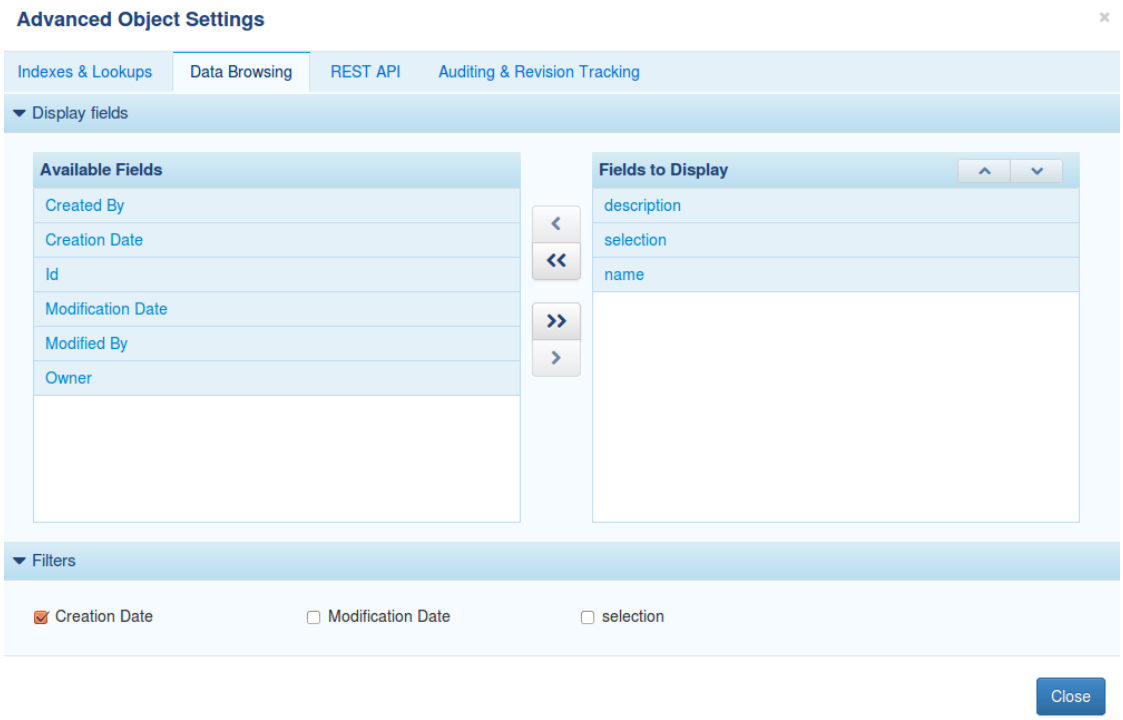
- The ordering of the entity fields
- The fields to display on the UI by default
- Allow filtering by chosen field values (only available for some types)

The automatically generated fields are not displayable by default, but all other fields are. The display order is determined based on the order in which they were added. No fields will be marked filterable by default.

Note: The data browser filters can currently only be generated for the Date, DateTime, LocalDate, Boolean and List types.

Changing the settings through the UI

To change the data browsing settings via UI, go to the Schema Editor and select an entity for which you wish to set the settings. Go to the **Advanced** view and pick the **Data Browsing** tab. The first section, called **Display fields**, contains two tables. The table to the right shows fields that have been selected to display by default. The table to the left shows all other fields. The order of the fields in the **Fields to display** table corresponds to the order of the fields in the data browser UI. You can move fields from one table to another and change their order, using provided buttons, or by dragging the fields to their destination. The second section, named **Filters** allows to pick fields, for which the data browser UI will generate filters. Please note that only fields of a certain types will be displayed. The filters are generated automatically and are adjusted to the field type. For example, for the date types, there will be an option to set a filter for today, this week, this month and this year, while for boolean, this will be only true and false. When you finish making the changes, close the Advanced window and click **Save changes**.



Changing the settings through annotations

The data browsing settings can also be set using MDS annotations. The two annotations that allow this are **@UIDisplayable** and **@UIFilterable**. Similar to the @Field annotation, they can be placed on fields, as well as on getters and setters. The **@UIFilterable** annotation will work only, when placed on the field of a supported type.

Note: If you use the **@UIDisplayable** annotation on any field of your entity, all other fields, that lack the annotation, will be marked as not displayable.

By default, all fields defined in the entity will be marked as displayable. The **@UIDisplayable** annotation allows changing this behaviour. If at least one field is marked with the **@UIDisplayable** annotation, the default behaviour will not be applied, and only annotated fields will be marked displayable. The annotation contains optional **position** parameter, that allows to pick the position of the field on the data browser UI. The ordering should start with the number zero. Fields are not UIFilterable by default. To allow filtering by field values on the data browser, simply annotate that field with **@UIFilterable**.

The following code presents the usage of the two annotations:

```
@Field
private String externalId;

@Field
@UIDisplayable(position = 0)
private String name;

@Field
@UIDisplayable(position = 2)
@UIFilterable
private DateTime dateTime;

@Field
@UIDisplayable(position = 3)
private Long priority;

@Field
@UIDisplayable(position = 1)
private String description;
```

2.3.13 The REST API

MDS REST API allows to perform CRUD operations on the instances of an entity. By default, no operations are allowed via REST, which means that an administrator, must explicitly allow an access via REST to an entity. Even when an access via REST is enabled for an entity, valid MOTECH credentials must be provided in order for a request to be processed. MDS REST API uses a BASIC access authentication method by default, but that can be changed using *dynamic security rules* (can be done on a per entity basis). Moreover the standard *MDS entity level security* will also apply.

REST endpoints

The general endpoint to the MDS REST operations is: `http://<motech-server-address>/module/mds/rest/<<path>>`

The table below explains what HTTP request method are supported for each of the CRUD operation, as well as how the "path" should look like.

Operation	HTTP requests	Paths	Notes

Create	POST	`/{moduleName}/{namespace}/{entityName}`
		`/{moduleName}/{entityName}`
		`/{entityName}`
			The data sent with the request should contain JSON representation of the object

Read	GET	`/{moduleName}/{namespace}/{entityName}`
		`/{moduleName}/{entityName}`
		`/{entityName}`
			Can take multiple params, like ?page=1&pageSize=20&sort=name

Read - Lookup	GET	`/lookup/{moduleName}/{namespace}/{entityName}/{lookupName}`
		`/lookup/{moduleName}/{entityName}/{lookupName}`
		`/lookup/{entityName}/{lookupName}`
			Can take multiple params, like ?page=1&pageSize=20&sort=name
			Lookup parameters should be provided as request parameters.

Update	PUT	`/{moduleName}/{namespace}/{entityName}`
		`/{moduleName}/{entityName}`
		`/{entityName}`
			The instance to update will be determined on the id, taken from included JSON representation

Delete	DELETE	`/{moduleName}/{namespace}/{entityName}/{instanceId}`
		`/{moduleName}/{entityName}/{instanceId}`
		`/{entityName}/{instanceId}`	

Note: EUDE are never assigned to any module. For DDE, the module name should not contain the "motech" or "motech-platform" prefix, if the module has one.

Response codes

These are the response codes returned by the MDS REST API:

- **200** *OK* - The operation was successful. Note that delete is idempotent, meaning 200 will be also returned for already deleted items.
- **400** *Bad Request* - The body or parameters provided in the request are invalid.
- **401** *Unauthorized* - The caller is not authorized and thus not permitted to execute the operation.
- **403** *Forbidden* - The user does not have necessary rights to execute the operation.
- **404** *Not Found* - Either the given entity or the requested object does not exist.
- **500** *Internal Server Error* - The request cannot be processed due to a server error.

Read response

In case of read operations Motech also adds metadata to the response. Response is divided into two sections: metadata and data. The metadata contains following fields:

Name	Description	Type
entity	The entity name of the instances.	String
class-Name	The name of the entity class.	String
module	The module name of the entity. Null in case of EUDE entity.	String
names-pace	The namespace in which the entity is defined.	String
total-Count	The total number of instances that match the search conditions. 1 i case of retrieving with `id` parameter or with a single object lookup.	Long
page	The page number.	Inte-ger
page-Size	The page size.	Inte-ger

Below you can find sample response:

```
{
  "metadata": {
    "entity": "EmailRecord",
    "className": "org.motechproject.email.domain.EmailRecord",
    "module": "MOTECH Platform Email",
    "namespace": "",
    "totalCount": 2,
    "page": 1,
    "pageSize": 20
  },
  "data": [
    {
      "id": 1,
      "creator": "admin",
      "owner": "admin",
      "modifiedBy": "admin",
      "deliveryStatus": "SENT",
      "toAddress": "adress1@organisation.com",
      "subject": "Subject 1",
      "message": "Sample message",
      "fromAddress": "adress2@organisation.com",
    },
    {
      "id": 2,
      "creator": "admin",
      "owner": "admin",
      "modifiedBy": "admin",
      "deliveryStatus": "SENT",
      "toAddress": "adress1@organisation.com",
      "subject": "Subject 2",
      "message": "Other message",
      "fromAddress": "adress2@organisation.com",
    }
  ]
}
```

Parameters and lookups

When retrieving the instances using MDS REST API (GET request), there's an ability to apply some parameters, to have a better control on the result of the request. The parameters are applied as any other GET request parameters.

- **id** Return a single instance, with the provided id
- **pageSize** Defines an amount of instances that should be returned per request (defaults to 20)
- **page** Defines a result page that should be returned (defaults to 1)
- **sort** Defines a column that should be used to sort the instances in the result
- **order** Either "asc" or "desc"
- **lookup** A name of lookup that should be used to retrieve the instances. A lookup must be marked as exposed via REST in order for this to work. The values used in the lookup should be provided as GET request parameters. This an alternative way of calling a lookup, rather than calling it through the lookup url described above.

Below, you will find some examples of valid REST URLs. Assume our entity is called MyEntity.

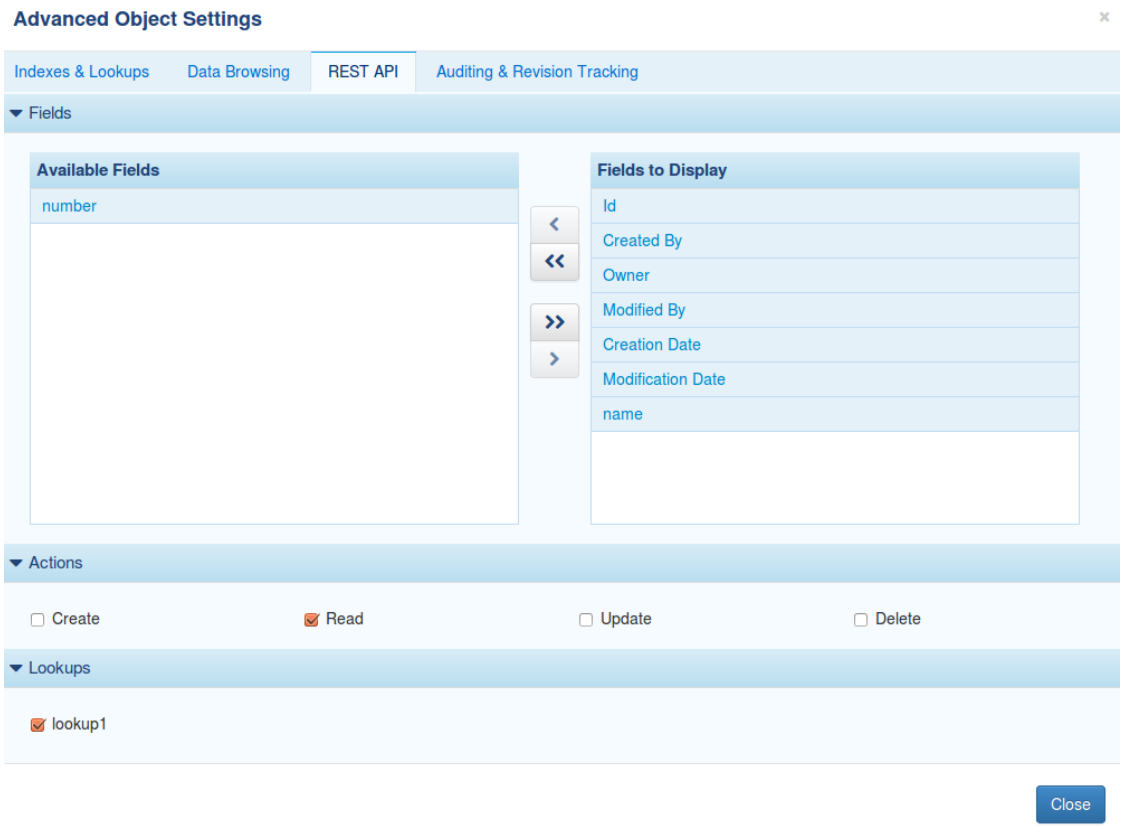
- `http://<<address>>:<<port>>/motech-platform-server/module/mds/rest/MyEntity`
  Return 20 records from the first page (default settings applied)
- `http://<<address>>:<<port>>/motech-platform-server/module/mds/rest/MyEntity?id=15`
  Return an instance with id 15
- `http://<<address>>:<<port>>/motech-platform-server/module/mds/rest/MyEntity?page=2&pag`
  Return 50 records from the second page, having sorted the instances by name field ascending
- `http://<<address>>:<<port>>/motech-platform-server/module/mds/rest/MyEntity?lookup=byN`
  Executes a lookup named "byName" with the lookup field "name" being "Laura" on the entity "MyEntity" and returns results.

REST fields exposed

By default all fields are marked as exposed via REST, both for DDE and EUDE. If you choose to hide some of them, they will simply be ignored, when performing CRUD operations via REST on them. When retrieving instances, the result will not contain the fields that are not exposed and when updating or creating instances, the hidden fields will be ignored, even if they are present in the provided JSON representation.

Changing REST settings through the UI

You can access the REST API settings by selecting an entity in the Schema Editor and then opening the advanced settings, by clicking on the **Advanced** button. On the new window, navigate to the **REST API** tab.



The settings may contain up to three sections:

- The first one, named **Fields** allows to pick fields that should be exposed via REST. Fields in the table to the right are exposed and fields in the table to the left are not. You can drag and drop fields from one table to another or select them and use provided buttons.
- The next section is named **Actions** and defines the operations on the instances that are allowed via REST for this entity. By default, no action is allowed. You can choose to change it, by selecting some or all of the actions.
- The last section, called **Lookups** will appear only if there is at least one lookup defined for an entity. This section allows to pick the lookups that can be executed via REST. Note, that to execute lookups at all, a "Read" action must be enabled.

Changing REST settings through annotations

The REST settings can also be applied using MDS annotations. The three annotations that allow this, are:

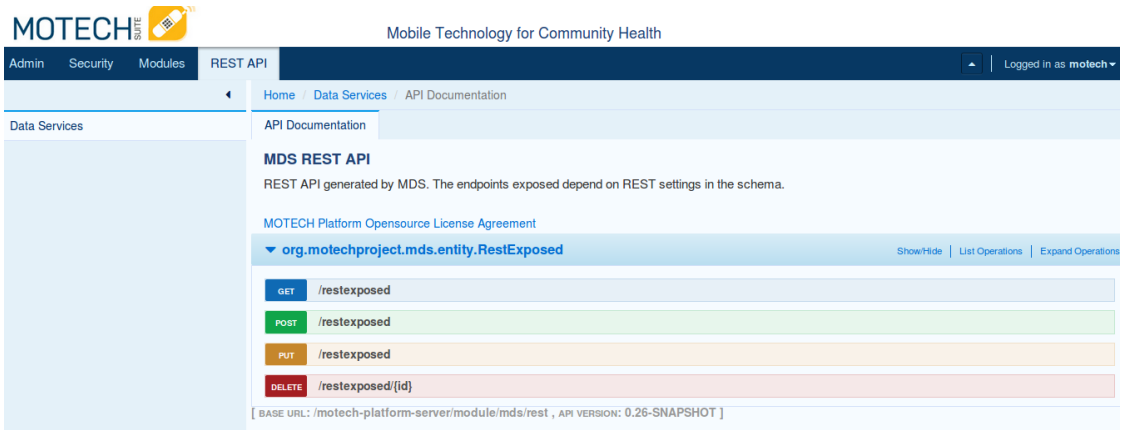
- **@org.motechproject.mds.annotations.RestIgnore** As stated in the previous sections, be default all fields are exposed via REST. You can adjust this behaviour using this annotation. Annotated fields will not be exposed via REST.
- **@org.motechproject.mds.annotations.RestOperations** Placed on the entity class definition, specifies the REST operations that should be allowed for this entity. The annotation takes an array of org.motechproject.mds.annotations.RestOperation, which is an enum of possible values.
- **@org.motechproject.mds.annotations.RestExposed** Placed on the lookup method definition, in the service interface. Annotated lookup methods will be marked as exposed via REST. By default, lookups are not exposed via REST.

The code below shows an example usage of the annotations:

```
@Entity
@RestOperations({RestOperation.CREATE, RestOperation.READ})
public class MyEntity {

    @Field
    @RestIgnore
    private Integer number;

    @Field
    private String emailAddress;

    @Field
    private String message;
}


public interface MyEntityService extends MotechDataService<MyEntity> {

    @Lookup(name = "By number")
    List<MyEntity> findByNumber(@LookupField(name = "number") Integer number);

    @Lookup(name = "By Email Address")
    @RestExposed
    List<MyEntity> findByEmailNumber(@LookupField(name = "emailAddress") String emailAddress);
}
```

REST documentation

MOTECH provides a user interface that documents and allows the testing of the REST API exposed by MDS. This interface is generated using Swagger. In order to access this UI, first select **REST API** in the top menu, then **Data Services** in the sub-menu.



The raw Swagger specification file (JSON format) is accessible at *<your_motech_url>/module/mds/rest-doc*.

2.3.14 Entity validations

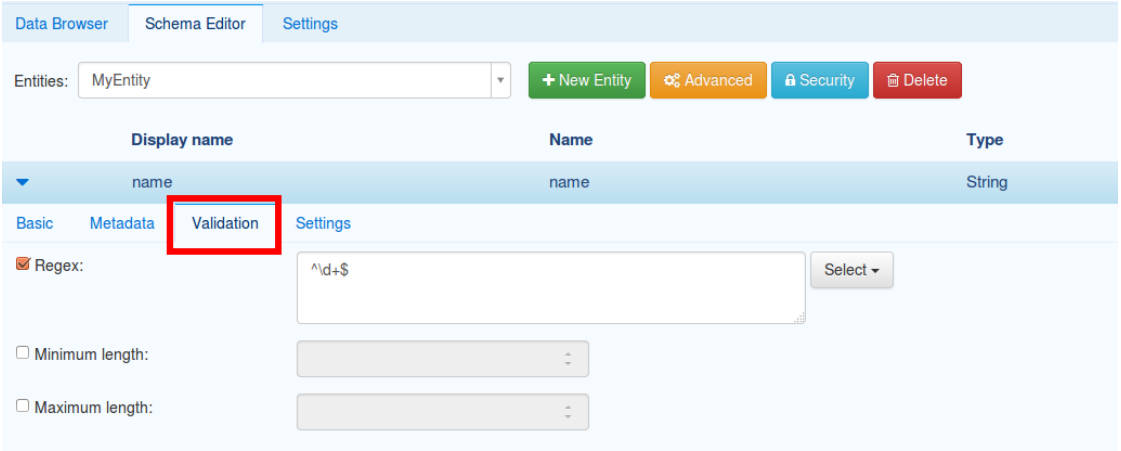
MDS allows to set up validations on the fields of an entity. A validation ensures that values of created instances will match some criteria. The validations are applied on two levels:

- UI - MDS UI will check the values when adding or editing instances and display hints or errors, when the value does not match some of the defined validations.

- Code - Attempting to save an instance that has got invalid values, using the retrieved MotechDataService, will result in a **ConstraintViolationException**.

Configuring validations through the UI

To set up validations for a field of an entity, open the Schema Editor and select an entity, for which you wish to set validations. Expand the field that should be validated and navigate to the **Validation** tab.

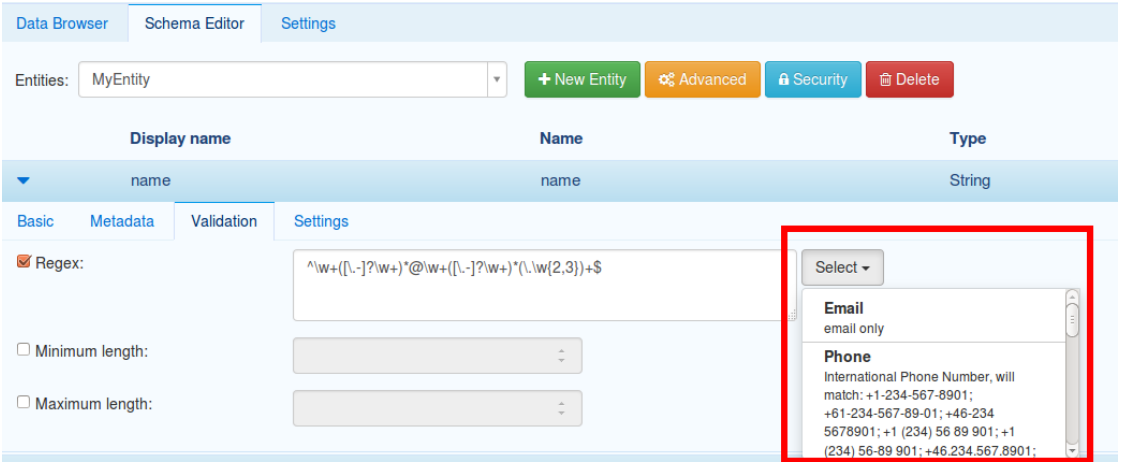


Only some of the MDS types support setting up validations via UI, so if a selected field is of a type that is not supported, the **Validation** tab will not appear. Please see the list of supported types and validations below.

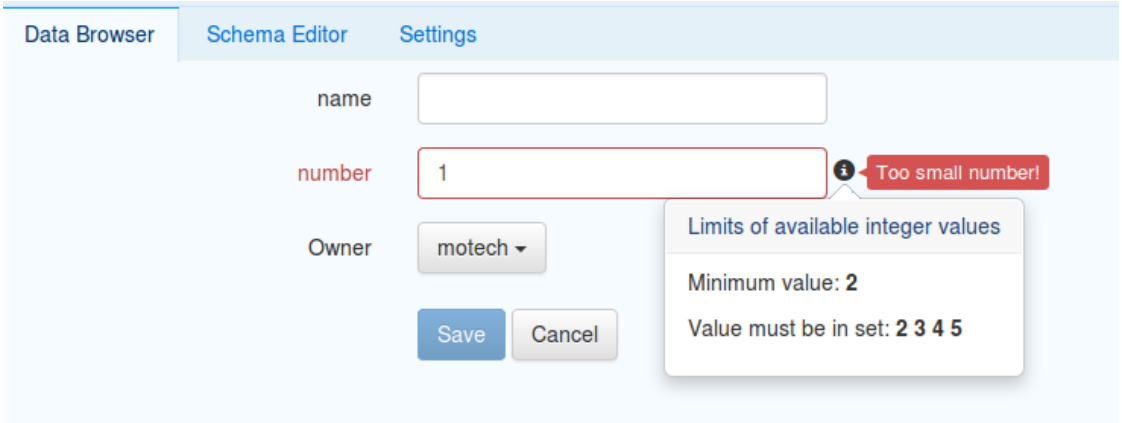
Type	Validation	Annotation	Description
String	Regex	@javax.validation.constraints.Pattern	Allows to set up a regular expression. Only strings that match the regex will be accepted.
String	Minimum length	@javax.validation.constraints.Size	Defines a minimal number of characters the strings must have.
String	Maximum length	@javax.validation.constraints.Size	Defines a number of characters the strings cannot exceed.
Integer / Decimal	Minimum value	@javax.validation.constraints.Min @javax.validation.constraints.DecimalMin	Defines a minimal number that will be accepted.
Integer / Decimal	Maximum value	@javax.validation.constraints.Max @javax.validation.constraints.DecimalMax	Defines a maximal number that will be accepted.
Integer / Decimal	Must be in set	@org.motechproject.mds.annotations.InSet	Only numbers that have been explicitly specified will be accepted.
Integer / Decimal	Cannot be in set	@org.motechproject.mds.annotations.NotInSet	All numbers that have not been explicitly specified will be accepted.

Note: Setting up validations via UI is only possible for the EUDE.

The **Regex** validation contains some predefined patterns, for the most common use cases. To view them, click **Select**, next to the Regex input field and pick one of the available, predefined expression. This will automatically, place the regular expression in the input field. Please note, that this operation will erase the current value in the field, if there's any provided.



Setting up validations will display hints while adding an instance of an entity, that has got validated fields. An attempt to add an instance with invalid values, will display an error and block the ability to save the instance.



Configuring validations using annotations

For DDEs, it is possible to set up validations using the annotations. MDS will recognize the @javax.validation.constraints annotations, as well as two MDS-defined annotations: **@org.motechproject.mds.annotations.InSet** and **@org.motechproject.mds.annotations.NotInSet**. See the code below, for an example of validation definition through annotations.

```
@Entity
public class MyEntity {

    @Field
    @Min(10)
    @Max(100)
    private Integer number;

    @Field
    @Pattern(regexp = "^\\w+([\\.-]?\\w+)*@\\w+([\\.-]?\\w+)*(\\.\\w{2,3})+$")
    private String emailAddress;

    @Field
    @AssertTrue
    private Boolean alwaysTrue;
```

```
    @Field
    @Size(min = 64, max = 2048)
    private String message;
}
```

Note: When using annotations, take into consideration what field types they can be applied to. Most of the annotations support only one or a few types.

Even though you can use any @javax.validation.constraints annotation on an entity field, the UI support (hints, error messages), will only be displayed for the validations listed in the previous section, about setting validation through UI. Other validations will not show up on the UI, but it still will not be possible to add an invalid value - a **ConstraintViolationException** will be thrown.

2.3.15 MDS Lookup Service

The org.motechproject.mds.service.MdsLookupService is an OSGi service which allows easy access to executing queries on entities without compile time access to their classes. It can also be useful for executing on entities without knowing the entity name at compile time. An example is the IVR module which exposes this service to velocity templates, allowing users data access.

Note: As with all MDS API, the MdsLookupService uses the underlying MotechDataService for the entity underneath. It is really just a facade for service access.

The service exposes these methods:

```
public interface MDSLookupService {

    <T> T findOne(Class<T> entityClass, String lookupName, Map<String, ?> lookupParams);
    <T> T findOne(String entityClassName, String lookupName, Map<String, ?> lookupParams);

    <T> List<T> findMany(Class<T> entityClass, String lookupName, Map<String, ?> lookupParams);
    <T> List<T> findMany(String entityClassName, String lookupName, Map<String, ?> lookupParams);
    <T> List<T> findMany(Class<T> entityClass, String lookupName, Map<String, ?> lookupParams,
                         QueryParams queryParams);
    <T> List<T> findMany(String entityClassName, String lookupName, Map<String, ?> lookupParams,
                         QueryParams queryParams);

    <T> List<T> retrieveAll(Class<T> entityClass);
    <T> List<T> retrieveAll(String entityClassName);
    <T> List<T> retrieveAll(Class<T> entityClass, QueryParams queryParams);
    <T> List<T> retrieveAll(String entityClassName, QueryParams queryParams);

    long count(Class entityClass, String lookupName, Map<String, ?> lookupParams);
    long count(String entityClassName, String lookupName, Map<String, ?> lookupParams);

    long countAll(Class entityClass);
    long countAll(String entityClassName);
}
```

For the examples below assume the following classes:

```
@Entity
public class Patient {

    @Field
```

```
    public String name;

    @Field
    public Integer age;

    @Field
    private Set<Visit> visits;
}

@Entity
public class Visit {

    @Field
    public Integer officeNumber;

    @Field
    public DateTime date;
}
```

with the following lookups defined in its data service:

```
public interface PatientService extends MotechDataService<Patient> {

    @Lookup
    Patient byName(@LookupField(name = "name") String name);

    @Lookup
    List<Patient> byAge(@LookupField(name = "age") Integer age);
}
```

The **findOne** methods can be used to execute single return lookups given the lookup name, the entity class name(or class object) and map consisting of the lookup params, where the key is the lookup parameter name and the value is the actual parameter. Usage example:

```
Map<String, ?> params = new HashMap<>();
params.put("name", "John");

// type safe method
Patient patient = mdsLookupService.findOne(Patient.class, "findByName", params);

// alternative method
Patient patient = (Patient) mdsLookupService.findOne("org.motechproject.example.Patient", "findByNam
```

The **findMany** method can be used to execute multiple result lookups. Additional versions of the method allow executing the lookup with QueryParams, which control/pagination ordering. Usage example:

```
Map<String, ?> params = new HashMap<>();
params.put("age", 29);

// type safe method
Patient patient = mdsLookupService.findOne(Patient.class, "findByAge", params);

// alternative method
List<Patient> patients = (List<Patient>) mdsLookupService.findOne("org.motechproject.example.Patient

// with QueryParams

// first page, with pages consisting of 10 records
// order by name, descending
```

```
QueryParams queryParams = new QueryParams(1, 10, new Order("name", Order.Direction.DESC));

// type safe method
Patient patient = mdsLookupService.findOne(Patient.class, "findByAge", params, queryParams);

// alternative method
List<Patient> patients = (List<Patient>) mdsLookupService.findOne("org.motechproject.example.Patient"
```

The **retrieveAll** methods can be used as above with omission of parameter maps, since instead of using a lookup, it retrieves all records from the database executing retrieveAll on the service.

The **count** and **countAll** methods are also no different in terms of usage. The only difference is that they return the number of instances returned by a lookup and the total number of instances respectively.

Lookups on relationship fields can be used like in the example below:

```
public interface PatientService extends MotechDataService<Patient> {

    @Lookup
    List<Patient> byVisitsDate(@LookupField(name = "visits.date") DateTime date);

    @Lookup
    List<Patient> byVisitsDateAndVisitsOffice(@LookupField(name = "visits.officeNumber") Integer off
                                              @LookupField(name = "visits.date") Range<DateTime> date
}
```

Note: MDS Lookups support only first depth level of relationships.

2.3.16 Executing custom queries

Executing JDO queries

MDS allows developers to use the JDO API offered by DataNucleus to execute any query they wish. A utility method for calling direct SQL queries through DataNucleus. Although the approach of executing custom queries gives the user all the flexibility he needs, the more easier and recommended approach is to use *Lookups* instead. This API remains in place however in order to fulfil the more complex requirements.

In order the execute a custom JDO query, the developer has to implement the org.motechproject.mds.query.QueryExecution interface and pass an instance of this implementation to the **executeQuery(QueryExecution)** method. This interface exposes one method - execute(javax.jdo.Query, org.motechproject.mds.util.InstanceSecurityRestriction). The first a parameter is the javax.jdo.Query instance class created using the PersistenceManager for the entity class of the data service being used, the second is an object describing security restrictions on the entity.

What is returned by the interface method will be also returned by the executeQuery() call on the data service. The interface is generic, the type parameter represents the return value.

Following is an example of executing a custom JDO query. Given a simple entity:

```
@Entity
public class Example {

    public Integer amount;

    public String name;
}
```

Here is an example of a JDO query that will check the amount value and based on that select only the names from the database:

```
// get the service for the entity you wish to execute the query on
MotechDataService<Example> service = getService();

QueryExecution<List<String>> queryExecution = new QueryExecution<List<String>>() {
    @Override
    public List<String> execute(Query query, InstanceSecurityRestriction restriction) {
        // return objects with the amount value either less then 1000 or greater then 1000
        query.setFilter("amount < 100 || amount > 1000");

        // select only the name column
        query.setResult("name");

        // limit the results
        query.setRange(0, 100);

        return (List<String>) query.execute();
    }
};

List<String> names = service.executeQuery(queryExecution);
```

More info on JDO queries can be found here: http://www.datanucleus.org/products/datanucleus/jdo/jdoql.html

Executing SQL queries

Similar to executing JDO queries MDS also provides developers with access to executing SQL queries. Instead of implementing the QueryExecution interface however, developers have to implement the **org.motechproject.mds.query.SqlQueryExecution** interface. This interface has two methods, **execute(javax.jdo.Query)** and **getSqlQuery()**. The contents of the SQL query should be returned by the **getSqlQuery** methods, so that MDS can construct the JDO query using that SQL.

Following is an example of executing a custom SQL query. Given a simple entity:

```
@Entity
public class Example {

    public Integer amount;

    public String name;
}
```

Here is an example of a SQL query that will return values with the given amount:

```
// there is really no impact on which data service is used, since this is raw sql
MotechDataService<Example> service = getService();

SqlQueryExecution<List<String>> sqlQueryExecution = new SqlQueryExecution<List<String>>() {
    @Override
    public List<String> execute(Query query) {
        // usage of params
        Map<String, Integer> params = new HashMap<>();
        params.put("param", 5);
        return (List<String>) query.executeWithMap(params);
    }
```

```
    @Override
    public String getSqlQuery() {
        // this query will be executed by MDS
        return "SELECT name FROM MDS_EXAMPLE WHERE amount = :param";
    }
};

List<String> names = service.executeSQLQuery(sqlQueryExecution);
```

Note that using raw SQL should be the absolute last resort, it is advised to stick to more high-level concepts in your code.

2.3.17 Using Spring Transactions with MDS

Spring transactions (the @Transactional annotation) can be used inside your MOTECH module with MDS, however this requires some setup inside the module that wishes to use these transactions.

Firstly, Spring annotation driven transactions must be configured in the Spring context. The transaction manager that is used, must be the one exposed by the MDS entities bundle as an OSGi service. Below is a minimal example configuration that defines a reference to the MDS transaction manager and uses it when declaring annotation driven transactions:

```
<?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:osgi="http://www.eclipse.org/gemini/blueprint/schema/blueprint"
       xmlns:tx="http://www.springframework.org/schema/tx"
       xsi:schemaLocation="http://www.springframework.org/schema/beans http://www.springframework.org
        http://www.eclipse.org/gemini/blueprint/schema/blueprint http://www.eclipse.org/gemini/bluepr
        http://www.springframework.org/schema/tx http://www.springframework.org/schema/tx/spring-tx-3

    <tx:annotation-driven transaction-manager="transactionManager"/>

    <osgi:reference id="transactionManager" interface="org.springframework.transaction.PlatformTransa

</beans>
```

Note: Setting the context-class-loader to unmanaged will prevent switching the context classlaoder to the incorrect one, since the platform transaction manager is treated as an OSGi proxy itself. This issue can manifest in bundle ITs, where the wrong context classloader can be used, leading to errors about missing metadata.

Thanks to this configuration, Spring transaction annotations should work properly in your module, take note however that you might be required to explicitly import the following packages (example of the bundle plugin configuration):

```
<Import-Package>
     net.sf.cglib.core,
     net.sf.cglib.proxy,
     net.sf.cglib.reflect,
     org.aopalliance.aop,
     org.springframework.aop,
     org.springframework.aop.framework,
     org.springframework.transaction,
     *
 </Import-Package>
```

After this you can simply use the @Transactional annotation to mark your methods as transactions. Make sure you are

using the correct @Transactional annotation (org.springframework.transaction.annotation.Transactional). Example of a bean using the annotation:

```
@Component
public class TransactionTestBean {

    @Autowired
    private BookDataService bookDataService;

    @Transactional
    public void addTwoBooks() {
        bookDataService.create(new Book("Book1"));
        bookDataService.create(new Book("Book2"));
    }

    @Transactional
    public void addTwoBooksAndRollback() {
        addTwoBooks();
        // throwing a runtime exception rolls back the entire transaction
        throw new IllegalStateException("Rollback the transaction");
    }
}
```

More information on Spring transactions can be found here: http://docs.spring.io/spring/docs/current/spring-framework-reference/html/transaction.html

Note: Take note that these annotations will work only with Spring beans.

2.3.18 Security

Access to the Data Services module

MDS registers three permissions, that restrict access to certain parts of the Data Services module via MOTECH UI. They are:

- mdsSchemaAccess (grants access to the Schema Editor)
- mdsDataAccess (grants access to the Data Browser)
- mdsSettingsAccess (grants access to the Settings panel)

The **MDS Admin** role contains all of these three permissions.

Access to the instances

Depending on the chosen option, two security levels can be recognised in MDS:

Security level	Description
Instance	Defines access to certain instances of an entity. Only permitted users will be able to see the instance and perform any CRUD operations on it.
Non-instance	Defines access to all the instances of an entity. Only permitted users will be able to see the link to the instances table and perform CRUD operations on them.

Security settings can be set through the UI or by the **@org.motechproject.mds.annotations.Access** annotation for DDE. It works only with the **@Entity** annotation.

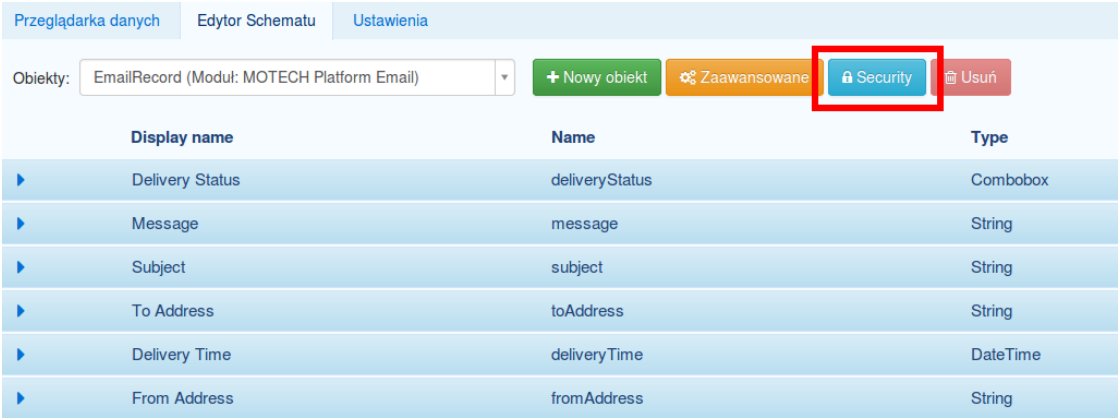
There are five security modes:

Option	Security level	Description
EVERYONE	None	The access to the instances is not limited in any way.
OWNER	Instance	Only the user that has been selected as an owner of the instance has got access. An owner can be selected while adding/editing instance.
CREATOR	Instance	Only the user that has created the instance has got access and can perform CRUD operations on it.
USERS	Non-instance	An additional input field will appear, where a list of permitted users should be placed. Permitted users will be able to view and perform CRUD operations on all instances of an entity.
ROLES	Non-instance	Similar to Users - an additional input field will appear, where a list of roles should be placed. Users that have got any of the permitted roles, will be able to view and perform CRUD operations on all instances of an entity.

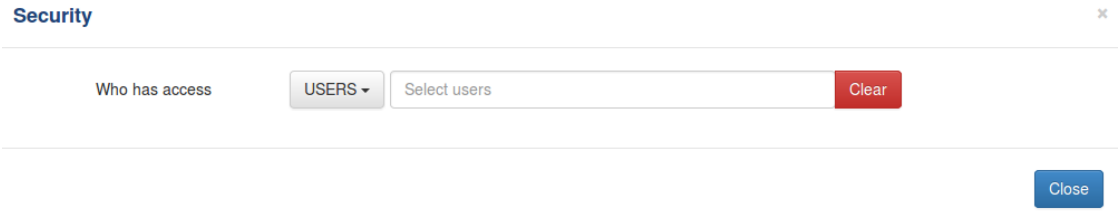
The code below shows an example usage of the annotation:

```
@Entity
@Access(value = SecurityMode.ROLES, members = {"admin"})
public class MyEntity { }
```

To update security settings via UI, pick the entity and click the **Security** button.



A new modal window will appear, where security settings can be updated.



Note: The security settings are applied to all means of access to the instances. It does not matter if an access is attempted via UI, through the code or REST - the necessary permissions will always be checked. This also means that it is possible to disallow the application itself to access the instances, so be careful when restricting access to the MOTECH entities.

2.3.19 CRUD Events

By default, MDS sends CRUD events after a Create/Update/Delete operation is completed, which can be optionally disabled through the UI or by the **@org.motechproject.mds.annotations.CrudEvents** annotation for DDE. It works only with the **@Entity** annotation.

The annotation has five options:

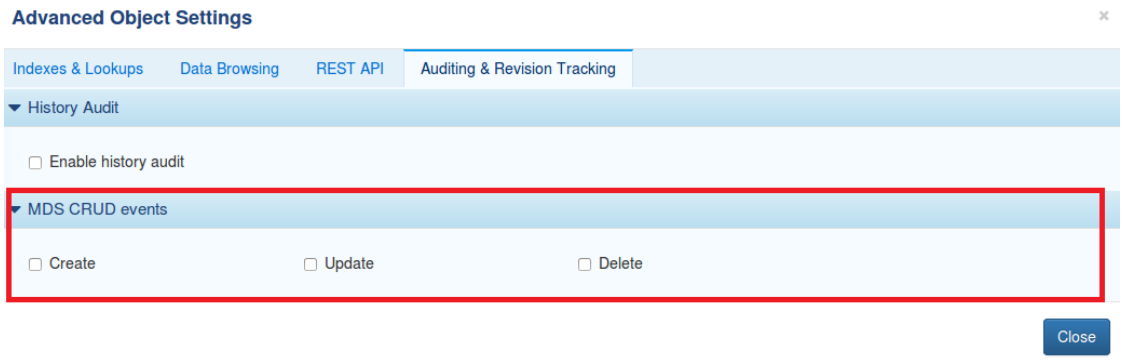
Option	Description
CREATE	Enable MDS to send events during creating instances of an entity.
UPDATE	Enable MDS to send events during updating instances of an entity
DELETE	Enable MDS to send events during deleting instances of an entity
ALL	Enable MDS to send events during creating, updating and deleting instances of an entity
NONE	None of the CRUD events will be sent by MDS

The code below shows an example usage of the annotation:

```
@Entity
@CrudEvents(CrudEventType.CREATE)
public class MyEntity {

    @Field
    private String message;
}
```

Note: Of course you can mix options (for example using CREATE and UPDATE).

To turn off sending events for an EUDE you have to disable the feature in the Advanced settings, 'Auditing & Revision Tracking' section. You can also do the same for a DDE. After changes are made, a flag modifiedByUser will be set to true, which means for a DDE, that the crud event settings will not be reloaded from the annotation upon restart.



The subject of MDS CRUD events takes the form of "mds.crud.<module name>.<namespace>.<entity name>.<action i.e. UPDATE|DELETE|CREATE>". The event payload contains 5 parameters:

- object_id - the ID of the object this event refers to
- entity_name - the name of the entity
- entity_class - the fully qualified class name of the entity
- module_name - the name of the module from which the entity comes from (optional)
- namespace - the namespace of the entity (optional)

A separate event is also fired once a CSV import is completed. The subject of the event is similar to a regular CRUD event and takes the form of "mds.crud.<module name>.<namespace>.<entity name>.csv-import.<success|failure>".

The payload for a CSV import success event contains the following parameters:

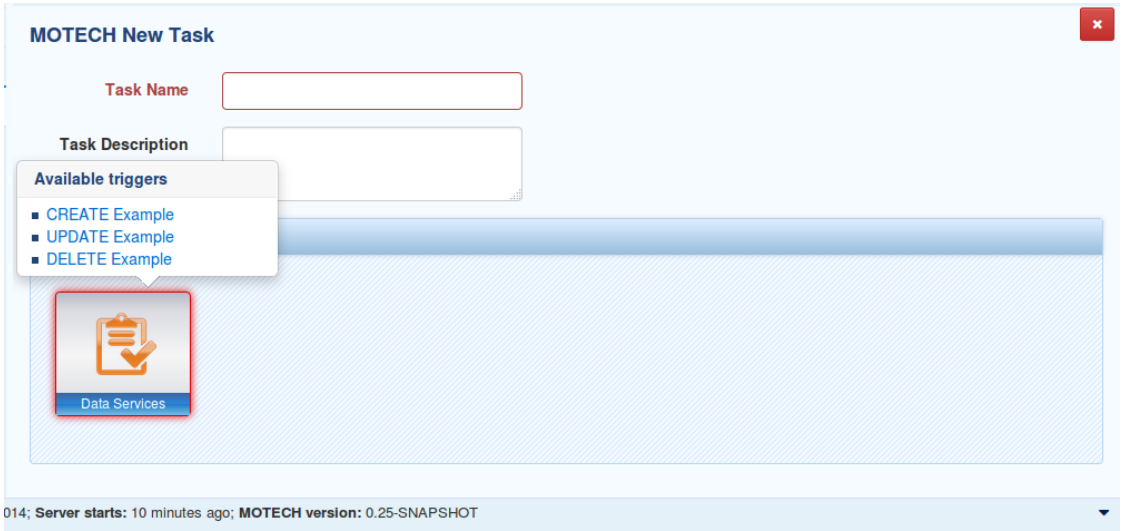
- entity_name - the name of the entity for which this import was performed
- entity_class - the fully qualified class name of the entity for which this import was performed
- module_name - the name of the module from which the entity comes from (optional)
- namespace - the namespace of the entity for which this import was performed (optional)
- csv-import.filename - the name of the imported file
- csv-import.created_ids - a list of IDs for instances newly created during import
- csv-import.updated_ids - a list of IDs for instances updated during import
- csv-import.created_count - the count of instances newly created during import
- csv-import.updated_count - the count of instances updated during import
- csv-import.total_count - total count of instances created/updated by this import(sum of the created count and updated count)

The payload for the import failure event is different:

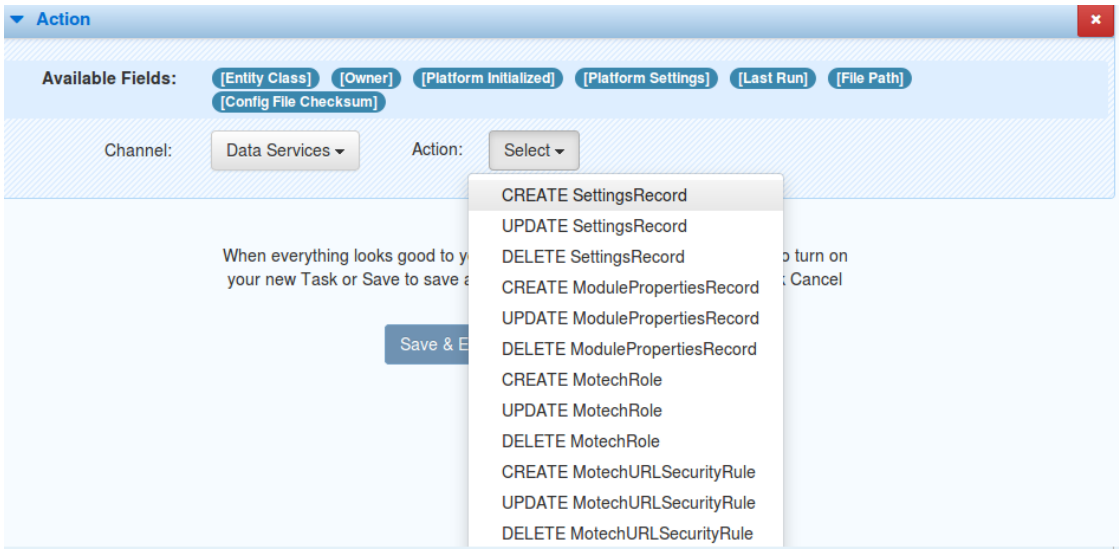
- entity_name - the name of the entity for which this import was performed
- entity_class - the fully qualified class name of the entity for which this import was performed
- module_name - the name of the module from which the entity comes from (optional)
- namespace - the namespace of the entity for which this import was performed (optional)
- csv-import.filename - the name of the imported file
- csv-import.failure_message - the message from the exception that caused the failure
- csv-import.failure_stacktrace - the stacktrace of the exception that caused the failure(as String)

Tasks integration

For the entities that expose these events, you can create tasks with these events as a trigger. To do it go to the Task module, click 'New task' and you should see the Data Services trigger list. A trigger is exposed for every crud event per entity:



In the Task module, you can also use Data Services as a channel and select an action you want :



2.3.20 Instance Lifecycle Listeners

In MDS you can register listeners for persistence events. You can provide listener to receive events for CREATE, DELETE, LOAD, and STORE of objects. To do this you have to use the **@org.motechproject.mds.annotations.InstanceLifecycleListener** annotation on service methods.

The annotation value is an array of one or more values :

Option	Description
POST_CREATE	Invoked after an instance is made persistent.
PRE_DELETE	Invoked before a persistent instance is deleted. Access to field values within this call are permitted.
POST_DELETE	Invoked after a persistent instance is deleted. This method is called after the instance transitions to persistent-deleted. Access to field values is not permitted.
POST_LOAD	Invoked after a persistent instance is loaded from the data store.
PRE_STORE	Invoked before a persistent instance is stored, for example during committing a transaction.
POST_STORE	Invoked after a persistent instance is stored. It is called after the field values have been stored.

Note: The listener is called within the same transaction as the operation being reported and so any changes they then make to the objects in question will be reflected in that objects state. Throwing a RuntimeException from a listener will fail the transaction.

The code below shows an example usage of the annotation:

```
public interface MyService {

    @InstanceLifecycleListener(InstanceLifecycleListenerType.POST_CREATE)
    void changeSubject(EmailRecord emailRecord);

    @InstanceLifecycleListener(InstanceLifecycleListenerType.POST_STORE, packageName = "org.motechpro
    void entityChanged(Object o);
}

@Service("myService")
public class MyServiceImpl implements MyService {

    public void changeSubject(EmailRecord emailRecord) {
        emailRecord.setSubject("newSubject");
    }
```

```
    public void entityChanged(Object o) {
        // process the entity
    }
}

<?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:osgi="http://www.eclipse.org/gemini/blueprint/schema/blueprint"
    xsi:schemaLocation="http://www.springframework.org/schema/beans
        http://www.springframework.org/schema/beans/spring-beans.xsd
        http://www.eclipse.org/gemini/blueprint/schema/blueprint
        http://www.eclipse.org/gemini/blueprint/schema/blueprint/gemini-blueprint.xsd">

    <osgi:service ref="myService" interface="org.motechproject.example.MyService"/>

</beans>
```

Note: If you want you can mix options (for example using POST_CREATE and POST_STORE).

You have to remember about the following when using InstanceLifecycleListeners :

- Methods annotated with **@org.motechproject.mds.annotations.InstanceLifecycleListener** must be in services exposed by OSGi
- Methods must have exactly one parameter and its type must be either a persistable class or java.lang.Object if the package is specified.
- You can annotate multiple methods for one type of event

The annotated method is a listener for class defined in the parameter type (in our example for EmailRecord).

2.3.21 Entities Migrations

In MDS you can use flyway migrations. These migrations will run after entities schema generation. MOTECH will automatically copy migration files from installed modules to the `.motech` directory. Files should be placed in `db/migration/default` directory(if you are using mysql then use `mysql` instead `default`) in the bundle. Each file muse have *a proper name <http://flywaydb.org/documentation/migration/sql.html>'_(e.g. :code:'V1__Description.sql*).

2.3.22 Javadoc

org.motechproject.mds.service

org.motechproject.mds.annotations

org.motechproject.mds.builder

org.motechproject.mds.config

org.motechproject.mds.domain

org.motechproject.mds.dto

org.motechproject.mds.enhancer

org.motechproject.mds.ex

org.motechproject.mds.filter

org.motechproject.mds.jdo

org.motechproject.mds.repository

org.motechproject.mds.util

`/org/motechproject/mds/web/package-index`

2.4 Messaging in MOTECH Overview

Table of Contents

2.4.1 Introduction

Messaging in MOTECH is independent from the scheduling or message campaigns that fire using the event system. It is up to the implementer to connect that (or any other for that matter) logic messaging, either by writing code or *creating tasks*. This document is a short overview of the messaging options provided by MOTECH: email, SMS and IVR. Each of these has its own module, which can be used by implementers for sending messages through different channels.

2.4.2 The Email Module

MOTECH provides an Email module which allows to easily send email messages after connecting to an SMTP server. This module is a part of the platform, so it will be always available. The module is capable of parsing Velocity templates into HTML that will be sent as messages. More information on configuring and using the Email module can be found in the `Email module documentation`.

2.4.3 The SMS Module

SMS messaging can be utilised thanks to the SMS module. This module is optional, so you have to install it in your MOTECH instance if you wish to use it. After connecting to an SMS provider, the module allows both sending and receiving SMS messages. Sending messages is done through HTTP requests to the provider, receiving messages is similarly done by listening for HTTP requests from the provider. MOTECH provides a predefined list of configuration templates for different providers, it also allows you to use your own custom templates. More information on configuring and using the SMS the module can be found in the *SMS module documentation*.

2.4.4 The IVR module

MOTECH also provides a module that allows making and receiving calls by integrating with an IVR provider. This module is optional, so you have to install it in your MOTECH instance if you wish to use it. Similarly to the SMS module, the IVR also relies on making and receiving HTTP calls from the IVR system. The module allows you to

upload Velocity templates, that will be processed and served to the provider. These are generally VoiceXML or a similar format, but the module can operate on any format the provider supports. More information on configuring and using the IVR module can be found in the *IVR module documentation*.

2.5 Connecting MOTECH to OpenMRS

Table of Contents

2.5.1 Introduction

MOTECH allows you to integrate with OpenMRS - an open source enterprise electronic medical record system platform. This is done using the *OpenMRS-19 module*, that communicates with OpenMRS through its REST API and exposes OSGi services which allow your implementation to easily integrate with OpenMRS. Refer to the module documentation for information on using the API it exposes, this document will describe setting up integration between MOTECH and OpenMRS.

You can easily configure the OpenMRS-19 module to integrate with MOTECH, and then use its API to retrieve or manipulate data in OpenMRS. MOTECH uses the REST API exposed by OpenMRS for integration and it will require HTTP access to OpenMRS and an account with rights to the operations you wish to perform.

Take note that the module was written with OpenMRS 1.9 in mind, and we will use that version for reference in this tutorial.

2.5.2 Getting OpenMRS

If you plan on hosting an OpenMRS instance yourself, you can download it from the OpenMRS website. Note that the MOTECH module was created and tested to work with version 1.9. You might encounter problems if choose to use different OpenMRS versions. Use the links below to get OpenMRS:

- OpenMRS website: http://openmrs.org
- OpenMRS 1.9.7 download: http://sourceforge.net/projects/openmrs/files/releases/OpenMRS_1.9.7/

Refer to the OpenMRS documentation for installation instructions: https://wiki.openmrs.org/display/docs/Installing+OpenMRS

Here is the simplest possible route of installing OpenMRS:

1. Install Tomcat
2. Install MySQL
3. In MySQL, create a database called **openmrs**
4. Place the OpenMRS war in the Tomcat webapps directory, making sure its name is **openmrs.war**
5. Start Tomcat
6. Go to http://localhost:8080/openmrs

7. Follow the installation wizard instructions

Again, refer to OpenMRS documentation for more details.

2.5.3 Configuring the OpenMRS-19 module

The MOTECH OpenMRS-19 module exposes a configuration file called openmrs.properties. This file is registered with the configuration system. Based on what configuration mode the system is configured with, you can change the settings either by using the Admin UI(UI Mode) or the file in the config file location(File Mode). The default username and password match the defaults from OpenMRS(make sure that you change these in production environment).

The table below describes the properties declared in the file and their default values, that will work with a default localhost OpenMRS installation. The openmrs.motechIdName setting needs to match an identifier type from OpenMRS. More information on creating the identifier type in OpenMRS can be found in the *next section*

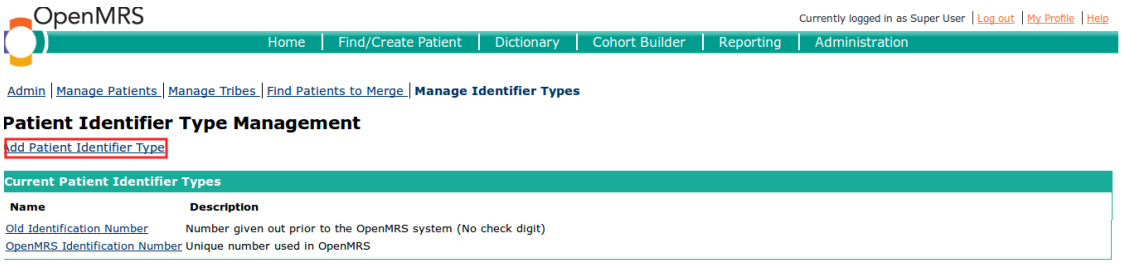
Key	Description	Default Value
openmrs.url	The top level url at which the OpenMRS Instance is accessible. Required since MOTECH integrates with OpenMRS through REST API calls.	*http://localhost:8080/openmrs*
openmrs.user	The OpenMRS username that MOTECH will use to identify with OpenMRS.	admin
openmrs.password	The OpenMRS user password that MOTECH will use to identify with OpenMRS.	Admin123
openmrs.motechIdName	The name of the OpenMRS identifier used by MOTECH. This must match the identifier that you will create in OpenMRS.	MOTECH Id

Note: The module must be restarted in order for configuration changes to take effect.

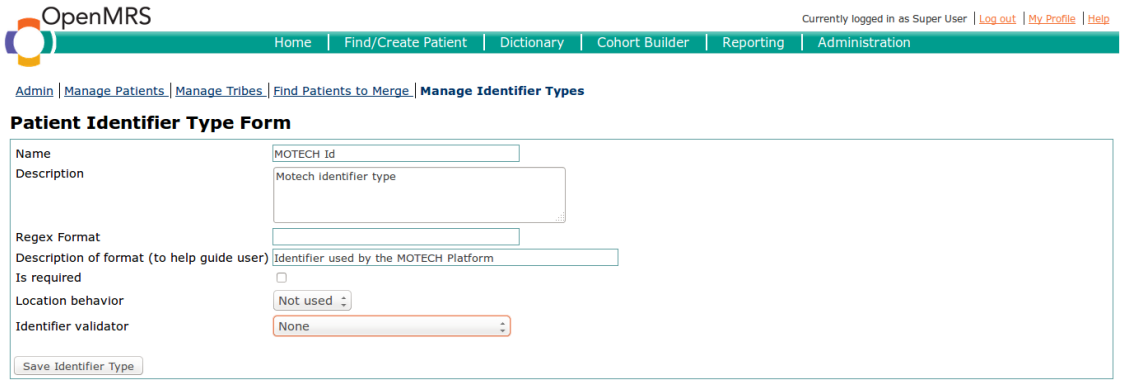
2.5.4 Creating a MOTECH Identifier in OpenMRS 1.9

In order to make the module work with OpenMRS, an identifier type that MOTECH will use for identifying patients must be created. The name of that identified must match the value of the configuration variable **openmrs.motechIdName**. In order to define the ID type, go to the Administration section of OpenMRS, then select **Manage Identifier Types** under the section **Patients**:

Next, select **Add Patient Identifier Type**:

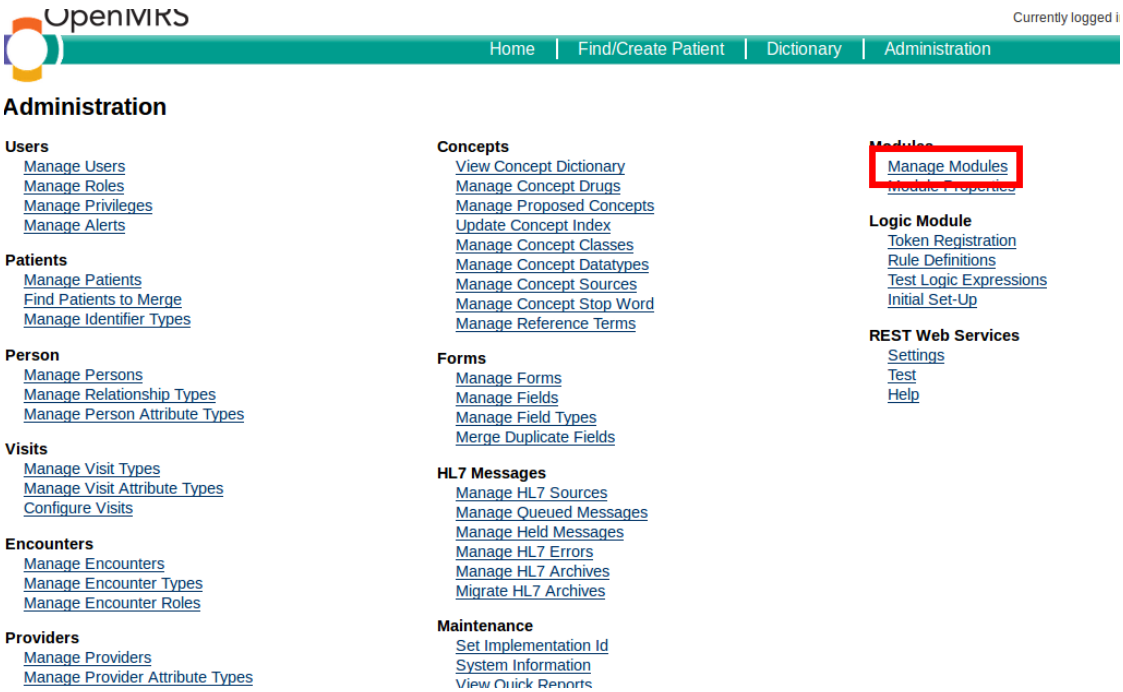


Finally, enter the details of the identifier type. The name must match the one in the **openmrs.motechIdName** setting variable. You can specify the settings as you wish, note that for example making locations required or adding a regex format for the identifier will restrict what values you can use. Refer to the OpenMRS documentation for more information.

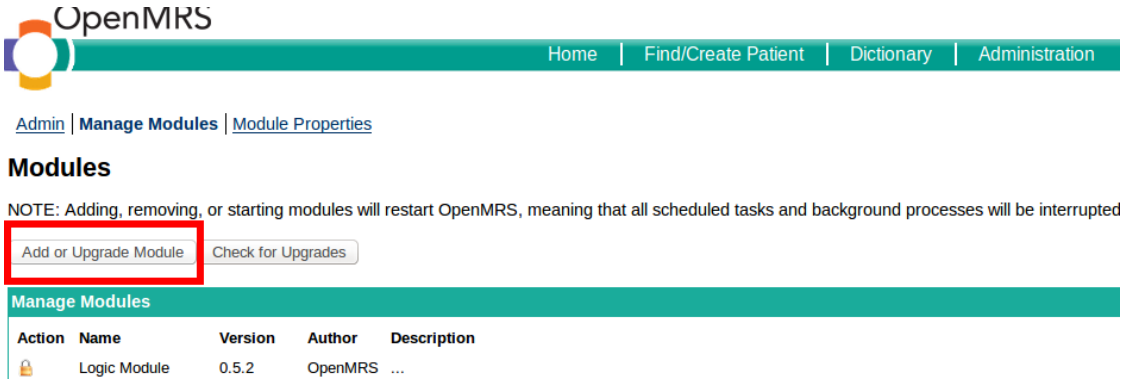


2.5.5 Installing Rest Web Services module in OpenMRS

MOTECH communicates with the OpenMRS via REST, which means that the OpenMRS instance must have a Rest Web Services module installed and activated. You can find the required module on the OpenMRS modules website. Pick the latest released version (2.9+) and download it. You can install the module using OpenMRS UI. Go to Administration tab, and select Manage Modules.



You will see the Add or Upgrade Module button. Click it, then select the downloaded file under "Add module" and upload it. The module will be installed and started. You can verify its status in the Manage Modules section.



You should now be able to use the OpenMRS-19 module. Refer to the module *documentation* for usage instructions.

2.6 Integrating MOTECH with CommCare

Table of Contents

2.6.1 Introduction

The Commcare module provides an ability to integrate MOTECH with CommCareHQ. At the moment, the Commcare module allows the following:

- View forms and cases, using MOTECH UI
- Receive notifications when a new form or case is received or when the application schema changes
- Use the exposed OSGi services, to query CommCareHQ for forms, cases, users and fixtures and upload certain data to CommCareHQ, like cases and data forwarding rules
- Use the data from CommCareHQ to model more advanced logic, using the Tasks module

Please note, that throughout this document, two similar expressions will be used:

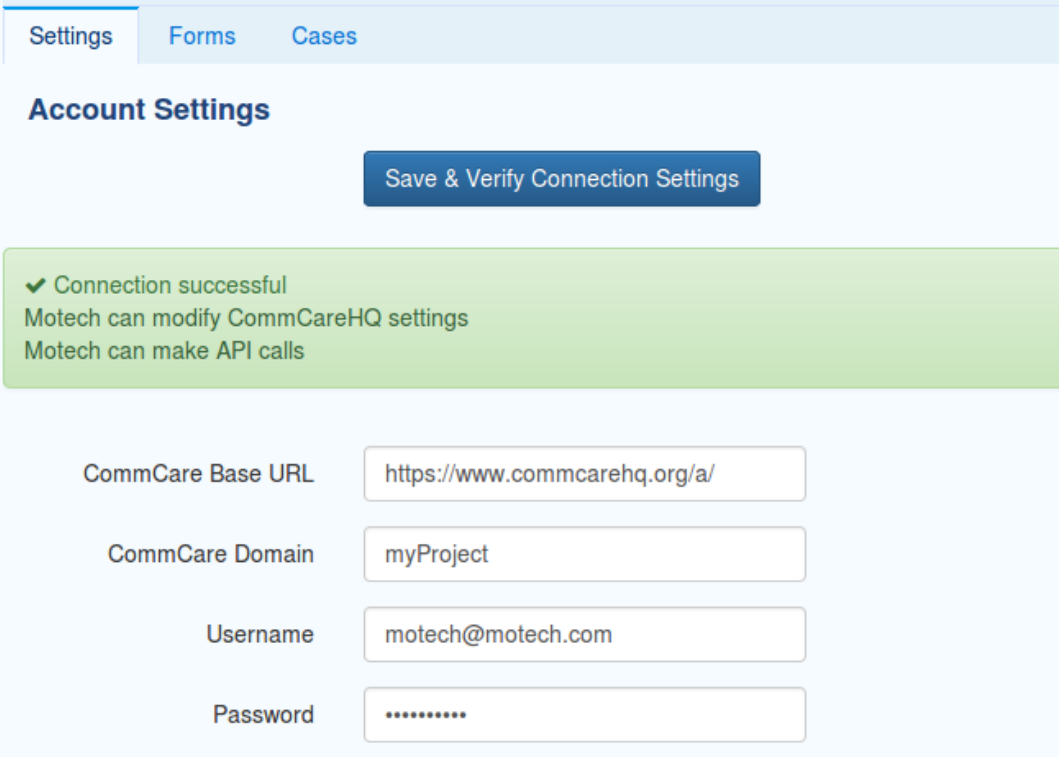
- **Commcare** - refers to the MOTECH module, that allows the integration with CommCareHQ
- **CommCareHQ** - refers to an external service, located under www.commcarehq.org

2.6.2 Configure Commcare module

Account settings

To allow MOTECH to connect to the CommCareHQ, you should first provide correct credentials to the CommCareHQ account:

- CommCare Base URL (a link to the CommCareHQ instance; by default www.commcarehq.org)
- CommCare Domain (project name on CommCareHQ)
- Username
- Password



To verify the provided data, click the **Verify** button, at the top of the page. Commcare module will send a test request to the CommCareHQ, attempting to authenticate with the credentials you have provided. If everything works OK, you will be notified about successful connection. If there were any problems connecting to the CommCareHQ, an error will be displayed and you will not be able to work with the Commcare module, until valid credentials are provided.

Event forwarding

This section allows you to configure the events, fired by the Commcare module. Currently, you can pick the **Event Strategy** for the forwarding of case events. There are three options available:

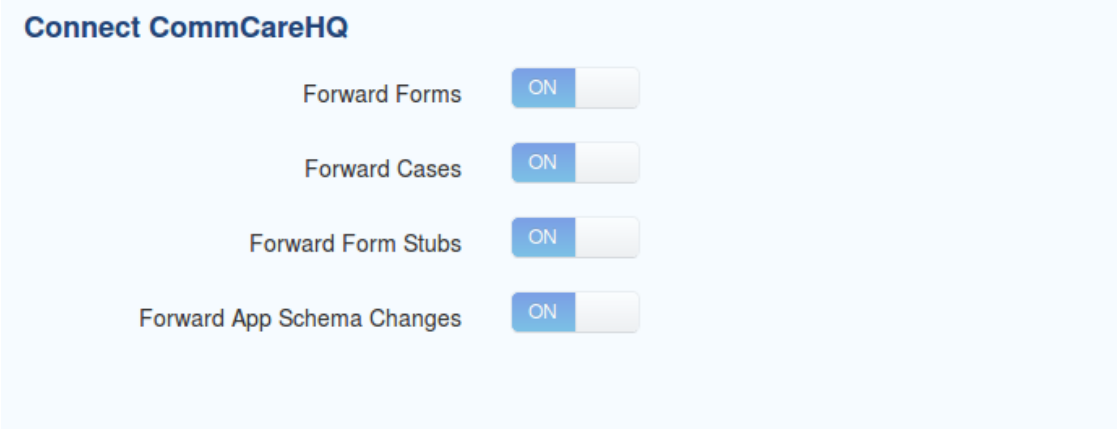
- minimal (the event will contain only the case ID)
- partial (the event will contain case ID, as well as other, not-case specific parameters (case metadata)
- full (the event will contain case ID, case metadata and all field values of a case)

Once you switch the option, the events fired by the Commcare module will contain only the fields you have chosen to forward.

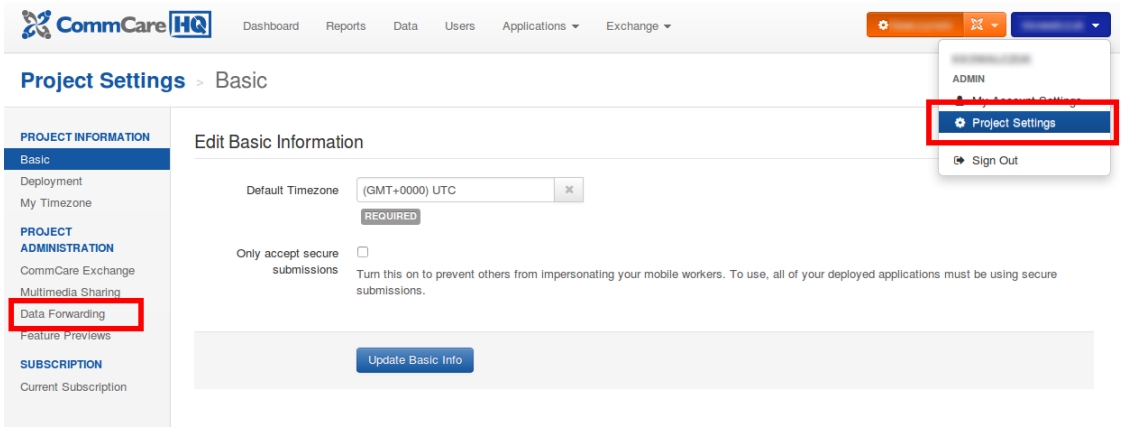
Note: When you switch to a more strict strategy (forwarding less details), make sure that no listeners rely on the content of these events (eg. Task, triggered by "Received case")

Connect CommCareHQ

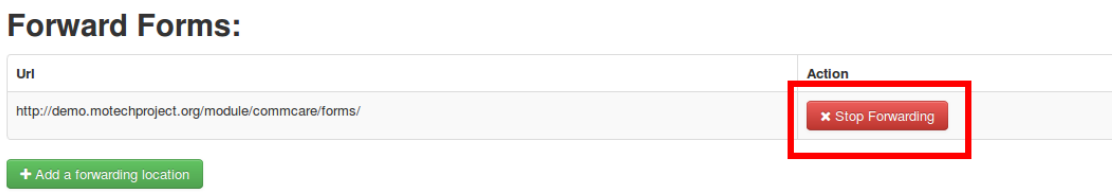
To let CommCareHQ know, where the data should be forwarded, you also need to set up data forwarding URLs. This can be achieved in two ways. The first way is to do it via the provided slider buttons. If you want to set up CommCareHQ to forward the data to the Commcare module, simply slide the button to **ON** and Commcare module will automatically set up an URL on your CommCareHQ account.



If for any reason, the first way doesn't work or sets up invalid URL, the data forwarding rules can also be set in the project settings on your CommCareHQ account. If you have enabled the rules via the slider buttons, you can also verify that the correct rules (with proper URL to your server) have been set up on the CommCareHQ.



To disable the data forwarding rules, you have to open the project settings on your CommCareHQ account and disable them from there. Commcare module can only set up the rules, but cannot disable them.



2.6.3 Fired events

Subject	Info
org.motechproject.commcare.api.schemaChange	Fired, when the project schema gets changed on the CommCareHQ (module added, form edited, etc.).
org.motechproject.commcare.api.forms	Fired, when a new form has been received on CommCareHQ. One event per received form.
org.motechproject.commcare.api.case	Fired, when a new form has been received on CommCareHQ. One event will be fired per affected case.
org.motechproject.commcare.api.formstub	Fired, when a new form has been received on CommCareHQ. Contains only IDs of affected form and cases.

There are three more events, that are fired, when an internal exception occurs while parsing XML file, received from CommCareHQ. They are:

- **org.motechproject.commcare.api.forms.failed** (when parsing of a form fails)
- **org.motechproject.commcare.api.formstub.failed** (when parsing of a form stub fails)
- **org.motechproject.commcare.api.exception** (when parsing of a case fails)

2.6.4 Integration with the Tasks module

The Commcare module will automatically update the Tasks triggers and data sources, each time a schema change event is received. For each form and for each case type, a separate trigger and data source object will be created. This means that you can trigger tasks, when a certain form or case is received and use its fields in an action you select. The fields of forms and cases are based on the schema received from CommCareHQ.

2.7 Using the Tasks Module

Table of Contents

2.7.1 Introduction

The Tasks module, as its name suggests, provides an ability to define and execute tasks. In MOTECH world, task is a piece of work (called action) that has to be performed in response to some event (trigger). In other words, the module provides a tool for defining simple logic that is ready to use without writing a single line of code.

The main features of the Tasks module include:

- Creating, managing and executing tasks
- Monitoring tasks execution
- Registering custom triggers, actions and data providers

2.7.2 Basic concepts

Channels

A channel contains information about all exposed triggers and actions within a given module. It can be considered as a module specific configuration that tells the Tasks module how it can make a use of it.

The most important elements of the channel are trigger and action definitions. In fact, if channel does not define neither triggers nor actions it is considered invalid. Other properties includes the channel display name, that will be visible on the Tasks UI (it may be a key from message.properties, in which case it will appear as a translated message). Module version and name are obtained at channel registration from the registering bundle. Additionally the channel may contain a short description.

Detailed definition:

Field	Attributes	Description
displayName	required	A channel name that will be displayed on the UI (may be an i18n key)
moduleName	required, derived	A name of the module that registered the channel. By default derived from the bundle information
moduleVersion	required, derived	A version of the module that registered the channel. By default derived from the bundle information
description	optional	A brief channel description that will be displayed on the UI (may be an i18n key)
triggerTaskEvents	optional	An array of *triggers* definitions
actionTaskEvents	optional	An array of *actions* definitions

Triggers

A trigger represents a precise definition of events exposed by module. In tasks, a trigger is something that, as the name suggests, triggers task executions. This means that when an event described by the trigger is published, all tasks with that trigger will get executed. Every trigger has an unique name and, in a simple case, corresponds to exactly one event. Parameters of this event are defined within the trigger. Each parameter contains its name (key) and *type*.

A good example of a trigger can be an inbound SMS. It would contain the following parameters: message (STRING), sender (STRING), recipients (LIST) etc. Those information will be accessible in the Tasks module.

In the basic case, the most important elements of a trigger are subject and event parameters. The subject corresponds to the event subject that is wrapped by this trigger, while event parameters are the parameters that will be exposed by the trigger. Providing this basic kind of the trigger makes Tasks module listen to the event with the given subject. Each time such an event is published, all active tasks with a corresponding trigger are executed.

However, in some cases the basic behaviour is not sufficient. Sometimes we want the event to correspond to many triggers. In this situation, the trigger listener subject comes in handy. It has to be used along with a *custom event parser*, which is a little more advanced component, thus it will be described later.

Detailed trigger definition:

Field	Attributes	Description
displayName	required	Trigger name that will be displayed on the UI (may be an i18n key)
subject	required	Trigger subject that will be delivered to the task
triggerListenerSubject	optional	Real event subject that is wrapped by this trigger. In a simple case it is identical to the subject above, so it can be omitted.
description	optional	A brief trigger description that will be displayed on the UI (may be an i18n key)
eventParameters	optional	An array of event parameters described below

Detailed trigger event parameter definition:

Field	Attributes	Description
displayName	required	Event parameter name that will be displayed on the UI (may be an i18n key)
eventKey	required, unique	Event parameter key. The event parameter value will be obtained from delivered event using this key
type	optional	*Type* of the delivered event parameter. Default is UNICODE

Actions

An action represents a definition of function that can be called in a response to a trigger. Every action can represent either a single method of an OSGi service that will be called or an event that will be sent. Each parameter contains it name (key), *type* and may contain its default value. In case of a method call, the way in which parameters will be passed may vary depending on the needs. They can be either passed directly to the method (matching its signature) or using a key-value pair map.

For instance, an action may correspond to sending an email message. That action would then contain some required fields such as recipients (as a LIST) and the message (STRING) and some optional fields, for example the delivery time (DATE).

As mentioned before, there are two forms in which an action can be represented. The first one is an event. In this case, the action must define a subject of that event. Action execution leads to creating an event with the defined subject and parameters that correspond to the exposed action parameters. The second form that action can take is a service method call. In that case, the action definition must contain the name of the OSGi exposed service interface and the method name to execute. Additionally, one can specify the way in which the method will be called. When it is specified as 'named parameters', the action parameters will be evaluated, casted and passed directly to the service method according to its signature and matching its parameter names. In the other case, when it is specified as 'map', the parameters are evaluated, packed into a hash map and passed to the method. In this situation the service method is supposed to take exactly one parameter of type java.util.Map<java.lang.String, java.lang.Object>.

An action is considered invalid if it does not define the method nor the event. However, it can define both of them, but the method call has the precedence before event passing. Thus event is send only if the method defining service is not available.

Detailed action definition:

Field	Attributes	Description
name	optional, unique	Action name
displayName	required	Action name that will be displayed on the UI (may be an i18n key)
description	optional	A brief action description that will be displayed on the UI (may be an i18n key)
subject	optional event-required	A subject of the event that is to be sent
serviceInterface	optional, method-required	A service containing a method that is to be called
serviceMethod	optional method-required	A service method that is to be called
serviceMethodCallManner	optional method-optional	A service method call manner. It can take one of two values: NAMED_PARAMETERS (default) - action parameters are passed to the function directly, matching its signature; MAP - action parameters are passed to the method as a map in which keys correspond to parameter names and values correspond to parameter values
actionParameters	optional	An array of action parameter definitions described below

Detailed action parameter definition:

Field	Attributes	Description
displayName	required	Action parameter name that will be displayed on the UI (may be an i18n key)
key	required, unique	Action parameter key. Depending on action method call manner it will correspond either to a method parameter name or a map key
value	optional	Action parameter default value. Depending on action method call manner it will correspond either to a method parameter value or a map value
type	optional	*Type* of the action parameter value. Default is UNICODE
required	optional	Indicates if this action parameter is mandatory. May be true or false. Default is false
hidden	optional	Indicates if this action parameter should not be visible on the UI. May be true or false. Default is false
order	optional	Specifies position at which this action parameter should appear among other parameters

Parameters types

Available types that can be used with action parameters and trigger event parameters in the Tasks module are listed below.

Type Name	Java Type	Description
UNICODE	java.lang.String	Short Unicode string
TEXTAREA	java.lang.String	Long Unicode string
INTEGER	java.lang.Integer	Signed number without a fraction component
LONG	java.lang.Long	Large signed number without a fraction component
DOUBLE	java.lang.Double	Double precision floating point number
DATE	org.joda.time.DateTime	Calendar date with time
TIME	org.joda.time.DateTime	Calendar time without date
PERIOD	org.joda.time.Period	Period of time
BOOLEAN	java.lang.Boolean	True or false
LIST	java.util.List	Collection of values
MAP	java.util.Map	Collection of key-value pairs

Data providers

A data provider can be considered as a source of various data that can be used in a task. It defines the structure of objects it supports as well as structure of the queries that it can perform. Each data provider is recognized by its name.

An example data provider is the one defined by the CMSLite module. It provides two types of objects: StreamContent and StringContent. For instance, StringContent objects contains several fields that can be used in the Tasks module. Those are value, language, name and metadata. It also contains two lookups. One of them is used to find a desired instance by id, the other one uses name and language fields.

Every data provider must implement the DataProvider interface. It contains a few methods responsible for retrieving the data provider name, performing a search, discriminating if a provided type is supported by this provider and finally, returning the provider JSON definition. The definition is in fact a TaskDataProvider object, thus it must follow its schema.

Detailed task data provider definition:

Field	Attributes	Description
name	required, unique	An unique data provider name
objects	required	An array of task data provider objects

Detailed task data provider object definition:

Field	Attributes	Description
displayName	required	Task data provider object name that will be displayed on the UI (may be an i18n key)
type	required, unique	The symbolic type name of the object backed by this task data provider object. As it will be used to distinguish this object from other objects within this data provider, it has to be unique
lookupFields	required	An array of lookup field parameters definitions used to in the lookup
fields	required	An array of fields parameters definitions available from this task data provider object

Detailed lookup field parameter definition:

Field	Attributes	Description
displayName	required	Lookup field parameter name that will be displayed on the UI (may be an i18n key)
fields	required	An array of field names required by this lookup

Detailed field parameter definition:

Field	Attributes	Description
displayName	required	Field parameter name that will be displayed on the UI (may be an i18n key)
fieldKey	required	A key used to identify this parameter
type	optional	*Type* of the field parameter value. Default is UNICODE

2.7.3 Channel registration

To expose a module actions or triggers in Tasks module, a channel containing their definitions has to be registered in the Tasks module. It can be done in one of three different ways: using a static channel definition file, task annotations or programmatically, utilizing the ChannelService.

Using the channel file

It is the most common way to register a task channel. It comes down to creating a json channel definition file named task-channel.json and placing it right in the classpath root of your bundle. It will be automatically discovered by the Tasks module at your bundle start or update.

The file content, written in JSON format, has to follow a well defined structure. The root element must be a channel object that matches a *specification* defined above.

Example channel file:

```
{
    "displayName": "sms",
    "triggerTaskEvents": [
        {
            "displayName": "sms.inbound_sms",
            "subject": "inbound_sms",
            "serviceInterface": "org.project.service.SmsService",
            "serviceMethod": "sendSms",
            "eventParameters": [
                {
                    "displayName": "sms.message",
                    "eventKey": "message"
                },
                {
                    "displayName": "sms.sender",
```

```
                    "eventKey": "sender"
                },
                {
                    "displayName": "sms.recipient",
                    "eventKey": "recipient"
                },
                {
                    "displayName": "sms.datetime",
                    "eventKey": "datetime",
                    "type": "DATE"
                }
            ]
        }
    ],
    "actionTaskEvents": [
        {
            "displayName" : "sms.send_sms",
            "subject" : "send_sms",
            "actionParameters" : [
                {
                    "displayName" : "sms.message",
                    "key" : "message"
                },
                {
                    "displayName" : "sms.recipients",
                    "key" : "recipients",
                    "type" : "LIST"
                },
                {
                    "displayName" : "sms.delivery_time",
                    "key" : "delivery_time",
                    "type" : "DATE",
                    "required": false
                }
            ]
        }
    ]
}
```

Note: The order of the elements in the action parameters array determines their order on the Tasks UI, unless an order parameter is specified.

Using annotations

This method allows to register a channel using the Tasks annotation processing mechanism and annotations from org.motechproject.tasks.annotations package. However, this approach is limited to registering actions only. In this scenario, channels correspond to classes and actions to their methods. To make a class recognized as a channel by the Tasks module, it has to be annotated with @TaskChannel. Channel display name can be provided as an annotation parameter. Additionally, module name and version have to be provided as annotation parameters.

Each channel class should have at least one method marked as @TaskAction. For this annotation as well, one can specify the action display name. Each parameter of the action method is considered as an action parameter with default properties: the parameter are marked as required, its type is set to UNICODE and its display name and key corresponds to the action method parameter name. Those default properties can be modified utilising the @TaskActionParam annotation.

Example channel class:

```
@Service
@TaskChannel(channelName = "sms", moduleName = "sms", moduleVersion = "1.0")
public class SmsServiceImpl implements SmsService {

    @TaskAction
    public void sendSms(
        @TaskActionParam(displayName = "sms.message", key = "message") String message,
        @TaskActionParam(displayName = "sms.recipients", key = "recipients", type = ParameterType.LIS
        @TaskActionParam(displayName = "sms.delivery_time", key = "delivery_time", type = ParameterTy
        ) {

        ...
    }

}
```

Using the ChannelService

The most elastic way to register a channel is to use the ChannelService. It allows to register both triggers and actions in a dynamic manner. The first step of typical usage of the ChannelService is to build a ChannelRequest object. The ChannelRequest is the Java representation of a channel, that follows already defined *channel specification*. Accordingly, TriggerEventRequest and EventParameterRequest corresponds to *trigger and trigger event parameters* and ActionEventRequest and ActionParameterRequest corresponds to *action and action parameters*. Note that in this scenario module name and module version must be provided manually as proper fields of the request. Once the ChannelRequest is ready, it can be passed to the ChannelService method called registerChannel. It will validate the request and register the tasks channel.

Example channel registration using the ChannelService:

```
@Component
public class SmsChannelRegistration {

    @Autowired
    private ChannelService channelService;

    ...

    private void registerSmsChannel() {

        EventParameterRequest inboundSmsMessage = new EventParameterRequest(
            "message", // event key
            "sms.message" // display name
        );

        EventParameterRequest inboundSmsSender = new EventParameterRequest(
            "sender", // event key
            "sms.sender" // display name
        );

        EventParameterRequest inboundSmsRecipient = new EventParameterRequest(
            "recipient", // event key
            "sms.recipient" // display name
        );

        EventParameterRequest inboundSmsDatetime = new EventParameterRequest(
```

```
        "datetime", // event key
        "sms.datetime", // display name
        "DATE" // type
    );

    TriggerEventRequest inboundSmsTrigger = new TriggerEventRequest(
        "sms.inbound_sms", // display name
        "inbound_sms", // subject
        null, // description
        Arrays.asList(inboundSmsMessage, inboundSmsSender, inboundSmsRecipient, inboundSmsDatetir
    );

    ActionParameterRequest sendSmsMessage = new ActionParameterRequest(
        "message", // key
        null, // default value
        "sms.message", // display name
        0, // order
        null, // type (default: UNICODE)
        true, // required
        false // hidden
    );

    ActionParameterRequest sendSms = new ActionParameterRequest(
        "recipients", // key
        null, // default value
        "sms.recipients", // display name
        1, // order
        "LIST", // type (default: UNICODE)
        true, // required
        false // hidden
    );

    ActionParameterRequest sendSms = new ActionParameterRequest(
        "delivery_time", // key
        null, // default value
        "sms.delivery_time", // display name
        2, // order
        "DATE", // type (default: UNICODE)
        false, // required
        false // hidden
    );

    ActionEventRequest sendSmsAction = new ActionEventRequest(
        null, // name
        "sms.send_sms", // display name
        "send_sms", // subject
        null, // description
        "org.project.service.SmsService", // service interface
        "sendSms", // service method
        null, // service method call manner (default: NAMED_PARAMETERS)
        Arrays.asList(sendSmsMessage, sendSmsRecipients, sendSmsDeliveryTime) // action paramete
    );

    ChannelRequest smsChannel = new ChannelRequest(
        "sms", // display name
        "sms", // module name
        "1.0", // module version
        null, // description
```

```
            Arrays.asList(inboundSmsTrigger), // trigger requests
            Arrays.asList(sendSmsAction) // action requests
        );

        channelService.registerChannel(smsChannel);
    }
}
```

2.7.4 Data provider registration

To register a custom data provider, two things have to be done. As it was said before, every data provider has to implement the DataProvider interface. For your convenience we provide an abstract base class that implements the DataProvider interface and removes the requirement to write needles boilerplate. That class is called AbstractDataProvider and is extend by most of our data providers. Usually, this class is used along with a JSON data provider definition stored somewhere in the classpath. The only thing to do then is to provide the string or resource containing the JSON. Once the data provider is ready to use, it is time to actually register it in the Tasks module, which comes down to publishing it as an OSGi service.

Example data provider:

```
@Service
public class ExternalPatientDataProvider extents AbstractDataProvider {

    @Autowired
    public ExternalPatientDataProvider(ResourceLoader resourceLoader) {
        setBody(resourceLoader.getResource("task-data-provider.json"));
    }

    @Override
    public String getName() {
        return "external-patient";
    }

    @Override
    public Object lookup(String type, String lookupName, Map<String, String> lookupFields) {
        if (supports(type) && lookupFields.containsKey("id")) {
            String id = lookupFields.get("id");
            return getExternalPatient(id);
        } else {
            return null;
        }
    }

    @Override
    public List<Class<?>> getSupportClasses() {
        return Arrays.asList(ExternalPatient.class);
    }

    @Override
    public String getPackageRoot() {
        return "org.project.domain";
    }

    private ExternalPatient getExternalPatient(String id) {

        ...
```

```
    }
}

{
    "name": "external-patient",
    "objects": [
        {
            "displayName": ext.external_patient,
            "type": "ExternalPatient",
            "lookupFields": [
                {
                    "displayName": "ext.lookup.id",
                    "fields": [
                        "id"
                    ]
                }
            ],
            "fields": [
                {
                    "displayName": "ext.field.firstName",
                    "fieldKey": "firstName"
                },
                {
                    "displayName": "ext.field.secondName",
                    "fieldKey": "secondName"
                }
            ]
        }
    ]
}
```

2.7.5 Custom event parser

As it was mentioned earlier, there also exists a more advanced way to handle a trigger, by using a custom event parser. It allows to change the real event subject and parameters to a form in which they will be represented in the task trigger. In other words, it converts an event model to a tasks model.

An example of custom event parser usage can be found in the Commcare module. Once the form-received event occurs, the parser transforms the event payload containing a generic representation of the form xml to a trigger definition based on the schema of the concrete form, giving end-users intuitive access to the fields of that form.

To use a custom event parser, one has to implement TasksEventParser interface and expose it as an OSGi service. To make a use of the custom parser, the incoming event should contain a parameter with 'org.motechproject.tasks.custom_event_parser' as a key and the parser name returned by the getName() method as a value.

Example event parser:

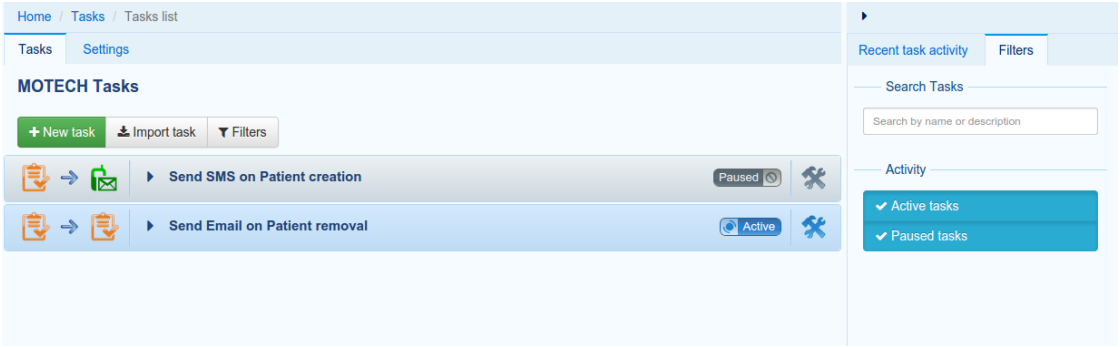
```
@Service
public class FormsEventParser implements TasksEventParser {

    @Override
    public Map<String, Object> parseEventParameters(String subject, Map<String, Object> parameters)
        Map<String, Object> parsedParameters = new HashMap<>();
        Map<String, Object> dataParameters = (Map<String, Object>) parameters.get("data");
        for (Map.Entry<String, Object> entry : dataParameters.entries()) {
            parsedParameters.put("data/".concat(entry.getKey()), entry.getValue());
```

```
        }
        return parsedParameters;
    }

    @Override
    public String parseEventSubject(String subject, Map<String, Object> parameters) {
        String formName = (String) parameters.get("name");
        return subject.concat(".").concat(formName);
    }

    @Override
    public String getName() {
        return "org.project.forms-event-parser";
    }
}
```

2.7.6 Tasks UI

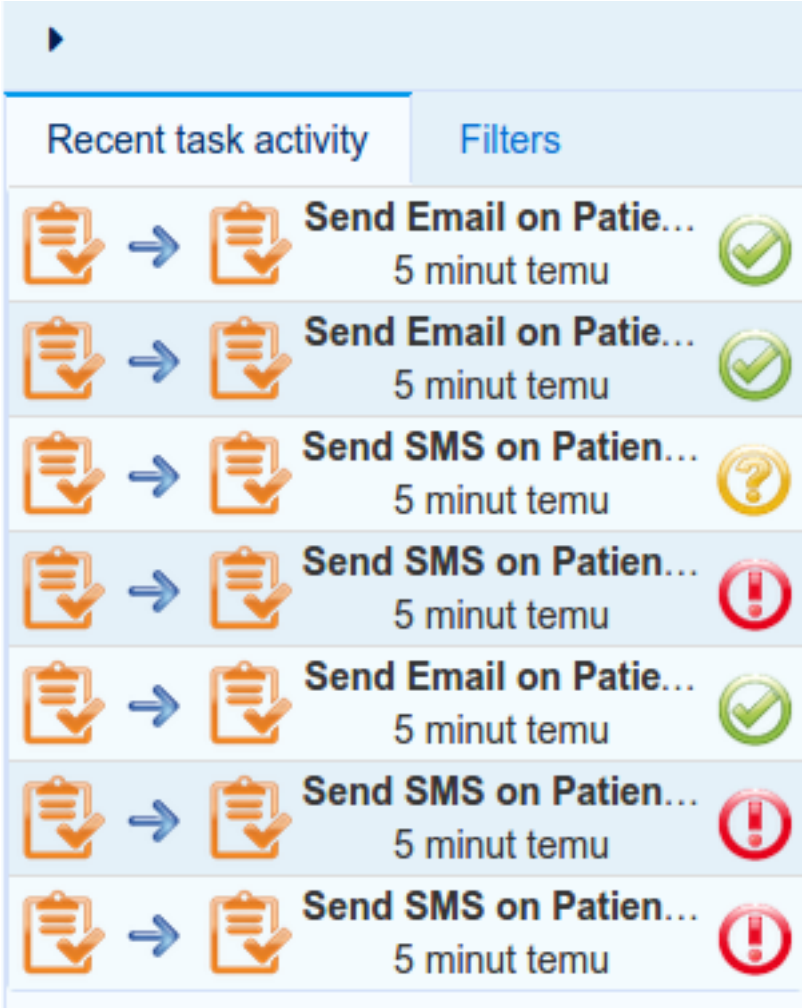
An important part of the Tasks module is the Tasks UI. It is used to create, edit, manage and monitor tasks.

Overview



The main Tasks view contains a few elements. Firstly, the action buttons are on the top. They allow creating tasks, importing previously exported tasks and toggling the visibility of the filter view on the right.

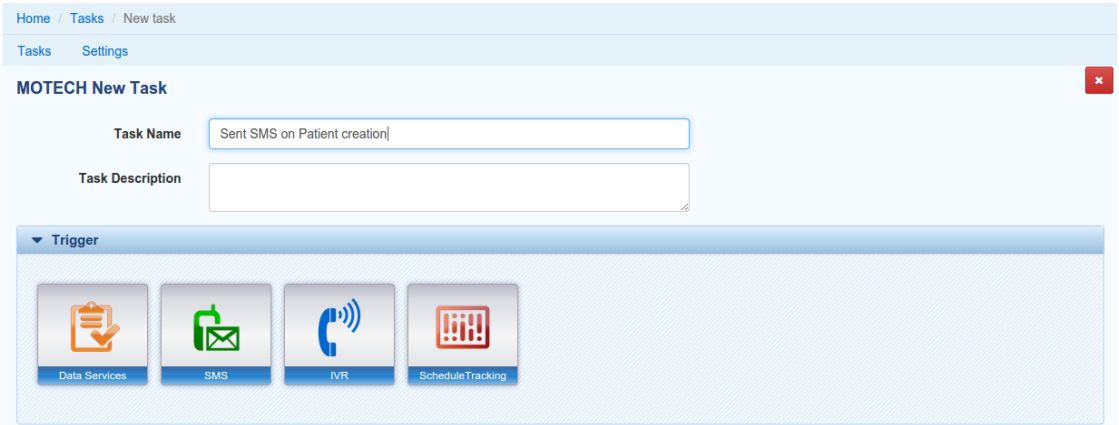
The main view lists all currently existing tasks in the form of expandable boxes, that provide actions related to the tasks they represent. The list can be filtered using filters tab mentioned before. One can search the tasks by their name or description as well as be their state (active/paused).



The right panel, besides filters, contains also a recent task activity tab. It provides an instant overview of latest task executions and their results.

Creating a task

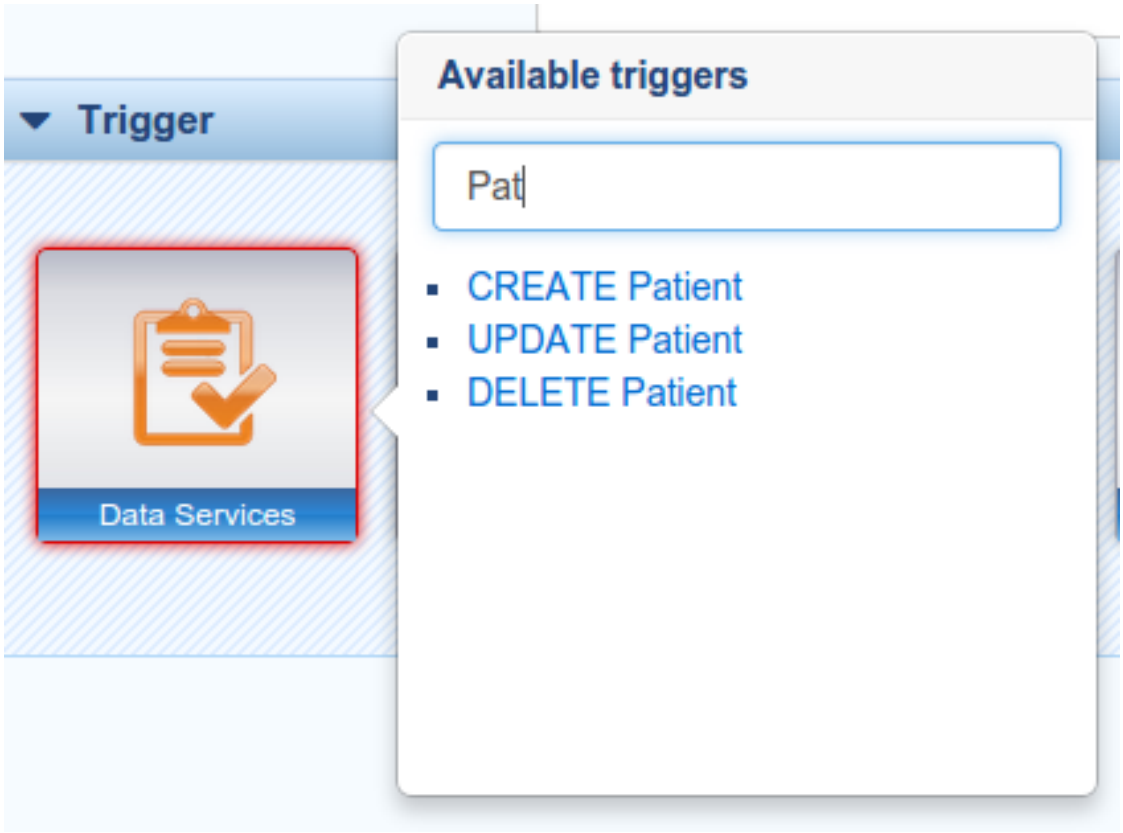
New task creation process begins with clicking the 'New task' button on the main view. The task creation view shows up.



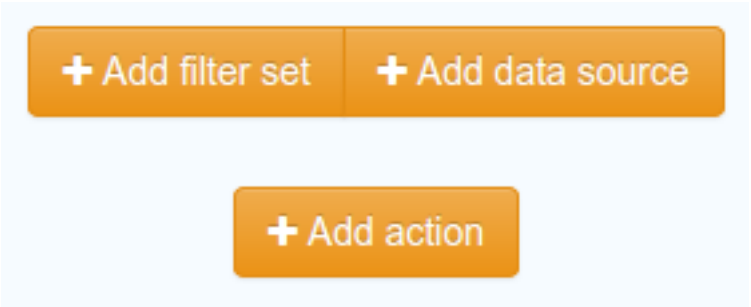
Starting from the top, one can see two properties to provide: task name and task description, from which the task name is mandatory.

Note: In the Tasks UI, if a property has invalid value it is signalled by highlighting its label and input field. There must not be any property with invalid value in order to save the task.

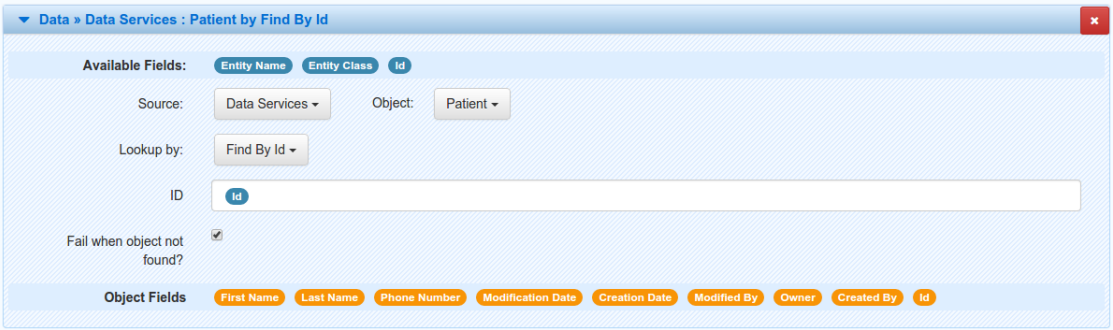
The trigger selection widget comes next. In this example there are four channels registered that expose at least one trigger. Once the channel icon is clicked, a popup shows up. It lists all triggers exposed by that channel.



Picking a trigger makes new actions available. One can add a data source, a filter set and finally select an action to execute.

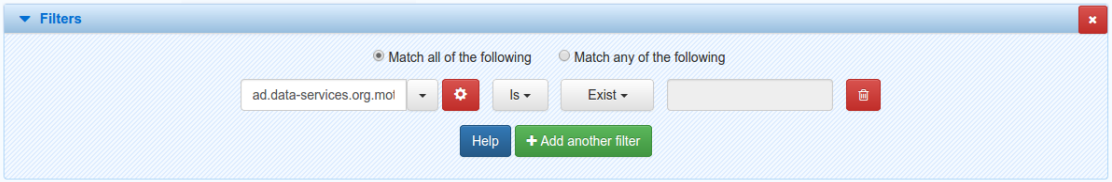


After clicking the 'Add data source' button, the data source widget shows up. The first step is to select an actual source. The dropdown lists all registered data sources. After picking one, a data source object must be selected. Now, one has to choose a lookup that will be used to retrieve an object and provide its arguments. In this example the 'find by id' lookup is used, thus the only lookup parameter is ID. The argument value may be either entered by hand (hardcoded) or composed from available fields listed on the top of the widget. Additionally, it is possible to set that task execution will fail if the lookup will not find the desired object.



Note: Available fields that may be noticed on the top of the data source and action widgets can be used to compose an argument values for the parameters. To include those fields they may be dragged and dropped into the desired input or written as a token in a text form. The syntax in case of trigger fields (the blue bubbles) is {{trigger.[field_name]}} where field_name is the key of the trigger parameter. In case of data source fields (the orange bubbles) the syntax is {{ad.[data_source_name].[object_name]#[object_index].[field_name]}} where provider_name and object_name corresponds to the selected data source and object respectively, object_index is the index of the object (in a situation when the same object was selected several times in the same task) and field_name corresponds to the object property.

Another available option is to add a filter set. The filter allows to setup a set of conditions that must evaluate to true in order for the task to execute. One can choose if all conditions should be satisfied or just one of them. If the entire condition set is not fulfilled, the task execution is canceled.

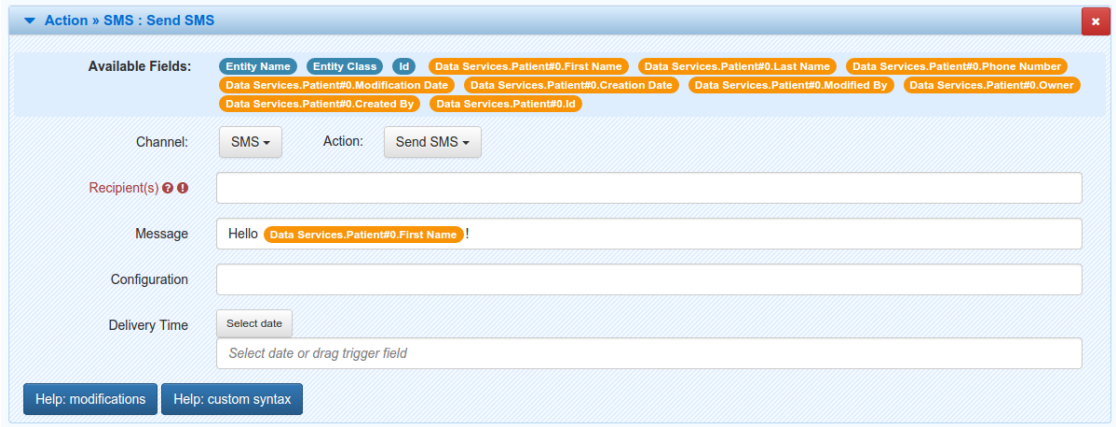


Each filter corresponds to a single field, either from trigger or data source. After selecting a field, there is a possibility to manipulate its value using Tasks *manipulations* by clicking on the gear icon next to it. Then it has to be set if the filter should be satisfied on a condition result or its negation. Finally, the condition can be selected. All conditions are grouped in three categories including Date, Number and String. Each of those contains basic checks that can be performed on the types they represents.

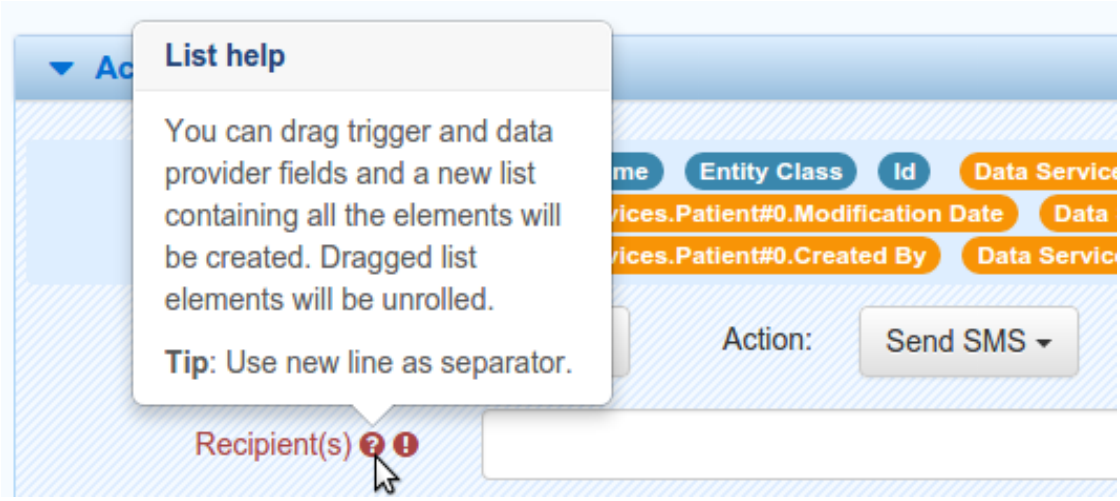
The filter is executed after all previous steps (data source lookups, filters) before were executed. In order for a filter to perform a check on a given data provider object, it has to be placed after that data source step. This can be used to abort the task execution before doing costly data lookups.

At last, with the 'Add action' button there comes a possibility to add an action to a task. There can be multiple actions in a single task, but unlike filters and data sources, the task is obligated to contain at least one action to be valid.

Note: If a task contains multiple actions and if some action execution fails, the remaining actions will not be executed.



All channels that expose at least one action are listed in the channel dropdown. When one of them is selected, the action dropdown contains all actions available. After selecting both channel and action, a list of action parameters is presented. In order to be valid, the action must not contain any parameter with an invalid value. Like in the case of data source lookups, the parameters may be filled with hardcoded values or combined field available either from the trigger or a data source. In case of fields, its values can be modified using Tasks *manipulations*. Sometimes, when a property has a complex type, a question mark can be visible next to its label. When hovered over, a popup with a short tooltip is shown.



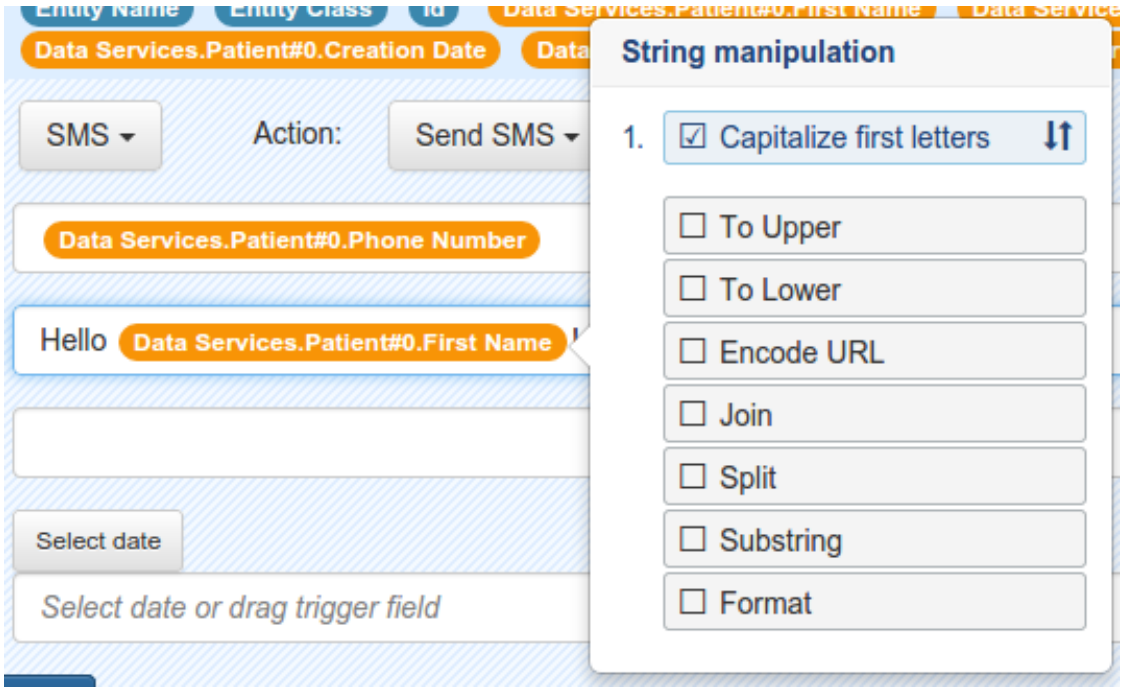
There are also two buttons at the bottom on the action widget. Once clicked, they provide a handy user manual related with fields syntax and string/date manipulations.

Note: Data sources, filter sets and actions can be removed from task by clicking an 'X' button at the top-right corner of the corresponding widget.

Once the task is defined, the last thing to do is to save it. There are two buttons on the bottom that allow to achieve this goal. One of them simply saves the task. This action is possible even if the task is not fully valid. The second option is to save and enable the task at once. In this case, the task must be valid. After clicking any of the buttons, the saved task can be seen in the main view.

Manipulations

In various situations related with task creation, there is a possibility to apply so called manipulations to fields originating from trigger or data sources. Manipulation allow to modify the incoming field values and transform them to something else. The most basic example might be changing all letters of a string value to uppercase.



Depending on the field type, distinct manipulations may be enabled. There are currently two categories of supported types: String and Date. An extensive description of them is available at the Tasks UI through a proper help button.

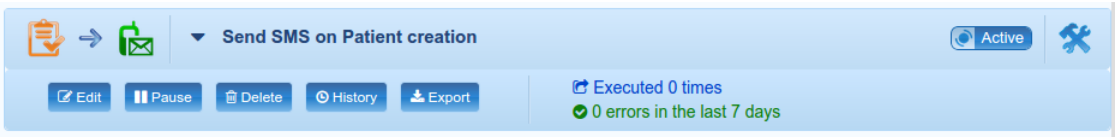
There might be multiple manipulations assigned to a single field. Moreover, they can be ordered in the widget by simply dragging and dropping them.

There is also possibility to define manipulations 'by hand' using plain text. The syntax in this case is {{[field]?[manipulation]}}, for example {{trigger.externalId?substring(0,8)?toUpper}}.

Note: Note that the modified value will not be written back to its source. For example, if the firstName field from the Motech Data Services Patient object will be edited with the uppercase manipulation, its value will not be changed in the database.

Managing and monitoring tasks

Once the task is created, it is shown in the main Tasks view in a widget form. In a basic form it contains information about modules related with the task, the task name, an icon indicating if the task is active or not and an icon that leads to a task editing view. Once the task name is clicked, the widget expands to expose all available actions related that task.



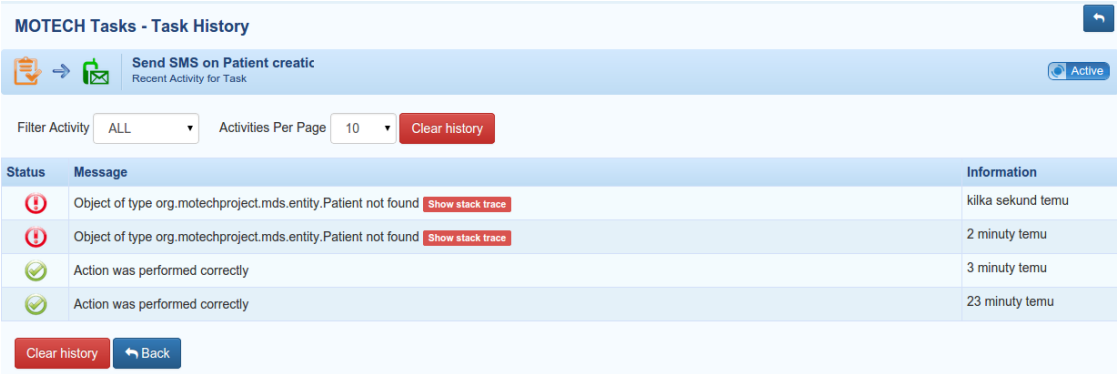
The first action allows to edit the task. The editin process is very similar to task creation. The editor view shows up and presents the task in a form in which it was saved. In a situation when a task is invalid, all validation errors are

visible at the top of the view.

Second button toggles the task state between paused and active. Active task will be executed when their corresponding trigger will occur, while paused task will not. The task may be paused in order to temporarily disable the task execution.

The delete button allows to permanently delete tasks. Once deleted, a task cannot be restored.

There is also a button that leads to the task history view. It allows to monitor all events related to the task and especially track an execution of the task. It provides information about the task result status and the message, which in case of failure contains stacktrace and failure reason. You can also clean the tasks history.



The last available option is to export the task. Selecting this action will trigger a file download. The file is a json representation of the task, that can be imported using 'Import task' action in the main view.

Settings

The only setting actually available is the limit of invalid executions for a single tasks. If the task will fails more times than it is allowed to by this parameter, it will be automatically paused until its manual activation. Once this happens, a message is added to the task events history. If the value of this parameter is set to 0, the task will be paused after only one failure.

It is worth mentioning that this parameter may be also set using file based config. The property name of the parameter is 'task.possible.errors'.

2.8 Configuring Pill Reminders

There will be more text here.

2.9 Configuring Your MOTECH App to be HIPAA Compliant

There will be more text here.

2.10 Tour of MOTECH UI

There will be more text here.

2.11 Security Rules - Dynamic URLs

Table of Contents

2.11.1 Security Rules

Security rules are used to build the Spring SecurityFilterChain which is used to filter incoming requests. By default, MOTECH blocks access to any resources if you are not logged in, therefore, accessing any URL will redirect to the login page. If you need an endpoint using a different configuration, you must add a new rule or edit an existing one.

Each rule contains the following parameters:

Display name	Parameter name	Description	Values
Active	active	You can enable the rule using this parameter	true, false
URL Pattern	pattern	URL pattern the security rule applies to (? matches one character, * matches zero or more characters, ** matches zero or more 'directories' in a path)	all
Protocol	protocol	Protocol which will be used for communication	HTTP or HTTPS
HTTP Method	methodsRequired	HTTP methods that have access to the endpoint	ANY, GET, POST, HEAD, OPTIONS, PUT, DELETE, TRACE
Rest	rest	Whether the endpoint is meant for a form login process or as an REST endpoint that does not create a session for the client	true, false
Priority	priority	Rule which has a higher priority will be checked first	priority value
Supported Schema	supportedSchemes	Specify which authentication is required	NO_SECURITY or USERNAME_PASSWORD, BASIC, OPEN_ID
User access	userAccess	Specify which users has access	list of users names
Permission Access	permissionAccess	Requires user has at least one of the listed permissions to access the URL	list of permissions names
(Not present in GUI)	origin	The module or user the rule originated from. Rules with SYSTEM_PLATFORM origin will be cleared at server start, so that they are always reloaded by the server	all
(Not present in GUI)	version	The version of the module or platform in which the rule was created	all

2.11.2 Priority

You can specify the order of processing using the priority parameter. Rules with greater priority will be checked first. In case of conflicting rules, the ones with higher priority will block the ones with lower priority. In this case it is worth considering to use more accurate URL patterns. It is very helpful for a hierarchy model of urls.

2.11.3 User access and permission access

When you are using permission access with user access in one rule you must know that these options operate separately. For example you gave User access to `sampleUser` and Permission access to `viewSecurity` permission. Access to the endpoint will be granted to `sampleUser` and each other user with `viewSecurity` permission.

2.11.4 Supported Schema, Rest and @PreAuthorize

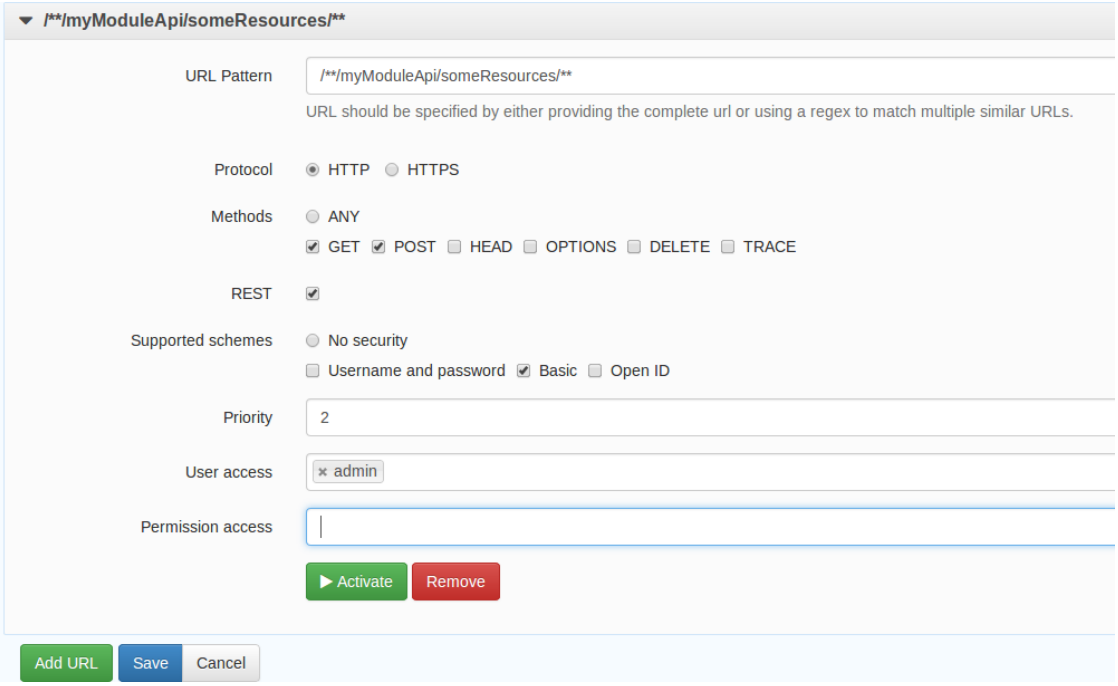
If resources are protected using @PreAuthorize annotation you must remember that `NO_SECURITY` schema will not work because access to these resources will be granted only to users with respective roles. If other schemas are used,

the user will still have to have the appropriate roles. The value of the rest option is important, you must know that if it's true then only `NO_SECURITY` and `BASIC` schemas will be supported.

2.11.5 Configuration via GUI

> **Attention:** Before saving configuration remember to check the correctness of the settings, because you can lock yourself access to change them or you could provide access to the whole system. If you have lost access to the system, read the *information on regaining access, due to incorrect security rules configuration*.

If you want edit those settings via GUI, your user account must have viewSecurity and updateSecurity permissions. To open the configuration you want to select 'Manage dynamic URLs' option under Security tab in the Admin panel. You should see a list of all security rules. When you start editing or adding a new security rule form will expand and you will see options that were described earlier. To activate current configuration you must save changes.



2.11.6 Configuration via files

You can add rules to your module using configuration files. To do this you must create a file named `securityRules.json` and place it in the resources directory and then build the module. Security rule configuration files are discovered automatically by MOTECH when the module starts.

Sample file:

```
[
    {
        "active": true,
        "pattern": "/**/myModuleApi/someResources/**",
        "supportedSchemes": [
            "NO_SECURITY"
        ],
        "protocol": "HTTP",
        "priority": 2,
```

```
        "rest": true,
        "origin": "SYSTEM_MODULE_MY_MODULE",
        "version": "0.25",
        "methodsRequired": [
            "GET",
            "POST"
        ]
    },
    {
        "active": true
        "pattern": "/**/myModuleApi/otherResources/**",
        "supportedSchemes": [
            "BASIC"
        ],
        "protocol": "HTTP",
        "userAccess": [
            "userName"
        ],
        "priority": 3,
        "rest": true,
        "origin": "SYSTEM_MODULE_MY_MODULE",
        "version": "0.25",
        "methodsRequired": [
            "ANY"
       ],
   }
]
```

2.11.7 Regaining access

To regain access to MOTECH, restart it. When server starts, default platform rules are always reloaded so it may help you regain access. If that doesn't work you should try drop database table holding security rules or delete only rules that block access.

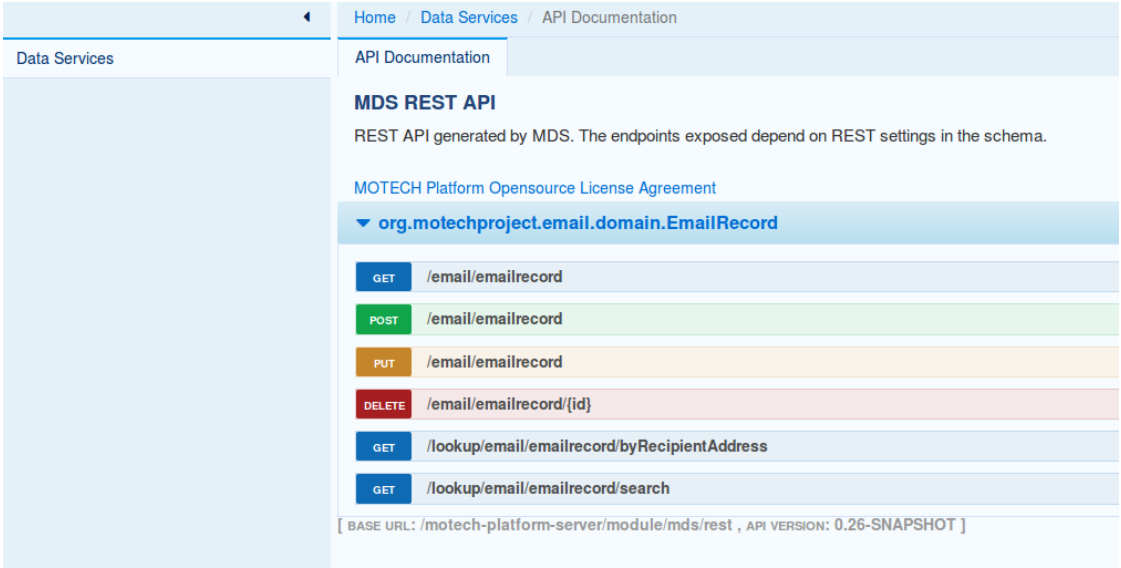
2.12 Automatic REST API documentation UI in MOTECH

MOTECH uses Swagger for generating a user interface that documents and allows testing of REST APIs. An interface can be generated for each module that wishes to register a REST API for documenting. This document will describe the process of registering a REST API for the module with the system.

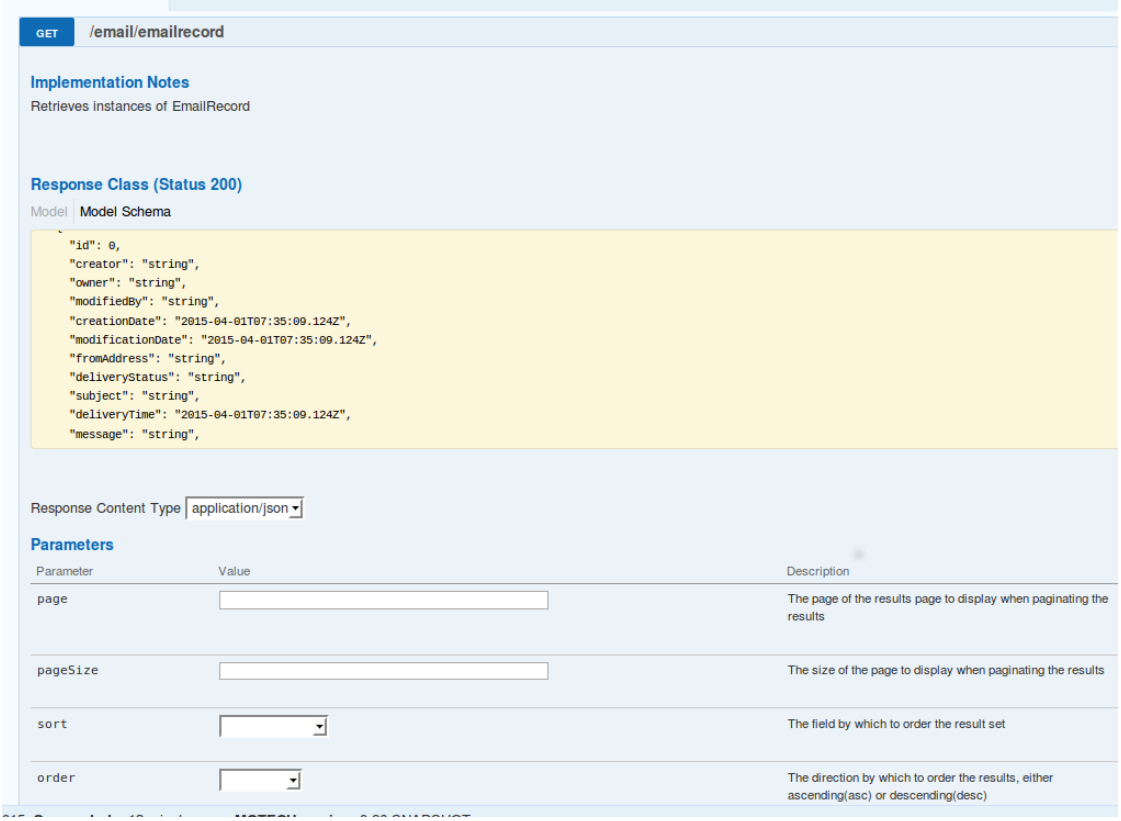
It is worth noting that this documentation will always be generated for *MDS* entities that have REST access enabled.

2.12.1 Overview of the UI

Swagger will generate documentation for each endpoint specified in the API description:



For each HTTP method allowed for a given endpoint, the user will be able to view the details of the operation, such as the structure of the response, structure of the expected request, allowed parameters.



Users can use that UI to easily execute REST calls against the API and view the responses.

2.12.2 Registering REST documentation

The first step for registering rest documentation is creating a Swagger spec file that will describe the API. More information on spec files, ways of generating them and so on can be found on the Swagger spec Wiki.

After generating the file, it has to be exposed by the module through HTTP. You can achieve this either by placing the file in your webapp resources or by creating a Spring controller that will serve this content. For more information on exposing a resource, refer to the *UI documentation*.

After the resource is exposed through the UI, its path should be specified in the ModuleRegistrationData bean using the **restDocsPath** property. Below is an example of a simple module registration that registers the spec file with the system.

```
<bean id="moduleRegistrationData" class="org.motechproject.osgi.web.ModuleRegistrationData">
    <constructor-arg name="url" value="../mymodule/resources/index.html"/>
    <constructor-arg name="moduleName" value="my-module"/>
    <property name="restDocsPath" value="/mymodule/resources/spec.json"/>
</bean>
```

After these steps, the module and its API will be incorporated into the Swagger UI exposed by MOTECH.

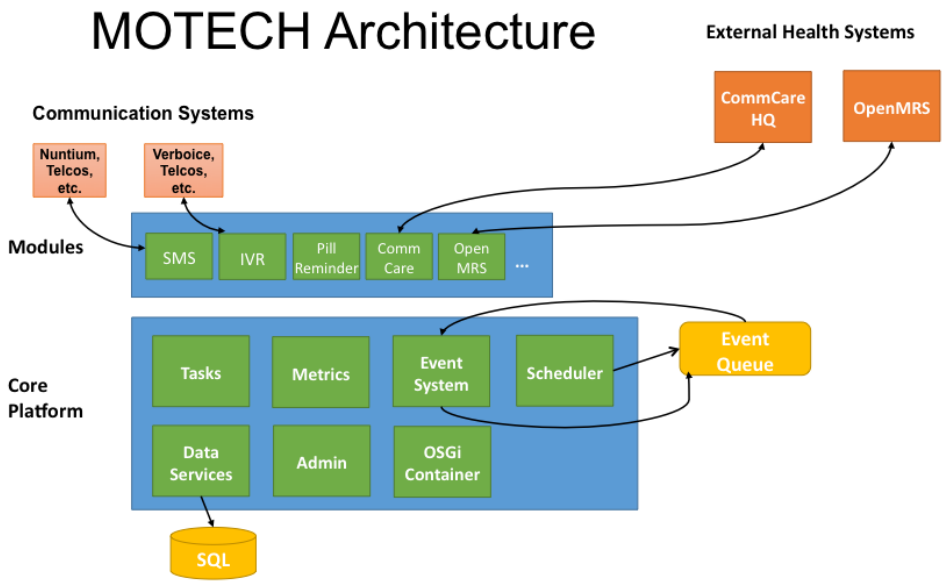
CHAPTER 3

Architecture and Technical Overviews

3.1 Core Architecture

3.1.1 Architecture Overview

MOTECH can logically be broken into the core platform and modules. The core platform wraps several well-known open source systems, and augments and exposes their features to the other components. The main functions of the core are to wrap ActiveMQ (which provides the message queue and the message topic) and present an internal pub/sub like event interface to the module and implementation layers. The core also provides a module loading environment (OSGi), an interface to the Scheduler, and access to the database.



Modules within MOTECH are self-contained bits of functionality that are loaded into the server via the OSGi host. Typically a module provides one type of functionality, such as SMS or access to an external health system. For more information, see *Modules Architecture*. For a list of current and planned modules, see *Modules*.

MOTECH is designed to be horizontally scalable with multiple MOTECHs all acting as workers connecting to the same message queue and topic.

3.1.2 Design Philosophy

Stateless

A core design principle of the MOTECH platform is that the server should be stateless across requests to allow for horizontal scalability. It is expected that code running within the MOTECH server should perform a single action per request and then return. The module should never persist any state in memory or local disk and expect that state to be available to later requests.

Events

To aid in the development of stateless services, the MOTECH engine provides a pub/sub like event system. (The event system follows the publish-subscribe pattern but does not implement the standard Java pub/sub protocol.) It helps to decouple emitters of events from the modules that wish to consume them. Any module can emit an event by calling the EventRelay and passing it a MotechEvent and a subject. To register for an event, a module just needs to annotate a method with the list of event subjects of interest.

For more information, see *Event and Scheduler Architecture*.

Scheduled Events & Timers

To assist in the development of a stateless event-based server, the MOTECH platform provides access to a flexible scheduling system. Using the open source Quartz engine, MOTECH can easily schedule events for future consumption. For more information, see *Event and Scheduler Architecture*.

3.1.3 Subsystems

Tasks System

The Tasks system allows you to connect modules without code by using tasks. Each task consists of three parts:

1. Trigger: an event raised by Module A (or the Scheduler)
2. Filter: a conditional statement specifying whether the task should run
3. Action: an action executed by Module B in response

In between the trigger and the action, tasks may use data loaders to look up data from other modules that are registered as data sources.

Data Services

MOTECH Data Services is a flexible data modeling system that allows users to define and share custom schemas without code, and provides auditing and revision tracking. It is a JDBC-based user configurable database abstraction layer on top of a standard SQL database. It provides generated POJOs and OSGi service interfaces for the data objects, generated CRUD events, and generated user interface for data browsing and editing. In a future release it will also support auto-generation of REST APIs.

3.1.4 Dependencies on Third-Party Systems

Quartz Scheduler

Quartz is an open source job scheduling engine that enables MOTECH modules to schedule events for future consumption.

Tomcat

Apache Tomcat provides the application container for MOTECH.

ActiveMQ

Apache ActiveMQ is an open source message broker that provides the message queue and the message topic.

OSGi

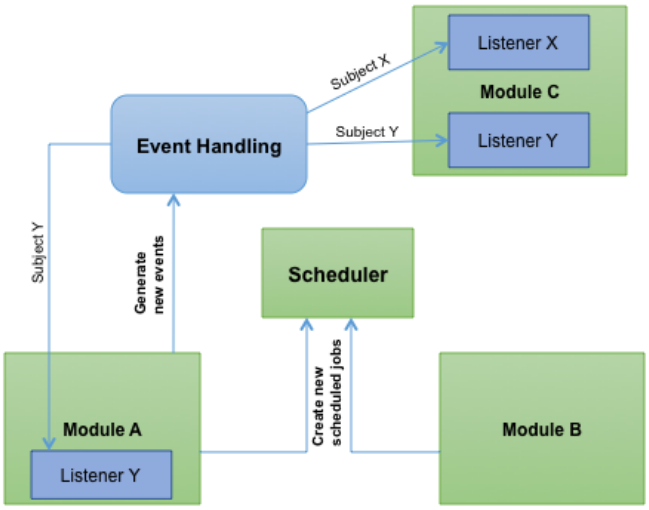
Each MOTECH module is an OSGi bundle. Using OSGi allows the platform to manage the bundle lifecycle (adding, removing, starting, and stopping modules), and allows modules to expose service interfaces. For more information, see *Modules Architecture*.

3.2 Event and Scheduler Architecture

3.2.1 Event Handling and Scheduling among Modules

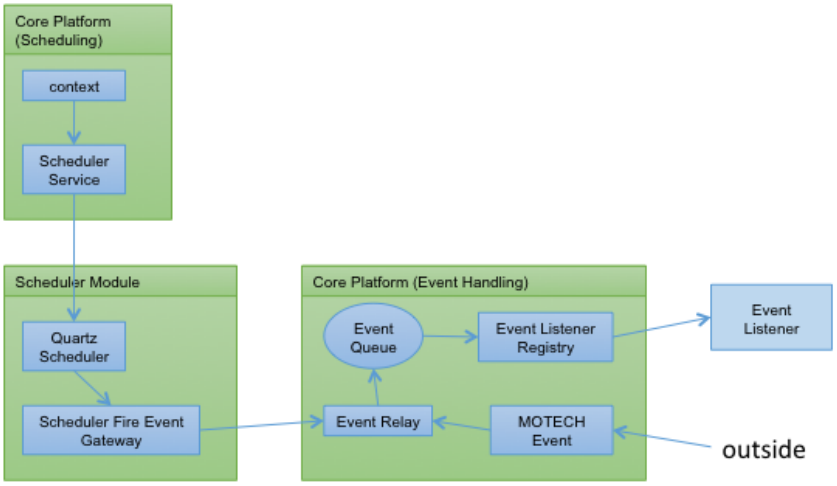
The following diagram provides three examples of Motech modules:

- Motech module A publishes events, schedules new jobs and listens for events. An example of a Motech platform that has these three responsibilities is the Message Campaign module.
- Motech module B schedules new jobs.
- Motech module C listens to events of subject X and Y: listener X listens to events of subject X, and listener Y listens to events of subject Y.



3.2.2 Event Handling and Scheduling Architecture

To schedule a job in MOTECH, the core platform exposes the MotechSchedulerService. Clients of the service have an instance of it injected into the class that uses it. This service employs Quartz to schedule MotechScheduledJobs. When triggered, MotechScheduledJobs raise events that are sent through an event relay to the event queue. These messages are dequeued and then received by a consumer, the event listener registry, which in turn discovers all of the listeners on the event and invokes the appropriate method that was listening on that event.



Notes

- The scheduler service retrieves the Quartz Scheduler from the SchedulerFactoryBean. The service can only schedule MotechScheduledJobs, which all must include a MotechEvent.
- When the trigger for the scheduled job is satisfied, the job is executed and the accompanying Motech event is sent to the SchedulerFireEventGateway interface. This gateway is defined in schedulerFiredEventChannelAdapter.xml and a proxy is generated by Spring at runtime.
- The SchedulerFireEventGateway serves as an outbound message gateway to shuttle messages to the event relay, which then sends JMS messages to the event queue.
- The event queue dequeues messages and routes them to the event listener registry. The core platform has one consumer for events, a channel that routes messages to an event listener registry.
- The event listener registry retrieves the listeners that are listening on the motech event it is relaying (listening is based on the value bound to the motech event's subject key).
- The core platform scans new bundles for any method annotated as a MotechListener, and registers those methods with the listener registry.
- If the event has a specific destination or only one listener, the listener's handle method is called. This will invoke the method that was annotated as a MotechListener. These listener classes can be project specific and will reside outside of the core platform.
- If the event has multiple listeners, a separate event is raised for each listener. The original event object is not handled by a listener in this case.
- The EventListener's handle method will invoke the method that was annotated as a MotechListener. The MotechEvent will be passed to that method as a parameter.

- If you wish to use ActiveMQ web console to view MotechEvents, please note that the server running the console must have the motech-platform-event bundle in its classpath. Therefore, either place the jar in the default location, or add a classpath that will contain the motech-platform-event jar.
- Note that MOTECH Scheduler is not a real-time system. There's no guarantee that your scheduled job will fire the exact same minute it has been scheduled for. Everything depends on the present CPU usage, other jobs that must be served and many more factors. Pay special attention to this, when scheduling jobs close to midnight, in case the date matters in your use case.

3.3 Modules Architecture

Modules within MOTECH are self-contained bundles of functionality that are loaded into the server via the OSGi host. Modules interact with the core platform through its APIs and with other modules, either through their APIs or by consuming their events. Modules can expose service interfaces of their own as well as emit their own events. Modules may also register servlet controllers, which allow them to respond to HTTP requests.

Through MOTECH Data Services, a modules may expose entities from its data model. This allows a module to provide a data editor, REST APIs, record-level security, and field-level auditing. Via the Tasks system, modules can expose triggers, data and actions to be orchestrated by other modules.

See *Modules* for the list of current and planned modules.

Reasons to create a module include developing application-specific UI, business logic, and data models. Another reason is to develop generic reusable functionality to share with the MOTECH community.

The MOTECH war already contains all platform modules required for operation and two additional modules: Admin Module and `/modules/scheduler`. The Admin Module allows installing additional modules at runtime, either by uploading a module jar or by selecting a module to be downloaded from our repository. Except from war modules, all MOTECH modules live in `~/.motech/bundles`. Module jars placed in that directory will be loaded by MOTECH at startup, if a module placed there has the same Bundle-Version and Bundle-SymbolicName as a module from the war, that module will override the platform one.

3.4 Data Services Architecture

There will be text here.

3.5 Security Model

Table of Contents

3.5.1 Introduction

Security aspects in MOTECH are handled by the web-security module. It is responsible for users, roles and permissions management, authentication, filtering requests and more. The web-security module uses Spring security in the backend and allows to use certain tools from the Spring security, like @PreAuthorize and @PostAuthorize annotations or SecurituContextHolder, that makes it possible to retrieve current authentication data. The web-security module provides an ability to administer users, roles, permissions and security rules via UI, for users that have been given the necessary permissions. The web-security module uses MDS as the datastore. User details, as well as roles, permissions and configured security rules are kept there. User passwords are hashed before storing, using bcrypt hashing algorithm.

3.5.2 MOTECH roles and permissions

MOTECH permissions can be used to secure an access to methods, functionalities or even whole modules. MOTECH roles are containers, that can keep one or more MOTECH permissions. MOTECH roles can be assigned to MOTECH users. MOTECH defines several permissions and roles that secure access to certain parts of the system, but both users and developers may create further roles and permissions, based on their needs.

Default MOTECH roles

MOTECH defines 5 default roles.

MOTECH role	Description
Motech Admin	Contains all MOTECH permissions. Any new permissions will get added to this role automatically.
Email Admin	Grants access to the Email module, allowing to send e-mails, change Email module settings and view detailed e-mail logs (containing e-mail addresses and e-mail contents).
Email Junior Admin	Similar to the Email Admin, grants access to the Email module, however the Junior Admin can only see basic e-mail logs (status and timestamp).
Bundle Admin	Allows to install, uninstall, update, start and stop modules.
Campaign Manager	Grants access to the Message Campaign module.

3.5.3 Access to modules

You can use the MOTECH permissions to secure an access to the module. To do so, you must set the **roleForPermission** property of the **org.motechproject.osgi.web.ModuleRegistrationData** bean. The value of this property can either be a String, that represents a permission name, or a list of permission names, in which case, the user will be granted an access if they have at least one of these permissions.

The sample definition of the ModuleRegistrationData bean in the blueprint context, with the roleForAccess property is shown below.

```
<bean id="moduleRegistrationData" class="org.motechproject.osgi.web.ModuleRegistrationData">
    <constructor-arg name="url" value="../email/resources/index.html"/>
    <constructor-arg name="moduleName" value="email"/>
    <property name="roleForAccess">
        <list>
            <value>viewBasicEmailLogs</value>
            <value>viewDetailedEmailLogs</value>
```

```
        </list>
    </property>
    <property name="settingsURL" value="/email/settings" />
    <property name="defaultURL" value="/email/send"/>
</bean>
```

3.5.4 Securing methods

MOTECH permissions can also be used to secure the execution of methods. This is achieved, using the Spring annotations. To enable Spring security annotations, add the following entry to the blueprint.xml context file of your module:

```
<security:global-method-security pre-post-annotations="enabled" proxy-target-class="true"/>
```

Once the security is enabled, you can annotate the methods to secure them using Spring security annotations. You can find the description of the supported annotations in the Spring documentation. The most commonly used annotation in the MOTECH modules is the **@PreAuthorize** annotation, which checks logged user credentials before executing the method and if they are insufficient, an attempt to execute the method will be denied. A method signature, that would get executed only for the users with the "admin" permission, could look like this:

```
@PreAuthorize("hasRole('admin')")
public void mySecureMethod() {
    doSomething();
}
```

Similar to the above, we can specify a set of roles. The execution will be allowed, if the user has got at least one of the listed permissions. The sample code could look like this:

```
@PreAuthorize("hasAnyRole('admin', 'junior_admin')")
public void mySecureMethod() {
    doSomething();
}
```

Note: Do not get fooled by the hasRole and hasAnyRole names. Despite the name suggesting otherwise, you should place MOTECH permissions there, not MOTECH roles!

The MOTECH web-security module will look for the **@PreAuthorize** and **@PostAuthorize** annotations in the modules, and add the permissions, that are not yet present in the system.

3.5.5 Retrieving security context

If you want to implement a custom security processor for your module or retrieve certain security information, you can do so, using the **org.springframework.security.core.context.SecurityContextHolder** util class. It allows you to retrieve information about current authentication. See the code below for the example on retrieving current user and his permissions.

```
Authentication auth = SecurityContextHolder.getContext().getAuthentication();
if (auth != null) {
    User user = (User) auth.getPrincipal(); // RETRIEVE USER
    Collection<GrantedAuthority> authorities = auth.getAuthorities(); // RETRIEVE PERMISSIONS
}
```

3.5.6 Login modes

The MOTECH platform allows two ways of authenticating users. The two modes are called:

- Repository
- Open ID

The login mode is chosen by the administrator, during first server startup or in the motech-settings.properties file, depending on the chosen *config source*.

Using the **Repository** login mode, MOTECH will provide a way to create an initial user, during first server startup. The initial user is granted all default permissions (Motech Admin role). New users can be created via UI, or using the MotechUserService from web-security module. All the users are stored in the MOTECH database, using MDS.

If the chosen login mode is **Open ID**, it is also necessary to provide a valid URL to the Open ID provider, that will handle authentication. For example, to set **Google** as your Open ID provider, the URL should be **https://www.google.com/accounts/o8/id**. It will be possible to authenticate to MOTECH, by logging in at the provider. The first user that logs in will be granted all default permissions (Motech Admin role). Next users that log in will not be given any permissions, but that may be altered via UI, or using the MotechUserService.

> **Warning:** When choosing **Open ID** as the login mode, please remember that everyone who has got an account at the specified provider will be able to access your server. If that's not what you want, use the **Repository** login mode.

3.5.7 HTTP access

If you are not authenticated, the access to any MOTECH resources is blocked by default. Therefore, most of the requests will get a 301 HTTP response, with a redirection to the login page. It is possible to configure exceptions to this rule, by creating *dynamic URLs*. They can be used to alter the security settings for specified URLs or to disable the security at all (meaning everyone will be able to access the resource). Security rules can be altered via UI or via properties file, placed in your module.

3.5.8 Security Configuration

The MOTECH platform allows you to configure security options. You can easily change those setting via Admin UI(`Settings` tab), or by editing `motech-settings.properties` file. For more details you should read the *configuration system section*. Below you can find available options.

Email required (`security.required.email`) - Indicates whether you must provide an email address when creating the user. Possible values: `true`, `false`.

Failure login limit (`security.failure.login.limit`) - The permissible number of incorrect login attempts, default value is 0. After this limit is reached the user is blocked. After a successful login counter is reset. If the value is 0 then blocking is inactive.

Session timeout (`security.session.timeout`) - The session timeout in seconds, default 30 minutes. After this time session will be closed.

Minimum password length (`security.password.minlength`) - The minimum length of the password, default 0. If the value is 0 then length checking is disabled.

Password restriction (`security.password.validator`) - Name of the password validator which will be used for checking passwords. Validator specifies password restriction e.g. 1 number, 1 special character. You can use 1 of 4 validators implemented in MOTECH(default is `none`) or you can *create your own password validator*. Below you can find the names of validators provided by MOTECH.

- `none`
- `lower_upper` - at least 1 uppercase and lowercase
- `lower_upper_digit` - at least 1 uppercase lowercase and digit
- `lower_upper_digit_special` - at least 1 uppercase, lowercase, digit and special character

3.5.9 Creating Password Validator

Description

To create your own password validator you must create new validator class which will implement `PasswordValidator` interface. The next step is to create OSGI service from your new validator. To do this you must add `@Service` annotation to the class and service reference information in the blueprint file.

`PasswordValidator` interface contains following methods:

- void validate(String password) throws PasswordValidatorException - validates password.
- String getValidationError(Locale locale) - returns the error message(should be treated as a literal) for the validator. Message should explain what is expected in password.
- String getName() - Returns the name of the validator used for retrieval. Must match the value from the configuration in order to be used.

Example Code

Below you can find example of a validator which requires 3 special characters in password.

```
@Service("serviceName")
public class MyValidator implements PasswordValidator {

    private String name = "validator_name";

    @Override
    public void validate(String password) {
        CharacterCount count = new CharacterCount(password);

        if (count.getSpecial() < 3) {
            throw new PasswordValidatorException("Invalid password, validator name - " + name);
        }
    }

    @Override
    public String getValidationError(Locale locale) {
        return "Password must have 3 special characters";
    }

    @Override
    public String getName() {
        return name;
    }
}
```

To enable it you must add `<osgi:service ref="serviceName" interface="org.motechproject.security.vali` to the blueprint file and set it in config(security.password.validator=validator_name).

CHAPTER 4

Modules

4.1 *Alerts*

Collects alerts for users in an inbox-like container

4.2 *Appointments*

Provides appointment scheduling and reminders

4.3 *Batch*

An implementation of Spring batch (version: 3.0.0.M3); it essentially deals with scheduling triggering of jobs

4.4 *CMS Lite*

Provides basic content storage and retrieval

4.5 *CommCare*

Integrates the MOTECH platform with CommCareHQ, an open-source platform to help manage community health workers

4.6 *Data Services*

Integrates data from external data sources and provides sharable data schemas

4.7 *Email*

Sends and logs email messages

4.8 *Event Logging*

Allows MOTECH modules to easily see each others' events

4.9 *Hindi Transliteration*

Supports transliteration of English strings to Hindi using ITRANS encoding

4.10 *Hub*

Provides an implementation of the PubSubHubbub Hub spec; exposes an API so other modules can act as publisher and make contents available to it for distribution

4.11 *IVR*

Integrates the MOTECH platform with Interactive Voice Response (IVR) providers thus enabling support for voice/audio dialogs

4.12 *Message Campaign*

Enrolls users in message campaigns with flexible content-scheduling rules

4.13 *mTraining*

Provides data containers and APIs for defining mobile (e.g. SMS or IVR-based) training courses and tracking user enrollment and progress

4.14 *OpenMRS*

Integrates the MOTECH platform with OpenMRS, an open source electronic medical record platform

4.15 *Pill Reminder*

A flexible reminder system that may be used to alert patients when it is time to take their medications

4.16 *Schedule Tracking*

Enrolls users for alerts based on complex scheduling rules

4.17 *Scheduler*

Publishes events on a schedule, using the open source Quartz engine.

4.18 *SMS*

Provides a basic specification for integrating the MOTECH platform with an SMS provider to send/receive SMS messages

4.19 *Tasks*

Allows administrative users to author simple "tasks" that wire up different modules; for example, a task can be created to enroll a patient in a message campaign in response to an incoming SMS message containing specific text

hidden

includehidden

glob

•

CHAPTER 5

Developing the MOTECH Platform

This section of the documentation is aimed at developers and maintainers of the MOTECH Platform. These docs help MOTECH devs get up and running, from configuring a machine, to getting the code, and committing and reviewing changes. If you find any errors in the topics below, or you have any other questions about MOTECH development, please contact us via the mailing list.

5.1 Setting up a Development Environment

The topics below will walk you through installing MOTECH on a Mac or Linux machine. Note that the Docker-based setup is much faster than the "official" method but the instructions are in beta. If you have any trouble with either approach, please feel free to contact us via motech-dev@googlegroups.com.

5.1.1 Installing MOTECH for Developers ("Official" Method)

Table of Contents

Installing on Ubuntu

The versions below may change, most likely the latest stable release will work for your purposes. If they do not, please feel free to send in feedback.

1. **Install Ubuntu Desktop 14.04.1 LTS 64bit** Installation instructions
2. Install Maven, Git, Curl, ActiveMQ, and a database of your choice
   (a) In terminal, type

```
sudo apt-get install curl git maven activemq
```

(b) The two datastores officially supported by MOTECH are MySQL and PostgreSQL. It is not required to install both of them to run MOTECH, but provided you intend to introduce some changes to the code, it may be required that you test the outcome on both databases.

```
sudo apt-get install mysql-server
sudo apt-get install postgresql
```

(c) On a fresh Ubuntu installation, you may need to run the following first

```
sudo apt-get update
```

3. Configure ActiveMQ

   Run the following

```
sudo ln -s /etc/activemq/instances-available/main /etc/activemq/instances-enabled/main
```

Note: For ActiveMQ scheduled delivery to work, you must set the attribute: **schedulerSupport="true"** for the broker element in your activemq.xml config file. This file should be located at (active-mq-folder)/conf/activemq.xml.See ActiveMQ docs.

4. Install JDK 7

   (a) Go to The Java JDK Download Page

   (b) Accept License Agreement

   (c) Click on jdk-7u51-linux-x64.tar.gz (or latest stable version)

   (d) Extract the file into your home directory, ie: `/home/*<user>*/jdk1.7.0_51`

   (e) Set the proper Java environment and change maven options:

      i. Start a new terminal session

      ii. Edit your .profile file

```
nano ~/.profile
```

      iii. append the following at the end of the file:

```
export PATH="$HOME/jdk1.7.0_21/bin:$PATH"
export JAVA_HOME=$HOME/jdk1.7.0_21
export MAVEN_OPTS="-Xmx512m -XX:MaxPermSize=128m"
export CATALINA_OPTS="-Xms1024m -Xmx2048m -XX:MaxPermSize=1024m"
```

      iv. Save the changes (Ctrl+X) and quit

      v. Confirm the settings are right

      vi. Log out & log back in & start a new terminal

      vii. Type

```
java -version && env | grep "\(MAVEN_OPTS\|CATALINA_OPTS\)"
```

      You should see something like:

```
java version "1.7.0_51"
Java(TM) SE Runtime Environment (build 1.7.0_51-b11)
Java HotSpot(TM) 64-Bit Server VM (build 23.21-b01, mixed mode)
MAVEN_OPTS=-Xmx512m -XX:MaxPermSize=128m
CATALINA_OPTS=-Xms1024m -Xmx2048m -XX:MaxPermSize=1024m
```

5. Install Tomcat7
   (a) Go to Tomcat's download page
   (b) Under 7.0.52 (or the latest stable version) - Binary Distributions - Core, click on tar.gz
   (c) Once downloaded, expand the file to your home directory, i.e.: `/home/*<user>*/apache-tomcat-7.0.52`
   (d) Edit the `tomcat-users.xml` file (located under `\etc\tomcat7\conf\`) to add an admin user:
   (e) In the terminal type

```
nano ~/apache-tomcat-7.0.52/conf/tomcat-users.xml
```

   (f) Insert a line similar to the following before the closing </tomcat-users> tag:

```
<user username="*<username>*" password="*<password>*" roles="manager-gui"/>
```

   (g) Save the changes (Ctrl+X) then quit
   (h) Edit the web.xml of the manager application(located under `\webapps\manager\WEB-INF\web.xml`):

```
nano ~/apache-tomcat-7.0.52/webapps/manager/WEB-INF/web.xml
```

   (i) Edit the lines in multipart-config defining the max upload value. Change it from 50MB to a bit more, 70MB should suffice:

```
<!-- Before changes -->

<multipart-config>
  <!-- 50MB max -->
  <max-file-size>52428800</max-file-size>
  <max-request-size>52428800</max-request-size>
  <file-size-threshold>0</file-size-threshold>
</multipart-config>


<!-- After changes -->

<multipart-config>
  <!-- 70MB max -->
  <max-file-size>71680000</max-file-size>
  <max-request-size>71680000</max-request-size>
  <file-size-threshold>0</file-size-threshold>
</multipart-config>
```

   (j) Save the changes by hitting `Ctrl+X` then quit
   (k) Now edit `~/.bashrc` to setup tomcat's environment variable

```
nano ~/.bashrc
```

(l) Append the following line:

```
export CATALINA_HOME=$HOME/apache-tomcat-7.0.52
```

(m) Save the changes (Ctrl+X) then quit

(n) Start a new terminal session or type

```
source ~/.bashrc
```

6. Setup MySQL (skip if you did not install MySQL server)

   (a) Access your database, by typing in the terminal:

   ```
   $ mysql -u root -p
   ```

   (b) Create required databases (note: when you're using account with privilages for DB connection, MOTECH will create necessary DBs and fill them with data; otherwise you have to create them yourself)

   ```
   sql> create database motechquartz;
   sql> create database motech_data_services;
   sql> exit;
   ```

   (c) (Optional) Create user for the motechquartz database. MOTECH will use the user and password from the bootstrap configuration by default, but you can adjust that in the Scheduler settings and provide different credentials.

   ```
   sql> create user 'quartz'@'localhost' identified by 'quartz2123';
   sql> grant all privileges on motechquartz.* to 'quartz'@'localhost';
   ```

   Note: Sometimes it is needed to set the proper database character encoding. For example, to create motech_data_services database with UTF-8 character encoding, change your sql query to:

   ```
   sql> create database motech_data_services
        default character set utf8 collate utf8_general_ci;
   ```

7. Setup PostgreSQL (skip if you did not install PostgreSQL server)

   (a) Access your database, by typing in the terminal:

   ```
   $ sudo -u postgres psql postgres
   ```

   (b) Set a password for the "postgres" database role

   ```
   postgres=# \password postgres
   ```

   and give your password when prompted.

   (c) Create required databases (note: when you're using account with privilages for DB connection, MOTECH will create necessary DBs and fill them with data; otherwise you have to create them yourself)

```
postgres=# create database motechquartz;
postgres=# create database motech_data_services;
postgres=# (ctrl + D)
```

(d) (Optional) Create user for the motechquartz database. MOTECH will use the user and password from the bootstrap configuration by default, but you can adjust that in the Scheduler settings and provide different credentials.

```
postgres=# create user quartz with password 'quartz2123';
postgres=# grant all privileges on database motechquartz to quartz;
```

Note: MD5 authentication is required and should be enabled by default in latest versions of PostgreSQL. If it's not the case, you might need to enable this by hand. For more information refer to: http://www.postgresql.org/docs/9.3/static/auth-methods.html

8. **Start Tomcat**

   (a) In terminal, type:

```
~/apache-tomcat-7.0.52/bin/catalina.sh jpda start
```

   (b) You should see messages similar to:

```
Using CATALINA_BASE:   /home/*<user>*/apache-tomcat-7.0.52
Using CATALINA_HOME:   /home/*<user>*/apache-tomcat-7.0.52
Using CATALINA_TMPDIR: /home/*<user>*/apache-tomcat-7.0.52/temp
Using JRE_HOME:        /home/*<user>*/jdk1.7.0_51
Using CLASSPATH:       /home/*<user>*/apache-tomcat-7.0.52/bin/bootstrap.jar:/home/*<use
```

   (c) You can also confirm tomcat was started by going to http://localhost:8080 in a browser

9. Jump to the Building and Installing MOTECH section to install MOTECH

Installing on a Macintosh

1. Installing Prerequisites for MOTECH

   (a) Installing HomeBrew

   To install Homebrew, run the following in the terminal

```
ruby -e "$(curl -fsSL https://raw.githubusercontent.com/Homebrew/install/master/install)
```

   (b) **Use Homebrew to install git, erlang, ActiveMQ, and Apache Tomcat:**

```
brew install git
brew install activemq
brew install tomcat
brew install maven
```

   (c) Homebrew installations are located `in /usr/local/Cellar` with symlinks in ``/usr/local/bin`, which should already be part of your $PATH environment variable.

   Note: Homebrew provides instructions about how to run these applications, as well as how to have launchd start them automatically on system startup.

(d) Configuring Tomcat

i. Edit the `tomcat-users.xml` file to add an admin user. Insert a line similar to the following before the closing `</tomcat-users>` tag:

```
<user username="motech" password="motech" roles="manager-gui"/>
```

ii. Edit the web.xml of the manager application(located under `\webapps\manager\WEB-INF\web.xml`) and change the lines in multipart-config defining the max upload value. Change it from 50MB to a bit more, 70MB should suffice:

```
<!-- Before changes -->

<multipart-config>
  <!-- 50MB max -->
  <max-file-size>52428800</max-file-size>
  <max-request-size>52428800</max-request-size>
  <file-size-threshold>0</file-size-threshold>
</multipart-config>


<!-- After changes -->

<multipart-config>
  <!-- 70MB max -->
  <max-file-size>71680000</max-file-size>
  <max-request-size>71680000</max-request-size>
  <file-size-threshold>0</file-size-threshold>
</multipart-config>
```

(e) Installing JDK 7:

Mac OS includes JDK6 by default, however JDK 7 is required for MOTECH. Use these instructions to install the latest version of the JDK.

(f) Installing MySQL:

i. Before installing MySQL, you will need Xcode from the App Store. This can take a while; it's a big download.

ii. Next start Xcode from the Launchpad (rocketship icon in the dock) and select Install. Then you can quit Xcode; you don't need to keep it running.

> **Note:** (Command Line Tools using Xcode are included in OS X Mavericks, but not previous OS versions. If you are running Mountain Lion, you can follow these instructions:)

iii. Go to http://dev.mysql.com/downloads/mysql/ and download the appropriate DMG archive. Open it, double-click on the installer, and follow directions.

d. Once mysql has finished installing, double-click the MySQL preferences pane in the DMG and follow instructions. For more details see these instructions .

> **Note:** Homebrew can be used to install MySQL, however Homebrew will not install the Mysql System Preferences control panel.

2. Setting up Symbolic Link and Environment Variables

(a) Create a symbolic link from the Tomcat directory (Homebrew installs into `/usr/local/Cellar/tomcat/<version number>/libexec`) to `/usr/local/tomcat`:

```
ln -s /usr/local/Cellar/tomcat/`brew info tomcat | grep stable | awk '{print $3}' | sed
```

(b) Edit your `~/.bash_profile` to set environment variables (catalina is Tomcat):

```
export JAVA_HOME="/Library/Java/Home"
export MAVEN_OPTS="-Xmx512m -XX:MaxPermSize=128m"
export CATALINA_HOME="/usr/local/tomcat"
export CATALINA_OPTS="-Xms1024m -Xmx2048m -XX:MaxPermSize=1024m"
export PATH=/usr/local/mysql/bin:$PATH
```

(c) **When you're done editing:**

```
source ~/.bash_profile
```

3. Jump to the Building and Installing MOTECH section to install MOTECH

Building and Installing MOTECH

1. Getting the MOTECH code

   List of MOTECH repositories

   Generic developer git workflow

2. Building MOTECH

   (a) Assuming you issued the git clone command in your home directory root, in the terminal

   ```
   $ cd ~/motech
   $ mvn install
   ```

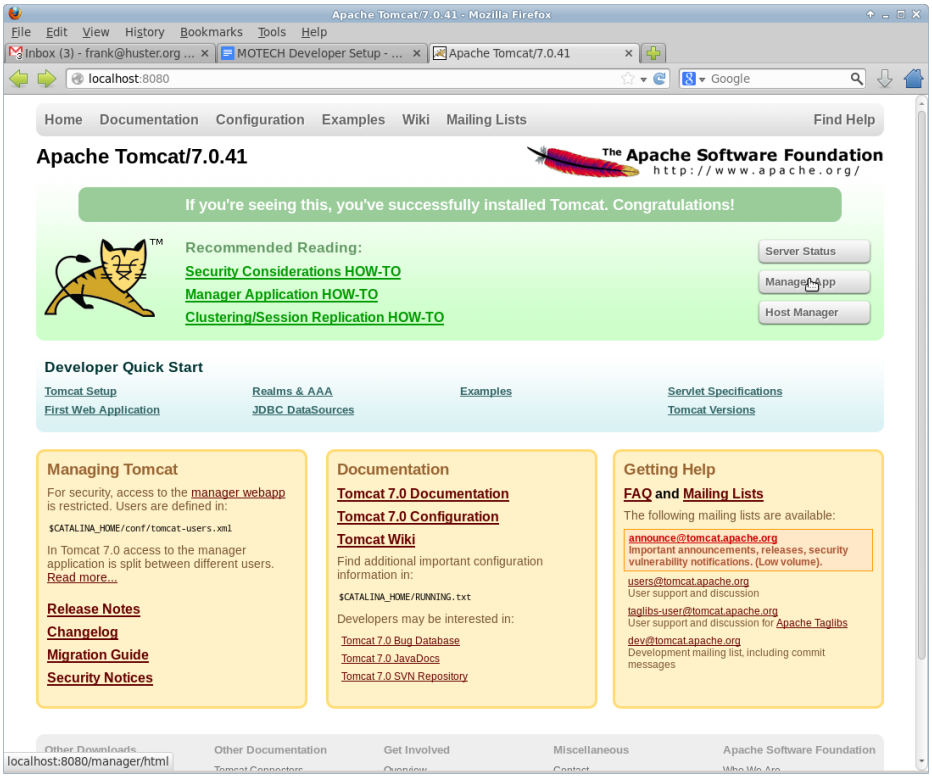
   b.) It takes some time to build MOTECH, but eventually you should see:

   ```
   [INFO] ------------------------------------------------------------------------
   [INFO] BUILD SUCCESS
   [INFO] ------------------------------------------------------------------------
   [INFO] Total time: 29:19.284s
   [INFO] Finished at: Fri Jun 07 12:12:43 PDT 2013
   [INFO] Final Memory: 152M/378M
   [INFO] ------------------------------------------------------------------------
   ```

   Note: Should you get a java.lang.OutOfMemoryError exception, it may be because you forgot to set MAVEN_OPT as described in [3.5]. But you may need to increase -Xmx. So something like -Xmx1024m might work.

3. Install MOTECH

   (a) In a browser, go to http://localhost:8080

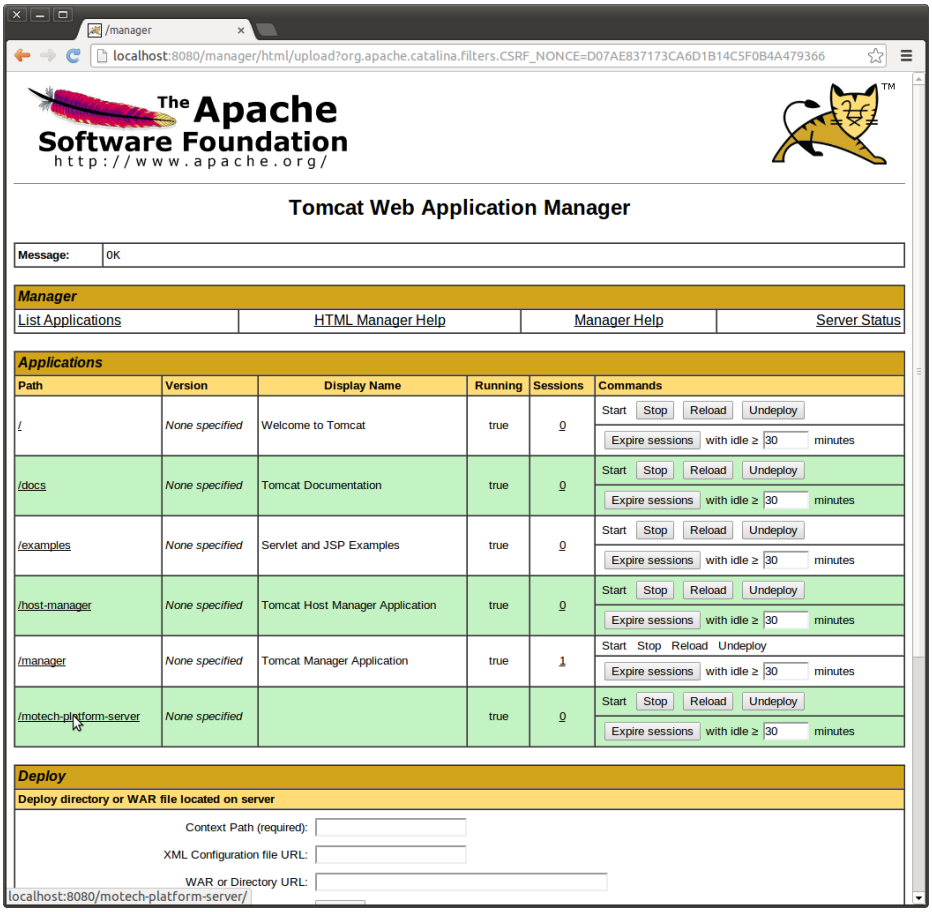


(b) Click on Manager App

(c) Type the user/password you used in tomcat-users.xml (if you installed via docker the default username/password is motech/s3cret).

> temporary hack you need to remove ~/.motech/config/motech-settings.conf to allow the create initial user wizard.

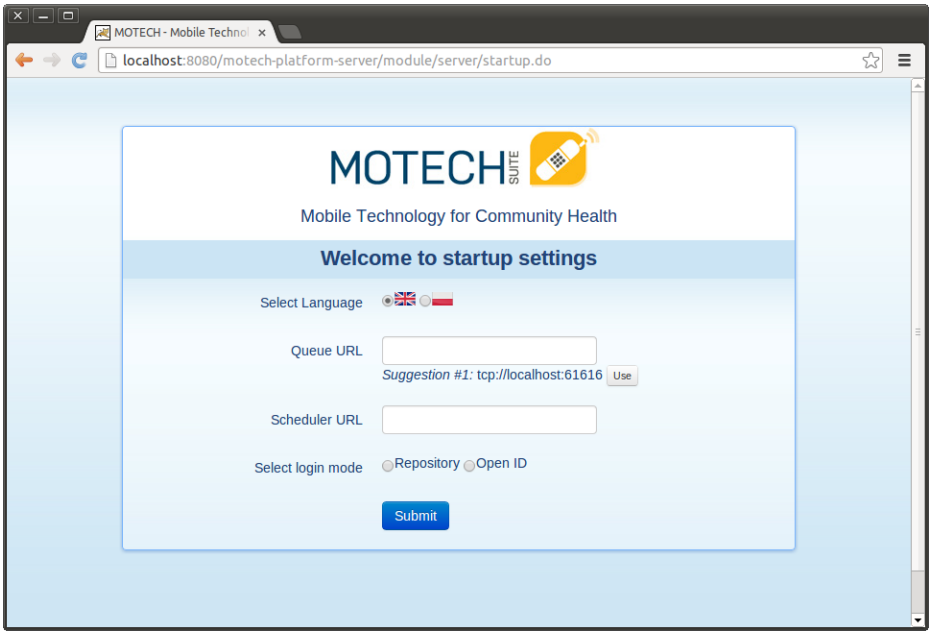
(d) In the Tomcat Web Application Manager, scroll down to the Deploy section and the WAR file to deploy subsection, click on Browse and select or navigate to `~/motech/platform/server/target/motech-platform-server.war` then click on Deploy



(e) Depending on your machine it could take a while for motech-platform-server to deploy

(f) If you get an error of the form: "the request was rejected because its size (68032892) exceeds the configured maximum (52428800)" follow these instructions to

(g) In the Tomcat Web Application Manager page, click on `/motech-platform-server`, you get the MOTECH initial user screen



Note: The war file contains all modules required for starting and managing MOTECH. You can either use the Admin UI to install additional modules at runtime or place them in the `~/.motech/bundles` directory and restart MOTECH. Note that doing a **mvn clean install** on any of our modules will place that module in the `~/.motech/bundles` directory automatically. Modules from that directory always override the ones contained in the war if their Bundle-Version and Bundle-SymbolicName are the same.

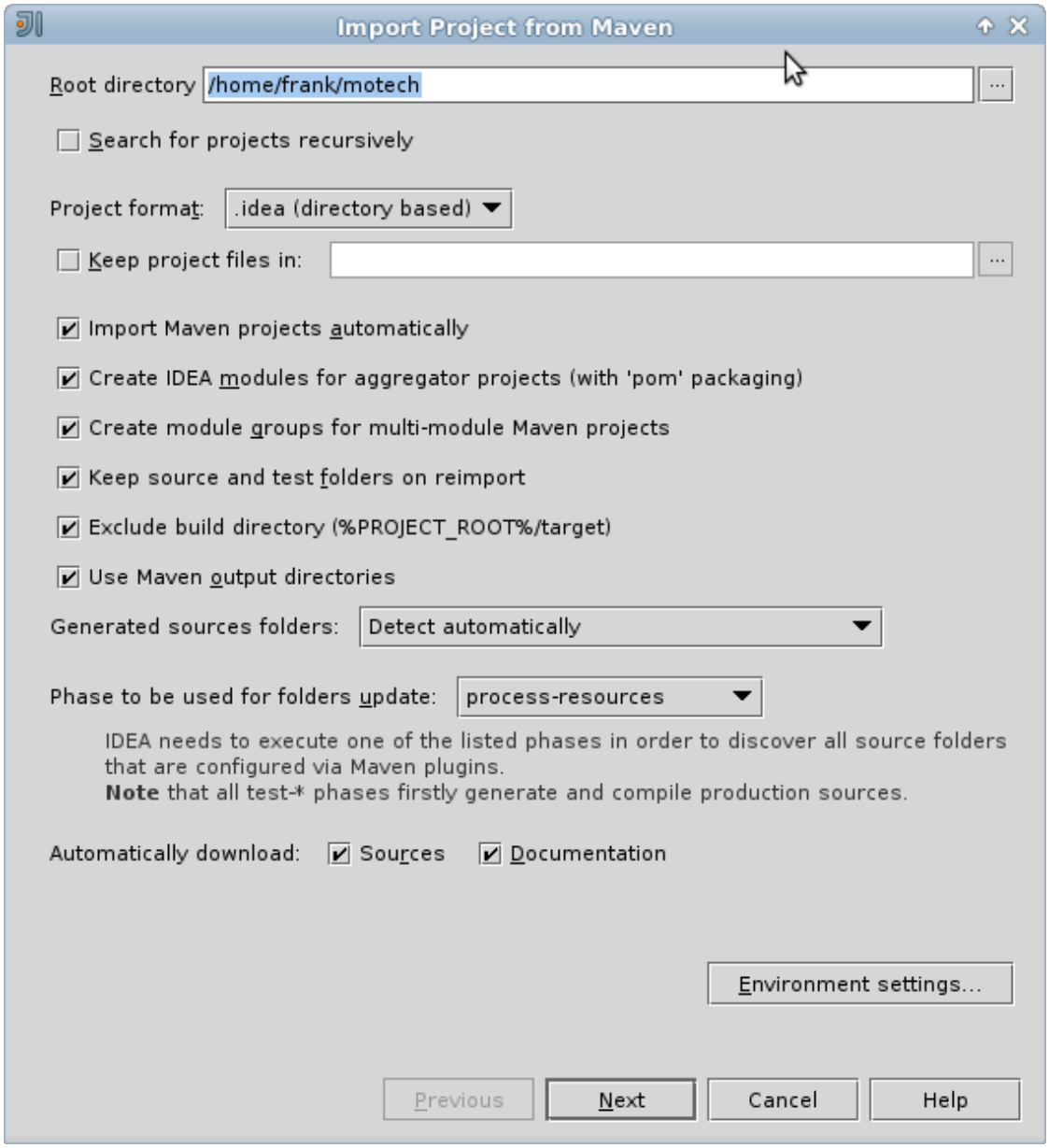
Installing the IDE, Intellij IDEA Community Edition & open MOTECH project

1. Go to the Jetbrains home page and click on Download Now in the Community Edition box, then expand the file to your home directory.

2. From a terminal, assuming you extracted IntelliJ to ~/idea-IC-129.713, start IntelliJ

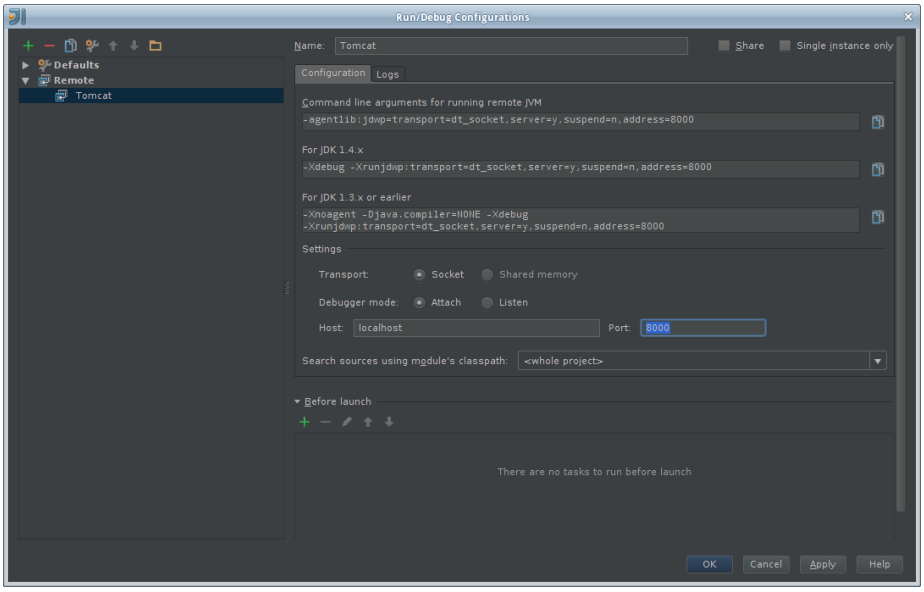
```
$ ~/idea-IC-129.713/bin/idea.sh
```

3. Select Import Project

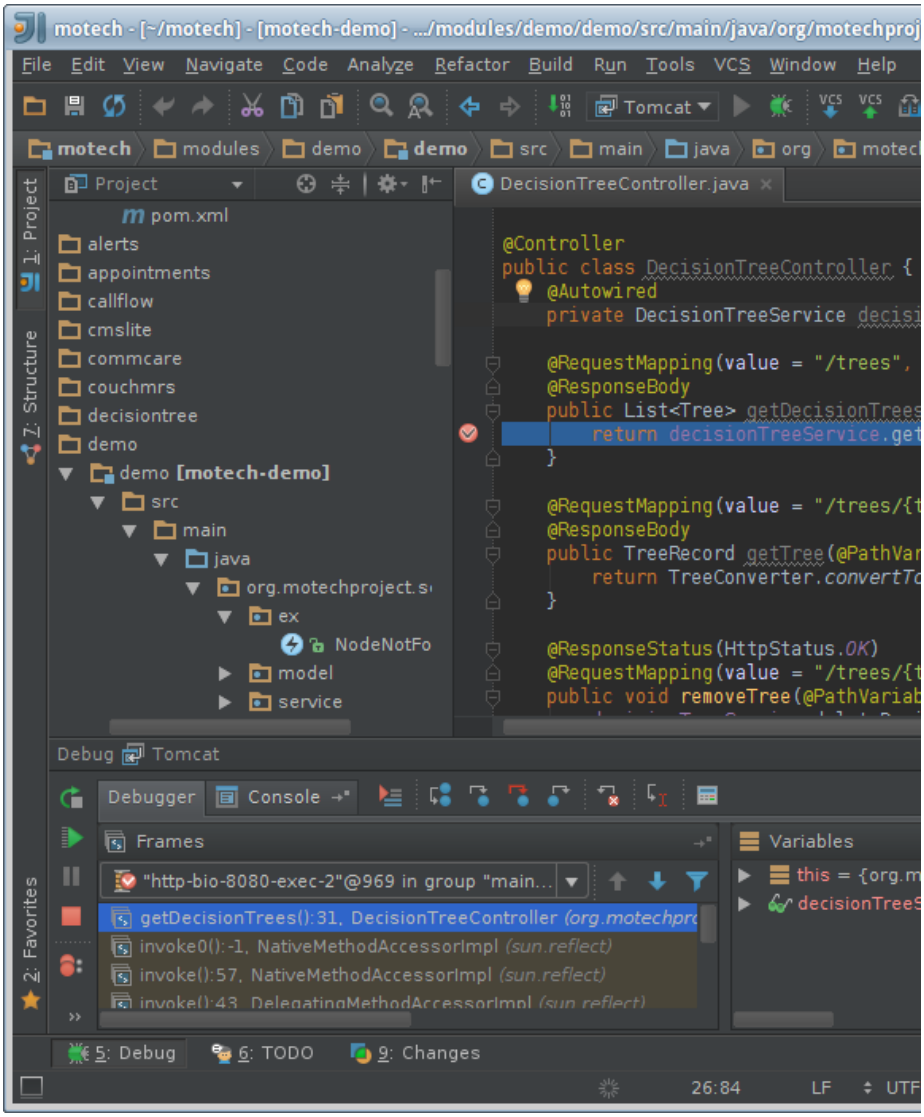


1. Select ~/motech/pom.xml, a dialog box will appear. Set the options as shown:
2. Click Next
3. In Select Profiles, do not select any profile, click Next
4. In Select Maven projects to Import, there should only be one project: org.motechproject:motech:0.20-SNAPSHOT, click Next
5. In Please select project SDK, if the 1.7.0_21 is present, select it, otherwise add it:
6. Click +
7. Select JDK
8. Select /home/frank/jdk1.7.0_21, then click OK
9. Click Next

10. Click Finish
11. Background processes will take a long time
12. You can also create a menu launcher, so you can start IntelliJ from the gui:
    (a) From the Tools menu select Create Desktop Entry
    (b) A IntelliJ menu item will be created in the Development application group
    (c) Debug demo module in IntelliJ
    (d) Start IntelliJ (from the command line, or from launcher icon if you created one)
    (e) It'll automatically open the motech project (if it was the last project you worked on)
    (f) From the Run menu select Edit Configurations
    (g) Click on the green +
    (h) Select Remote
    (i) Give a name to your Run/Debug configuration and change the port to 8000 as:



    (j) Hit OK
    (k) Set a breakpoint somewhere in the demo module code, i.e.:
    (l) From the Run menu, select Debug 'Tomcat' where Tomcat is the name of your configuration.
    (m) In the browser go to the place that will hit the breakpoint, i.e.: if you setup a breakpoint as in the previous screen, then in the Demo module, click the Decision Trees tab, and you should hit the breakpoint!



5.1.2 Installing MOTECH Using Docker ("Beta")

Note: These instructions assume you're running on Ubuntu. This setup is also possible on Mac OSX but the steps are slightly different. We hope to provide OSX instructions in future. Docker Compose isn't currently supported in Boot2Docker because it uses Tiny Core Linux. The Docker team is working on native Windows support and will integrate Docker Compose in the future. If using Windows, it's best to setup an Ubuntu virtual machine using Vagrant and installing it to the Vagrant virtual machine. Make sure to dedicate at least 2GB of RAM to this virtual machine.

This document provides instructions for creating a MOTECH environment using Docker containers. These instructions are "in beta" (the *official* installation guide is still the one *here*), but many members of the MOTECH dev team have been following this approach with success. This installation method is much faster than the official route.

There are two supported ways to install MOTECH with Docker:

1. As an implementer - follow this approach if you want to install a released version of MOTECH.
2. As a developer - follow this approach if you will be developing MOTECH and want to build the platform and modules from source code.

Get Docker, Docker-Compose and motech-docker

Whether you're installing as an implementer or a developer, you'll need Docker and Docker-Compose:

Docker

1. Follow the instructions on the Docker website to get the latest version of Docker.
2. Execute the following to configure Docker to work for non-root users:

```
sudo groupadd docker
sudo gpasswd -a ${USER} docker (logout and re-login)
sudo service docker restart
```

Docker-Compose —

Execute the following to install Docker-Compose in Linux:

```
sudo curl -L https://github.com/docker/compose/releases/download/1.2.0/docker-compose-`uname -s`-`una
sudo chmod +x /usr/local/bin/docker-compose
```

If you get a memory error, you may need to run these commands in root (sudo -i)

motech-docker

Clone the motech-docker project from GitHub or download it as a zip file and extract it. You'll need to run all commands from the motech-docker directory.

```
sudo apt-get install git
git clone https://github.com/motech/motech-docker
cd motech-docker
```

Implementer Setup

Go to your motech-docker directory. To setup as an implementer (everything is automagically installed):

```
./setup_as_imp.sh
```

Type the following to start MOTECH in the background:

```
docker-compose up -d
```

Voila! MOTECH has started. Wait a little bit (about 30s) then direct your browser to: http://localhost:8080/motech-platform-server

Note: 'docker-compose up' ERASES ALL YOUR DATA (well not really all, but pretend it does). You have to run it at least once to setup MOTECH. If you run it again, it'll erase everything you did in MOTECH. It's useful to start afresh, but remember: it nukes everything!

Developer Setup

Go to your motech-docker directory. To setup as a dev:

```
./setup_as_dev.sh
```

Type the following to start all the pieces that MOTECH needs to run in the background:

```
docker-compose up -d
```

Once you start the containers with the docker-compose up -d command above and *before* you build MOTECH for the first time. If you wish to add additional modules to MOTECH, then you can either use the Admin UI or copy them into /root/.motech/bundles directory of the container.

Conveniently, the container's /root/.motech/bundles directory is exposed as the docker-motech-bundles directory (with a-rw access) in your home directory (also note that the container's /root/.motech/config dir is also exposed as ~/docker-motech-config). So, you can either manually copy the binaries you require, or you can create a symbolic link to ~/docker-motech-bundles from ~/.motech/bundles.

Assuming the latter, and that you never built MOTECH before, you'd run the following commands:

```
# go to your home dir
cd
# create the .motech dir
mkdir .motech
# create the symlink
ln -s ~/docker-motech-bundles .motech/bundles
```

If you built MOTECH before, you can just delete the bundles directory and create the symlink using the command above.

Build, deploy and run MOTECH: see :doc:dev_install.

Note: For your convenience, the max upload in the Tomcat Manager is already increased to accept the MOTECH war.

Some Useful Docker Compose Commands

Stop MOTECH

```
docker-compose stop
```

Restart MOTECH

```
docker-compose start
```

Watching logs

To watch all the logs (very verbose):

```
docker-compose logs
```

To watch only the tomcat logs:

```
docker-compose logs tomcat
```

See the sections in the generated docker-compose.yml to see what other logs you can watch.

5.1.3 Configuring MOTECH

There will be more text here. This doc may belong in "Setting up a Development Environment" or in "Getting Started".

5.2 Introduction

Motech provides Maven archetypes in its Nexus repository which you can use to create a new Motech module. The archetypes supply basic source code needed for a new module, as well as configuration files for packaging the module as a bundle to be loaded into a Motech server.

The first archetype is the *minimal bundle archetype*. This supplies just enough source code and configuration to make a "Hello World" module.

Additional archetypes can add functionality to the minimal archetype:

- The *http bundle archetype* adds a servlet to respond to HTTP requests, and a simple web user interface.
- The *repository bundle archetype* adds a repository layer for storing and retrieving data from `MOTECH's data services`.
- The *settings bundle archetype* adds a properties file to store custom module configuration, and exposes the configuration through Motech's web user interface

Any combination of these additional archetypes may be added to the minimal archetype.

5.2.1 Minimal Bundle Archetype

To create a new minimal bundle from the minimal bundle archetype, use the following command:

```
mvn archetype:generate -DinteractiveMode=false -DarchetypeRepository=http://nexus.motechproject.org/c
```

This will create a new Maven project in your current directory.

This is a long command. Here is an explanation of the parameters:

parameter	*value*	*explanation*
-DinteractiveMode	false	no need to wait for user input
-DarchetypeRepository	http://nexus.motechproject.org/content/repositories/release	where to find the archetype
-DarchetypeGroupId	org.motechproject	group name for the archetype
-DarchetypeArtifactId	minimal-bundle-archetype	which archetype to use
-DmotechVersion	0.26-SNAPSHOT	Motech version to use with the new module
-DgroupId	org.motechproject	group name for the new module
-DartifactId	motech-test-module	artifact name for the new module
-Dpackage	archetype.test	Java package for new module classes
-Dversion	0.1-SNAPSHOT	version of the new module itself
-DbundleName	"Archetype Test Module"	name of the new module

5.2.2 HTTP Bundle Archetype

To create a new bundle that has HTTP support, use the following two commands from the same directory.

Create a minimal bundle with configuration modified for HTTP:

```
mvn archetype:generate -DinteractiveMode=false -DarchetypeRepository=http://nexus.motechproject.org/c
```

Note the new parameter:

-Dhttp	true

Add new source files from the HTTP archetype:

```
mvn archetype:generate -DinteractiveMode=false -DarchetypeRepository=http://nexus.motechproject.org/c
```

Note the new archetype Id:

-DarchetypeArtifactId	http-bundle-archetype

> **Attention:** For the names of the new controllers in angular you should add prefix associated with the module so that the name is unique.

5.2.3 Repository Bundle Archetype

To create a new bundle that has support for `MOTECH's data services`, use the following two commands from the same directory.

Create a minimal bundle with configuration modified for repository:

```
mvn archetype:generate -DinteractiveMode=false -DarchetypeRepository=http://nexus.motechproject.org/c
```

Add new source files from the repository archetype:

```
mvn archetype:generate -DinteractiveMode=false -DarchetypeRepository=http://nexus.motechproject.org/c
```

5.2.4 Settings Bundle Archetype

To create a new bundle that has module settings support, use the following two commands from the same directory.

Create a minimal bundle with configuration modified for settings:

```
mvn archetype:generate -DinteractiveMode=false -DarchetypeRepository=http://nexus.motechproject.org/c
```

Add new source files from the settings archetype:

```
mvn archetype:generate -DinteractiveMode=false -DarchetypeRepository=http://nexus.motechproject.org/c
```

5.2.5 Combined Bundle Archetype

The minimal bundle archetype can be supplemented with any combination of additional archetypes. To create a bundle that uses them all, use all the following commands from the same directory.

Create a minimal bundle with configuration modified for all additional archetypes:

```
mvn archetype:generate -DinteractiveMode=false -DarchetypeRepository=http://nexus.motechproject.org/c
```

Add source files from all the additional archetypes:

```
mvn archetype:generate -DinteractiveMode=false -DarchetypeRepository=http://nexus.motechproject.org/c
```

```
mvn archetype:generate -DinteractiveMode=false -DarchetypeRepository=http://nexus.motechproject.org/c
```

```
mvn archetype:generate -DinteractiveMode=false -DarchetypeRepository=http://nexus.motechproject.org/c
```

5.2.6 Using Archetypes Locally

You can also use the archetypes locally, without the Motech Nexus repository. First, you must build the archetypes locally. You can either follow the `developer guidelines` to set up your developer environemt, or to build locally without commiting:

```
git clone https://github.com/motech/motech/
cd motech
mvn clean install
```

Then you can use the archetypes from your Maven local catalog:

```
mvn archetype:generate -DinteractiveMode=false -DarchetypeCatalog=local -DarchetypeGroupId=org.motech
```

Note the new parameter:

-DarchetypeCatalog	local

5.3 Developing and Submitting a Patch

We use a web-based code review system called Gerrit. Using this system, anyone can comment on a proposed change before it is merged to our Git repository. It's pretty easy to use; the instructions below will get you started.

5.3.1 Create a Gerrit Account

1. Navigate to http://review.motechproject.org/
2. Click sign in on top-right
3. Select Open ID provider (only **Ubuntu One** and **Yahoo!** accounts are permitted at the moment)
4. Select user name
5. Upload your SSH public key

5.3.2 Configuring Your Git Client to Use Gerrit

Follow these steps once for each *MOTECH code repository* that you clone.

1. Get source code

   ```
   git clone ssh://<userid>@review.motechproject.org:29418/motech
   ```

2. Set up review branch

   ```
   cd motech
   git config remote.origin.push refs/heads/*:refs/for/*
   ```

3. Install change-id generation hook

```
scp -p -P 29418 <userid>@review.motechproject.org:hooks/commit-msg .git/hooks/
```

5.3.3 Development Workflow

1. Checkout to feature branch

   ```
   git checkout -b newfeature
   ```

2. Make changes/test/multiple commits
3. When ready to submit changes: update master, squash commits and merge feature branch

   ```
   git checkout master && git pull --rebase
   git merge --squash newfeature
   git gui
   ```

4. Edit commit message using the proper *commit message format*
5. Push changes

   ```
   git push origin
   ```

5.3.4 Submitting Changes (Patch Set) to Incorporate Review Comments

If you've received some code review feedback and you'd like to make some changes, follow the steps below to add your changes as a new "patch set" to the existing Gerrit code review.

1. **Checkout patch from gerrit change:**
   (a) Navigate to http://review.motechproject.org/#/c/<change id>/
   (b) Copy pull url under patch set section and run
2. Make changes
3. Copy change ID from Gerrit (top section in Gerrit change page)
4. Amend change ID in commit message
5. Squash commits
6. Push changes

5.3.5 Pushing to Remote Branches (Not for Review)

This practice enables developers to share in-progress feature work with others without actually submitting the changes for review.

1. Use branch namespace dev

   ```
   git checkout -b dev/newfeature
   git add . && git commit -m "message"
   git push -u origin dev/newfeature:dev/newfeature
   ```

2. Once done with feature, squash commits and merge with master. Submit for review as mentioned above.

5.3.6 Additional Information

- http://review.motechproject.org/Documentation/user-upload.html
- http://review.motechproject.org/Documentation/user-changeid.html

5.4 MOTECH Code Repositories

Each repository can be cloned from GitHub or Gerrit. If you're only interested in a copy of the code and will not be contributing, use the GitHub repo (no sign-in required). Otherwise use the Gerrit repo (sign-in required) where each commit will trigger a Jenkins build and be submitted for code review. Jenkins is our continuous integration (CI) system. Once your change is approved and merged by an authorized Gerrit reviewer, it will show up on the GitHub repo.

5.4.1 MOTECH Platform

The platform repo contains the motech-platform-server Tomcat servlet. In addition it also contains the essential Admin, Config, Tasks, Motech Data Services, Email, and Scheduler modules.

- GitHub Repository

```
git clone https://github.com/motech/motech
```

- Gerrit Repository

```
git clone ssh://<userid>@review.motechproject.org:29418/motech
```

5.4.2 MOTECH Modules

This repo is the home of all optional MOTECH modules.

- GitHub Repository

```
git clone https://github.com/motech/modules
```

- Gerrit Repository

```
git clone ssh://<userid>@review.motechproject.org:29418/modules
```

5.5 Commit Message Format

To ensure that our commit messages are both concise and informative, all MOTECH committers are asked to follow the git commit message format outlined below. For reference, Linus Torvalds provides a description of a good commit message here.

5.5.1 Format

1. First line is the Jira issue ID + subject (max 50 chars)
2. Leave a blank line (if not, the entire message is interpreted as subject)

3. Summary (ensure it is wrapped to a reasonable number of characters, e.g. 72, because git log does not handle wrapping).

The description of the change should be both brief and informative. The Linux kernel documentation has this to say about good commit summaries:

> ...the "summary" must be no more than 70-75 characters, and it must describe both what the patch changes, as well as why the patch might be necessary. It is challenging to be both succinct and descriptive, but that is what a well-written summary should do.

5.5.2 Author/Committer

There is no need to include the names of the developers who developed the change in the subject line. If there is more than one person working on a change, the committer is asked to include the –author parameter in his/her git commit to specify the second person's name.

5.5.3 Example

Here is an example of a MOTECH commit message:

> MOTECH-678 Moves campaign modules to github
>
> Disables pillreminder, message-campaign and scheduletracking modules from main pom.xml as they've moved to a new repo on github - https://github.com/motech/platform-campaigns
>
> Change-Id: I5964249887160c868fa9598c413aebb93a49fa32

5.6 Coding Conventions

Table of Contents

5.6.1 Introduction

Goals

The primary goal of the MOTECH coding guidelines is to minimize the cost of maintaining and innovating on code through its lifetime.

Additional goals are:

- Increased developer efficiency when working with code written by others.
- Promoting engineering excellence.
- Compliance with legal and company policies.
- Consistency of the codebase.

There are many guidelines in this document. The decision to introduce a guideline as well as choices between alternative approaches was informed by the following assumptions:

- The benefit of each guideline must significantly surpass the cost of enforcing it.
- During its lifetime, each unit of code is read many more times than it is written or modified. Therefore, the guidelines are optimized for code readability. Cost of writing or modifying code is a secondary consideration.

Many of the following guidelines are adapted from Sun's Java Coding Conventions document (PDF).

Guideline Presentation

The guidelines are organized as simple recommendations using **Do**, **Consider**, **Avoid**, and **Do not**. Each guideline describes either a good or bad practice and all have a consistent presentation. Good practices have a check ( ) in front of them, and bad practices have an 'x' ( ) in front of them. The wording of each guideline also indicates how strong the recommendation is.

A **Do** guideline is one that should be always followed.

On the other hand, **Consider** guidelines should be generally followed, but if you fully understand the reasoning behind a guideline and have a good reason to not follow it anyway, it is ok to break the rule.

Similarly, **Do not** guidelines indicate something you should never do.

Less strong, **Avoid** guidelines, indicate that something is generally not a good idea, but there are known cases where breaking the rule makes sense.

Some more complex guidelines are followed with additional background information, illustrative code samples, and rationale.

5.6.2 Coding Guidelines

Naming

Class and Interface Names

Do use PascalCasing (the first letter of each internal word is capitalized) for interface, and class names. Class names should be nouns. Keep your class names simple and descriptive. Use whole words — avoid acronyms and abbreviations. If the abbreviation is much more widely used than the long form, such as URL or HTML, capitalize only the first letter of the acronym.

```
class Raster;
class SurveyResults;
class HtmlView;
```

Service Interfaces & Implementations

Do append Impl to the implementations class names and place the implementations in a /impl/ subdirectory. So for example the Foo service interface should have the following structure:

/ Foo.java /impl/

FooImpl.java

Method Names

Do use camelCasing for method names (the first letter is lowercase, with the first letter of each additional word capitalized). Methods should be verbs, for example:

```
run();
runFast();
getBackground();
```

Variables

All instance, class, and class variables are in camelCase. Additional words start with capital letters. Variable names should be short yet meaningful. The choice of a variable name should be mnemonic — that is, designed to indicate to the casual observer the intent of its use. One-character variable names should be avoided with the possible exception of temporary "throwaway" variables, e.g. for loops. Even in these cases, more readable names can be provided (e.g. "surveyIndex" instead of "i").

Do not use a prefix for member fields or methods (for example do not start your names with: underscore, m, s, etc.)

Do use camelCasing for member variables

Do use camelCasing for parameters

Do use camelCasing for local variables

Do not prefix enums or classes with any letter

Correct:

```
public class Button
```

Incorrect:

```
public class CButton
```

Do not make local declarations that hide declarations at higher levels. For example, do not declare a previously occurring variable name in an inner block:

```
int count;
...
func() {
    if (condition) {
        int count; // DON'T DO THIS!
        ...
    }
    ...
}
```

Do not declare more than one variable per line, even if the language supports it.

Correct:

```
int startIndex;
int endIndex;
```

Incorrect:

```
int startIndex, endIndex;
```

Do not assign a value to more than one variable per statement, even if the language supports it.

Correct:

```
int surveyCount = 10;
int farmerCount = 10;
```

Incorrect:

```
int surveyCount = farmerCount = 10;
```

Constants

Do name constants with all uppercase words separated by underscores.

```
int MIN_WIDTH = 4;
int MAX_WIDTH = 999;
```

Enum Values

Do name enum values the same way as constants - all uppercase, with words separated by underscores.

```
public enum Day {
    SUNDAY, MONDAY, TUESDAY, WEDNESDAY,
    THURSDAY, FRIDAY, SATURDAY
}
```

File Names

Do not have more than one public type in a source file. Each Java source file contains a single public class or interface.

Do name the source file with the name of the type it contains. For example, MotechScheduler class should be in the MotechScheduler.java file.

Do use the same casing when mapping the type name to file name.

File Content

Do put package and import statements (in that order) directly following the copyright banner, and prior to the class definition:

```
import java.applet.Applet;
import java.util.List;
import java.util.Map;
```

Do group class members into the following sections in the specified order:

1. Static fields
2. Instance fields
3. Constructors
4. Methods
5. Inner classes

Do order fields by public, then protected, then private.

Do group methods by related functionality.

Consider organizing overloads from the simplest to the most complex number of parameters (which often corresponds to complexity of the body).

Do not declare imports not used within the file.

Code Comments

Do use code comments to document code whose operation is not self-evident to the professional developer (e.g. code reviewer). For example, consider commenting:

- Pre-conditions not evident in code, e.g. thread-safety assumptions
- Complex algorithms
- Complex flow of control, e.g. chained asynchronous calls
- Dependencies on global state
- Security considerations
- Return values, e.g. returning either an object or null
- DateTime parameters, are we expecting UTC or local date/times, or is the timezone encapsulated in the DateTime object?

Avoid using comments that repeat self-commenting information found in many code structures. For example, do not add vacuous comments such as "Constructors", "Properties", "Using Statements". Avoid commenting:

- Type declarations (e.g. method signatures)
- Assertions
- Method overloads
- Well-understood patterns (e.g. enumerators)

Do use Javadoc comments before your public field and method definitions.

```
/**
 * Short one line description.
 *
 * Longer description. If there were any, it would be
 * here.
 *
 * @param  variable Description text text text.
 * @return Description text text text.
 */
```

Do use // commenting style for both single and multi-line prose comments. For example:

```
// This method assumes synchronization is done by the caller
Byte[] ReadData(Stream stream)
```

or

```
// This AsyncResult implementation allows chaining of two
// asynchronous operations. It executes the second operation only
// after the first operation completes.
```

Avoid leaving unused code in a file, for example by commenting it out. There are occasions when leaving unused code in a file is useful (for example implementing a single feature over multiple checkins), but this should be rare and short in duration.

Avoid using #if/#endif commenting style for purposes other than excluding code from the compilation process:

```
Console.WriteLine("Hello");
#if false
    Console.WriteLine("Press  to continue...");
    Console.Readline();
#endif
    Console.WriteLine("Finished");
```

Syntax

Braces

Do use braces with if, else, while, do, and dowhile statements.

Do not omit braces, even if the language allows it.

Braces should not be considered optional. Even for single statement blocks, you should use braces. This increases code readability and maintainability.

```
for (int i = 0; i < 100; i++) {
    doSomething(i);
}
```

The only exception to the rule is braces in case statements. These braces can be omitted as the case and break statements indicate the beginning and the end of the block.

```
case 0:
    doSomething();
    break;
```

Do place opening braces on the same line as their associated statement, with a space before the opening brace.

Do place closing braces in their own line.

Do align the closing brace with its corresponding opening statement.

```
if (someExpression) {
    doSomething();
}
```

Indents and Tabs

Do use 4 consecutive space characters for indents.

Do not use the tab character for indents.

Do indent contents of code blocks.

```
if (someExpression) {
    doSomething();
}
```

Do indent case blocks even if not using braces.

```
switch (someExpression) {
    case 0:
        doSomething();
        break;
}
```

Spacing

Do use a single space after a comma between function arguments.

Correct:

```
read(myChar, 0, 1);
```

Incorrect:

```
read(myChar,0,1);
```

Do not use a space after the parenthesis and function arguments

Correct:

```
createFoo(myChar, 0, 1)
```

Incorrect:

```
createFoo( myChar, 0, 1 )
```

Do not use spaces between a function name and parenthesis.

Correct:

```
createFoo()
```

Incorrect:

```
createFoo ()
```

Do not use spaces inside brackets.

Correct:

```
x = dataArray[TDG:index];
```

Incorrect:

```
x = dataArray[TDG: index ];
```

Do use a single space before flow control statements

Correct:

```
while (x == y)
```

Incorrect:

```
while(x==y)
```

Do use a single space before and after comparison operators

Correct:

```
if (x == y)
```

Incorrect:

```
if (x==y)
```

Do use a single space before and after arithmetic operators

Correct:

```
x = x + y;
```

Incorrect:

```
x = x+y;
```

Do use a single space before and after assignment operations

Correct:

```
x = y;
```

Incorrect:

```
x=y;
```

Do use a space or newline before and after the conditional operator

Correct:

```
x = ((p > q) ? y : z);
```

Incorrect:

```
x = (p > q)?y:z;
```

Do use parenthesis around the conditional operator

Correct:

```
x = (foo ? y : z);
```

Incorrect:

```
x = foo ? y : z;
```

Do use a single space for class derivation

Correct:

```
class Button extends Control
```

Do use a single space for variable declarations.

Do not use multiple spaces to try and align variable names separately from their types.

Correct:

```
int groupSize = 10;
```

Do use a single blank line in between method definitions.

Page Width

Do try to limit the width of your code to 120 characters.

Do Use common sense. If changing an existing file with obvious 80 column formatting keep it that way. If a particular line will be much more readable but break the width rule, use common sense.

Changing Existing Code

Do comply with the 'when in Rome, do as the Romans do' principle. When working on an existing file, please limit your changes to the issue you're working on so as to not overwhelm the person reviewing your code with unnecessary changes.

Do feel responsible to fix a really messy file. Making overall changes to a file to make it look good, outside the needs of your actual change, is an acceptable exception to the preceding rule when dealing with a real mess.

5.7 Code Review Workflow and Checklist

5.7.1 Principles

In general, we aim to have all code reviewed prior to submission, with very few – hopefully rare – exceptions Please be conscientious about turning around reviews quickly – investing in good code review karma will pay off when it comes time for you to solicit reviews from your peers

5.7.2 Exceptions to Code Review Requirement

Urgent fix for a build break. Unless the fix is completely trivial, please remember to get a code review after submission.

5.7.3 Workflow

1. **Write code:** Follow the development workflow described *here* to develop code changes and submit for review
2. **Choose code reviewers:** Add your chosen code reviewers as reviewers in Gerrit. For most changes, one reviewer should be sufficient; if your change is complex and/or represents core platform functionality, it is ideal to involve more than one reviewer. If a particular individual has significant expertise in the code you are changing, it's a good practice to include that person as a revewier. If not, it's a great practice to choose someone from a different organization to review your code, in order to facilitate cross-pollination of ideas across the project.
3. **Wait for reviews to come in:** Code reviewers should attempt to provide comments within 2 business days – in the case of time-sensitive fixes, reviews should be turned around faster. (TIP: If you aren't seeing email notifications for code reviews assigned to you, check your junk mail folder.)
4. **Address feedback:** It should be possible to address most code review feedback prior to submission. If changes are non-trivial, create a new patch and send it out for re-review as described *here*.
5. **Track any future work identified:** For larger issues that are flagged during code review, discuss with the reviewer whether the feedback should block submission. If not, open a new Trello card to track the suggestion for a later code change.
6. **Submit:** Once reviewers have signed off, submit change. Gerrit has a button that allows you to merge directly from the site.

5.7.4 Checklist

Correctness

- Does the code completely and correctly implement the design?
- Does every code change in the submission map to a ticket in Jira?

Maintainability

- Does the code make sense?
- Code reuse
  - Are there any blocks of repeated code that could be encapsulated into methods?
  - Can the desired functionality be achieved by reusing any existing code?

- Does the code comply with the accepted *Coding Conventions*? (Indentations, varable/method names, bracket style, commenting, etc.)

Error Handling

- Are all thrown exceptions handled properly?
- Does the code catch (or throw) general exceptions, e.g. java.lang.Exception?
- If a method could return null, does the caller check for null?

Control Flow and Structure

- Are loop termination conditions obvious and invariably achievable?

Test Coverage

- Have sufficient unit tests been provided to cover the basic functionality being provided?
- Do unit tests cover all error conditions of a method call?

Documentation

- Are the classes properly documented? Do they contain at least a short description of what they do?
- Is the code self-documenting? If its very cryptic what a piece of code does or why, it should either get reworked or properly documented.
- Are documentation changes required? If the change is modifying the API or behavior of already documented components, it should also contain appropriate updates to the documentation.
- Is new documentation required? In case of a new module or new functionality, it should get properly documented.

The documentation should either be included with the change or a ticket for documenting the functionality should exist. If there is no documentation and no ticket on Jira, a ticket for adding the documentation should get created on Jira with "Inbox" set as the fix version. That ticket will get discussed during our weekly review call and scheduled accordingly.

Resources

The resources below contain a number of good ideas (and some bad ones). Many of the checklist items above were borrowed from these documents, with some tweaking.

- http://www.javacodegeeks.com/2011/06/not-doing-code-reviews-whats-your.html
- http://scientopia.org/blogs/goodmath/2011/07/06/things-everyone-should-do-code-review/
- http://www.perlmonks.org/?node_id=744932
- https://wiki.openmrs.org/display/docs/Code+Review+Checklist

5.8 Authoring Documentation

This document provides information specific to setting up and authoring Sphinx documentation for MOTECH. The Sphinx Documentation Guide is also a good resource.

Each MOTECH repository contains a *docs* directory populated with reStructured Text (reST) files. These files are built by Sphinx and hosted at http://readthedocs.org. A Sphinx plugin, called Javasphinx, builds Javadoc for the MOTECH codebase.

5.8.1 Installing Sphinx and Javasphinx

If you are working on MOTECH documentation, it is helpful to build the docs locally and review changes in advance of checkin. The steps below will get your environment configured to build Sphinx docs.

Install python utils

```
sudo apt-get install python-setuptools python-dev python-pip
```

Install javasphinx and motechjavasphinx

```
sudo apt-get install libxslt-dev libxml2-dev zlib1g-dev
cd docs
sudo pip install -r requirements.txt
```

5.8.2 Building Docs

ReadTheDocs.org automatically builds Sphinx projects, but it is a good idea to build your documents and make sure they render correctly before you push your changes up to the code repository.

To build Sphinx docs, go to the docs directory and type:

```
make html
```

You can then use a web browser to view docs/build/html/index.html and make sure everything looks good.

5.8.3 Authoring and Submitting New Documents

To create a new documentation topic, simply create a file with extension *.rst* under the docs directory. Create a table-of-contents entry for your new doc in the appropriate index.rst file. The reStructuredText Primer is good to have handy as you write your doc.

When your document is ready for review, follow the instructions for *creating and submitting a patch* to submit it for code review.

5.9 Managing External OSGi dependencies

5.9.1 OSGi dependencies

All MOTECH modules run in a Felix OSGi framework instance embedded within the platform war. Because of this all libraries you make use of in your module should be packaged as actual OSGi bundles. Since everything that is (theoretically) needed to make a library ready for OSGi is adding a few entries in the **META-INF/MANIFEST.MF** many popular java libraries are already OSGi enabled by default. However this might not be the case for all the

libraries you would wish to use. While you are urged to use libraries that are OSGi bundles, it is still possible to use non OSGi compatible libraries. This document will take you through the process of evaluating a given dependency and using it in MOTECH.

5.9.2 Recognizing whether the library is an OSGi bundle

First of all you should check whether the library itself is an OSGi dependency. An easy way to do this is to open its jar file using any of the tools that allow browsing zip files (jars are actually zip files after all). After opening the jar, check the **META_INF/MANIFEST.MF** file for presence of OSGi related entries such as: Bundle-SybmolicName, Bundle-Version, Import-Package, Export-Package, etc. If they are present it means that the library supports OSGi.

Note: Always check the source library for OSGi support before doing anything else, like looking for alternatives or OSGifying it yourself.

5.9.3 Finding an existing OSGi version of the dependency

If your dependency is not an OSGi bundle at source, don't worry, it's possible that someone has already OSGified the bundle for you. You can search for an OSGified version using Google (or any other search engine). There are however a few particular repositories excelling in providing OSGi versions of common libraries:

- **Spring EBR - http://ebr.springsource.com/repository/app/ - while the future is not certain for this project, the** Enterprise Bundle Repository remains a good source of OSGi bundles. This repository is proxied through our Nexus.
- **ServiceMix Bundles - http://servicemix.apache.org/developers/source/bundles-source.html - this repository is a good place** to search for existing OSGi compatible versions of popular java libraries. The bundled versions from this repository go to maven central.
- **Pax Tipi - https://ops4j1.jira.com/wiki/display/PAXTIPI/Pax+Tipi - OPS4J's answer to this problem. Pax Tipi is an umbr** for third-party artifacts repackaged as OSGi bundles. These OSGi bundles are all available from Maven Central with groupId org.ops4j.pax.tipi.

If none of these resources provide the OSGi version of the library you are looking for, it seems you will have to get your hands dirty and follow the good old principle - if you want something done, do it yourself.

5.9.4 Creating an OSGi-ready version of the dependency

Creating an OSGi version of given library is not an awfully complicated process. All you have to do is to create a version with the correct **META-INF/MANIFEST.MF** file. There are tools that can help you with this and you are not required to have access to the library code in order to accomplish this task. In MOTECH we use the Felix Bundle Plugin for Maven for creating bundles. A dependency to OSGify should be added as a maven module in our External OSGi Bundles repository. Everything you should need to place there is the pom file with the correct configuration for creating the bundle. The following code is an example of such a pom.xml file for creating a bundle from org.example.example-artifact, version 1.0:

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

<project xmlns="http://maven.apache.org/POM/4.0.0"
         xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
         xsi:schemaLocation="http://maven.apache.org/POM/4.0.0 http://maven.apache.org/xsd/maven-4.0

    <modelVersion>4.0.0</modelVersion>

    <properties>
```

```
    <example.version>1.0<example.version>
</properties>

<parent>
    <groupId>org.motechproject</groupId>
    <artifactId>external-osgi-bundles</artifactId>
    <version>1.0.8</version>
</parent>

<!-- We prefix the groupId for our bundles with org.motechproject -->
<groupId>org.motechproject.org.example</groupId>
<artifactId>example-artifact</artifactId>
<!-- The release tag property is important. It allows us to update the version of the bundle
     without making changes to the base version -->
<version>${example.version}-${release.tag}</version>


<!-- The library we are OSGifying has to be declared as a dependency -->
<dependencies>
    <dependency>
        <groupId>org.example</groupId>
        <artifactId>example-artifact</artifactId>
        <version>${example.version}</version>
        <scope>provided</scope>
    </dependency>
</dependencies>

<build>
    <plugins>
        <!-- This will make Felix scan the library for imports -->
        <plugin>
            <artifactId>maven-dependency-plugin</artifactId>
            <executions>
                <execution>
                    <id>unpack-sources</id>
                    <goals>
                        <goal>unpack</goal>
                    </goals>
                    <phase>package</phase>
                    <configuration>
                        <outputDirectory>${project.build.directory}/sources</outputDirectory>
                        <artifactItems>
                            <artifactItem>
                                <groupId>org.example</groupId>
                                <artifactId>example-artifact</artifactId>
                                <version>${example.version}</version>
                                <classifier>sources</classifier>
                            </artifactItem>
                        </artifactItems>
                    </configuration>
                </execution>
            </executions>
        </plugin>

        <!-- This configuration will tell the Felix bundle plugin to generate
             the bundle for the library. The original library will be embedded in the newly
             created bundle jar -->
        <plugin>
```

```
                <groupId>org.apache.felix</groupId>
                <artifactId>maven-bundle-plugin</artifactId>
                <version>2.3.4</version>
                <extensions>true</extensions>
                <configuration>
                    <instructions>
                        <!-- All library packages that are supposed to be exposed must be declared as
                        <Export-Package>
                            org.example;version=${project.version},
                            org.example.subpackage;version=${project.version}
                        </Export-Package>
                        <!-- You can specify additional imports that were not found by Felix -->
                        <Import-Package>
                            hidden.import,
                            *
                        </Import-Package>
                        <!-- Bundle metadata for the newly created bundle -->
                        <Bundle-SymbolicName>${project.artifactId}</Bundle-SymbolicName>
                        <Bundle-Vendor>Example.com</Bundle-Vendor>
                        <!-- We embed the original library -->
                        <Embed-Dependency>example-artifact;inline=true</Embed-Dependency>
                        <Embed-Transitive>true</Embed-Transitive>
                    </instructions>
                </configuration>
            </plugin>
        </plugins>
    </build>
</project>
```

It is important to note that by adding the release tag to the version of the bundle, we allow ourselves to make updates i.e. add an export we forgot about, without changing the base version. The value of the release tag is in the main pom of the external OSGi bundles repository.

5.9.5 Releasing a new version of external OSGi dependencies

For making a release of the external bundles, three steps are required:

1. Update the version in the parent pom. You can use the following command, where X is the new old number increased by one.

   ```
   mvn versions:set -DnewVersion=1.0.X
   ```

2. Increment the release tag in the parent pom. For example if its value is r30, change it to r31
3. Trigger the external-osgi-bundles build on Jenkins, it should trigger automatically when new commits come in to the repository.

After the repository gets updated, the release tag values defined in parent poms for the MOTECH platform and MOTECH modules must be updated in order to use the new versions.

Note: We would prefer to keep the number of bundles we maintain to a minimum. So please only commit additional bundles when it's necessary and you are absolutely sure the library is not already an OSGi bundle itself and there are no existing OSGi compatible versions. Also, if you have knowledge that one of the dependencies we maintain was OSGified at source, please let us know, so that we can get rid off the burden of maintaining it. Remember that in an ideal world, the external-osgi-bundles repository would not exist.

5.10 Project Management

There will be text here.

5.11 Release Process

MOTECH releases are created roughly quarterly, with some bigger releases taking longer. See the *Roadmap* for guidance on when to expect the next release.

This page provides step-by-step instructions for creating an official release. Many of these steps require elevated permissions in Gerrit and Jenkins, and are intended to be performed by a member of the release management team.

5.11.1 Create the Release Branch and Associated CI jobs

There are two ways to create a release branch and the associated release jobs in our CI:

1. Using our handy release script that automates most of the steps
2. Manually

Option 1 - Release Script

We have a Python script at https://github.com/motech/release that automates all of the steps for branching. It expects the following paramaters:

Required

--jenkinsUsername	A user in jenkins who has permission to create new jobs
--jenkinsPassword	The password for the jenkins user
--gerritUsername	A username in gerrit who is in the Bypass Review group.
--version	The version of the release i.e. 0.22
--developmentVersion	The next development version on the branch i.e. 0.22.1-SNAPSHOT
--nextMasterVersion	The next working version on master i.e. 0.23-SNAPSHOT

Optional

--buildDirectory	The base location to check out all source to. Will delete if it exists. Defaults to ./builds
--verbose	Be a little chatty on stdout

Option 2 - Manual Process

1. Cut and paste the following commands to set your environment variables to the appropriate values:

   VERSION={version}

   USERNAME={username}

2. Create branches for the release (one for each repo):

```
mvn release:branch -!DbranchName=0.$VERSION.X -Dscm.connection=scm:git:ssh://$USERNAME@revie
mvn release:branch -!DbranchName=0.$VERSION.X -Dscm.connection=scm:git:ssh://$USERNAME@revie
```

3. In the Jenkins UI, create CI jobs for the new branches. List of jobs:

   Platform-{VERSION}.X

   Modules-{VERSION}.X

The most straightforward way to create the jobs is to copy the config from a previous release. Any other method will be error-prone, as there are many fields to configure, several of which are hidden under "Advanced" sections.

5.11.2 Test the Release Candidate

After the release branch has been created, user acceptance testing (UAT) begins. The release management team will make an initial pass over the tickets that were resolved for the release, and close those that are low-priority for UAT. All others should be tested and closed before building the official release. If testing reveals that an issue requires more work, the ticket should be reopened with fixVersion=Inbox so that it can be triaged at the next Inbox Review Meeting. A decision will be made as to whether the issue should be fixed on the release branch or moved to the next release.

Additional release criteria or functional areas for UAT may be added for specific releases, depending on the needs of partners or the desire to stress major new functionality. In future, release criteria will also include API freeze tools passing (API freeze tools aren't in place yet), documentation requirements, and a code coverage minimum.

5.11.3 Build the Release

Once the build has been tested and you are ready to create the official release, you can do so from the Jenkins UI.

1. Trigger the platform release build from the platform-{VERSION}.X CI job first.
2. Wait and ensure that the platform build completes successfully.
3. Trigger the modules release build from the modules-{VERSION}.X CI job.

If Something Goes Wrong...

If the release build fails for any reason, it will be necessary to do some cleanup before attempting a retry.

1. Depending on how far along the build process got before failing, some artifacts may have been uploaded to Nexus. You will need to expand the directories for each bundle and delete these artifacts manually if they exist. Subsequent release attempts will fail if these artifacts aren't removed.
2. **Jenkins makes two checkins when preparing the release and preparing for the next development iteration. These will need**

   [maven-release-plugin] prepare for next development iteration

   [maven-release-plugin] prepare release motech-{VERSION}

Push changes to the remote branch via Gerrit using git push origin HEAD:refs/for/{VERSION}.X

3. **If the tag for the new release was created, it will need to be deleted.** git push –delete origin motech-{VERSION}

After these issues are addressed (and the root cause of the release failure is investigated/fixed), it should be safe to retry the release build.

5.11.4 Release Notes

Release notes should be published under the *Release Notes* section on our documentation site. They should contain pointers to the binaries and source code, a summary of major changes delivered in the new release, and a list of known issues (with workarounds when applicable).

5.12 Browser Compatibility

5.12.1 Supported Browsers

MOTECH project uses new technologies (HTML 5, CSS3) and new frameworks like AngularJS, Bootstrap which means that the compatibility of the browsers depends on the extent to which these frameworks support old browsers.

Specifically, we support the latest versions of the following browsers and platforms.

Platform	IE 10+	Chrome	Firefox	Safari	Opera
Windows	Supported	Supported	Supported	Not Supported	Supported
Mac OS X	Not Supported	Supported	Supported	Supported	Supported
Linux	Not Supported	Supported	Supported	Not Supported	Supported

Unofficially, MOTECH should look and behave well enough in Chromium for Linux, though it is not officially supported.

5.12.2 Internet Explorer Compatibility

Bootstrap is built to work best in the latest desktop browsers, meaning older browsers might display differently styled, though fully functional, renderings of certain components.

AngularJS 1.3 has dropped support for IE8. Read more about it on this blog.

The AngularJS project currently supports and will attempt to fix bugs for IE9 and above. Their continuous integration server runs all the tests against IE9, IE10, and IE11. See Travis CI and ci.angularjs.org.

5.12.3 Screen Resolution

Currently we tested MOTECH on screen resolution 1024 x 768 px and higher, but the best results will be achieved using resolution 1680 x 1050 px or higher, especially for wide tables. That means we do not support mobile and tablet devices.

5.12.4 Browser Settings

All browsers must have cookies and JavaScript enabled to use MOTECH.

5.12.5 Our approach to maintain compatibility with browsers

1. Even if we declare the latest HTML 5 in MOTECH, we try not to use the latest tags, if possible.
2. Newer versions of browsers provide debugging tools directly or as a plugin. These developer tools also allow you to inspect specific HTML and alter CSS styles.

3. We do not use inline JavaScript events inside HTML markup. An example would be <button onclick="validate()">Validate</button>. This practice breaks the clean separation that should exist between markup, presentation, and behavior.

5.13 Static resources - faster UI development

This short manual explains how to enable hot deployment of static files (.js,.css files etc). This will help to view the changes made to the static files without having to redeploy the module/bundle each time a change is made. After following the described steps, static files will be read from specified location on your local machine, rather than from bundle resources.

5.13.1 Details

You need to do two things:

- Set environment as development
- For the bundle for which static resources are to be hot deployed, create an environment variable with the "bundle symbolic name" but after replacing all special characters and spaces with underscore; this variable should contain a path to the module resources directory

Note: By default, the bundle symbolic name is constructed, based on the groupId and artifactId, in the following way: **{groupId}.{artifactId}**.

5.13.2 Example

Let's assume the following properties of a module:

groupId:	org.motechproject
artifactId:	sms
bundle symbolic name:	org.motechproject.sms
path to resources:	/home/me/modules/sms/src/main/resources

Set up two environment variables (from the same shell which starts up tomcat). Remember to replace all spaces and special characters with an underscore. Note, that the configuration is case-sensitive.

```
export ENVIRONMENT=DEVELOPMENT
export org_motechproject_sms=/home/me/modules/sms/src/main/resources
```

In case you find the changes are not being reflected even after correctly setting up the environment variables, clear browser cache and delete the directory ${tomcat_installation_dir}/work/Catalina/localhost/${motech_dir}/

5.13.3 Hot deployment with Docker container

It is also possible to configure hot deployment of static files running MOTECH with the Docker container. To do so, you must make some edits in the **fig.yml** file.

First of all, you must link your local directory, containing module resources to a volume visible in the Docker container. This can be achieved by adding an appropriate entry in the volumes section of the tomcat configuration. The entry must be in the form of "yourLocalDirectory: virtualDirectoryOnDocker". This is how exposing your local "/home/you/modules/sms/src/main/resources" directory could look like:

```
volumes:
    - /home/you/modules/sms/src/main/resources:/home/modules/sms/resources
```

Adding this entry will cause that your local directory will be visible in the Docker container, under the virtual path "home/modules/sms/resources".

You must also set up the environment variables in your configuration. In your file find tomcat section and then "environment". Add required entries. The names and values of the environment variables must follow the rules stated above. Note, however, that Docker will only see resources that you have manually exposed as volumes (virtual directories within Docker). It might look like this:

```
environment:
    JAVA_OPTS: -Xms1024m -Xmx2048m -XX:MaxPermSize=1024m
    DB_TYPE: mysql
    DB_USER: root
    DB_PASSWORD: password
    DB_DRIVER: com.mysql.jdbc.Driver
    ENVIRONMENT: DEVELOPMENT
    org_motechproject_sms: home/modules/sms/resources
```

The variables must be added in the "variableName: variableValue" format. Set environment as development and add paths to the resource directories, for the modules you wish to have hot deployment for.

The complete configuration for the tomcat section in the **fig.yml** could look like this:

```
tomcat:
    image: motech/tomcat:7.0.53
    ports:
        - "8080:8080"
        - "8000:8000"
    links:
        - couchdb
        - db
        - activemq
    environment:
        JAVA_OPTS: -Xms1024m -Xmx2048m -XX:MaxPermSize=1024m
        DB_TYPE: mysql
        DB_USER: root
        DB_PASSWORD: password
        DB_DRIVER: com.mysql.jdbc.Driver
        ENVIRONMENT: DEVELOPMENT
        org_motechproject_sms: home/modules/sms/resources
    volumes:
        - /home/you/docker-motech-config:/root/.motech/config
        - /home/you/docker-motech-bundles:/root/.motech/bundles
        - /home/you/modules/sms/src/main/resources:/home/modules/sms/resources
```

CHAPTER 6

Deployment

6.1 Load Balancing with Apache 2 WebServer - Sticky Session

Table of Contents

6.1.1 Overview

Sticky Session is a method used with Load Balancing, to achieve server affinity. In other words, it assigns a particular client with a particular server instance behind Load Balancer, so that HTTP session doesn't get lost across application instances. It is essential if we are deploying Motech in a cluster configuration and we want to be able to access its UI. This tutorial describes two methods of setting up Sticky Session feature with Apache Server used for Load Balancing.

6.1.2 Method 1: Using existing session cookie

Attention: This method requires Apache Tomcat to be used as instance server.

In this approach you will have to configure both Load Balance Server (LBS) and all Tomcat Instance Servers (IS). For session tracking, we will use existing session cookie - JSESSIONID - and let IS to modify it a bit.

In LBS VirtualHost configuration, you have to provide names for your instances (If you haven't done so yet). To do so, for each BalancerMembers you have to specify a route parameter:

```
BalancerMember http://{i'th instance ip:port} route=tomcat_instance_i
```

where tomcat_instance_i is the i'th instance unique name.

In the same file you have to provide session ID cookie name to LBS under its VirtualHost Proxy definition:

```
<Proxy balancer://mycluster>
    (...)
    ProxySet stickysession=JSESSIONID
```

```
    (...)
</Proxy>
```

Finally, for each tomcat, you have to edit $CATALINA_HOME/conf/server.xml. In the <Engine> node add jvmRoute attribute which corresponds with BalancerMember route of this particular IS from LBS configuration:

```
(...)
<Engine name="Catalina" defaultHost="localhost" jvmRoute="tomcat_instance_i">
(...)
```

After restarting all Instance Servers and Load Balance Server, sticky session should work.

6.1.3 Method 2: Using additional session cookie

This approach requires changes only to Load Balance Server (LBS) VirtualHost configuration. For session tracking, a brand new cookie named ROUTEID will be used. It will be managed by LBS, so we need to enable its ability to modify headers (thus cookies). The advantage of this method is that it doesn't require any Instance Server (IS) configuration, so it potentially can be used with any backend web servers.

Same as in previous method, you have to name your instances:

```
BalancerMember http://{i'th instance ip:port} route=tomcat_instance_i
```

Then, ensure that 'headers' Apache module is enabled. To enable this module in Ubuntu, you can use following command:

```
sudo a2enmod headers
```

Next, you have to tell LBS to store its own cookie (ROUTEID) on client side. That cookie will contain instance name (route):

```
<VirtualHost *:80>
    (...)
    Header add Set-Cookie "ROUTEID=.%{BALANCER_WORKER_ROUTE}e; path=/" env=BALANCER_ROUTE_CHANGED
    (...)
</VirtualHost>
```

Finally, tell LBS the cookie name:

```
<Proxy balancer://mycluster>
    (...)
    ProxySet stickysession=ROUTEID
    (...)
</Proxy>
```

After restarting Load Balance Server, sticky session should work.

6.2 Using MOTECH with multibyte characters

Table of Contents

6.2.1 Overview

In most of Tomcat, MySQL and PostgreSQL versions by default **ISO-8859-1** character encoding is used. If you want to use multibyte characters like for example polish or chinese letters this tutorial will help you.

6.2.2 MySQL multibyte characters support

If you want to add support for a new character set that includes multi-byte characters, you need to add the following lines into **my.cnf** file:

```
[client]
default-character-set = utf8

[mysqld]
collation-server = utf8_unicode_ci
init-connect='SET NAMES utf8'
character-set-server = utf8
```

After settings this, restart the MySQL server.

6.2.3 PostgreSQL multibyte characters support

To create a new database with UTF-8 encoding you have to type the following line in terminal:

```
postgres=# create database motech_data_services with encoding='UTF-8' lc_collate='en_US.utf8' lc_cty
```

However you can encounter the following error:

```
ERROR: new encoding (UTF8) is incompatible with the encoding of the template database (SQL_ASCII)
```

It means that you have to change template's encoding to UTF-8. You can do this like so:

```
postgres=# update pg_database set datallowconn = TRUE where datname = 'template0';
postgres=# \c template0.
template0=# update pg_database set datistemplate = FALSE where datname = 'template1';
template0=# drop database template1;
template0=# create database template1 with template = template0 encoding = 'UTF8';
template0=# update pg_database set datistemplate = TRUE where datname = 'template1';
template0=# \c template1
template1=# update pg_database set datallowconn = FALSE where datname = 'template0';
```

After setting this, you can once again type the database creation command.

6.2.4 Tomcat multibyte characters support

To support the feature you have to configure the connector in Tomcat's **server.xml** like so :

```
<Connector port="8080" protocol="HTTP/1.1"
           connectionTimeout="20000"
           redirectPort="8443"
           URIEncoding="UTF-8"
           useBodyEncodingForURI="true"/>
```

Now if you want to use multibyte characters in a request url properly you can add manually the following header to the request headers:

```
Content-Type : application/x-www-form-urlencoded; charset=UTF-8
```

However you can set this globally by adding the following filter in Tomcat's **web.xml**, otherwise your controller's request mapping will not work correctly :

```
<filter>
    <filter-name>encodingFilter</filter-name>
    <filter-class>org.springframework.web.filter.CharacterEncodingFilter</filter-class>
    <init-param>
        <param-name>encoding</param-name>
        <param-value>UTF-8</param-value>
    </init-param>
    <init-param>
        <param-name>forceEncoding</param-name>
        <param-value>true</param-value>
    </init-param>
</filter>
<filter-mapping>
    <filter-name>encodingFilter</filter-name>
    <url-pattern>/*</url-pattern>
</filter-mapping>
```

CHAPTER 7

Demos

7.1 Demo: Hello World

Table of Contents

7.1.1 Overview

This "Hello World" tutorial aims to get you started with Motech development. The tutorial is separated into two phases. In the first, we generate the Hello World module incrementally from a series of archetypes provided by the Motech platform. We briefly tour the project code, configuration, and layout that make up a minimal Motech module, and then add in additional archetypes to support web requests and data services. Finally, we build and deploy the module to our Motech server.

In the second phase, we introduce an essential feature of the Motech platform: the Event system. The Event system allows our module to communicate with other modules by emitting and listening for events. We also introduce the Tasks module, which allows us to wire up Motech events using a graphical user interface. Finally, we introduce Motech Data Services, which we use to persist entities. Using these tools, we modify the archetype-generated code to save a new record whenever users request a URL defined by our module.

This tutorial assumes you are at least somewhat familiar with Java, Maven, and the Spring Framework, and have completed the instructions to *set up your development machine*. We use Intellij IDEA 13 Community Edition as the

integrated development environment, but this is not a requirement.

This tutorial was written with Motech 0.24.

7.1.2 Generating the Module from Archetypes

Motech is a modular system. Modules are units of functionality that encapsulate application-specific business logic in a reusable package. The Open Service Gateway initiative specification (OSGi) provides the framework to describe this modular architecture: each Motech module is also an OSGi bundle. The OSGi host manages the lifecycle of a module (adding, starting, stopping, and removing), and allows a module to expose services for use by other modules.

To mitigate some of the complexity of configuring OSGi bundles, the Motech platform provides a number of Maven archetypes. The archetypes are also modular: each new module begins from the minimal bundle archetype, and additional capabilities–serving web requests, managing data, and more–are enabled by adding their respective archetypes to the minimal bundle's foundation.

Generating the Minimal Bundle

The minimal bundle archetype generates the basic project layout and configuration sufficient to begin module development in Motech. To generate the bundle, open a terminal from the directory you wish to contain the project, then enter the following command:

```
mvn archetype:generate -DinteractiveMode=false -DarchetypeRepository=http://nexus.motechproject.org/c
```

The flags given with the Maven command instruct Maven to generate a minimal bundle from the 0.24-SNAPSHOT branch of the Motech repository. Once the process completes, the generated project resides in a folder named after the artifact ID, in this case "helloworld."

Tour of the Minimal Bundle

First, import the module into your favorite IDE. If using Intellij IDEA, from the "Welcome to Intellij IDEA" splash screen, select *Import Project*, navigate to the *helloworld* folder, and click *OK*. Select *Import project from external model*, select *Maven*, and accept the defaults by clicking "Next" and finally "Finish" through the remaining options.

The minimal bundle archetype generates a standard Maven project layout. In the Java folder, the archetype created a service package under our top level `org.motechproject.helloworld package`. Inside the package, the archetype created the interface and implementation of a very simple Spring service. We'll modify this service later in the tutorial to make it a bit more interesting.

As for configuration, the *resources/META-INF* directory contains folders for Motech and Spring XML files. In the *motech* folder, the *applicationContext.xml* file enables Spring component scanning of our base Java package, and declares details of our module as a `ModuleRegistrationData` bean. In the *spring* folder, the *blueprint.xml* file exposes our `HelloWorldService` as an OSGi service. Since we activated repository support when generating the minimal bundle, the `HelloWorldRecordService` is also exposed. This code will be added later in the tutorial, when we add the repository archetype. Similarly, we get an OSGi reference to the `HelloWorldRecordsDataService`, also to be added later. Any additional references to Motech platform services will be added to this blueprint file.

Adding the HTTP Archetype

Now that we have some perspective on the basic module layout, let's add the HTTP archetype into the mix. With a terminal open at the same folder you issued the first Maven command, enter the following:

```
mvn archetype:generate -DinteractiveMode=false -DarchetypeRepository=http://nexus.motechproject.org/c
```

This archetype pulls in additional dependencies to interact with and integration test Spring web servlets. Returning to the IDE, notice that our top level package now contains an additional subpackage called web, with a simple Spring controller. The controller makes our module a little more interesting by injecting our previously generated `HelloWorldService` and providing a URL route to exercise the service's public API.

The HTTP archetype also creates a *webapp* folder in the *resources* directory. This folder contains the module's static files, including HTML partials, stylesheets, JavaScripts, and internationalized messages. Motech modules use the AngularJS framework to drive the front-end client, so the archetype created the top-level module, controllers, directives, and services necessary for a simple Angular client in the *js* folder.

Adding the Repository Archetype

As the final step in setting up our basic Hello World module, let's generate the repository archetype code with the following Maven command:

```
mvn archetype:generate -DinteractiveMode=false -DarchetypeRepository=http://nexus.motechproject.org/c
```

The repository archetype created two new packages, `domain` and `repository`, which contain a simple data model and repository service, respectively. In addition, the archetype added an interface and implementation to the service package, so we can interact with the data layer.

Taking a closer look at the domain and repository packages, the `HelloWorldRecord` is a typical Java bean that models a record with name and message fields. The class-level `@Entity` annotation identifies the record as a data type to be managed and persisted by the core `Motech Data Services (MDS)` module. The MDS `@Field` annotations provide object-relational mappings between the bean's fields and columns in the database. The `HelloWorldRecordsDataService` interface extends the base `MotechDataService` interface, inheriting functionality to provide basic CRUD operations for our `HelloWorldRecord` objects. Using the MDS `@Lookup` annotation, we provide a custom method by which to find a `HelloWorldRecord`, in this case by name. Additional custom lookups can be defined here.

In the service package, the `HelloWorldRecordService` injects the data service and exposes a public interface by which to retrieve and persist records.

Building and Deploying the Module

To build the project in Maven, create a new run configuration in Intellij by clicking *Run -> Edit Configurations...*. Click the green plus sign to add a new configuration, and select Maven. Name the configuration "Maven clean install", enter "clean install" in the command line field, and click *OK*. Finally, click the *Run* button in the upper right hand corner to package the module.

Alternatively, from the command line, change directory to the *helloworld* module's folder and type "mvn clean install".

Once the build succeeds, the project directory will contain our module packaged as a jar in the */target* subfolder. With your Tomcat server running, open a browser and navigate to http://localhost:8080/motech-platform-server/, then log in with your administrator credentials. Click the *Admin* tab, then click *Manage Modules* from the sidebar navigation. In the dropdown labeled *Install module from*, choose *File*, then click the *Select File* and choose our packaged *helloworld-0.1-SNAPSHOT.jar*. Click the *Start on Install* button to toggle the icon to a check mark, and then finally click *Install or Update*.

If all goes well, our *Hello World Module* should appear alongside the other installed modules in the user interface. The state should be "Active."

As a small test of our web controller, navigate your browser to http://localhost:8080/motech-platform-server/module/helloworld/sayHello. The controller should respond with the JSON string `{"message":"Hello World"}`.

7.1.3 An Event-Driven Hello World

In this second phase of the tutorial, we implement a feature that creates a new record whenever a user requests the `sayHello` URL route defined by our module. To accomplish this, we emit an event when the URL is accessed, listen for an event instructing our module to create a new record, and finally wire these two separate events together using the `Tasks` module.

Declaring the Event System Dependency

To include events in our module, we first need to declare the dependency on the event system in our *pom.xml* file. Between the `<dependencies>` opening and closing tags, add the following:

```
<dependency>
    <groupId>org.motechproject</groupId>
    <artifactId>motech-platform-event</artifactId>
    <version>${motech.version}</version>
</dependency>
```

Secondly, our module will hook into Motech's Event Relay, so we need to add a reference to the relay alongside our other OSGi configuration in *blueprint.xml*:

```
<osgi:reference id="eventRelayOsgi" cardinality="0..1" interface="org.motechproject.event.listener.Ev
```

With these dependencies, we are now able to inject the event relay into our own code, send events, and define event listeners.

Event Subjects and Event Parameters

Motech events are identified by a String value called the subject. To encapsulate these values as constant values, let's create an event package under our main module package to contain the following class:

```
package org.motechproject.helloworld.event;

public final class HelloWorldEventSubjects {

    public static final String SEND_HELLO = "helloworld_hello_event";

    public static final String CREATE_HELLO_RECORD = "helloworld_create_hello_record";

    private HelloWorldEventSubjects() {}
}
```

Similarly, each event can be supplied with parameters, represented by a map of key-value pairs in which the keys are also constant String values. This class encapsulates those values:

```
package org.motechproject.helloworld.event;

public final class HelloWorldEventParams {

    public static final String NAME = "name";

    public static final String MESSAGE = "message";

    private HelloWorldEventParams() {}
}
```

Finally, let's create a helper class to simplify packaging our parameters together into a functional Motech event:

```java
package org.motechproject.helloworld.event;

import org.motechproject.event.MotechEvent;

import java.util.HashMap;
import java.util.Map;

public final class HelloWorldEvents {
    public static MotechEvent sendHelloWorldEvent(String name, String message) {
        Map<String, Object> params = new HashMap<>();
        params.put(HelloWorldEventParams.NAME, name);
        params.put(HelloWorldEventParams.MESSAGE, message);
        return new MotechEvent(HelloWorldEventSubjects.SEND_HELLO, params);
    }

    private HelloWorldEvents() {}
}
```

Given a name and a message, our static helper method will bundle up the parameters and return a Motech event, which in turn will be passed to the event relay.

Sending an Event

To send our event, let's modify our implementation of the `HelloWorldService` to send events whenever the `sayHello` method is called:

```java
package org.motechproject.helloworld.service.impl;

import org.motechproject.event.listener.EventRelay;
import org.motechproject.helloworld.event.HelloWorldEvents;
import org.motechproject.helloworld.service.HelloWorldService;
import org.springframework.beans.factory.annotation.Autowired;
import org.springframework.stereotype.Service;

@Service("helloWorldService")
public class HelloWorldServiceImpl implements HelloWorldService {

    @Autowired
    private EventRelay eventRelay;

    @Override
    public String sayHello() {
        eventRelay.sendEventMessage(HelloWorldEvents.sendHelloWorldEvent("HWEvent", "Hello world!"));
        return "Hello World";
    }

}
```

In the `HelloWorldServiceImpl` listing, we declare a private field to contain a reference to the Event Relay. Annotating the field with Spring's `@Autowired` annotation injects the dependency we declared earlier in the *blueprint.xml* configuration file. Then, in the body of the `sayHello` method, prior to the return statement, we use the event relay's `sendEventMessage` method to send our Motech event.

And that's it! Any Motech module that defines a listener for our `SEND_HELLO` subject can inspect our event and its parameters.

Listening for Events

In our Hello World module, we provide a data service layer to create `HelloWorldRecord` instances, but currently do not make use of it. Rather than keep that functionality to ourselves, let's define an event listener that allows other modules to send us requests to create new records. First, create a new subpackage to the event package called handler, then add the following class:

```
package org.motechproject.helloworld.event.handler;

import org.motechproject.event.MotechEvent;
import org.motechproject.event.listener.annotations.MotechListener;
import org.motechproject.helloworld.domain.HelloWorldRecord;
import org.motechproject.helloworld.event.HelloWorldEventParams;
import org.motechproject.helloworld.event.HelloWorldEventSubjects;
import org.motechproject.helloworld.service.HelloWorldRecordService;
import org.springframework.beans.factory.annotation.Autowired;

@Component
public class HelloWorldEventHandler {

    @Autowired
    private HelloWorldRecordService helloWorldRecordService;

    @MotechListener(subjects = {HelloWorldEventSubjects.CREATE_HELLO_RECORD})
    public void handleCreateHelloEvents(MotechEvent event) {
        String name = (String)event.getParameters().get(HelloWorldEventParams.NAME);
        String message = (String)event.getParameters().get(HelloWorldEventParams.MESSAGE);
        HelloWorldRecord record = new HelloWorldRecord(name, message);
        helloWorldRecordService.add(record);
    }
}
```

In our event handling class, the `@MotechListener` annotation does the work of transforming a standard Java method into an event-handling method. Using the annotation's subjects element, we declare our interest in receiving `CREATE_HELLO_RECORD` events. Our event-handling method accepts the `MotechEvent` as an argument, extracts the name and message parameters, creates a new `HelloWorldRecord`, and finally uses the injected data service to persist the new record.

Since we are listening for a different event than the one we emit, there is no interaction between our module's event-emitting functionality and event-listening functionality. In the next section, we look at a way to integrate these two systems with the Tasks module's graphical user interface.

Task Triggers and Task Actions

The Tasks module allows implementers to define interactions between modules through a graphical user interface. A Task consists of a trigger and an action. A trigger is a standard Motech event that has been exposed to the Task module. An action is an event for which the implementer's module provides a listener. Between the trigger and action, an implementer can add filters to control whether a task should run based on conditional logic. Also, tasks may use data loaders to query Motech Data Services for additional information.

In the listings that follow, we will declare a Task channel for our module in the form of a JSON document. We will expose our `SEND_HELLO` event as a task trigger, declare our `CREATE_HELLO_RECORD` event as a Task action, and then link the two in Motech's GUI.

First, create a new file called *task-channel.json* in our module's *main/resources* directory. Each Task channel must define a display name, a module name, and a module version as JSON properties. A Task channel may provide a list of triggers, identified by the `triggerTaskEvents` property. Each trigger must define a display name and event

subject, and may provide a list of parameters passed with the event, in which case each event has an event key and a display name. Similarly, a Task channel may provide a list of Task actions, identified by the `actionTaskEvents` property. Like triggers, each action must provide a display name and an event subject, and an optional list of parameters identified by key and display name:

```
{
    "displayName": "helloworld.task.channel.name",
    "moduleName": "${project.artifactId}",
    "moduleVersion": "${parsedVersion.osgiVersion}",
    "triggerTaskEvents" : [
        {
            "displayName" : "helloworld.task.trigger.send_hello.name",
            "subject" : "helloworld_hello_event",
            "eventParameters" : [
                {
                    "eventKey" : "name",
                    "displayName" : "helloworld.task.trigger.send_hello.param.name"
                },
                {
                    "eventKey" : "message",
                    "displayName" : "helloworld.task.trigger.send_hello.param.message"
                }
            ]
        }
    ],
    "actionTaskEvents" : [
        {
            "displayName" : "helloworld.task.action.create_hello_record.name",
            "subject" : "helloworld_create_hello_record",
            "actionParameters" : [
                {
                    "key" : "name",
                    "displayName" : "helloworld.task.action.create_hello_record.param.name"
                },
                {
                    "key" : "message",
                    "displayName" : "helloworld.task.action.create_hello_record.param.message"
                }
            ]
        }
    ]
}
```

In the listing above, the subjects for the trigger and action are the hard-coded String constants we defined in our `HelloWorldEventSubjects` class. Since our `HelloWorldRecord` provides fields for name and message, we pass that data as parameters on the trigger, and accept that data as parameters of the action.

Throughout the JSON listing, the values we provide as display names are not String literals, but rather references to Strings in our message properties files. To provide the String literals for the references above, open our *main/resources/webapp/messages/messages.properties* file and append the following:

```
#Tasks
helloworld.task.channel.name=Hello World

helloworld.task.trigger.send_hello.name=Hello World
helloworld.task.trigger.send_hello.param.name=Name
helloworld.task.trigger.send_hello.param.message=Message

helloworld.task.action.create_hello_record.name=Create Hello World Record
```

```
helloworld.task.action.create_hello_record.param.name=Name
helloworld.task.action.create_hello_record.param.message=Message
```

Now, all of the components of our task channel will display correct English values in Motech's graphical user interface. Before we turn to the user interface, let's rebuild the module with mvn clean install and install our module in the *Admin -> Manage Modules* user interface, as we did when touring the minimal bundle.

Creating a Task in the User Interface

Once the module has been updated, we'll create the relationship between task trigger and action in the Motech user interface. Navigate to the Tasks interface by clicking *Modules -> Tasks*. Click the *New Task* button and enter "Create Hello Record" in the *Task Name* field. Our module appears in the Trigger dropdown with the default module icon, a clipboard. Click on the icon, and select Hello World from the available triggers.

New buttons will appear allowing us to add filters, data sources, and actions. Click the *Add action* button, then, from the *Channel* dropdown menu, select our Hello World module. Click the *Action* dropdown menu, and select the *Create Hello World Record* action that we defined as part of our Task channel.

Finally, notice that the fields we declared as parameters of our Task trigger appear as draggable elements in the *Available Fields* list. Also notice that our Task action allows us to enter a name and message as text in two form input fields. To complete the association between the trigger and action parameters, drag the *Name* element to the *Name* input field, and the *Message* element to the *Message* input field. Once the form fields are populated, click the *Save & Enable* button to activate our new task.

Final Walkthrough

To review, in the Event-driven Hello World section of the tutorial, we added the Motech event system to our module. We defined two event subjects: a `SEND_HELLO` event that our module emits to other modules in the system, and a `CREATE_HELLO_RECORD` event to which our module listens for and responds. Our module emits the `SEND_HELLO` event whenever the HelloWorldService's `sayHello` method is called (in this case, when a user loads the /sayHello route as defined by the `HelloWorldController`). Using the Tasks module's user interface, we defined a task linking the `SEND_HELLO` and `CREATE_HELLO_RECORD` events as a task trigger and task action, respectively. Our `HelloWorldEventHandler` listens for `CREATE_HELLO_RECORD` and adds new :code:'HelloWorldRecord'instances to our Motech Data Services repository.

To verify that we implemented our feature correctly, load http://localhost:8080/motech-platform-server/module/helloworld/sayHello in your browser. The response sent to the browser remains the same. To check for the new record, click *Modules -> Data Services* in the Motech user interface. Select the *Data Browser* tab, then click on the `HelloWorldRecord` under the heading for our *Hello World* module. In the list of record instances that appear, we should find a record with the name "HWEvent" and the message "Hello world!"

7.2 IVR Demos

First of all, just in case you didn't know, IVR (or Interactive Voice Response) is a system that enables computers to interact with humans using a phone. The computer interacts with the human using pre-recorded or synthesized messages and the human either speaks back or uses her phone keypad (also called DTMF) to interact with the computer.

7.2.1 Initial Setup

In addition to the Motech platform, and the IVR module, you'll also need to build and/or install the SMS Module.

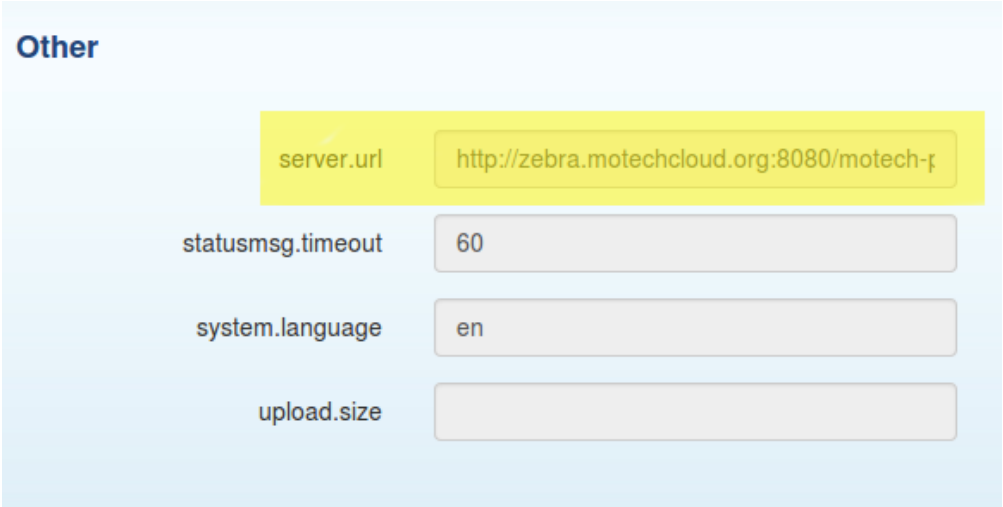
7.2.2 Server Settings

Also be sure that the `server.url` property is properly set to your server's URL and that your server is publicly reachable from the internet. If your server's config source is file based, locate the `motech-settings.properties` file and make sure the `server.url` is set. If your server's config is done through the UI, the navigate to **Admin / Settings** and set the `server.url` property there.

So, for example, if your server's public address was `zebra.motechcloud.org` and it was accessible on port 8080, then you should see:

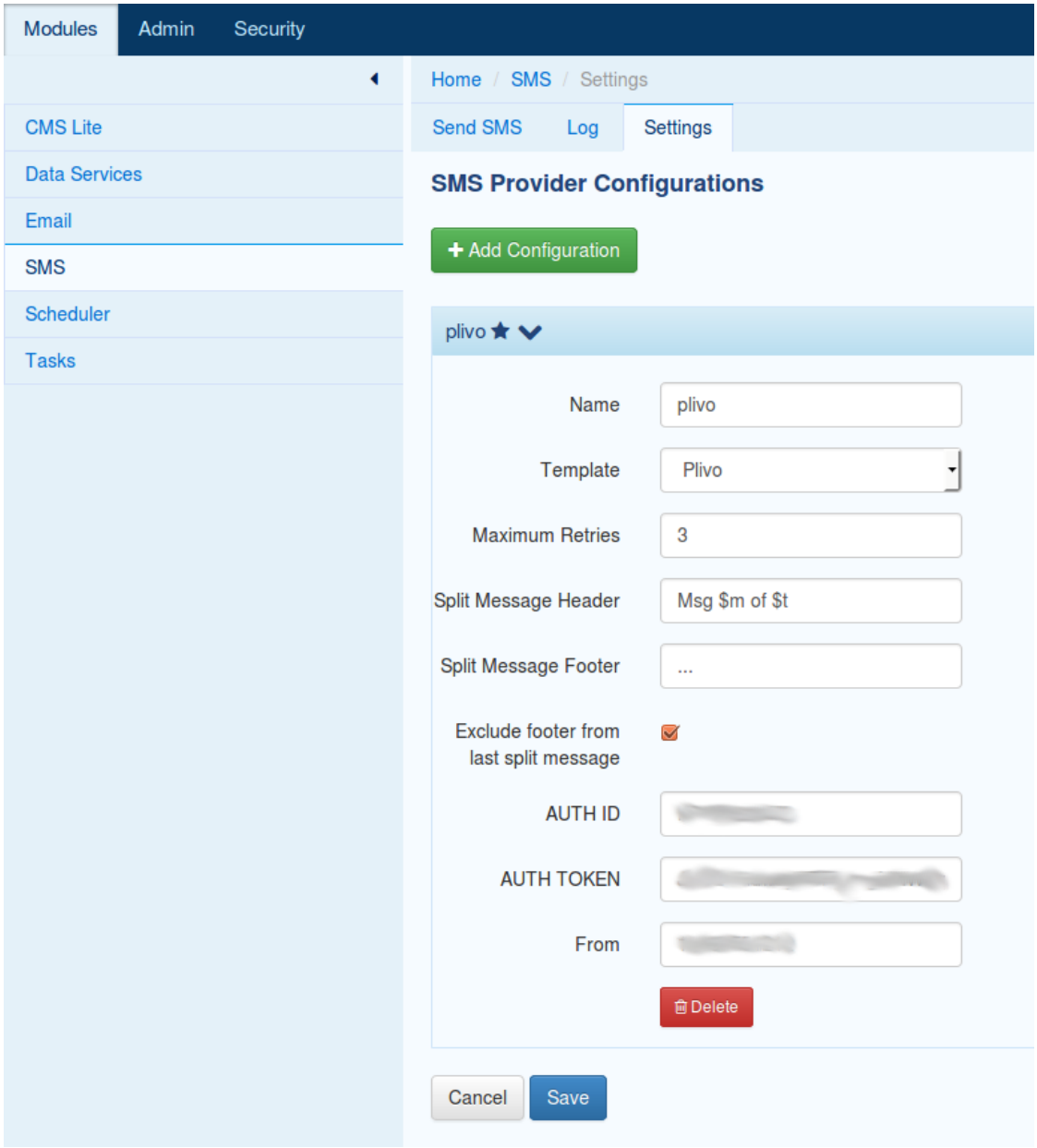
```
server.url=http://zebra.motechcloud.org:8080/motech-platform-server
```

Confirm the setting (or set it [#]) by clicking **Admin / Settings**:



7.2.3 SMS Module Config

For the demos that have you send or receive SMS, you need a valid SMS config. You'll need to establish an account with an SMS provider and then configure the SMS Module accordingly. In these demos we're using Plivo. To confirm your SMS Settings, click **Modules / SMS / Settings**:



If, in addition to sending, you're going to be receiving SMS, you must also tell your SMS provider what to do when they receive an SMS. There typically will be some way of setting that up on their website. They probably need a URL to send an HTTP request to. This is where you give them the address of your Motech server. So if your server is accessible on the web at `http://zebra.motechcloud.org:8080` the complete URL you would provide your SMS provider would be `http://zebra.motechcloud.org:8080/motech-platform-server/module/sms/incoming/plivo` where `plivo` is the name of the SMS Config you created for that SMS provider.

7.2.4 The Demos, Finally...

There are three IVR demos:

Generic VXML IVR Provider Demo: Receiving Calls

You accept phone calls, prompt for and record a code, and then send the code in an SMS to the caller.

In more details:

1. You call the IVR provider [1].
2. The IVR provider calls Motech [2] to request the `hello` VXML template which will tell it what to do. This demo uses VXML but, depending on the provider, it could also be CCXML or some proprietary language.
3. Motech returns the requested `hello` VXML template.
4. The IVR provider executes the instructions in `hello` which prompt the caller to type in a number.
5. The caller (that's you!) types in a number.
6. The IVR provider sends the number to Motech along with a request for the `thankyou` template.
7. Motech receives the number you typed, sends a Motech Event, and returns the requested `thankyou` template.
8. The IVR provider executes the instructions in `thankyou`: say 'Thank you' and hang up.
9. Meanwhile, on the Motech side, the Tasks module was waiting for a *[we received a template request and the value of the* `callStatus` *parameter is ANSWERED]* [3] Motech Event. When it finally receives it, it extracts the number you typed, creates a 'You chose *x*' message and sends an SMS [4] to the number that originated the call, which it also receives as a standard parameter from the IVR provider.
10. You receive 'You picked *x*' SMS. Nifty, eh?

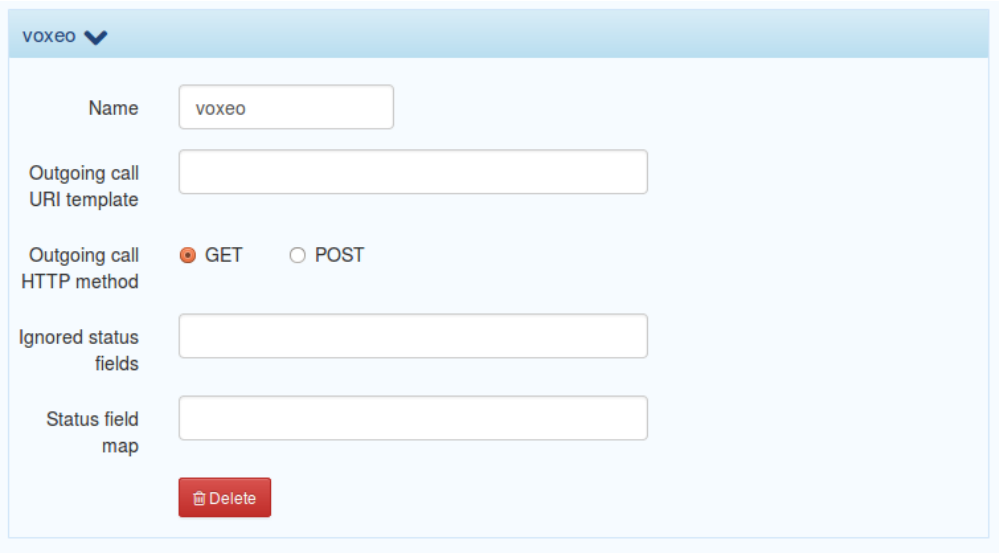
IVR Provider

For this demo we used Voxeo, a generic VXML/CCXML provider.

IVR Settings

We need to create an IVR Config so the IVR module knows what to do when it receives HTTP requests from IVR providers. Click on **Modules / IVR / Settings**:

[1] IVR providers will typically give you a phone number you can use to call your application on their system. They also will require that you give them a publicly accessible URL so they cal request the VXML that you want to be executed when a call is received at this number.
[2] More precisely makes a REST call to the IVR module at the `/template` HTTP endpoint.
[3] To differentiate it from other template requests.
[4] From the SMS provider you configured.



Note: Because we're using the Voxeo IVR provider we named our config **voxeo**. The name isn't important, but it's a good idea to use a simple one as it'll be part of the URL you'll provide your IVR provider so it knows where to get the VXML.

A little VXML

Here's the `hello` VXML template:

```
<?xml version="1.0" encoding="UTF-8"?>
  <vxml version = "2.1">
  <form id="enterCode">
    <field name="code" type="digits?minlength=1;maxlength=1">
      <prompt>
        Hello! Please pick a number between 0 and 9.
      </prompt>
    </field>
    <filled>
      <prompt>
        You picked <value expr="code" />.
      </prompt>
      <assign name="from" expr="session.callerid" />
      <assign name="providerCallId" expr="session.sessionid" />
      <assign name="callStatus" expr="'ANSWERED'" />
      <submit name="sendCode" next="http://zebra.motechproject.org:8080/motech-platform-serv
    </filled>
  </form>
</vxml>
```

Note: The selected number is passed as one of query parameters (`code`) the IVR provider sends along with its request for the `thankyou` template.

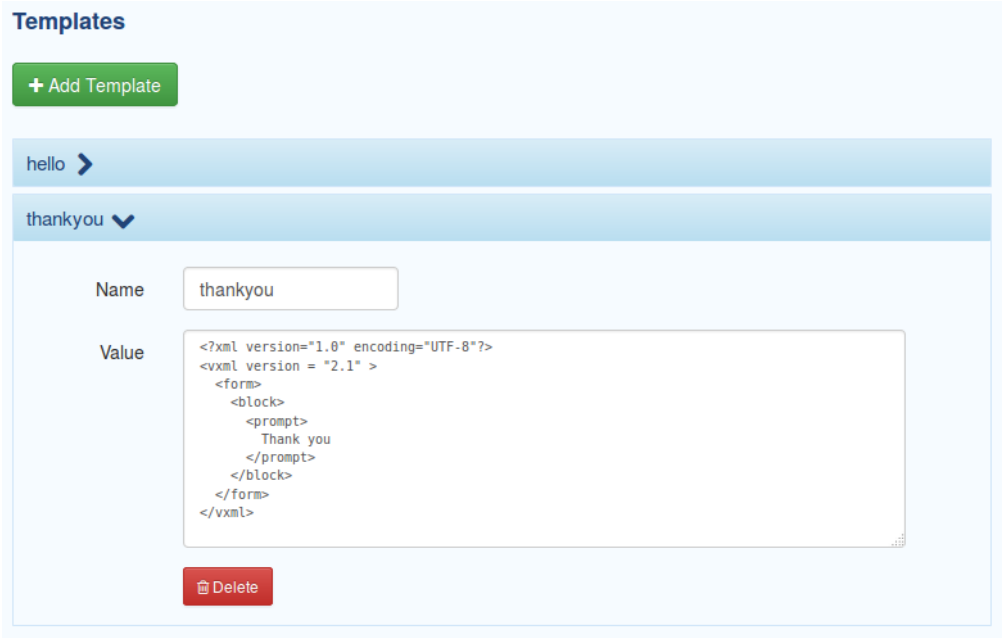
And the `thankyou` VXML template, which simply says 'Thank you' and hangs up:

```
<?xml version="1.0" encoding="UTF-8"?>
<vxml version = "2.1" >
  <form>
```

```
    <block>
      <prompt>
        Thank you
      </prompt>
    </block>
  </form>
</vxml>
```

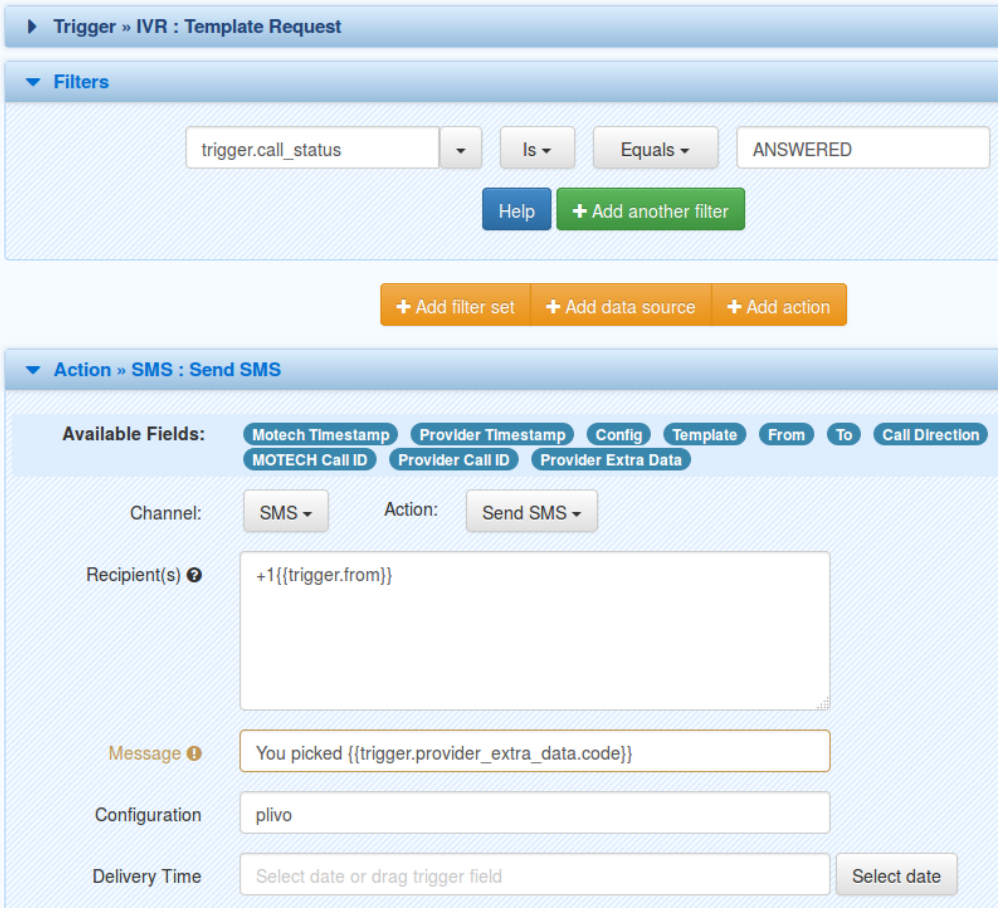
Creating the Templates

To add these two templates, click **Module / IVR / Settings** and then **+ Add Template**:



Creating the Task

We need to create a task where the trigger is an IVR template request where the call status is `ANSWERED` and the action is to send an SMS to the original caller with the code she entered in the message:

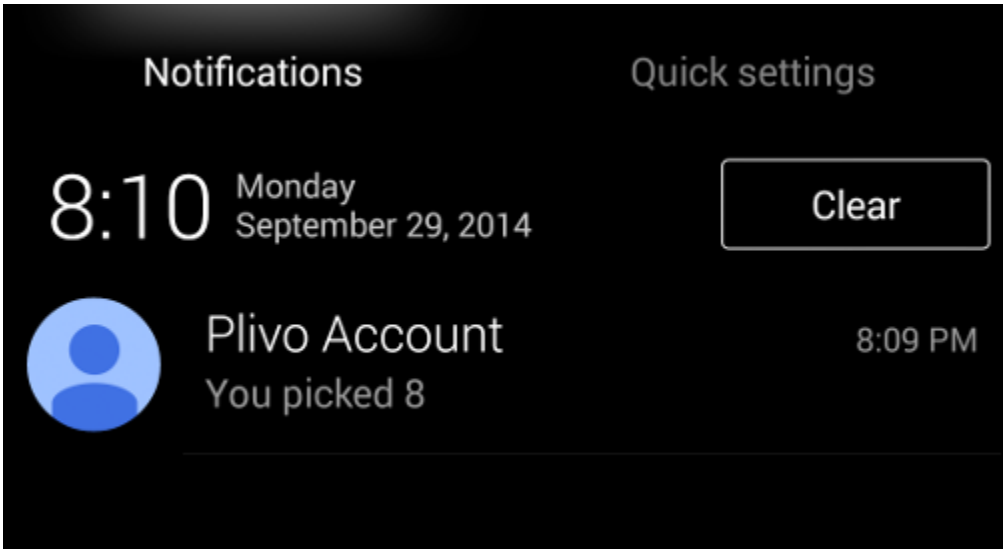


Note: **code** is extracted from the Motech event payload with `{{trigger.provider_extra_data.code}}`

Note: A **+1** is added to the SMS recipient because our sample SMS provider, Plivo, needs it.

Et Voila!

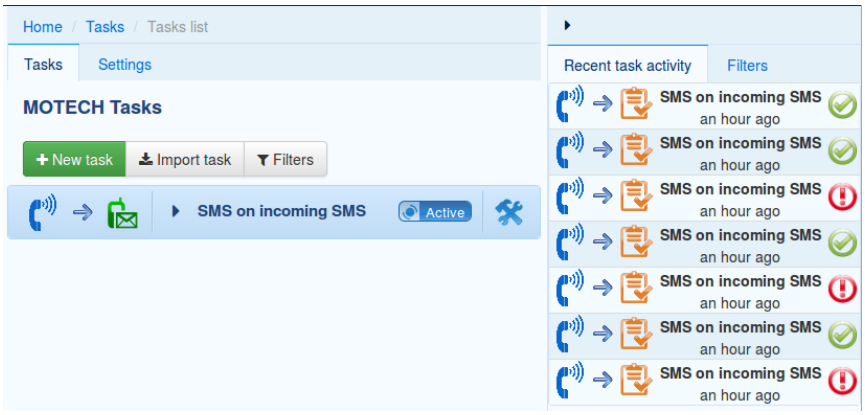
Now call your application at the phone number that your IVR provider gave you, then listen to the "Hello! Please pick a number between 0 and 9." prompt, type in a number (say 8). The IVR system will say "You picked 8. Thank you", then the call will disconnect and soon enough you should receive an SMS:



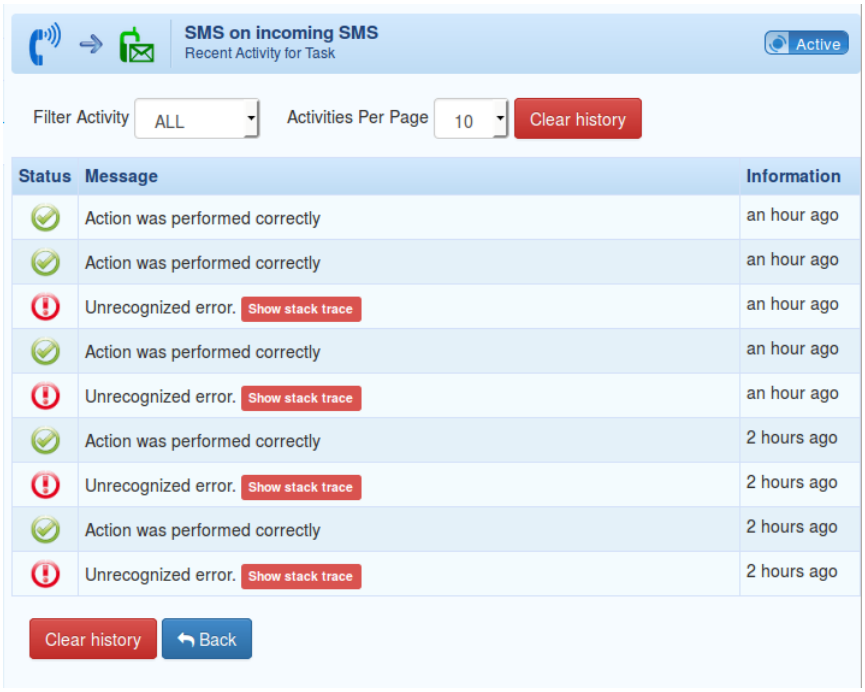
Did it work?

In addition to the obvious sign that you're receiving an SMS from your SMS provider, there are other ways you can check your application works.

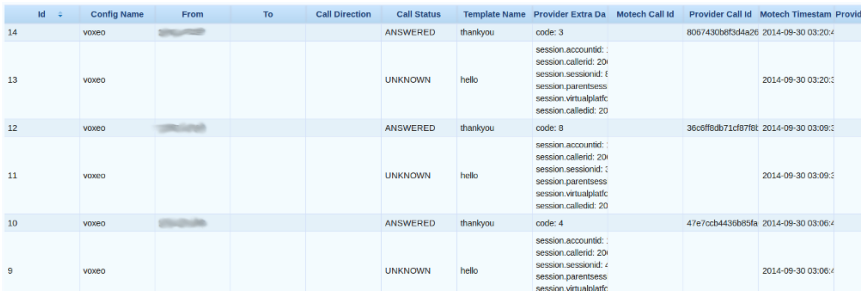
- You can look at the Tasks module's **Recent task activity** list to see if your task was executed:



- Or you can look at your task's history:



- You can also browse the IVR CallDetailRecord entity in the database using the MDS Data Browser:



Id	Config Name	From	To	Call Direction	Call Status	Template Name	Provider Extra Da	Motech Call Id	Provider Call Id	Motech Timestam	Provid
14	voxeo				ANSWERED	thankyou	code: 3		8067430b8f3d4a26	2014-09-30 03:20:4	
13	voxeo				UNKNOWN	hello	session.accountid: 1 session.callerid: 20 session.sessionid: 8 session.parentsess session.virtualplatfc session.calledid: 20			2014-09-30 03:20:3	
12	voxeo				ANSWERED	thankyou	code: 8		36c6ff8db71cf87f8t	2014-09-30 03:09:3	
11	voxeo				UNKNOWN	hello	session.accountid: 1 session.callerid: 20 session.sessionid: 3 session.parentsess session.virtualplatfc session.calledid: 20			2014-09-30 03:09:3	
10	voxeo				ANSWERED	thankyou	code: 4		47e7ccb4436b85fa	2014-09-30 03:06:4	
9	voxeo				UNKNOWN	hello	session.accountid: 1 session.callerid: 20 session.sessionid: 4 session.parentsess session.virtualplatfc			2014-09-30 03:06:4	

Note: Our simple VXML application did not bother to set the CallDirection nor many other fields in its status callback to Motech.

- Yet another way to see how your application would be to be to look at the SMS log or, for even more details, the Server Log.

Generic VXML IVR Provider Demo: Making Calls

Upon receiving an SMS, call the sender back and speak the content of the SMS.

The details:

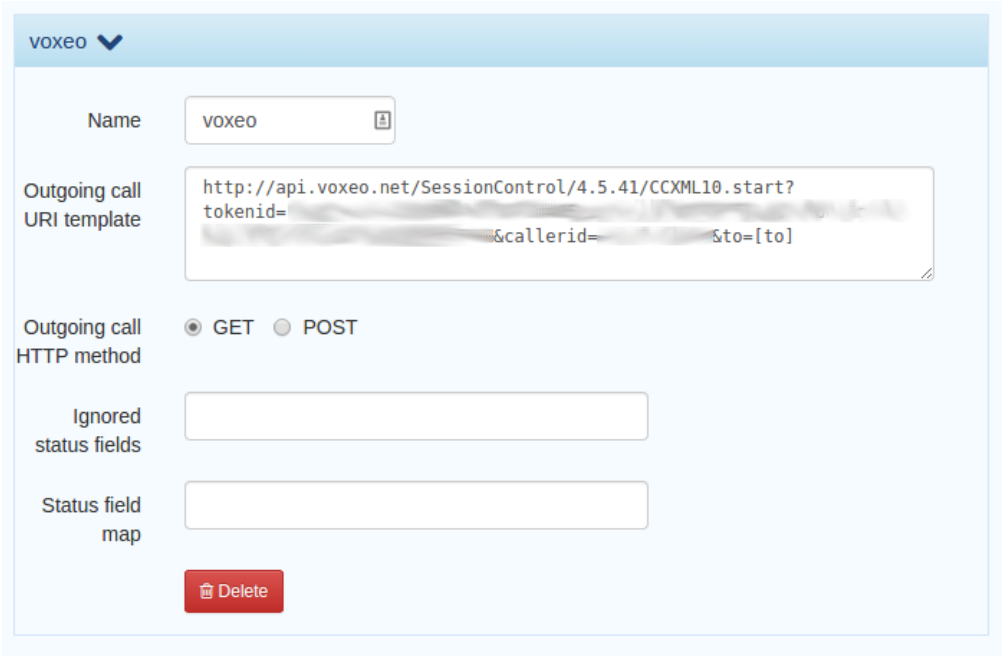
1. You send an SMS to the number provided to you by your SMS provider [5].
2. The SMS module receives a `/incoming` HTTP request from the SMS provider and sends a corresponding `inbound_sms` Motech Event.

[5] You must also tell your SMS provider what to do when they receive an SMS, remember?

3. The Tasks module listens to the `inbound_sms` Motech event and triggers [6] an outbound IVR call, passing the text of the SMS as a parameter named `message`.
4. Your IVR provider receives the outbound call request.
5. Your IVR provider then asks Motech for the VXML template [7], executes the VXML.
6. You receive a phone call, pick up, hear the IVR computer voice speak the content of your SMS.

Creating a Config

In order for the IVR provider to initiate a call, we need to create a Config, click **Modules / IVR / Settings**:



Note: We named ours **voxeo**. Note that it's a bit different than the one we created in the *incoming-calls* demo, we need to tell the IVR module how to reach the IVR provider by settings the `outgoingCallUriTemplate` and `outgoingCallMethod` properties.

The VXML

We need a simple VXML script that will say something that was passed to the IVR provider via the outgoing call initiation HTTP request:

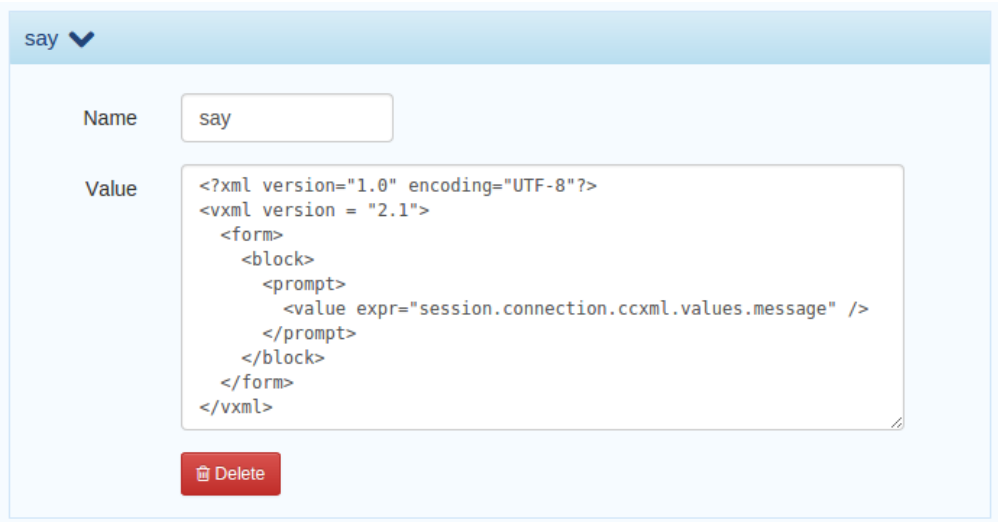
```
<?xml version="1.0" encoding="UTF-8"?>
<vxml version = "2.1" >
<form>
    <block>
        <prompt>
```

[6] By issuing an HTTP request to an URL provided by your IVR provider.
[7] At the URI you told your provider to find the VXML for outgoing calls from your number.

```
            <value expr="session.connection.ccxml.values.message" />
        </prompt>
    </block>
</form>
</vxml>
```

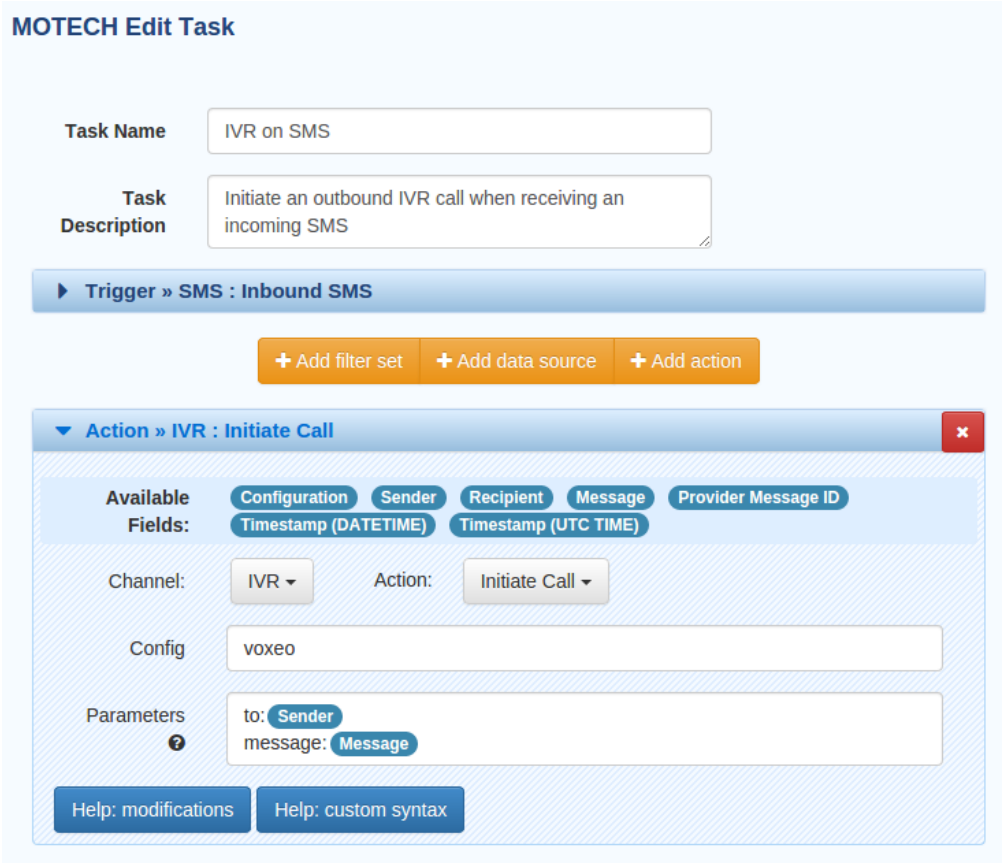
Note: `session.connection.ccxml.values.message` implies Motech will have to add a parameter named `message` to the HTTP request querystring to the IVR provider.

We'll name this template `say`:



Gluing things together with the Tasks module

Let's create a task which, upon receipt of an SMS, initiates an outgoing call and passes a message for the VXML script to say:



Note: we specify the number to call (in this case the sender of the SMS) and what do say (the content of the SMS) using a map notation in the action `Parameters` field.

Drum roll...

Now send an SMS with a simple 'hello world'. Wait a few seconds [8]. You should receive a 'hello world' voice call from your IVR provider. Et voila!

Notes

As in the previous example, you can check the **Recent tasks activity** pane on the Tasks module, or check the SMS or the IVR log to see what happened.

It's important to note that this very crude & simple demo does not deal with call status, so the IVR CallDetailRecord log will not be very useful.

Proprietary IVR Provider Demo: KooKoo

Introduction

KooKoo is a popular IVR provider from India. They are not a standard CCXML/VXML provider, but instead offer a language that's somewhat similar to a very simplified version of

[8] Crossing your fingers always helps

VXML, named KooKoo Tunes.

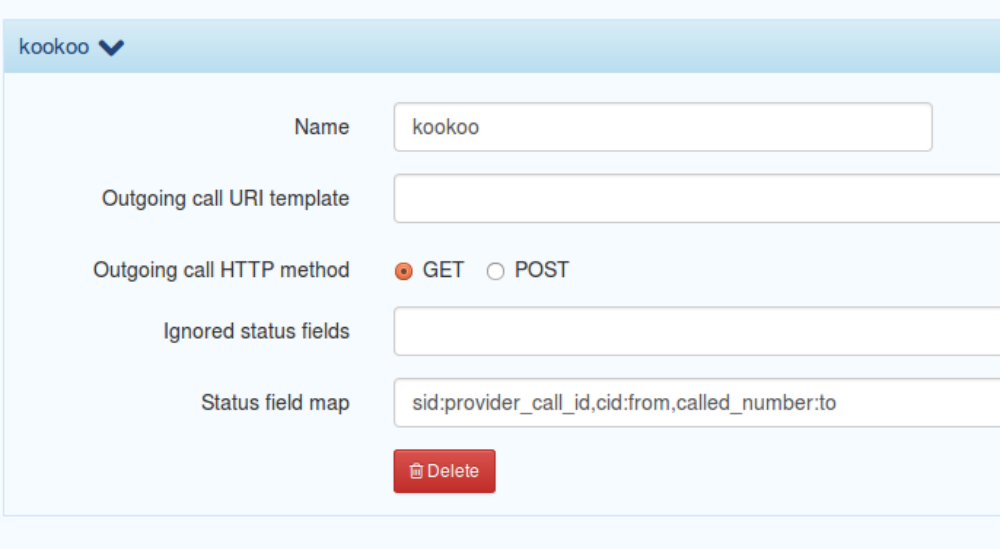
This demo is similar to the Incoming SMS Demo but uses KooKoo's simplified XML language instead of standard VXML. We'll only explain the IVR specific parts here, to create the full demo that sends you an SMS, please see the demo.

Initial Setup

You'll need to setup a KooKoo account.

IVR Config

In this demo we're only receiving calls (to the phone number provided to us by KooKoo) so we only need create a minimal config, click **Modules / IVR**:



Note: We're mapping three of the parameters sent back to Motech by KooKoo: `sid` will be mapped to `provider_call_id`, `cid` will be mapped to `from` (the number of the phone placing the call) and `called_number` will be mapped to `to` (the number of the phone receiving the phone, in this case the number assigned to you by KooKoo)

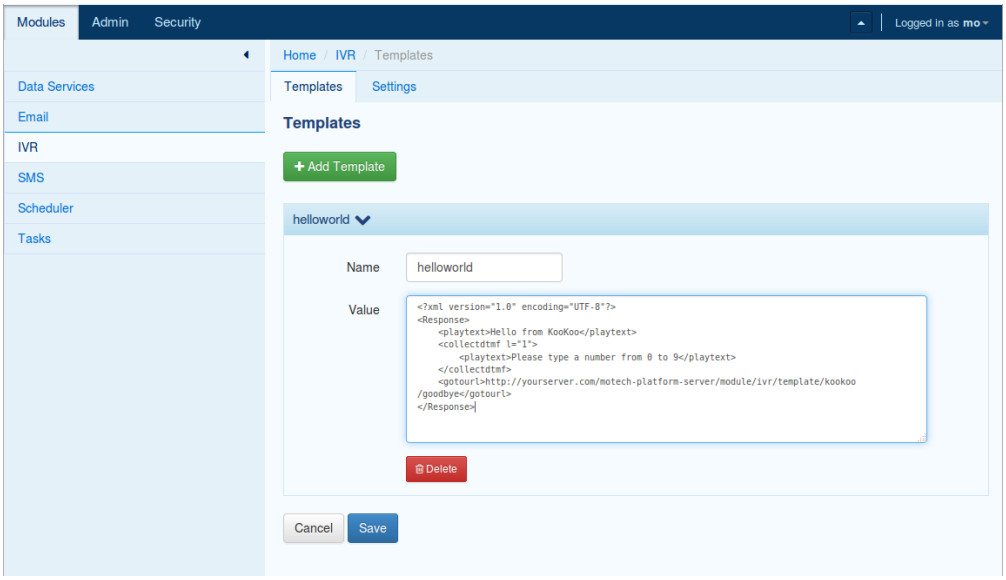
Two KooKoo Tunes

The funny people at KooKoo call an XML file a "Tune". We're creating two by going to **Modules / IVR** and clicking on **+ Add Template**. The first one says "Hello from KooKoo, please type a number from 0 to 9" and then sends the response back in a `data` parameter and requests the next thing to do from `goodbye` template:

```
<?xml version="1.0" encoding="UTF-8"?>
<Response>
    <playtext>Hello from KooKoo</playtext>
    <collectdtmf l="1">
        <playtext>Please type a number from 0 to 9</playtext>
    </collectdtmf>
```

```
    <gotourl>http://yourserver.com/motech-platform-server/module/ivr/template/kookoo/goodbye
</Response>
```

Name it **helloworld** and copy/paste the XML above in the value text area:



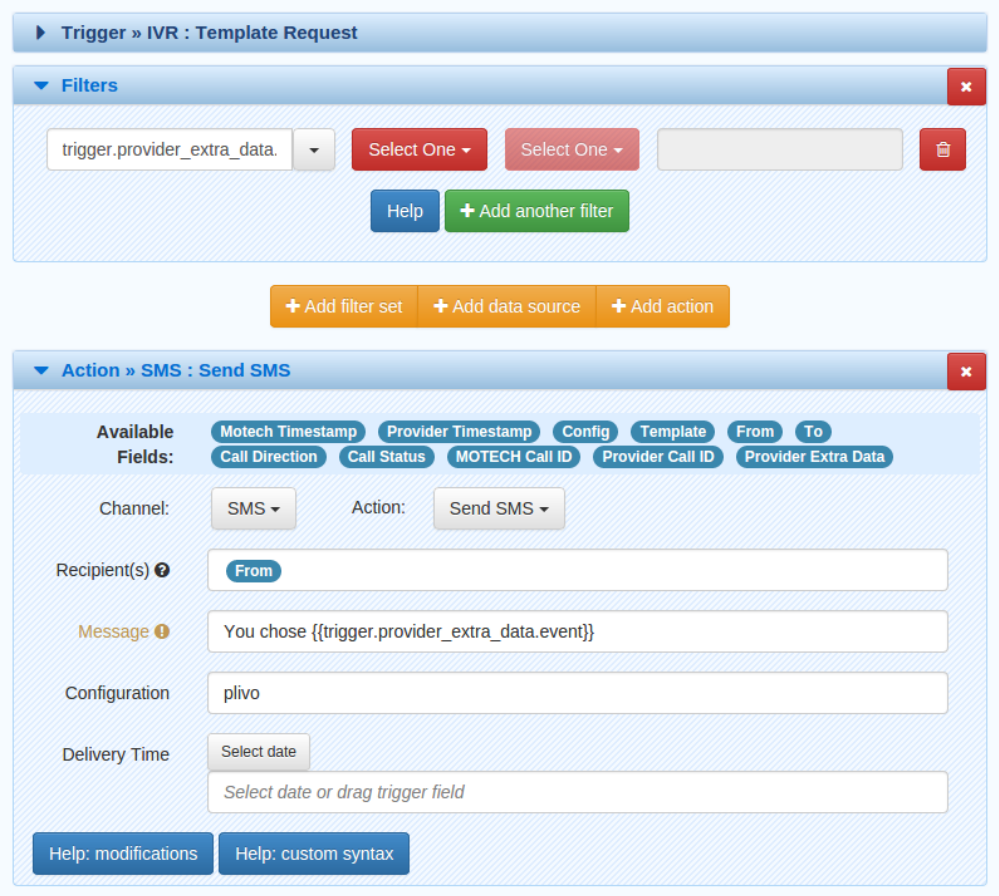
The `goodbye` template simply says "Thank you" and hangs up:

```
<?xml version="1.0" encoding="UTF-8"?>
<response>
    <playtext>Thank you</playtext>
    <hangup></hangup>
</response>
```

The first script sends an additional query parameter named `data` containing the key pressed during the call to Motech. Since `data` is not a standard property, it will be added to the `CallDetailRecord`'s `providerExtraData` map property. Actually KooKoo sends yet another query parameter named `event` with the value `GotDTMF`, we'll use `event` to filter the callback from KooKoo such that we pick the one which contains the `data` parameter.

Let's Create a Task

We need to create a task where the trigger is an IVR template request and where the **event** key in the `providerExtraData` map field is equal to `GotDTMF`. We also want the action to send an SMS to the original caller with the code she entered in the message:



Note: The filter source is partially obscured, here it is in full: `{{trigger.provider_extra_data.event}}`

Note: **data** *(the parameter containing the key pressed, remember?)* is extracted from the Motech event payload with `{{trigger.provider_extra_data.data}}`

Et Voila!

Now call your application at the phone number that KooKoo gave you, then listen to the lady [9] say 'Hello from KooKoo, please type a number from 0 to 9', type in a number (say 4). She'll say 'Thank you' and will hang up. Soon enough you should receive an SMS with the following message: 'You chose 4'.

7.3 Demo: Modeling a New System with MOTECH Data Services

`MOTECH Data Services (MDS)` is a user-configurable data store, that allows a MOTECH admin or developer to define the objects that are relevant for her application. To illustrate the use of MDS, we'll build out the data model for a simple electronic medical records (EMR) system stored in MOTECH. This tutorial will describe two different methods for defining the data model:

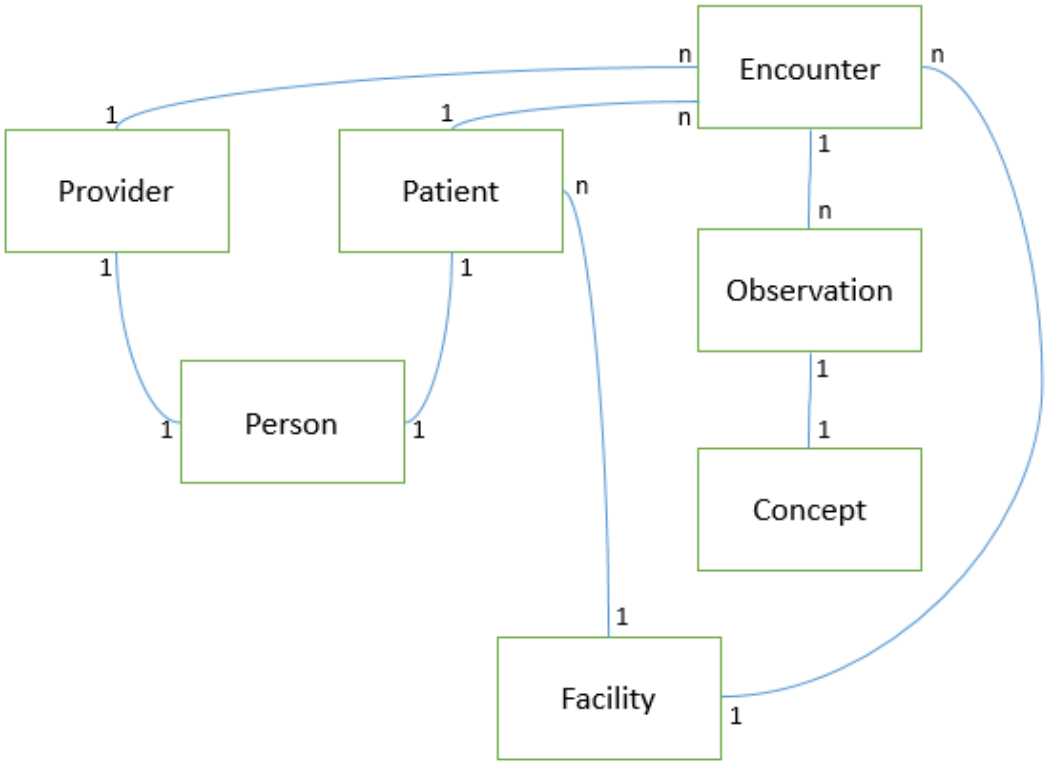
[9] The default might not be a lady's voice on your IVR provider, it was on mine.

1. `Using code annotations in a custom module`
2. `Using the MDS Schema Editor user interface`

Both demos will define a simple EMR with the following entities:

- **Person** - contains basic demographic information (name, gender, DOB) about a person in the EMR.
- **Patient** - represents a patient; has a 1:1 relationship to a Person object.
- **Provider** - represents a health care provider (nurse, physician, community health worker); has a 1:1 relationship to a Person object.
- **Facility** - represents a clinic or other health care facility.
- **Observation** - represents an observation made by a provider about a specific patient; consists of a concept (question) and a value
- **Concept** - an individual data point or question collected from patients (e.g. blood type or eye color). OpenMRS provides detailed documentation for their concept dictionary; we'll develop only a very limited version of it for this demo.
- **Encounter** - represents a provider's encounter with a patient, and has a 1:n relationship with the Observation entity.

These entities are inspired by (but represent only a small subset of) the domain model of OpenMRS. MOTECH's OpenMRS module implements a similar domain model. In our simplified model, the relationships among these entities can be represented by the following diagram:

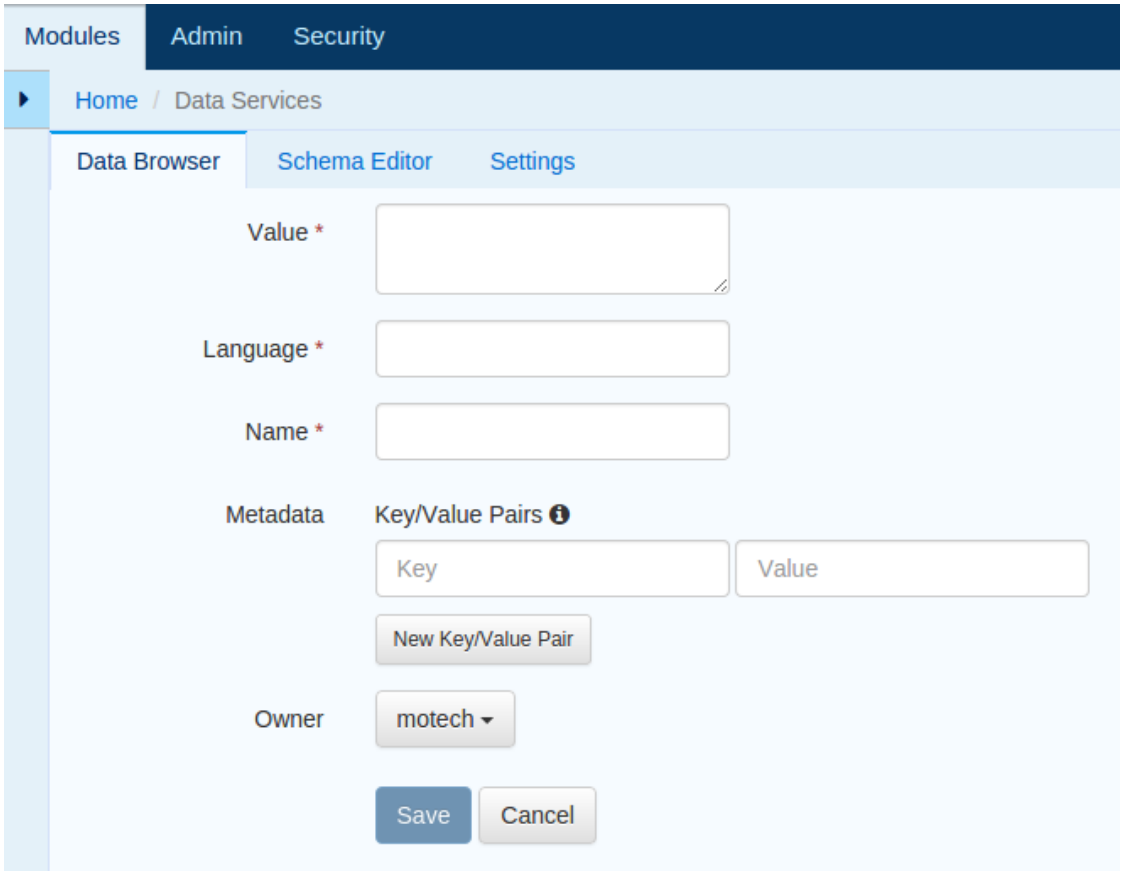


Choose your adventure (`code` or `UI`) and let's get started!

7.4 Demo: MOTECH Data Services Bulk Import

As of MOTECH 0.25, it is now possible to bulk import MDS entity instances in CSV format. This short tutorial describes the process. For this tutorial, we'll use the *CMS Lite module*, which has a simple data model and represents a real-world use case where bulk upload is useful.

First let's take a look at CMS Lite in the data browser, in particular the StringContent entity. StringContent has four user-supplied fields: value, language, name, and metadata (a key-value map). The former three fields are required.

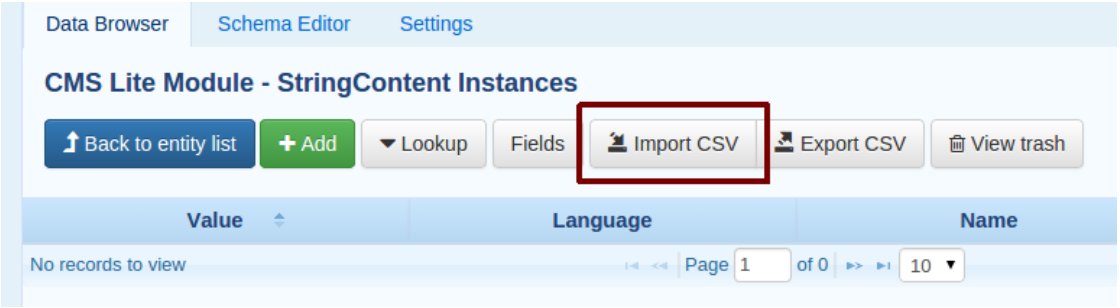


Now let's create a CSV file that defines a few entities for bulk import. You may use your text editor of choice, although for large data sets it's going to be easiest to use a spreadsheet tool like LibreOffice or MS Excel. Here's a sample with a header row and four entities:

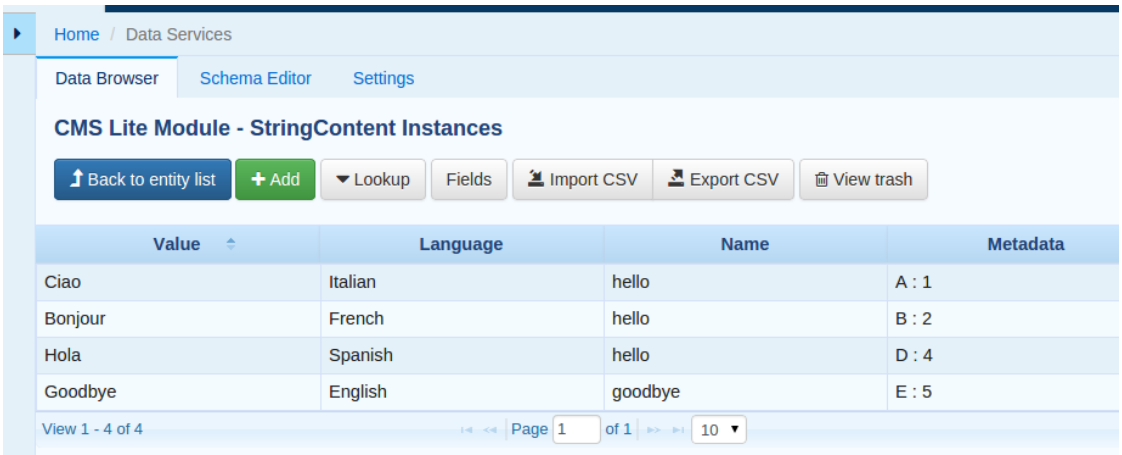
```
value,language,name,metadata
Ciao,Italian,hello,A:1
Bonjour,French,hello,B:2
Hola,Spanish,hello,D:4
Goodbye,English,goodbye,E:5
```

Note that it's not necessary to set many of the entity's fields in your CSV file (e.g. owner, creationDate, createdBy, modifiedBy, etc.); these will be set by the system. If you include the id field and it matches an existing entity's ID, the import will be handled as an update to the existing row.

Save the file in .csv format, and you're ready to go. Now let's upload the file. In the MDS Data Browser, navigate once again to StringContent, and click on Import CSV:



Browse to the location of your saved CSV file and select it. Et voila, your new entities will appear in the data browser:



7.5 Demo: OpenMRS Schedule Tracking

Table of Contents

7.5.1 Getting started

The source code for the demo is available on our GitHub repository (branch v1.0).

You will need to set up the following external systems for the demo implementation:

- CommCareHQ
- OpenMRS 1.9 (with Rest Web Services module, version 2.4)

You may check our documentation, for complete guide on *Connecting MOTECH to OpenMRS*.

Before starting to work with the demo implementation, you must prepare your CommCareHQ and OpenMRS instance. First, access the CommCareHQ and upload the forms schema, present in the commcare.schema folder. Then access MOTECH Commcare module and connect it with your CommCareHQ instance, by providing necessary data. Finally, enable data forwarding for forms, either via MOTECH Commcare module or via CommCareHQ instance. For OpenMRS, you are supposed to prepare some sample data. Access your OpenMRS instance and under Administration tab,

click "View Concept Dictionary" and add four new concepts named: Demo Concept Question #1, Demo Concept Question #2, Demo Concept Question #3, Demo Concept Question #4. Make sure to set the datatype in all of them to "Date". Get back to Administration and click on locations. Add at least one location of any name. Get back to Administration and click "Manage providers". Click "Add Provider" and add provider for person linked with the initial admin user (by default the person name is "Super User"). Get back to Administration and click on "Manage Encounter Types". Click "Add Encounter Type" and add type of name "ADULTRETURN", with any description.

You can build the demo project invoking the following command from the demo project directory:

```
$ mvn clean install
```

The demo module exposes its options via a very simple UI, available under: <motech-platform-url>/module/scheduletracking-demo/enroll/scheduleTracking

7.5.2 Demo specification

In this demo, a milestone corresponds to an observation of a particular concept having been made on or after the initial date of enrollment into the Demo Schedule. The four milestones are the Demo concepts, defined in the previous step.

Milestones have "windows" which represent a period of time. For example, today through five days from now represents a five day window. In the demo, each milestone has the same window periods. 0-1 minutes is the "early" window and no messages or alerts are raised.

At 1 minute in, a due message is raised and a phone call as well as text message are placed to the patient indicating that they are due for that particular Concept Question.

At 3 minutes, they enter the late window, and a late message is dispatched in the same manner. They receive a second late message one minute later at the 4 minute mark. The late window lasts until the 6th minute.

After this, no more late messages are sent and they enter the "max" window. They may still complete the milestone during this period. If they go beyond 15 minutes from the start of a milestone without fulfilling it, they are defaulted and their enrollment in the schedule is no longer active.

If a patient fulfills a milestone before they have defaulted, they will move on to the next milestone (unless there are no more, then the enrollment is completed) with new messages scheduled following the same format outlined above.

A patient may only have one active enrollment in the schedule at any given time, but may unenroll and start a new enrollment, or enroll again after their enrollment is completed or defaulted.

The three Commcare forms, that you have uploaded in the previous step are:

- **Patient Registration** - This form will create an OpenMRS patient, create a MOTECH patient, and optionally enroll the patient into the Demo Schedule.
- **Patient Enrollment** - This form will enroll the MOTECH patient into the Demo Schedule (A corresponding OpenMRS patient with the same MOTECH Id must exist to use this form.)
- **Patient Encounter** - This form will create an encounter for an OpenMRS patient. You must provide the MOTECH Id of the patient in OpenMRS, an observation date, and an observed concept.

7.5.3 Demo workflow

In order to use the demo, you must register a patient into MOTECH with a phone number. The same patient ID must also be registered into OpenMRS. This can be achieved by registering the patient directly in OpenMRS or by sending a Commcare "Registration form".

You may then enroll that patient in the schedule. You may view the definition of a schedule in the simple-schedule.json file, located in the resources directory of the scheduletracking demo module. This schedule will be automatically created during demo startup. A patient can be enrolled to the schedule from the scheduleTracking page or by sending

the Commcare Enrollment form. If an enrolled patient is not found in both the demo MOTECH phone number database and in OpenMRS, they will not be enrolled in the schedule. Once a patient is enrolled, SMS messages and phone calls will be placed, indicating that the patient is due for a particular concept.

To complete a concept, a Patient Encounter form should be submitted via CommCareHQ. You must provide the MOTECH Id of the patient in OpenMRS, an observation date, and an observed concept id. After you have completed all 4 concepts, you will be removed from the Demo Schedule. You are also free to complete concepts (e.g. Demo Concept Question #1), before you enroll. In this case, when you do enroll, you will be enrolled at the next required concept milestone. For example, if you complete Demo Concept Question #1 and #2, then enroll in the Demo Schedule, you will be scheduled for the third milestone (Demo Concept Question #3).

Possible failures

Each form received by the scheduletracking demo is validated before processing. A few of the reasons a form may fail include:

- Bad phone number format (must be in form XXX-XXX-XXXX)
- Out of sequence Concept, e.g. you can't complete "Demo Concept Question #3" before completing "Demo Concept Question #2"

If a validation of a form fails, an information will be printed in the logs and no further action (eg. enrollment, encounter) will be executed.

7.6 Demo: SMS-Based Pregnancy Message Campaign

This demo will illustrate how to create an outgoing SMS-based Message Campaign with MOTECH. In order to follow along, you'll need to have a MOTECH server with the following modules installed:

- *Message Campaign*
- *SMS*
- *CMS Lite*
- *Tasks*

Further, to send campaign messages via SMS, you need to configure an SMS provider as described in the *SMS* module documentation.

7.6.1 Defining the Campaign

Our informational message campaign will be aimed at pregnant mothers, and will provide timely information to help women engage in healthy behaviors. The campaign will consist of one message per week, tailored to the specific week of pregnancy. Participants can subscribe to the service at any point in their pregnancies, and when they do, they will start with the message corresponding to current gestational age (i.e. if someone joins the program 20 weeks into her pregnancy, she can start with the message for week 20).

To meet the above requirements, we can use an Offset Campaign (refer to the *Message Campaign* documentation for a discussion of the different campaign types and how to configure each). The JSON definition of our Offset Campaign will contain 40 messages, one for each week of pregnancy. A snippet of the definition, for the first five weeks, is shown below. You may view the contents of the entire 40-week definition `here`.

```
[{
  "name" : "Pregnancy Campaign",
  "type" : "OFFSET",
```

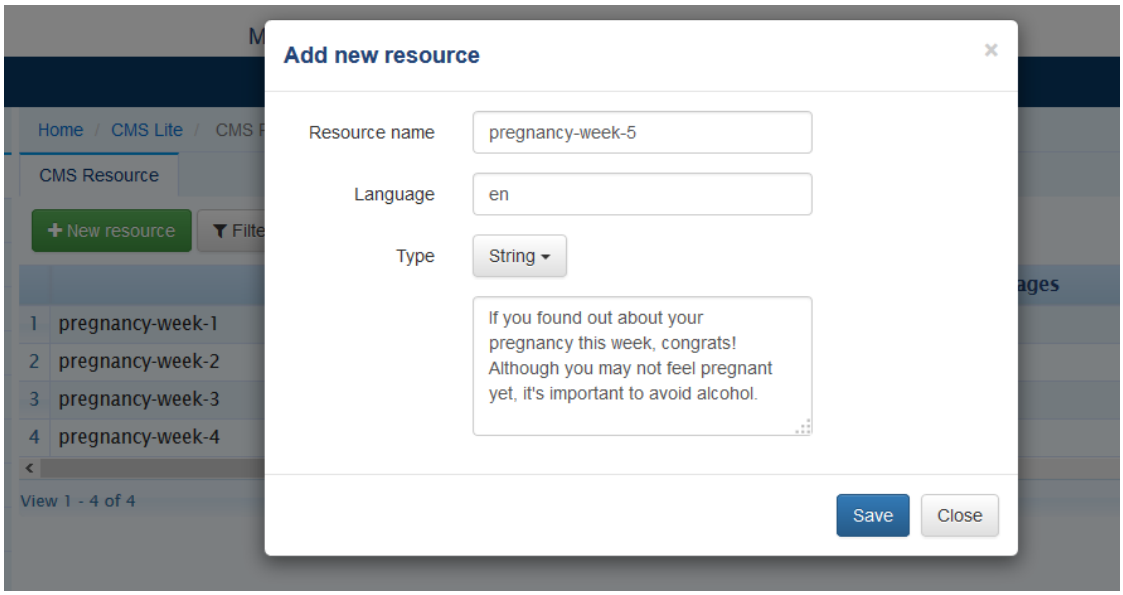
```
  "messages" : [
    {
      "name" : "Week 1",
      "formats" : ["SMS"],
      "languages" : ["en"],
      "messageKey": "pregnancy-week-1",
      "timeOffset" : "1 Week",
      "startTime" : "10:30"
    },
    {
      "name" : "Week 2",
      "formats" : ["SMS"],
      "languages" : ["en"],
      "messageKey": "pregnancy-week-2",
      "timeOffset" : "2 Weeks",
      "startTime" : "10:30"
    },
    {
      "name" : "Week 3",
      "formats" : ["SMS"],
      "languages" : ["en"],
      "messageKey": "pregnancy-week-3",
      "timeOffset" : "3 Weeks",
      "startTime" : "10:30"
    },
    {
      "name" : "Week 4",
      "formats" : ["SMS"],
      "languages" : ["en"],
      "messageKey": "pregnancy-week-4",
      "timeOffset" : "4 Weeks",
      "startTime" : "10:30"
    },
    {
      "name" : "Week 5",
      "formats" : ["SMS"],
      "languages" : ["en"],
      "messageKey": "pregnancy-week-5",
      "timeOffset" : "5 Weeks",
      "startTime" : "10:30"
    }
  ]
}]
```

The campaign JSON may be uploaded using the file upload UI or by placing the message-campaigns.json file in the message-campaign directory. Both methods are described in the *Message Campaign* documentation.

7.6.2 Creating Campaign Messages

The text content for our SMS messages may be conveniently managed within MOTECH using the *CMS Lite* module. We'll define one message for each week of pregnancy, using the "messageKey" specified in our campaign definition as the identifier for each message.

To create a string resource in the CMS, we'll navigate to the CMS Lite module within the MOTECH UI, and click on the "New Resource" button. A popup will appear, and we can enter our content. Here's what we might enter for Week 5:

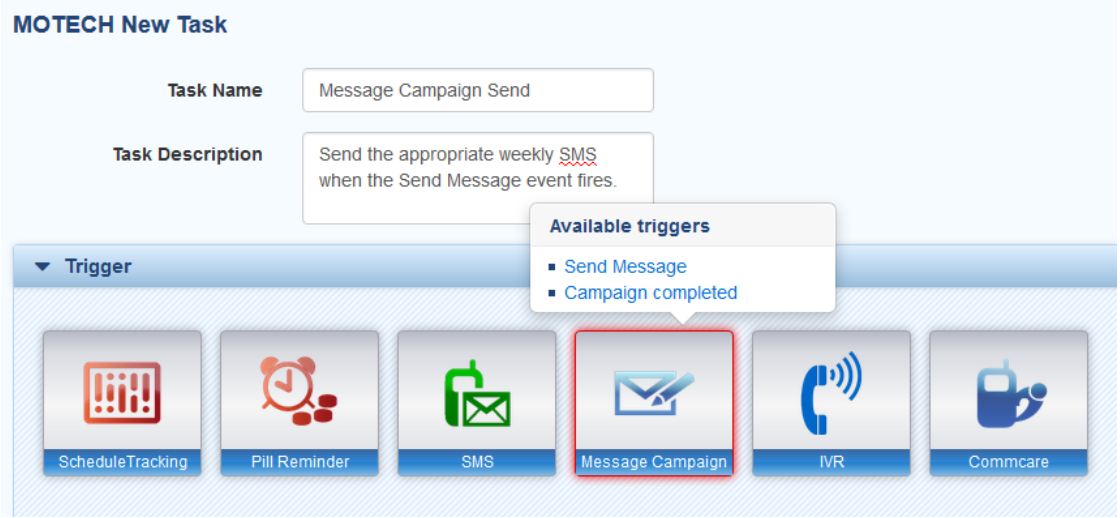


Keep in mind that SMS messages are limited to 140 characters, so we need to keep our prose concise.

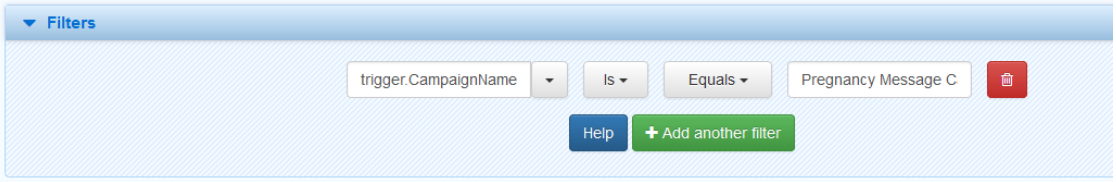
7.6.3 Wiring Up Events

Now that we have a campaign schedule and message content defined, we need to configure MOTECH to send out the appropriate messages according to the schedule. This can be accomplished using the *Tasks* module.

To get started, we navigate to the Tasks module, and click on "New Task". For the Trigger, we select Message Campaign's Send Message event.

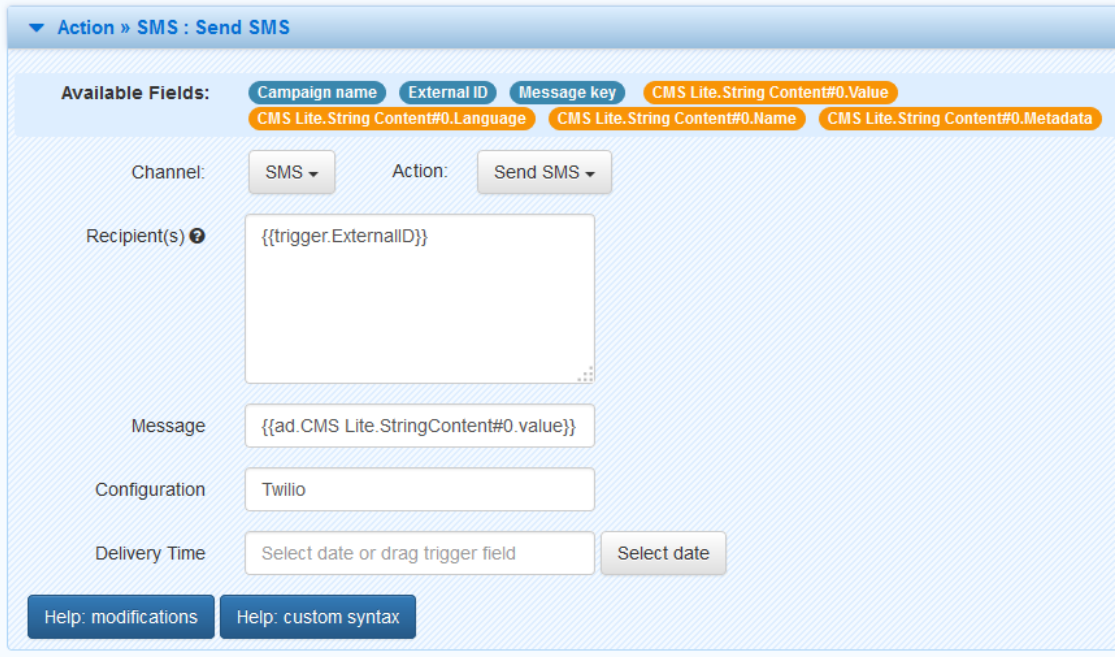


Next, we want to ensure that this Task is only executed for our specific message campaign. We can do this by adding a simple Filter to our Task:



In order to access messages in the CMS, we need to add a Data Source to our Task. We can do this by clicking on "Add Data Source" and selecting "CMS Lite" as the source. Notice that when configuring the data source, the fields contained in the Message Campaign Send Message event are available to be used for lookups in the CMS. These appear as blue ovals and can be dragged/dropped to the input fields below. We want to drag the "Message Key" field and drop it in the CMS Lite "Name" field. Once we've configured this data source lookup, the data retrieved from the CMS will be available to the downstream steps in our Task.

The last step is adding an Action for our Task – this will be where we send the SMS, of course. To construct the Action, first we select the Channel and Action (SMS and Send SMS), and then we can drag/drop the blue and orange ovals (the fields from the Trigger event and our Data Source, respectively) to configure the Action. We drop "External ID" in the recipient field (we haven't discussed campaign enrollment yet, but this field will hold the recipient's phone number). Then we can drop the CMS Lite content "value" in the "Message" field. For "Configuration", we enter the name of the SMS configuration that we want to use to send the message (see the *SMS* module documentation for instructions on configuring a provider).



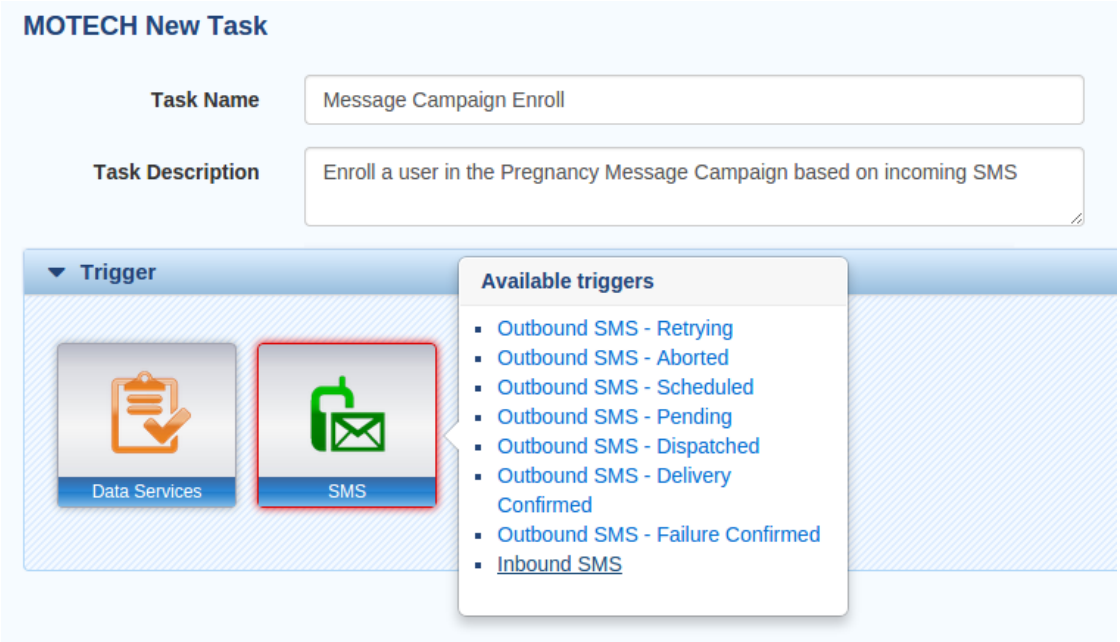
And now our Task is complete! Once we click "Save and Enable" it will be active and ready to handle events.

7.6.4 User Enrollment Via SMS

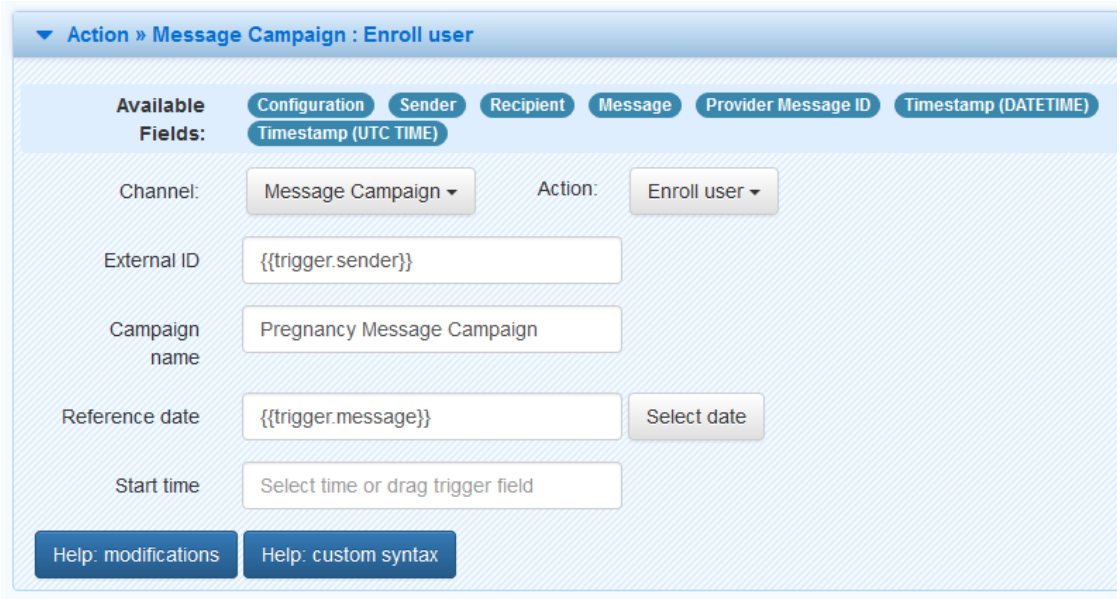
There's just one more piece of the puzzle – enrolling actual people in the campaign. The *Message Campaign* documentation describes two standard methods for enrolling subscribers in campaigns: manually using the enrollment UI, or with code. For our campaign, however, it would be nice to allow recipients to self-enroll by sending an SMS. For this, we can use the Tasks module again.

Let's create a new Task that is triggered by an Incoming SMS, that will create a Message Campaign enrollment corresponding to the information contained in the body of the SMS. For this example, we'll assume the SMS body is very simple – that it contains the date of the potential enrollee's last menstrual period (LMP). We'll use the LMP as the Reference Date for the enrollment in the Message Campaign.

The Trigger part is quite simple:



For the Action, we drag the Recipient and Message ovals into the appropriate fields to configure the enrollment:



If desired, we could support additional Message Campaign enrollment actions in response to an inbound SMS – e.g. unsubscribing from a campaign or allowing the user to specify which campaign to subscribe for if our system defines more than one. These variations can also be defined using Tasks, with more sophistication possible if we use filters and/or apply string manipulation functions to the message text in order to parse multi-word messages.

7.7 Demo: Create a Care Schedule

Text to come

CHAPTER 8

Contribute

Thank you for your interest in contributing to MOTECH. There are many different ways to get involved - regardless of your area of expertise and time commitment, there is likely a useful way for you to contribute. Please peruse the sections below for some instructions on how to get started contributing in specific areas. If there is another way you'd like to help that isn't listed, just let us know what your interests are and we can help find a project for you.

8.1 Development

Want to help us develop MOTECH? The step by step guide below will help you get started. The instructions may be incomplete, or may contain some errors, so please just let us know if something is missing or incorrect. Our mailing list is populated with helpful people who will help you get going - and will get this documentation up to date with any issues you encounter.

Here is how to get started:

8.1.1 Mailing Lists and Accounts

1. Sign up for the MOTECH Developer mailing list - this is the best place to get help with any questions about MOTECH development.
2. Create an account on Gerrit, our code review system.
3. Get a Jira account if you'll need to report/track issues - there is no self-serve way to request a Jira account yet, so please just email the MOTECH Developer list and we'll get you set up.

8.1.2 Dev Environment & Tools

1. Configure your *dev machine*.
2. Configure your Git client to *submit changes via Gerrit*.
3. Familiarize yourself with our CI.

8.1.3 Finding Something to Work On

1. Some MOTECH devs have found that reviewing code written by others before they jump in and develop their own is a good way to get their feet wet. If that sounds like your style, feel free to jump into any code review on Gerrit and add your comments.

2. If you'd rather dive right in, we'd recommend that you find a "community" issue from our issue tracker - these are bugs and user stories that we think are good introductory items for MOTECH newbies. See a list of all current "community" issues here.
3. If you're already building your own system on top of MOTECH and you'd like us to incorporate your changes for an issue you've found and fixed, please first check whether the issue already exists in our issue tracker. If you are not sure, please email the mailing list and we'll help you determine whether the issue is already known. Please track your work with a new issue so that we can evaluate it for inclusion in the Platform.

8.1.4 Developing and Submitting Your Code

1. If your fix will be nontrivial, please leverage the mailing list for feedback on your design before you get too far - we are a friendly bunch and can help ensure you are headed in the right direction.
2. When you're ready to push your changes, please squash your commits to keep the change history concise, and write a commit message that conforms to our *guidelines*.
3. Submit your code using **git push origin** - if you configured your environment correctly, this sends your changes to Gerrit, our code review system.
4. Please incorporate code review feedback and update your patch as needed - once your change has passed code review, one of the project maintainers will merge your change to the repo.
5. Resolve the relevant issue(s) in Jira.

8.2 Documentation

We could really use some help telling our story, and we'd love your help.

First, a bit about how our docs are stored and managed. Each MOTECH code repository contains a *docs* directory populated with reStructured Text (reST) files. These files are built by Sphinx and hosted at http://readthedocs.org. *This page* contains more information about reStructuredText and how to build the docs on your local machine. Once you've written and built your docs locally, you can just check them in and they'll automatically appear on our docs site after the next doc build.

The instructions below will let you know how to get started with adding/editing MOTECH documentation.

Note: If you are a writer (not a software developer!) and you find that committing documentation to git is a bit daunting, drop us a line. We can provide extra support through the process (or even check in your docs for you).

8.2.1 Mailing Lists and Accounts

1. Sign up for the MOTECH Developer mailing list - this is the best place to get help with any questions about MOTECH documentation or code.
2. Create an account on Gerrit, our code review system.
3. Get a Jira account if you'll be working on documentation tickets - there is no self-serve way to request a Jira account yet, so please just email the MOTECH Developer list and we'll get you set up.

8.2.2 Doc Environment & Tools

1. Install Sphinx and Javasphinx, and test out building the docs locally. Full instructions *here*.

2. Configure your Git client to *submit changes via Gerrit*.
3. Install an editor for reStructuredText. Any editor will work, but we find that Sublime works pretty well.

8.2.3 Finding Something to Work On

1. All planned documentation topics for MOTECH are tracked in Jira – you are welcome to pick any unassigned ticket from this query. Note that many of the tickets have sub-tickets as well, so you can drill down to find additional unassigned topics.
2. If the topic you want to write about doesn't appear to be tracked in Jira, email the list and let us know. We'll help you determine where the topic fits in our ToC and create a Jira ticket for it.
3. Assign the Jira ticket to yourself and click "Start Progress" when you're ready to start writing.

8.2.4 Writing and Submitting Your Doc

1. As you are writing your doc, we recommend building periodically to ensure that the doc looks the way you expect. It can take some trial and error to get the hang of reStructuredText markup.
2. When you're ready to push your changes, please squash your commits to keep the change history concise, and write a commit message that conforms to our *guidelines*.
3. Submit your doc using **git push origin** - if you configured your environment correctly, this sends your changes to Gerrit, our code review system.
4. Please incorporate review feedback and update your patch as needed - once your change has passed code review, one of the project maintainers will merge your change to the repo.
5. Resolve the relevant issue(s) in Jira.

8.3 Translation

Bonjour!

We are using Transifex to manage MOTECH translations, which makes it easy for non-geeks to help. If you speak multiple languages and would like to help us make MOTECH multilingual, please start by checking out our translation page to determine whether your language(s) are on our list. Do we have a translation in progress for your language? Great! We'd love your help translating additional strings. Do we not have a translation started for your language? Also great! We'd love your help getting one started.

Either way, please sign up for Transifex (free), and then contact us. Let us know your Transifex user ID and which language(s) you'd like to work on, and we'll help you get started.

CHAPTER 9

Release Notes

9.1 Current Version

Version 0.24.1 Release Notes

9.2 Older Versions

Version 0.24 Release Notes
Version 0.19 Release Notes
Version 0.18 Release Notes
Version 0.17 Release Notes
Version 0.16 Release Notes
Version 0.15 Release Notes
Version 0.14 Release Notes
Version 0.13 Release Notes
Version 0.12 Release Notes
Version 0.11 Release Notes
Version 0.10 Release Notes
Version 0.9 Release Notes
Version 0.8 Release Notes
Version 0.7 Release Notes
Version 0.6 Release Notes
Version 0.5 Release Notes
Version 0.4 Release Notes

9.2.1 Version 0.24 Release Notes

Release Date: September 3, 2014

Release Summary

The 0.24 release is primarily dedicated to removing MOTECH's dependency on CouchDB, as well as enhancing MOTECH Data Services (MDS) to make it usable as the data layer for most MOTECH applications. All modules

have now been migrated to MDS, with a few exceptions noted below. One notable enhancement to MDS in this release is the support for relationships between entities (1:1, 1:many, master-detail), in order to enable a number of the module migrations.

This release also features some consolidation of our code repositories (details below) and deprecation of a few modules. The Platform modules were also refactored to reduce their public surface area. Three new modules have been developed: mTraining, Batch, and Hub.

Where to Get it

Source Code: Platform | Modules

Binary Distribution

Platform WAR

Modules:

- Alerts
- Appointments
- Batch
- CMS Lite
- CommCare
- Event Logging
- Hindi Transliteration
- HTTP Agent
- Hub
- Message Campaign
- mTraining
- OpenMRS 1.9
- Pill Reminder
- Schedule Tracking
- SMS

Major Changes

Modules Migrated from CouchDB to MDS

All MOTECH modules (with the exception of the IVR modules mentioned later) are now using MOTECH Data Services for data storage and retrieval. Any modules that are used in MOTECH implementations should likewise be migrated from CouchDB to MDS. The best reference for using MDS at this time is the source code for the existing modules; soon we will provide developer-focused `documentation for MDS`.

New MDS Features

A number of new features were added to MDS in order to support migration of the existing modules. These features include:

- Support for relationships among MDS entities. This includes 1:1, 1:many, and master-detail relationships (bi-directional and many:many relationships are expected in a future release)
- UI support for browsing and restoring items from Trash
- UI support for Filters
- Support for additional types: Locale, Date, File, Map
- Various UI changes to improve the schema editor and data browser

Repository Consolidation

In order to simplify development, some of the MOTECH source code repositories have been merged. Rather than maintaining four separate repositories for MOTECH source code (motech, platform-campaigns, platform-communcations, and platform-medical-records), there are now two repositories: motech and modules. Please see the *MOTECH Code Repositories* topic for more information.

Modules Moved to motech-contrib

As part of the repository cleanup effort, a few modules were moved out to the motech-contrib repository. The specific modules that were chosen are those that have very few (or in some cases no) consumers. These modules will no longer be maintained by Grameen Foundation, but MOTECH implementers are welcome to continue using them and either fork the code or submit pull requests to the repo as needed. The list of migrated modules is as follows:

- Event Aggregation
- Mobile Forms
- OpenMRS OMOD
- Outbox

New Modules

Several new modules were developed as part of this release:

- **mTraining** - The mTraining module provides data containers and APIs for defining training courses and tracking user enrollment and progress. A typical use case might be an IVR-based training course for a health worker.
- **Batch** - The Batch module is an implementation of Spring batch (version:3.0.0.M3). It essentially deals with scheduling triggering of jobs.
- **Hub** - The Hub module is the implementation of Google's PubSubHubbub Hub 0.4 Specifications. It exposes an API so other modules can act as publisher and make contents available to it for distribution whenever there is an update in a topic.

Legacy IVR Modules Deprecated - Replacements Coming

The following modules have been deprecated:

- Call Flow
- Decision Tree
- IVR (including API, Asterisk, Kookoo, Verboice, and Voxeo)

These modules are still present in the source code repository, but they were not built as part of the release. If you need to use one or more of these modules, you can build each module that you require by executing **mvn install** from the module's base directory. Note that these modules depend on CouchDB.

These legacy modules will be replaced in 0.25 by new generic modules for handling VXML and CCXML. There will also be a re-worked version of the Verboice module in a coming release that removes the dependencies on call-flow and decision-tree.

Known Issues

- MOTECH-818 - Able to remove a field from a lookup even if the lookup is being used in a Task

  Summary: User should not be able to remove a field from an MDS Lookup if the Lookup is used as a data source in a Task. Currently this is not prevented.

  Workaround: When modifying a Lookup, the user will need to verify manually that it is not being used as a data source in a Task (this can be checked via the Tasks UI).

- MOTECH-1084 - MDS ComboBox UI Bugs

  Summary: There are some problems with combobox fields when we add two or more of them to an MDS entity:

  1. After opening instance view in Data Browser we can see error message "This field is required" under dropboxes of all comboboxes (except for the first one) even though they aren't.
  2. After clicking "Add option" button and filling text field with any value on more than one combobox, when we click "Save" for any of them, save buttons for all others will just gray out and turn off. They'll become active again when we enter anything in text fields in any combobox.

  Workaround:

  1. A number of workarounds may exist depending on the nature of your application. For example, one could create a dummy default option for the ComboBox with a name like "Empty" or "No Value" when defining the ComboBox field.
  2. Enter anything in the text fields of any combobox, and the "Save" buttons will become active again.

- MOTECH-1125 - Getters starting with "is" are not recognized by the MDS annotation processor

  Summary: The MDS annotation processor should recognize boolean getters, starting with "is", eg. for field "completed", the getter method "isCompleted()" should be recognized. Currently, it seems to only recognize getters starting with "get".

  Workaround: This issue may be temporarily avoided by prefixing getters on MDS entity classes with "get".

- MOTECH-1147 - Default value for Date type fields doesn't work

  Summary: Create an entity and add a Date type field. Set a default value to any date and save changes. Notice that the default value field is clear and when you add an instance of that entity, there's no default value inserted.

  Workaround: When using an MDS Entity with a field of type Date, the values for all Date fields will need to be set explicitly.

- MOTECH-1153 - Creating an MDS entity with an enum field fails if the enum has many values

  Summary: Attempting to create an entity that has an enum field (allow user supplied option disabled), that has got many values causes failures. This does not happen when user supplied option is enabled (as it creates a list, instead of enum).

  Workaround: Some possible workarounds for this issue (depending on the nature of the application) include: - Enabling user supplied values on enum fields, if appropriate for the application - If possible, splitting the enum into multiple enums

- MOTECH-1156 - Error when adding MDS Entity with space in name

  Summary: If a user adds an Entity with spaces in the name then there is an error. After that it is impossible to add other Entities. Entity name should be validated against spaces in the name or they should be deleted.

  Workaround: Avoid spaces in Entity names.

Tickets

You can browse the list of tickets resolved for this release on our issue tracker.

9.2.2 Version 0.24.1 Release Notes

Release Date: October 8, 2014

Release Summary

0.24.1 contains a few bug fixes, and two notable new MOTECH Data Services features: bi-directional relationships and many-to-many relationships.

Where to Get it

Source Code: Platform | Modules

Binary Distribution

Platform WAR

Modules:

- Alerts
- Appointments
- Batch
- CMS Lite
- CommCare
- Event Logging
- Hindi Transliteration
- HTTP Agent
- Hub
- Message Campaign

- mTraining
- OpenMRS 1.9
- Pill Reminder
- Schedule Tracking
- SMS

Tickets

The following tickets were resolved for this release:

MOTECH-1108 - Support two-way relationships

MA-471 - Expose retriveAll() by both properties list and query params

MOTECH-1047 - Support many-to-many relationships - DDE only

MOTECH-1255 - MDS doesn't recognize an array of byte primitives

NO BUG - Fix job id type of reminder event in PillReminder

9.2.3 Version 0.25 Release Notes

Release Date: March 31, 2015

Release Summary

The 0.25 release is primarily dedicated to *MOTECH Data Services* improvements, including MDS REST APIs and the ability to import entities to MDS via CSV file.

This release also features a new, simplified IVR module to replace the assorted IVR modules that were deprecated in 0.24.

Where to Get it

Source Code: Platform | Modules

Binary Distribution

Platform WAR

Modules:

- Alerts
- Appointments
- Batch
- CMS Lite
- CommCare
- Event Logging
- Hindi Transliteration
- HTTP Agent

- Hub
- IVR
- Message Campaign
- mTraining
- OpenMRS 1.9
- Pill Reminder
- Schedule Tracking
- SMS

Major Changes

CSV Import of MDS Entities

It is now possible to bulk import MDS entity data in CSV format. See the *bulk upload tutorial* for more information.

MDS REST API Generation

MDS CRUD Events

Other MDS Improvements

- Two-way relationships

New IVR Module

WAR File Changes

CouchDB Removal

Support for CouchDB, which was deprecated in version 0.24, has been completely removed in version 0.25. As part of the removal process, the development team extensively tested MDS performance to ensure that the migration to MDS had not introduced performance regressions in MOTECH modules.

Tickets

You can browse the list of tickets resolved for this release on our issue tracker.

CHAPTER 10

Roadmap

This page describes the high-level roadmap for the next few MOTECH Platform releases. The specific features that comprise the releases listed below may be rescheduled as additional information comes to light. For more granular and up-to-date information about release plans, click on the issue tracker links for each of the releases below.

10.1 Version 0.25 - Early 2015

Issue Tracker

10.1.1 MOTECH Data Services Performance

The goal of this effort will be to ensure that MDS performance is as good as (or better than) MOTECH running on CouchDB. We will begin by conducting performance testing and benchmarking of MDS against CouchDB to find performance bottlenecks. These will be prioritized, and the most important ones will be fixed for this release (others may be postponed to future releases).

10.1.2 MOTECH Scale Testing

A test environment will be created to simulate MOTECH running at scale. Once this environtment is created, we will test performance of MOTECH under various configurations including clustering. Through this process, we expect to identify, document, and/or fix specific issues discovered, including making clustered mode operational and performant, as well as providing configuration recommendations for ActiveMQ at scale.

10.1.3 IVR Support

The legacy MOTECH IVR modules were deprecated in the 0.24 release. Starting with version 0.25, there will be one IVR module that supports VXML/CCXML as well as Verboice.

10.1.4 MDS REST API Generation

MOTECH will automatically generate REST APIs for CRUD operations on entities defined in MOTECH Data Services.

10.2 Version 1.0 - mid-late 2015

Issue Tracker

10.2.1 DHIS2 Module

A new module will be created to support pushing individual level anonymous data (DHIS2 Event Capture) to DHIS2. DHIS2 data push will be exposed as a new Action through MOTECH Tasks. Support for additional DHIS2 use cases will likely come in future releases.

10.2.2 Stable Semantic Versioning

We will apply a semantic version scheme to MOTECH Platform and all modules thus making it easier to determine backward compatibility. We will stabilize our core system and API providing implementers with a level of confidence that their system will be compatible with future releases of MOTECH.

10.2.3 End User Install

End users should be able to install MOTECH without compiling. Install should be scriptable and unattended. We will provide an example install script that operations engineers may reuse or modify as desired for their purposes.

10.2.4 Platform API Documentation

Every public API will be documented with standard javadoc that is published with each release.

10.2.5 API Sanitization

The process of cleaning up MOTECH's public API - which was started in v0.24 - will be completed, resulting in a stable public API for MOTECH.

CHAPTER 11

MOTECH Mailing Lists

The mailing lists below are the best way to keep in touch with us. Please join the discussion!

11.1 MOTECH Developers

Mailing list for regular contributors to the MOTECH Platform source repository - used for design discussions and other issues impacting the developer community.

motech-dev@googlegroups.com	Join Dev List

11.2 MOTECH Implementers

Mailing list for implementers and other users of MOTECH - used mostly for communication related to releases. Traffic is moderated and very low volume.

motech-implementers@googlegroups.com	Join Implementers List

CHAPTER 12

Javadoc

12.1 org.motechproject.admin.domain

12.1.1 NotificationRule

public class **NotificationRule**

A notification rule persisted in the database. Represents a rule for sending out a single notification after a matching `org.motechproject.admin.domain.StatusMessage` is registered in the system. The message is matched against its level and the module to which it is tied to. This class also contains information about this notification's recipient and the `ActionType` representing a method used for notifying the recipient.

See also: `org.motechproject.admin.domain.StatusMessage`

Constructors

NotificationRule

public **NotificationRule**()

NotificationRule

public **NotificationRule**(String *recipient*, ActionType *actionType*, Level *level*, String *moduleName*)

Constructor.

Parameters

- **recipient** – the recipient of the notification
- **actionType** – the type of action which will be performed
- **level** – the minimal level for which the notification will trigger
- **moduleName** – the module name for which this rule will trigger, leave null or blank for every module

Methods

getActionType

public ActionType **getActionType**()

Returns the action that should be performed for this notification rule

getId

public Long **getId**()

Returns the database ID

getLevel

public Level **getLevel**()

Returns the minimal level for which the notification will trigger

getModuleName

public String **getModuleName**()

Returns the module name for which this rule will trigger, null or blank for every module

getRecipient

public String **getRecipient**()

Returns the recipient of the notification

matches

public boolean **matches**(StatusMessage *message*)

Checks if the message matches this rule.

Parameters

- **message** – the message which will be checked against this rule

Returns true if message matches this notification rule, false otherwise

setActionType

public void **setActionType**(ActionType *actionType*)

Parameters

- **actionType** – the action that should be performed for this notification rule

setId

public void **setId** (Long *id*)

Parameters

- **id** – the database ID

setLevel

public void **setLevel** (Level *level*)

Parameters

- **level** – the minimal level for which the notification will trigger

setModuleName

public void **setModuleName** (String *moduleName*)

Parameters

- **moduleName** – the module name for which this rule will trigger, leave null or blank for every module

setRecipient

public void **setRecipient** (String *recipient*)

Parameters

- **recipient** – the recipient of the notification

12.1.2 QueueMBean

public class **QueueMBean**

Represents a JMS queue. Holds information about the queue statistics. This information is retrieved using JMX.

Constructors

QueueMBean

public **QueueMBean** (String *destination*)

Parameters

- **destination** – the name of the queue

Methods

getConsumerCount

public long **getConsumerCount**()

Returns number of consumers for this queue (most likely MOTECH instances)

getDequeueCount

public long **getDequeueCount**()

Returns the total number of messages removed from the queue (ack'd by consumer) since last restart

getDestination

public String **getDestination**()

Returns the name of the queue

getEnqueueCount

public long **getEnqueueCount**()

Returns the total number of messages sent to the queue since the last restart

getExpiredCount

public long **getExpiredCount**()

Returns the number of messages that were not delivered because they were expired

getQueueSize

public long **getQueueSize**()

Returns the total number of messages in the queue/store that have not been ack'd by a consumer. This can become confusing at times when compared to the Enqueue Count because the Enqueue Count is a count over a period of time (since the last broker restart) while the Queue Size is not dependent on a period of time but instead on the actual number of messages in the store.

Returns the total number of messages in the queue/store that have not been ack'd by a consumer, not dependent on a period of time

setConsumerCount

public void **setConsumerCount**(long *consumerCount*)

Parameters

- **consumerCount** – number of consumers for this queue (most likely MOTECH instances)

setDequeueCount

public void **setDequeueCount** (long *dequeueCount*)

Parameters

- **dequeueCount** – the total number of messages removed from the queue (ack'd by consumer) since last restart

setDestination

public void **setDestination** (String *destination*)

Parameters

- **destination** – the name of the queue

setEnqueueCount

public void **setEnqueueCount** (long *enqueueCount*)

Parameters

- **enqueueCount** – the total number of messages sent to the queue since the last restart

setExpiredCount

public void **setExpiredCount** (long *expiredCount*)

Parameters

- **expiredCount** – the number of messages that were not delivered because they were expired

setQueueSize

public void **setQueueSize** (long *queueSize*)

Sets the total number of messages in the queue/store that have not been ack'd by a consumer. This can become confusing at times when compared to the Enqueue Count because the Enqueue Count is a count over a period of time (since the last broker restart) while the Queue Size is not dependent on a period of time but instead on the actual number of messages in the store.

Parameters

- **queueSize** – the total number of messages in the queue/store that have not been ack'd by a consumer, not dependent on a period of time

12.1.3 QueueMessage

public class **QueueMessage**

Represents a message from the JMS queue.

Constructors

QueueMessage

public **QueueMessage** (String *messageId*, Boolean *redelivered*, DateTime *timestamp*)

Constructor.

Parameters

- **messageId** – unique identifier for the message
- **redelivered** – whether the message was delivered
- **timestamp** – the timestamp of when the message was sent

Methods

getMessageId

public String **getMessageId** ()

Returns unique identifier for the message

getRedelivered

public Boolean **getRedelivered** ()

Returns true if the message is being resent to the consumer

getTimestamp

public String **getTimestamp** ()

Returns the timestamp of when the message was sent

12.1.4 StatusMessage

public class **StatusMessage**

Represents a message displayed in the 'messages' section of the Admin UI. Persisted by MDS. Apart from the message and its `Level`, it contains also information about the module that sent the message. The timeout field represents the `DateTime` of the message expiration. Status messages are matched against notification rules.

See also: `org.motechproject.admin.domain.NotificationRule`

Constructors

StatusMessage

public **StatusMessage** ()

Constructor. Defaults the level of this message to INFO and the expiration date to 60 minutes from now.

StatusMessage

public **StatusMessage** (String *text*, String *moduleName*, Level *level*)

Constructor. Defaults the expiration date to 60 minutes from now.

Parameters

- **text** – the message content
- **moduleName** – the name of the module to which this message relates to
- **level** – the message level

StatusMessage

public **StatusMessage** (String *text*, String *moduleName*, Level *level*, DateTime *timeout*)

Constructor.

Parameters

- **text** – the message content
- **moduleName** – the name of the module to which this message relates to
- **level** – the message level
- **timeout** – the message expiry date

Methods

getDate

public DateTime **getDate** ()

Returns the date and time at which this message was published

getLevel

public Level **getLevel** ()

Returns the level of the message

getModuleName

public String **getModuleName** ()

Returns the name of the module to which this message relates to

getText

public String **getText** ()

Returns the message content

getTimeout

public DateTime **getTimeout** ()

Returns the message expiry date

setDate

public void **setDate** (DateTime *date*)

Parameters

- **date** – the date and time at which this message was published

setLevel

public void **setLevel** (Level *level*)

Parameters

- **level** – the level of the message

setModuleName

public void **setModuleName** (String *moduleName*)

Parameters

- **moduleName** – the name of the module to which this message relates to

setText

public void **setText** (String *text*)

Parameters

- **text** – the message content

setTimeout

public void **setTimeout** (DateTime *timeout*)

Parameters

- **timeout** – the message expiry date

12.1.5 TopicMBean

public class **TopicMBean**

Represents a JMS topic. Holds information about the topic statistics. This information is retrieved using JMX.

Constructors

TopicMBean

public **TopicMBean** (String *destination*)

Parameters

- **destination** – the name of the topic

Methods

getConsumerCount

public long **getConsumerCount** ()

Returns number of consumers for this topic (most likely MOTECH instances)

getDequeueCount

public long **getDequeueCount** ()

Returns the total number of messages removed from the topic since last restart

getDestination

public String **getDestination** ()

Returns the name of the topic

getEnqueueCount

public long **getEnqueueCount** ()

Returns the total number of messages sent to the topic since the last restart

getExpiredCount

public long **getExpiredCount** ()

Returns the number of messages that were not delivered because they were expired

setConsumerCount

public void **setConsumerCount** (long *consumerCount*)

Parameters

- **consumerCount** – number of consumers for this topic (most likely MOTECH instances)

setDequeueCount

public void **setDequeueCount** (long *dequeueCount*)

Parameters

- **dequeueCount** – the total number of messages removed from the topic since last restart

setDestination

public void **setDestination** (String *destination*)

Parameters

- **destination** – the name of the topic

setEnqueueCount

public void **setEnqueueCount** (long *enqueueCount*)

Parameters

- **enqueueCount** – the total number of messages sent to the topic since the last restart

setExpiredCount

public void **setExpiredCount** (long *expiredCount*)

Parameters

- **expiredCount** – the number of messages that were not delivered because they were expired

12.2 org.motechproject.admin.mds

12.2.1 NotificationRulesDataService

public interface **NotificationRulesDataService** extends MotechDataService<NotificationRule>

MDS data service for `NotificationRule`s.

12.2.2 StatusMessagesDataService

public interface **StatusMessagesDataService** extends MotechDataService<StatusMessage>

MDS data service for `StatusMessage`s.

Methods

findByTimeout

List<StatusMessage> **findByTimeout** (Range<DateTime> *timeout*)

Returns status messages with their timeout value in a given range. Leaving the min value empty will result in

retrieving all messages timing out before the max value. Leaving the max value empty will result in retrieving all messages timing out after the min value.

Parameters

- **timeout** – the range in which the timeout date-time must fall

Returns a list of messages matching the timeout criteria

12.3 org.motechproject.admin.messages

12.3.1 ActionType

public enum **`ActionType`**

Represents an action which should be taken for a notification, in particular which message channel should be used to communicate the notification. Currently the two channels supported are sms and email.

Enum Constants

EMAIL

public static final ActionType **`EMAIL`**

SMS

public static final ActionType **`SMS`**

12.3.2 Level

public enum **`Level`**

Represents the level of a `org.motechproject.admin.domain.StatusMessage`, which is reflected on the UI. Message levels are taken into consideration when processing notification rules.

See also: `org.motechproject.admin.domain.StatusMessage`, `org.motechproject.admin.domain.NotificationRule`

Enum Constants

CRITICAL

public static final Level **`CRITICAL`**

DEBUG

public static final Level **`DEBUG`**

ERROR

public static final Level **ERROR**

INFO

public static final Level **INFO**

WARN

public static final Level **WARN**

12.4 org.motechproject.admin.service

12.4.1 StatusMessageService

public interface **StatusMessageService**

Message service used to send status messages and manage notification rules.

See also: `org.motechproject.admin.domain.StatusMessage`, `org.motechproject.admin.domain.NotificationRule`

Methods

critical

void **critical** (String *text*, String *moduleName*)

Creates a status message with CRITICAL level and posts it in the system. If the message matches any notification rules, appropriate notifications will be triggered. The message will be visible in the message UI until it expires. The value of timeout is set to the default.

Parameters

- **text** – the message content
- **moduleName** – the name of the module this message is related with

critical

void **critical** (String *text*, String *moduleName*, DateTime *timeout*)

Creates a status message with CRITICAL level and posts it in the system. If the message matches any notification rules, appropriate notifications will be triggered. The message will be visible in the message UI until it expires.

Parameters

- **text** – the message content
- **moduleName** – the name of the module this message is related with
- **timeout** – the message expiry date

debug

void **debug** (String *text*, String *moduleName*)

Creates a status message with DEBUG level and posts it in the system. If the message matches any notification rules, appropriate notifications will be triggered. The message will be visible in the message UI until it expires. The value of timeout is set to the default.

Parameters

- **text** – the message content
- **moduleName** – the name of the module this message is related with

debug

void **debug** (String *text*, String *moduleName*, DateTime *timeout*)

Creates a status message with DEBUG level and posts it in the system. If the message matches any notification rules, appropriate notifications will be triggered. The message will be visible in the message UI until it expires.

Parameters

- **text** – the message content
- **moduleName** – the name of the module this message is related with
- **timeout** – the message expiry date

error

void **error** (String *text*, String *moduleName*)

Creates a status message with ERROR level and posts it in the system. If the message matches any notification rules, appropriate notifications will be triggered. The message will be visible in the message UI until it expires. The value of timeout is set to the default.

Parameters

- **text** – the message content
- **moduleName** – the name of the module this message is related with

error

void **error** (String *text*, String *moduleName*, DateTime *timeout*)

Creates a status message with ERROR level and posts it in the system. If the message matches any notification rules, appropriate notifications will be triggered. The message will be visible in the message UI until it expires.

Parameters

- **text** – the message content
- **moduleName** – the name of the module this message is related with
- **timeout** – the message expiry date

getActiveMessages

List<StatusMessage> **getActiveMessages**()

Retrieves status messages that have not expired.

Returns list of active status messages

getAllMessages

List<StatusMessage> **getAllMessages**()

Retrieves all status messages, including those that expired.

Returns list of all status messages

getNotificationRules

List<NotificationRule> **getNotificationRules**()

Retrieves all notification rules.

Returns list of all notification rules

info

void **info**(String *text*, String *moduleName*)

Creates a status message and posts it in the system. If the message matches any notification rules, appropriate notifications will be triggered. The message will be visible in the message UI until it expires. The value of timeout is set to the default.

Parameters

- **text** – the message content
- **moduleName** – the name of the module this message is related with

info

void **info**(String *text*, String *moduleName*, DateTime *timeout*)

Creates a status message with INFO level and posts it in the system. If the message matches any notification rules, appropriate notifications will be triggered. The message will be visible in the message UI until it expires.

Parameters

- **text** – the message content
- **moduleName** – the name of the module this message is related with
- **timeout** – the message expiry date

postMessage

void **postMessage**(StatusMessage *message*)

Creates a status message and posts it in the system. If the message matches any notification rules, appropriate notifications will be triggered. The message will be visible in the message UI until it expires.

Parameters

- **message** – the message to send

postMessage

void **postMessage** (String *text*, String *moduleName*, Level *level*)

Creates a status message and posts it in the system. If the message matches any notification rules, appropriate notifications will be triggered. The message will be visible in the message UI until it expires. The value of timeout is set to the default.

Parameters

- **text** – the message content
- **moduleName** – the name of the module this message is related with
- **level** – the message level

postMessage

void **postMessage** (String *text*, String *moduleName*, Level *level*, DateTime *timeout*)

Creates a status message and posts it in the system. If the message matches any notification rules, appropriate notifications will be triggered. The message will be visible in the message UI until it expires.

Parameters

- **text** – the message content
- **moduleName** – the name of the module this message is related with
- **level** – the message level
- **timeout** – the message expiry date

removeMessage

void **removeMessage** (StatusMessage *message*)

Removes the given message.

Parameters

- **message** – the message to remove

removeNotificationRule

void **removeNotificationRule** (String *id*)

Removes notification rule with given id. Method is deprecated because now Motech is using Long type for primary key.

Parameters

- **id** – the id of notification rule which will be removed

removeNotificationRule

void **removeNotificationRule** (Long *id*)

Removes notification rule with the given id.

Parameters

- **id** – the id of notification rule which will be removed

saveNotificationRules

void **saveNotificationRules** (List<NotificationRule> *notificationRules*)

Creates or updates notification rules. Rule is updated when it has the id field set.

Parameters

- **notificationRules** – the list of notification rules to create/update

saveRule

void **saveRule** (NotificationRule *notificationRule*)

Creates or updates a notification rule. Rule is updated when it has the id field set.

Parameters

- **notificationRule** – the rule to create/update

warn

void **warn** (String *text*, String *moduleName*)

Creates a status message with WARN level and posts it in the system. If the message matches any notification rules, appropriate notifications will be triggered. The message will be visible in the message UI until it expires. The value of timeout is set to the default.

Parameters

- **text** – the message content
- **moduleName** – the name of the module this message is related with

warn

void **warn** (String *text*, String *moduleName*, DateTime *timeout*)

Creates a status message with WARN level and posts it in the system. If the message matches any notification rules, appropriate notifications will be triggered. The message will be visible in the message UI until it expires.

Parameters

- **text** – the message content
- **moduleName** – the name of the module this message is related with
- **timeout** – the message expiry date

12.5 org.motechproject.bundle.extender

12.5.1 MotechExtenderConfigFactory

public class **MotechExtenderConfigFactory** implements FactoryBean<Properties>

Creates the extender configuration for MOTECH, currently blueprint dependency wait time is the only option supported. In order to change the blueprint extender dependency wait time used during platform runtime, `org.motechproject.blueprint.dependencies.waittime` should be set with the wait time in milliseconds. The default blueprint timeout is 5 minutes.

Fields

DEP_WAIT_TIME_ENV

public static final String **DEP_WAIT_TIME_ENV**

DEP_WAIT_TIME_KEY

public static final String **DEP_WAIT_TIME_KEY**

Methods

getObject

public Properties **getObject**()

getObjectType

public Class<?> **getObjectType**()

isSingleton

public boolean **isSingleton**()

12.5.2 MotechOsgiApplicationContextCreator

public class **MotechOsgiApplicationContextCreator** implements OsgiApplicationContextCreator

This is the class responsible for creating Spring application contexts for Spring enabled bundles. Scans bundles and creates a `org.motechproject.bundle.extender.MotechOsgiConfigurableApplicationContext` for Spring enabled bundles. In most cases such bundles have their Spring configuration in their META-INF/spring directory. The context created is then managed by the Blueprint extender.

Methods

createApplicationContext

public DelegatedExecutionOsgiBundleApplicationContext **`createApplicationContext`** (BundleContext *bundleContext*)

setConfigurationScanner

public void **`setConfigurationScanner`** (ConfigurationScanner *configurationScanner*)

Parameters

- **configurationScanner** – the configuration scanner used for scanning bundles

12.5.3 MotechOsgiConfigurableApplicationContext

public class **`MotechOsgiConfigurableApplicationContext`** extends OsgiBundleXmlApplicationContext implements Confi

This is the context class, which will be used by the extender for creating application contexts.

Constructors

MotechOsgiConfigurableApplicationContext

public **`MotechOsgiConfigurableApplicationContext`** (String[] *configurationLocations*)

Constructs the new context using the provided configuration locations.

Parameters

- **configurationLocations** – the configuration location (Spring xml configuration files)

Methods

getNamespace

public String **`getNamespace`** ()

getServletConfig

public ServletConfig **`getServletConfig`** ()

getServletContext

public ServletContext **`getServletContext`** ()

setConfigLocation

public void **`setConfigLocation`** (String *configLocation*)

setNamespace

public void **setNamespace** (String *namespace*)

setServletConfig

public void **setServletConfig** (ServletConfig *servletConfig*)

setServletContext

public void **setServletContext** (ServletContext *servletContext*)

waitForContext

public void **waitForContext** (int *waitTimeInMillis*)

Utility for waiting until the context is ready, which is signalled by firing the `org.springframework.context.event.ContextRefreshedEvent`.

Parameters

- **waitTimeInMillis** – the max wait in milliseconds

12.6 org.motechproject.commons.api

12.6.1 AbstractDataProvider

public abstract class **AbstractDataProvider** implements DataProvider

Base class for every data provider.

Methods

getBody

protected String **getBody** ()

getClassForType

protected Class<?> **getClassForType** (String *type*)

getLogger

protected Logger **getLogger** ()

getPackageRoot

public abstract String **getPackageRoot** ()
Returns root package for this data provider.

Returns root package

getSupportClasses

public abstract List<Class<?>> **getSupportClasses** ()
Returns list of classes supported by this data provider.

Returns the list of supported classes

isAssignable

protected boolean **isAssignable** (Class<?> *check*, List<Class<?>> *classes*)

setBody

protected void **setBody** (String *body*)

setBody

protected void **setBody** (Resource *resource*)

supports

public boolean **supports** (String *type*)

toJSON

public String **toJSON** ()

12.6.2 ApplicationContextServiceReferenceUtils

public final class **ApplicationContextServiceReferenceUtils**
Utility class for `ServiceReference` class.

Fields

SERVICE_NAME

public static final String **SERVICE_NAME**

Methods

isNotValid

public static boolean **isNotValid** (ServiceReference *serviceReference*)
Checks if given `ServiceReference` is not valid.

Parameters

- **serviceReference** – the `ServiceReference` to be validated

Returns true if given `ServiceReference` is not valid, false otherwise

isValid

public static boolean **isValid** (ServiceReference *serviceReference*)
Checks whether given `ServiceReference` is valid or not.

Parameters

- **serviceReference** – the `ServiceReference` to be validated

Returns true if given `ServiceReference` is valid, false otherwise

12.6.3 ClassUtils

public final class **ClassUtils**
Utility class responsible for casting classes.

Methods

filterByClass

public static <T> List<T> **filterByClass** (Class<T> *clazz*, Enumeration *enumeration*)
Filters the given `Enumeration` searching for instances of the given class.

Parameters

- **clazz** – the class used for filtering
- **enumeration** – the filtered elements
- **<T>** – the class used for filtering and returning properly cast objects

Returns the list of instances of given class found in the enumeration

12.6.4 DataProvider

public interface **DataProvider**
Interface for classes that act as data providers for Tasks.

Methods

getName

String **getName**()

Returns data provider name.

Returns the data provider name

lookup

Object **lookup**(String *type*, String *lookupName*, Map<String, String> *lookupFields*)

Returns single object matching given conditions.

Parameters

- **type** – the type of searched object
- **lookupName** – the name of used lookup
- **lookupFields** – the map of fields names and expected values

Returns single object matching conditions

supports

boolean **supports**(String *type*)

Checks if given type is supported by the `DataProvider`.

Parameters

- **type** – the type to be checked

Returns true if type is supported, false otherwise

toJSON

String **toJSON**()

Converts data provider to json.

Returns json stored as `String`

12.6.5 IdentityProvider

public interface **IdentityProvider**

Interface for classes providing identity.

Methods

getIdentity

String **getIdentity**()

Returns identity of class implementing this interface.

Returns the identity

12.6.6 MotechEnumUtils

public final class **MotechEnumUtils**
Misc enum-related helper functions

Methods

toEnumSet

public static <T extends Enum> Set<T> **toEnumSet** (Class<T> *enumClass*, Set<String> *strings*)
Returns a set of enums given a set of strings and an enum class

Parameters

- **enumClass** – the enum class
- **strings** – a set of strings

Returns the enum set constructed from the given string set

toEnumSet

public static <T extends Enum> Set<T> **toEnumSet** (Class<T> *enumClass*, String *csv*)
Returns a set of enums given a comma separated string and an enum class

Parameters

- **enumClass** – the enum class
- **csv** – a comma separated string representing a set of enum values

Returns the enum set constructed from the given string

toString

public static String **toString** (Set<? extends Enum> *items*)
Returns a csv string given a set of enums

Parameters

- **items** – a set of enums

Returns the csv string constructed from the given enum set

toStringSet

public static Set<String> **toStringSet** (Set<? extends Enum> *items*)
Returns a set of strings given a set of enums

Parameters

- **items** – a set of enums

Returns the string set constructed from the given enum set

12.6.7 MotechException

public class **MotechException** extends RuntimeException

Constructors

MotechException

public **MotechException** (String *s*, Throwable *throwable*)

MotechException

public **MotechException** (String *message*)

12.6.8 MotechMapUtils

public final class **MotechMapUtils**

The `MotechMapUtils` class contains methods that allow modifications and operations on maps

Methods

asProperties

public static Properties **asProperties** (Map<Object, Object> *map*)

Converts java.util.Map into java.util.Properties

Parameters

- **map** – Map to convert

Returns Properties, created from given map

mergeMaps

public static Map<Object, Object> **mergeMaps** (Map<Object, Object> *overridingMap*, Map<Object, Object> *baseMap*)

Null-safe merge of two maps. If both parameters are null it returns empty map. If one of the maps is null, it returns the other one. If a key is present in two maps, the value in the merged map will be taken from the overriding map.

Parameters

- **overridingMap** – The map overriding values in the base map
- **baseMap** – The base map

Returns merged map

12.6.9 NanoStopWatch

public class **NanoStopWatch**

Simple class for measuring time.

Constructors

NanoStopWatch

public **NanoStopWatch** ( )
Default constructor.

Methods

duration

public long **duration** ( )
Return time elapsed since this timer was started.

Returns the time elapsed since timer started

start

public NanoStopWatch **start** ( )
Starts the timer.

Returns this instance of class

12.6.10 Range

public class **Range**<T>
Class representing range between two objects of same type.

Parameters

- **<T>** –

Constructors

Range

public **Range** (T *min*, T *max*)
Constructor.

Parameters

- **min** – minimum value for range
- **max** – maximum value for range

Methods

equals

public boolean **equals** (Object *o*)

getMax

public T **getMax**()

getMin

public T **getMin**()

hashCode

public int **hashCode**()

12.6.11 SystemIdentityProvider

public class **SystemIdentityProvider** implements IdentityProvider

Implementation of `IdentityProvider`.

Methods

getIdentity

public String **getIdentity**()

12.6.12 TasksEventParser

public interface **TasksEventParser**

The `TasksEventParser` interface provides a way for modules to define a custom way to handle trigger events. Before event parameters or subject are parsed, the Tasks module will first check if received event contains parameter with key custom_tasks_event_parser and if so, it will look for custom parser that matches the name exposed via `getName()` method. If any module wants to use a custom event parser, they should simply implement this interface and expose it as OSGi service.

Fields

CUSTOM_PARSER_EVENT_KEY

String **CUSTOM_PARSER_EVENT_KEY**

Methods

getName

String **getName**()

Returns the name of the module that registers custom parser. Tasks module will look for all registered event parsers and try to match the name passed in the event parameter `org.motechproject.tasks.custom_event_parser` with the name returned by this method. If there's a match, a custom parser with matched name will be used.

Returns Module name that registers custom event parser

parseEventParameters

Map<String, Object> **parseEventParameters** (String *eventSubject*, Map<String, Object> *eventParameters*)

Given a map of event parameters, parses them in a user-defined way to receive a custom map of event parameters

Parameters

- **eventSubject** – The original event subject
- **eventParameters** – Initial event parameters

Returns Custom, parsed event parameters, that will be used instead of initial params

parseEventSubject

String **parseEventSubject** (String *eventSubject*, Map<String, Object> *eventParameters*)

Adjusts event subject of the event. Thanks to this, we are able to map events of the same event subject to different triggers. If there's no need to modify the subject of an event (eg. one event should map only one trigger), simply return the original subject, that is passed as an argument.

Parameters

- **eventSubject** – The original event subject
- **eventParameters** – Initial event parameters

Returns Custom event subject

12.6.13 Tenant

public class **Tenant**

Class representing tenant used for getting queue statistics.

Constructors

Tenant

Tenant (TenantIdentity *identity*)

Constructor.

Parameters

- **identity** – the identity of created tenant

Methods

canHaveQueue

public boolean **canHaveQueue** (String *queueName*)

Checks whether tenant can have given queue.

Parameters

- **queueName** – the queue name to be checked

Returns true if tenant cant have given queue, false otherwise

current

public static Tenant **current** ()
Returns current tenant.

Returns the current tenant

getId

public String **getId** ()

getSuffixedId

public String **getSuffixedId** ()

12.6.14 TenantIdentity

public class **TenantIdentity**
Holds identity of a tenant.

Constructors

TenantIdentity

public **TenantIdentity** ()
Constructor.

TenantIdentity

public **TenantIdentity** (IdentityProvider *identityProvider*)
Constructor.

Parameters

- **identityProvider** – the identity provider to be used

Methods

getId

public String **getId** ()

12.6.15 ThreadSuspender

public final class **ThreadSuspender**

Util class, that allows to put current thread to sleep.

Methods

sleep

public static void **sleep** (int *millis*, String *interruptedMessage*)

Causes the currently executing thread to sleep (temporarily cease execution) for the specified number of milliseconds, subject to the precision and accuracy of system timers and schedulers.

Parameters

- **millis** – the length of time to sleep in milliseconds
- **interruptedMessage** – the message to put in logs, in case the waiting gets interrupted

sleep

public static void **sleep** (int *millis*)

Causes the currently executing thread to sleep (temporarily cease execution) for the specified number of milliseconds, subject to the precision and accuracy of system timers and schedulers.

Parameters

- **millis** – the length of time to sleep in milliseconds

12.7 org.motechproject.commons.api.json

12.7.1 MotechJsonReader

public class **MotechJsonReader**

Class responsible for creating objects from json. It can use `InputStream`, `String` or file classpath. This class uses Gson underneath.

See also: Google Gson

Constructors

MotechJsonReader

public **MotechJsonReader** ()

Constructor.

MotechJsonReader

public **MotechJsonReader** (FieldNamingStrategy *fieldNamingStrategy*)

Constructor.

Parameters

- **fieldNamingStrategy** – the field naming strategy to be used when deserializing object

Methods

readFromFile

public Object **readFromFile** (String *classpathFile*, Type *ofType*)

Creates object of type `ofType` from file under given classpath.

Parameters

- **classpathFile** – the file to deserialize
- **ofType** – the type of created object

Returns object of type `ofType`

readFromStream

public Object **readFromStream** (InputStream *stream*, Type *ofType*)

Creates object of type `ofType` from input stream.

Parameters

- **stream** – the stream to deserialize
- **ofType** – the type of created object

Returns object of type `ofType`

readFromStreamOnlyExposeAnnotations

public Object **readFromStreamOnlyExposeAnnotations** (InputStream *stream*, Type *ofType*)

Creates object of type `ofType` from input stream. Will only deserialize fields with `Expose` annotation.

Parameters

- **stream** – the stream to deserialize
- **ofType** – the type of created object

Returns object of type `ofType`

readFromString

public Object **readFromString** (String *text*, Type *ofType*)

Creates object of type `ofType` from given `String`.

Parameters

- **text** – the `String` to deserialize
- **ofType** – the type of created object

Returns object of type `ofType`

readFromString

public Object **readFromString** (String *text*, Type *ofType*, Map<Type, Object> *typeAdapters*)

Creates object of type `ofType` from given `String` using user-specified adapters.

Parameters

- **text** – the `String` to deserialize
- **ofType** – the type of created object
- **typeAdapters** – custom adapters to use for deserialization

Returns object of type `ofType`

readFromStringOnlyExposeAnnotations

public Object **readFromStringOnlyExposeAnnotations** (String *text*, Type *ofType*)

Creates object of type `ofType` from given `String`. Will only deserialize fields with `Expose` annotation.

Parameters

- **text** – the `String` to deserialize
- **ofType** – the type of created object

Returns object of type `ofType`

12.8 org.motechproject.commons.api.model

12.8.1 MotechProperties

public class **MotechProperties** extends HashMap<String, String>

12.9 org.motechproject.commons.date.exception

12.9.1 ParseException

public class **ParseException** extends RuntimeException

Signals a problem with parsing dates or periods.

Constructors

ParseException

public **ParseException** (String *s*)

12.10 org.motechproject.commons.date.model

12.10.1 DayOfWeek

public enum **DayOfWeek**
Represents a single day of week.

Enum Constants

Friday

public static final DayOfWeek **Friday**

Monday

public static final DayOfWeek **Monday**

Saturday

public static final DayOfWeek **Saturday**

Sunday

public static final DayOfWeek **Sunday**

Thursday

public static final DayOfWeek **Thursday**

Tuesday

public static final DayOfWeek **Tuesday**

Wednesday

public static final DayOfWeek **Wednesday**

12.10.2 Time

public class **Time** implements Comparable<Time>, Serializable
Represents time as number of hours and minutes.

Constructors

Time

public **Time** ()
Constructor.

Time

public **Time** (int *hour*, int *minute*)
Constructor.

Parameters

- **hour** – the hour to be stored, not null
- **minute** – the minute to be stored, not null

Time

public **Time** (LocalTime *localTime*)
Constructor.

Parameters

- **localTime** – the time to be stored, not null

Time

public **Time** (String *timeStr*)
Constructor.

Parameters

- **timeStr** – the time represented as `String`

Throws

- **IllegalArgumentException** – if `timeStr` doesn't match "HH:MM" pattern

Methods

compareTo

public int **compareTo** (Time *otherTime*)

equals

public boolean **equals** (Object *obj*)

getHour

public Integer **getHour** ()

getMinute

public Integer **getMinute**()

gt

public boolean **gt**(Time *toCompare*)

Checks if given object is before this object or both objects represents the same time.

Parameters

- **toCompare** – the object to be compared with this object

Returns false if given `Time` is after this, true otherwise

hashCode

public int **hashCode**()

isAfter

public boolean **isAfter**(Time *other*)

Checks whether this is after the given time.

Parameters

- **other** – the `Time` to be compared with this object

Returns true if this time is after other, false otherwise

isBefore

public boolean **isBefore**(Time *other*)

Checks whether this is before the given time.

Parameters

- **other** – the `Time` to be compared with this object

Returns true if this time is before other, false otherwise

lt

public boolean **lt**(Time *toCompare*)

Checks if given object is after this object or both objects represents the same time.

Parameters

- **toCompare** – the object to be compared with this object

Returns false if given `Time` is before this, true otherwise

parseTime

public static Time **parseTime** (String *time*, String *separator*)

Parses given `String` using the separator.

Parameters

- **time** – the `String` to be parsed, null returns null
- **separator** – the separator used to distinguish minute from hour, not null

Throws

- **IllegalArgumentException** – if `time` doesn't match "HHMM" pattern

Returns the instance of `Time`

setHour

public void **setHour** (Integer *hour*)

setMinute

public void **setMinute** (Integer *minute*)

timeStr

public String **timeStr** ()

Returns `String` representation of stored time.

Returns the time stored as a `String`

toDateTime

public DateTime **toDateTime** (DateTime *dateTime*)

Creates `DateTime` instance with time stored in this object.

Parameters

- **dateTime** – the `DateTime` to be used as base for the new `DateTime`

Returns the `DateTime` with stored time

toDateTime

public DateTime **toDateTime** (LocalDate *date*)

Creates `DateTime` instance with time stored in this object.

Parameters

- **date** – the `LocalDate` to be used as base for the new `DateTime`

Returns the `DateTime` with stored time

toString

public String **toString**()

valueOf

public static Time **valueOf**(String *str*)

Creates instance of `Time` for given `String`.

Parameters

- **str** – the `String` to be parsed to `Time`

Returns the `Time` parsed from given `String`

12.11 org.motechproject.commons.date.util

12.11.1 DateTimeSourceUtil

public final class **DateTimeSourceUtil**

Utility class for `DateTimeSource`.

Methods

now

public static DateTime **now**()

Returns current time as an instance of `DateTime`.

Returns the current time

setSourceInstance

public static void **setSourceInstance**(DateTimeSource *sourceInstance*)

timeZone

public static DateTimeZone **timeZone**()

Returns time zone used by class.

Returns time zone used by class

today

public static LocalDate **today**()

Returns current local date.

Returns the current local date as an instance of `LocalDate`

12.11.2 DateUtil

public final class **DateUtil**

Utility class for various classes from `org.joda.time` package. Using this class for retrieving the current date and time will allow mocking time, so should be always used throughout the platform instead of calling the underlying date-time API directly.

Methods

daysPast

public static int **daysPast** (LocalDate *localDate*, DayOfWeek *dayOfWeek*)

Counts the days passed between given date and given day of week.

Parameters

- **localDate** – the given date
- **dayOfWeek** – the given day of week

Returns the number of days passed between `localDate` and a `org.motechproject.commons.date.model.DayOfWeek`

daysStarting

public static List<DayOfWeek> **daysStarting** (DayOfWeek *day*, int *numberOfDays*)

Returns list of days equal to numberOfDays from given day.

Parameters

- **day** – the day of week from which list should begin
- **numberOfDays** – the number of days which should be included in the list

Returns the list of days

daysToCalendarWeekEnd

public static int **daysToCalendarWeekEnd** (LocalDate *date*, int *calendarWeekStartDay*)

Returns number of days left until weekend.

Parameters

- **date** – the date from which days should be counted
- **calendarWeekStartDay** – the day at which weekend starts

Returns days left until weekend

endOfDay

public static DateTime **endOfDay** (Date *dateTime*)

Creates `DateTime` with time set to 23:59:59:999 and date equal to the given one.

Parameters

- **dateTime** – the `Date` to be stored

Returns the new `DateTime` instance

getDifferenceOfDatesInYears

public static int **`getDifferenceOfDatesInYears`** (Date *startDate*)

Returns difference, in years, between given date and today.

Parameters

- **startDate** – the date from which year are counted

Returns difference in years

greaterThanOrEqualTo

public static List<DateTime> **`greaterThanOrEqualTo`** (DateTime *date*, List<DateTime> *dates*)

Filters given dates and returns only those that are past or at the given date.

Parameters

- **date** – the date to be used as filter
- **dates** – dates to be filtered

Returns list of date that are equal or greater than given date

inRange

public static boolean **`inRange`** (DateTime *reference*, DateTime *start*, DateTime *end*)

Checks if first date is in period between second and third.

Parameters

- **reference** – the date to be checked
- **start** – the start of the period
- **end** – the end of the period

Returns true if first date is in range, false otherwise

isOnOrAfter

public static boolean **`isOnOrAfter`** (DateTime *firstDate*, DateTime *secondDate*)

Checks whether first date is on or after second one.

Parameters

- **firstDate** – the date what should be on or after
- **secondDate** – the date to compare to

Returns false if first date is before second, true otherwise

isOnOrBefore

public static boolean **isOnOrBefore** (DateTime *firstDate*, DateTime *secondDate*)

Checks whether first date is on or before second one.

Parameters

- **firstDate** – the date what should be on or before
- **secondDate** – the date to compare to

Returns false if first date is after second, true otherwise

lessThan

public static List<DateTime> **lessThan** (DateTime *date*, List<DateTime> *dates*)

Filters given dates and returns only those that are before the given date.

Parameters

- **date** – the date to be used as filter
- **dates** – dates to be filtered

Returns list of date that are before the given date

newDate

public static LocalDate **newDate** (int *year*, int *month*, int *day*)

Creates new instance of `LocalDate`.

Parameters

- **year** – the year to be stored in created instance
- **month** – the month to be stored in created instance
- **day** – the day to be stored in created instance

Returns the instance of `LocalDate` with given year, month and day

newDate

public static LocalDate **newDate** (Date *date*)

Creates new `LocalDate` from given `Date`.

Parameters

- **date** – the `Date` to be parsed to `LocalDate`

Returns the new instance of `LocalTime`, null if `date` was null

newDate

public static LocalDate **newDate** (DateTime *dateTime*)

Creates new `LocalDate` from given `DateTime`.

Parameters

- **dateTime** – the `DateTime` to be parsed to `LocalDate`

Returns the new instance of `LocalTime`, null if `date` was null

newDateTime

public static DateTime **`newDateTime`** (LocalDate *localDate*, int *hour*, int *minute*, int *second*)

Creates new instance of `DateTime`.

Parameters

- **localDate** – the date to be stored in created instance
- **hour** – the hour to be stored in created instance
- **minute** – the minute to be stored in created instance
- **second** – the second to be stored in created instance

Returns the instance of `DateTime` with given date and time

newDateTime

public static DateTime **`newDateTime`** (Date *date*)

Creates new `DateTime` from given `Date`.

Parameters

- **date** – the `Date` to be parsed to `DateTime`

Returns the new `DateTime` instance

newDateTime

public static DateTime **`newDateTime`** (LocalDate *date*)

Creates new `DateTime` from given `LocalDate`. Time is set to 00:00:00.

Parameters

- **date** – the `Date` to be parsed to `DateTime`

Returns the new `DateTime` instance

newDateTime

public static DateTime **`newDateTime`** (LocalDate *localDate*, Time *time*)

Creates new `DateTime` from given `LocalDate` and `Time`.

Parameters

- **localDate** – the `LocalDate` to be stored
- **time** – the `Time` to be stored

Returns the new `DateTime` instance

newDateTime

public static DateTime **newDateTime** (int *year*, int *month*, int *day*, Time *time*)

Creates new `DateTime` from given information.

Parameters

- **year** – the year to be stored
- **month** – the month to be stored
- **day** – the day to be stored
- **time** – the time to be stored

Returns the new `DateTime` instance

newDateTime

public static DateTime **newDateTime** (int *year*, int *month*, int *day*)

Creates new `DateTime` from given information. Time is set to 00:00:00.

Parameters

- **year** – the year to be stored
- **month** – the month to be stored
- **day** – the day to be stored

Returns the new `DateTime` instance

newDateTime

public static DateTime **newDateTime** (int *year*, int *month*, int *day*, int *hour*, int *minute*, int *second*)

Creates new `DateTime` from given information.

Parameters

- **year** – the year to be stored
- **month** – the month to be stored
- **day** – the day to be stored
- **hour** – the hour to be stored
- **minute** – the minute to be stored
- **second** – the second to be stored

Returns the new `DateTime` instance

nextApplicableWeekDay

public static DateTime **nextApplicableWeekDay** (DateTime *fromDate*, List<DayOfWeek> *applicableDays*)

Returns first next applicable week day.

Parameters

- **fromDate** – the date from which next day should be searched

- **applicableDays** – list of applicable days

Returns next applicable week day

nextApplicableWeekDayIncludingFromDate

public static DateTime **nextApplicableWeekDayIncludingFromDate** (DateTime *fromDate*, List<DayOfWeek> *applicableDays*)

Returns first next applicable week day(including current day).

Parameters

- **fromDate** – the date from which next day should be searched
- **applicableDays** – list of applicable days

Returns next applicable week day

now

public static DateTime **now** ()

Returns current time as an instance of `DateTime`.

Returns the current time

nowUTC

public static DateTime **nowUTC** ()

Returns current time in UTC time zone.

Returns the current time as instance of `DateTime`

setTimeZone

public static DateTime **setTimeZone** (DateTime *dateTime*)

Sets time zone, for given `DateTime`, to default.

Parameters

- **dateTime** – the `DateTime` to have time zone set

Returns the `DateTime` with time zone set

setTimeZoneUTC

public static DateTime **setTimeZoneUTC** (DateTime *dateTime*)

Sets time zone, for given `DateTime`, to UTC time zone.

Parameters

- **dateTime** – the `DateTime` to have time zone set

Returns the `DateTime` with time zone set

time

public static Time **time** (DateTime *dateTime*)

Extracts time from given `DateTime` and converts it to an instance of `Time`.

Parameters

- **dateTime** – the `DateTime` storing the time

Returns the new instance of `Time`

today

public static LocalDate **today** ()

Returns current local date.

Returns the current local date as an instance of `LocalDate`

tomorrow

public static LocalDate **tomorrow** ()

Returns tomorrow local date.

Returns the tomorrow local date as an instance of `LocalDate`

12.11.3 JodaFormatter

public class **JodaFormatter**

Class responsible for parsing and formatting several classes from `org.joda.time` package.

Constructors

JodaFormatter

public **JodaFormatter** ()

Default constructor.

Methods

formatDateTime

public String **formatDateTime** (DateTime *dateTime*)

Formats `DateTime` as text.

Parameters

- **dateTime** – the `DateTime` to be formatted.

Returns the text representing the `DateTime`

formatPeriod

public String **formatPeriod** (Period *period*)
Formats Joda period as text, eg: "1 year"

Parameters

- **period** – time interval

Returns the text representing the period

parse

public Period **parse** (String *intervalString*, Locale *locale*)
Parses time interval in different units, eg: "1 year"

Parameters

- **intervalString** – time interval format number: integer unit : year, month, week, day, hour, minute, second (can use plural forms also) currently compound units like 1 year and 2 months are not supported
- **locale** – the locale to be used when parsing given `String`

Returns the given `String` parsed to `import org.joda.time.Period`

parseDateTime

public DateTime **parseDateTime** (String *isoDateTime*)
Parses given `String` to `DateTime`.

Parameters

- **isoDateTime** – the string to be parsed, must be using ISO-8601 standard

Returns the `DateTime` parsed from `String`

parsePeriod

public Period **parsePeriod** (String *intervalString*)
Parses time interval in different units, eg: "1 year"

Parameters

- **intervalString** – time interval format number: integer unit : year, month, week, day, hour, minute, second (can use plural forms also) currently compound units like 1 year and 2 months are not supported

Returns the given `String` parsed to `import org.joda.time.Period`

12.12 org.motechproject.commons.date.util.datetime

12.12.1 DateTimeSource

public interface **DateTimeSource**
A datetime source for the application. Allows mocking of time.

Methods

now

DateTime **now**()
Used for retrieving the current date and time.

Returns `org.joda.time.DateTime` representing the current date and time

timeZone

DateTimeZone **timeZone**()
Returns the timezone we are in.

Returns the timezone

today

LocalDate **today**()
Used for retrieving the current date.

Returns `org.joda.time.DateTime` representing the current date

12.12.2 DefaultDateTimeSource

public class **DefaultDateTimeSource** implements DateTimeSource
Default implementation of `DateTimeSource`.

Constructors

DefaultDateTimeSource

public **DefaultDateTimeSource**()

Methods

now

public DateTime **now**()

timeZone

public DateTimeZone **timeZone**()

today

public LocalDate **today**()

12.13 org.motechproject.commons.sql.service

12.13.1 SqlDBManager

public interface **SqlDBManager**

Classes implementing this interface are responsible for retrieving sql properties from the bootstrap configuration, updating sql-related properties for modules and creating databases for given properties.

Methods

checkForDatabase

boolean **checkForDatabase** (String *dbName*)

Check if a database with the given name exists

Parameters

- **dbName** – database name

Returns true if database exists, otherwise false

createDatabase

boolean **createDatabase** (String *dbName*)

Create a database with the given name

Returns true if the database was created properly

getChosenSQLDriver

String **getChosenSQLDriver** ()

Returns the SQL driver class name that has been chosen during bootstrap configuration of the system.

Returns Class name of the SQL driver, chosen during bootstrap configuration.

getSqlProperties

Properties **getSqlProperties** (Properties *propertiesToUpdate*)

Being passed raw properties, inserts correct SQL configuration in the correct places. Current replacement codes are:

•${sql.driver}

•${sql.user}

•${sql.password}

•${sql.url}

•${sql.quartz.delegateClass}

As a result of calling this method, all occurrences of the above keys get replaced with actual sql properties, retrieved from the provided bootstrap configuration.

Parameters

- **propertiesToUpdate** – Raw properties, containing replacement codes

Throws

- **IOException** – In case an I/O exception occurs when loading / merging properties

Returns Actual properties, with sql configuration from bootstrap

hasColumn

boolean **hasColumn** (String *database*, String *table*, String *column*)

Checks whether table with the given name has a column with the given name.

Parameters

- **table** – the name of the table
- **column** – the name of the column

Throws

- **SQLException** – when incorrect data was given

Returns true if the table has that column, false otherwise

12.14 org.motechproject.commons.sql.util

12.14.1 Drivers

public final class **Drivers**

Utility for storing supported SQL and Quartz driver class names.

Fields

MYSQL_DRIVER

public static final String **MYSQL_DRIVER**

POSTGRESQL_DRIVER

public static final String **POSTGRESQL_DRIVER**

QUARTZ_POSTGRESQL_DELEGATE

public static final String **QUARTZ_POSTGRESQL_DELEGATE**

QUARTZ_STD_JDBC_DELEGATE

public static final String **QUARTZ_STD_JDBC_DELEGATE**

12.14.2 JdbcUrl

public class **JdbcUrl**

Represents a JDBC URL, allows getting the url or database name parts. Support primarily Postgresql and MySQL.

Constructors

JdbcUrl

public **JdbcUrl** (String *url*)

Constructs this from the provided url.

Parameters

- **url** – the url to constructs this object from

Throws

- **URISyntaxException** – if the url is invalid

Methods

getDbName

public String **getDbName** ()

Returns the name of the database from this url

getUrlForDbServer

public String **getUrlForDbServer** ()

Returns get an url for connecting with the server without the database part

getValue

public String **getValue** ()

Returns returns the entire url

12.15 org.motechproject.config.core

12.15.1 MotechConfigurationException

public class **MotechConfigurationException** extends RuntimeException

The object of this class is thrown when there is a problem with reading the configuration from the predefined sources.

Constructors

MotechConfigurationException

public **MotechConfigurationException** (String *message*, Exception *exception*)

Parameters

- **message** – A descriptive message explaining the nature of the problem resulted in exception
- **exception** – Actual exception (if any) that this exception resulted from

MotechConfigurationException

public **MotechConfigurationException** (String *message*)

12.16 org.motechproject.config.core.constants

12.16.1 ConfigurationConstants

public final class **ConfigurationConstants**
Provides all the configuration constants.

Fields

AMQ_BROKER_URL

public static final String **AMQ_BROKER_URL**

AMQ_CONCURRENT_CONSUMERS

public static final String **AMQ_CONCURRENT_CONSUMERS**

AMQ_MAX_CONCURRENT_CONSUMERS

public static final String **AMQ_MAX_CONCURRENT_CONSUMERS**

AMQ_MAX_REDELIVERIES

public static final String **AMQ_MAX_REDELIVERIES**

AMQ_QUEUE_EVENTS

public static final String **AMQ_QUEUE_EVENTS**

AMQ_QUEUE_SCHEDULER

public static final String **AMQ_QUEUE_SCHEDULER**

AMQ_REDELIVERY_DELAY_IN_MILLIS

public static final String **AMQ_REDELIVERY_DELAY_IN_MILLIS**

BUNDLE_ID

public static final String **BUNDLE_ID**

BUNDLE_SECTION

public static final String **BUNDLE_SECTION**

BUNDLE_SETTINGS_CHANGED_EVENT_SUBJECT

public static final String **BUNDLE_SETTINGS_CHANGED_EVENT_SUBJECT**

BUNDLE_SYMBOLIC_NAME

public static final String **BUNDLE_SYMBOLIC_NAME**

CONFIG_MODULE_DIR_PREFIX

public static final String **CONFIG_MODULE_DIR_PREFIX**

DATANUCLEUS_SETTINGS_FILE_NAME

public static final String **DATANUCLEUS_SETTINGS_FILE_NAME**

EMAIL_REQUIRED

public static final String **EMAIL_REQUIRED**

EVENT_RELAY_CLASS_NAME

public static final String **EVENT_RELAY_CLASS_NAME**

FAILURE_LOGIN_LIMIT

public static final String **FAILURE_LOGIN_LIMIT**

FILE_CHANGED_EVENT_SUBJECT

public static final String **FILE_CHANGED_EVENT_SUBJECT**

FILE_CREATED_EVENT_SUBJECT

public static final String **FILE_CREATED_EVENT_SUBJECT**

FILE_DELETED_EVENT_SUBJECT

public static final String **FILE_DELETED_EVENT_SUBJECT**

FILE_PATH

public static final String **FILE_PATH**

JMX_BROKER

public static final String **JMX_BROKER**

JMX_HOST

public static final String **JMX_HOST**

JSON_EXTENSION

public static final String **JSON_EXTENSION**

LANGUAGE

public static final String **LANGUAGE**

LOGINMODE

public static final String **LOGINMODE**

MIN_PASSWORD_LENGTH

public static final String **MIN_PASSWORD_LENGTH**

MOTECH_EVENT_CLASS_NAME

public static final String **MOTECH_EVENT_CLASS_NAME**

PASSWORD_VALIDATOR

public static final String **PASSWORD_VALIDATOR**

PLATFORM_SETTINGS_CHANGED_EVENT_SUBJECT

public static final String **PLATFORM_SETTINGS_CHANGED_EVENT_SUBJECT**

PROPERTIES_EXTENSION

public static final String **PROPERTIES_EXTENSION**

PROVIDER_NAME

public static final String **PROVIDER_NAME**

PROVIDER_URL

public static final String **PROVIDER_URL**

RAW_DIR

public static final String **RAW_DIR**

SERVER_URL

public static final String **SERVER_URL**

SESSION_TIMEOUT

public static final String **SESSION_TIMEOUT**

SETTINGS

public static final String **SETTINGS**

SETTINGS_FILE_NAME

public static final String **SETTINGS_FILE_NAME**

STATUS_MSG_TIMEOUT

public static final String **STATUS_MSG_TIMEOUT**

UPLOAD_SIZE

public static final String **UPLOAD_SIZE**

12.17 org.motechproject.config.core.domain

12.17.1 AbstractDBConfig

public abstract class **AbstractDBConfig**

This abstract class encapsulates the database configuration, composed of as db url, username and password.

Constructors

AbstractDBConfig

public **AbstractDBConfig** (String *url*, String *driver*, String *username*, String *password*)

Constructor.

Parameters

- **url** – the url of the database
- **driver** – the driver class name for the database
- **username** – the username to the database
- **password** – the password for the database

Methods

equals

public boolean **equals** (Object *o*)

getDriver

public String **getDriver** ()

getPassword

public String **getPassword** ()

getUrl

public String **getUrl** ()

getUsername

public String **getUsername**()

hashCode

public int **hashCode**()

toString

public String **toString**()

12.17.2 BootstrapConfig

public class **BootstrapConfig**

Represents the bootstrap configuration object. It is composed of:

1. DBConfig - represents the database related bootstrap object.
2. Tenant ID - represents the identifier of the tenant.
3. Configuration source - represents the source of configuration (FILE / UI).
4. ActiveMq Config - represents the properties of ActiveMq.

Fields

CONFIG_SOURCE

public static final String **CONFIG_SOURCE**

DEFAULT_OSGI_FRAMEWORK_STORAGE

public static final String **DEFAULT_OSGI_FRAMEWORK_STORAGE**

DEFAULT_TENANT_ID

public static final String **DEFAULT_TENANT_ID**

OSGI_FRAMEWORK_STORAGE

public static final String **OSGI_FRAMEWORK_STORAGE**

QUEUE_URL

public static final String **QUEUE_URL**

SQL_DRIVER

public static final String **SQL_DRIVER**

SQL_PASSWORD

public static final String **SQL_PASSWORD**

SQL_URL

public static final String **SQL_URL**

SQL_USER

public static final String **SQL_USER**

TENANT_ID

public static final String **TENANT_ID**

Constructors

BootstrapConfig

public **BootstrapConfig** (SQLDBConfig *sqlConfig*, String *tenantId*, ConfigSource *configSource*, String *osgiFrameworkStorage*, String *queueUrl*)

Constructor.

Parameters

- **sqlConfig** – the configuration of a SQL database
- **tenantId** – the ID of a tenant
- **configSource** – the source from which MOTECH configuration should be read
- **osgiFrameworkStorage** – the directory used as the bundle cache
- **queueUrl** – the URL of the JMS broker

BootstrapConfig

public **BootstrapConfig** (SQLDBConfig *sqlConfig*, String *tenantId*, ConfigSource *configSource*, String *osgiFrameworkStorage*, String *queueUrl*, Properties *activeMqProperties*)

Constructor.

Parameters

- **sqlConfig** – the configuration of a SQL database
- **tenantId** – the ID of a tenant
- **configSource** – the source from which MOTECH configuration should be read

- **osgiFrameworkStorage** – the directory used as the bundle cache
- **queueUrl** – the URL of the JMS broker
- **activeMqProperties** – the ActiveMQ properties

Throws

- **org.motechproject.config.core.MotechConfigurationException** – if sqlConfig is null.

Methods

equals

public boolean **`equals`** (Object *o*)

getActiveMqProperties

public Properties **`getActiveMqProperties`** ()

getConfigSource

public ConfigSource **`getConfigSource`** ()

getOsgiFrameworkStorage

public String **`getOsgiFrameworkStorage`** ()

getQueueUrl

public String **`getQueueUrl`** ()

getSqlConfig

public SQLDBConfig **`getSqlConfig`** ()

getTenantId

public String **`getTenantId`** ()

hashCode

public int **`hashCode`** ()

setQueueUrl

public final void **`setQueueUrl`** (String *queueUrl*)

toString

public String **toString**()

12.17.3 ConfigLocation

public class **ConfigLocation**

Defines a MOTECH configuration location. If the given location starts with a leading file separator character, the location is treated as a file system directory. Otherwise, it is treated as a classpath location.

Constructors

ConfigLocation

public **ConfigLocation**(String *configLocation*)

Methods

equals

public boolean **equals**(Object *o*)

getExistingConfigFiles

public List<File> **getExistingConfigFiles**()

getFile

public File **getFile**(String *fileName*, FileAccessType *accessType*)

This method Returns the `java.io.File` object for the given file name relative to the config location. It also checks for the requested file accessibility. If the requested access type check is `ConfigLocation.FileAccessType.READABLE`, the file's existence and readability will be checked. Similarly, if the requested access type check is `ConfigLocation.FileAccessType.WRITABLE`, then the write accessibility to the file will be checked. If the file does not exists, write accessibility of its ancestors will be checked.

Parameters

- **fileName** – Name of the file to be added to the config location.
- **accessType** – One of `ConfigLocation.FileAccessType.READABLE` or `ConfigLocation.FileAccessType.WRITABLE`.

Throws

- **MotechConfigurationException** – if the file is not readable or writable depending on the given access type.

Returns File relative to the config location.

getLocation

public String **getLocation**()

getUrlResource

UrlResource **getUrlResource**()

hasPlatformConfigurationFile

public boolean **hasPlatformConfigurationFile**()

hashCode

public int **hashCode**()

toResource

public Resource **toResource**()
Resource corresponding to the config location.

Returns resource

toString

public String **toString**()

12.17.4 ConfigLocation.FileAccessType

public static enum **FileAccessType**
Defines the access check required.

Enum Constants

READABLE

public static final ConfigLocation.FileAccessType **READABLE**

WRITABLE

public static final ConfigLocation.FileAccessType **WRITABLE**

12.17.5 ConfigSource

public final class **ConfigSource**
Represents the source from which MOTECH configuration should be read.

Fields

FILE

public static final ConfigSource **FILE**

UI

public static final ConfigSource **UI**

Methods

getName

public String **getName**()

isFile

public boolean **isFile**()

Checks whether this configuration source is FILE or not.

Returns true if this configuration source is file, false otherwise

isValid

public static boolean **isValid**(String *name*)

Checks whether given name is a name of supported configuration source.

Parameters

- **name** – the name to be checked

Returns true if name is valid, false otherwise

toString

public String **toString**()

valueOf

public static ConfigSource **valueOf**(String *name*)

Creates proper object of `ConfigSource` class for given name. The correct values are "UI" and "FILE". If name isn't one of above MotechConfigurationException will be thrown.

Parameters

- **name** – the name of the configuration source, null and blank String treated as "UI"

Throws

- **org.motechproject.config.core.MotechConfigurationException** – when name is neither "FILE" nor "UI"

Returns proper instance of `ConfigSource`

12.17.6 DBConfig

public class **`DBConfig`** extends AbstractDBConfig

DBConfig encapsulates the database configuration, composed of as db url, username and password.

Constructors

DBConfig

public **`DBConfig`** (String *url*, String *username*, String *password*)

Constructor.

Parameters

- **url** – the URL to the database
- **username** – the username for the database
- **password** – the password for the database

Throws

- **org.motechproject.config.core.MotechConfigurationException** – if given url is invalid.

12.17.7 SQLDBConfig

public class **`SQLDBConfig`** extends AbstractDBConfig

This class encapsulates the SQL database configuration, composed of as db url, username and password.

Constructors

SQLDBConfig

public **`SQLDBConfig`** (String *url*, String *driver*, String *username*, String *password*)

Constructor.

Parameters

- **url** – the URL to the database
- **driver** – the driver class name for the database
- **username** – the username for the database
- **password** – the password for the database

Throws

- **org.motechproject.config.core.MotechConfigurationException** – if given url is invalid.

12.18 org.motechproject.config.core.filestore

12.18.1 ConfigLocationFileStore

public class **ConfigLocationFileStore**

Used to read default platform config location(s) from `config-location.properties` and also to save in the file in the default location.

`config-location.properties` file will be loaded according to the behaviour of `org.apache.commons.configuration.PropertiesConfiguration` as specified here.

Fields

CONFIG_LOCATION_PROPERTY_KEY

public static final String **CONFIG_LOCATION_PROPERTY_KEY**

Constructors

ConfigLocationFileStore

public **ConfigLocationFileStore** (PropertiesConfiguration *propertiesConfiguration*)

Methods

add

public void **add** (String *location*)

Adds the given location to the store.

Parameters

- **location** – the location to be stored

getAll

public Iterable<ConfigLocation> **getAll** ()

Returns all the configuration locations stored by this object.

Returns the list of configuration locations

12.18.2 ConfigPropertiesUtils

public final class **ConfigPropertiesUtils**

A utility class for loading properties.

Methods

getDefaultPropertiesFile

public static File **getDefaultPropertiesFile** (ConfigLocation.FileAccessType *accessType*, Iterable<ConfigLocation> *configLocations*, String *fileName*)

Returns default config file location.

Parameters

- **accessType** – the access required for returned File object
- **configLocations** – the config locations specified by MOTECH in config-locations.properties
- **fileName** – the file name

getPropertiesFromFile

public static Properties **getPropertiesFromFile** (File *file*)

Loads the properties from given `File`.

Parameters

- **file** – the file with properties

Throws

- **IOException** – if I/O error occurred

Returns the loaded properties

getPropertiesFromSystemVarString

public static Properties **getPropertiesFromSystemVarString** (String *string*)

Loads the properties from given `String`. The format of this `String` should be "key1=value1;key2=value2;key3=value3;...".

Parameters

- **string** – the string with properties

Returns the loaded properties

saveConfig

public static void **saveConfig** (File *file*, Properties *properties*)

Saves properties to the given `File`.

Parameters

- **file** – the file
- **properties** –

12.19 org.motechproject.config.core.filters

12.19.1 ConfigFileFilter

public class **ConfigFileFilter** extends FileFileFilter
FileFilter implementation to filter configuration files.

Fields

PLATFORM_CORE_CONFIG_FILTER

public static final FileFileFilter **PLATFORM_CORE_CONFIG_FILTER**

Methods

accept

public boolean **accept** (File *file*)

isFileSupported

public static boolean **isFileSupported** (File *file*)
Checks whether given file is supported.

Parameters

- **file** – the file to be checked

Returns true if file is supported, else otherwise

isPlatformCoreConfigFile

public static boolean **isPlatformCoreConfigFile** (File *file*)
Checks whether given file is platform core configuration file.

Parameters

- **file** – the file to be checked, null returns false

Returns true if file is platform core configuration, false otherwise

12.20 org.motechproject.config.core.service

12.20.1 CoreConfigurationService

public interface **CoreConfigurationService**
Loads and saves the core configuration required to start the Motech instance.

Fields

CORE_SETTINGS_CACHE_NAME

String **CORE_SETTINGS_CACHE_NAME**

Methods

addConfigLocation

void **addConfigLocation** (String *location*)

Adds the new config location to the list of existing config locations where configurations are loaded from in the file system.

Parameters

- **location** – config location to add.

Throws

- **FileSystemException** –

evictMotechCoreSettingsCache

void **evictMotechCoreSettingsCache** ()

Removes all cached MOTECH settings.

getActiveMqConfig

Properties **getActiveMqConfig** ()

Returns the ActiveMq properties.

Returns activeMq properties.

getConfigLocation

ConfigLocation **getConfigLocation** ()

Returns the config location where all the config files are present.

Returns configLocation.

loadBootstrapConfig

BootstrapConfig **loadBootstrapConfig** ()

Loads the bootstrap configuration.

Returns bootstrap configuration.

loadDatanucleusConfig

Properties **loadDatanucleusConfig**()

Loads the datanucleus configuration

Returns datanucleus configuration

saveBootstrapConfig

void **saveBootstrapConfig**(BootstrapConfig *bootstrapConfig*)

Saves the bootstrap configuration

Parameters

- **bootstrapConfig** – Bootstrap config

12.21 org.motechproject.config.domain

12.21.1 ModulePropertiesRecord

public class **ModulePropertiesRecord**

Class representing a record of a certain module properties. This class is exposed as an `org.motechproject.mds.annotations.Entity` through Motech Data Services.

See also: `org.motechproject.mds.annotations`

Fields

PROPERTIES_FILE_EXTENSION

public static final String **PROPERTIES_FILE_EXTENSION**

Constructors

ModulePropertiesRecord

public **ModulePropertiesRecord**()

Default constructor.

ModulePropertiesRecord

public **ModulePropertiesRecord**(Map<String, Object> *properties*, String *bundle*, String *version*, String *filename*, boolean *raw*)

Constructor.

Parameters

- **properties** – the module properties
- **bundle** – the modules bundle symbolic name
- **version** – the version of the module

- **filename** – the name of the file containing module properties
- **raw** – the flag defining whether the properties are raw or not

ModulePropertiesRecord

public **ModulePropertiesRecord**(Properties *props*, String *bundle*, String *version*, String *filename*, boolean *raw*)

Constructor.

Parameters

- **props** – the module properties
- **bundle** – the modules bundle symbolic name
- **version** – the version of the module
- **filename** – the name of the file containing modules properties
- **raw** – the flag defining whether the properties are raw or not

Methods

buildFrom

public static ModulePropertiesRecord **buildFrom**(File *file*)

Builds an instance of `ModulePropertiesRecord` from given file. Content of the file must match format specified in `Properties.load(Reader)`. Properties files are treated as raw configuration files.

Parameters

- **file** – the source file, null returns null

Returns the instance of `ModulePropertiesRecord`, null if error occurred

equals

public boolean **equals**(Object *obj*)

getBundle

public String **getBundle**()

getFilename

public String **getFilename**()

getProperties

public Map<String, Object> **getProperties**()

getVersion

public String **getVersion**()

hashCode

public int **hashCode**()

isRaw

public boolean **isRaw**()

sameAs

public boolean **sameAs**(Object *dataObject*)

Checks whether given object is the same as this object.

Parameters

- **dataObject** – the object to be compared

Returns true if objects are the same, false otherwise

setBundle

public void **setBundle**(String *bundle*)

setFilename

public void **setFilename**(String *filename*)

setProperties

public void **setProperties**(Map<String, Object> *properties*)

setRaw

public void **setRaw**(boolean *raw*)

setVersion

public void **setVersion**(String *version*)

toString

public String **toString**()

12.22 org.motechproject.config.monitor

12.22.1 ConfigFileMonitor

public class **ConfigFileMonitor** implements FileListener

Class used for monitoring changes in configuration files and sending appropriate events.

Methods

fileChanged

public void **fileChanged**(FileChangeEvent *fileChangeEvent*)

fileCreated

public void **fileCreated**(FileChangeEvent *fileChangeEvent*)

fileDeleted

public void **fileDeleted**(FileChangeEvent *fileChangeEvent*)

init

public void **init**()

Initializes the configuration file monitor. This method will be automatically called after creation and dependency injection. It is done to make sure that injected dependencies are set and ready to use.

setFileMonitor

public void **setFileMonitor**(DefaultFileMonitor *fileMonitor*)

stop

public void **stop**()

Stops the file monitor.

updateFileMonitor

public void **updateFileMonitor**()

Updates the file monitor.

12.23 org.motechproject.config.service

12.23.1 BundlePropertiesService

public interface **BundlePropertiesService** extends MotechDataService<ModulePropertiesRecord>

This service provides data access for `org.motechproject.config.domain.ModulePropertiesRecord`. The implementation is generated by Motech Data Services and published as an OSGi service.

Methods

findByBundle

List<ModulePropertiesRecord> **findByBundle** (String *bundle*)

Returns list of `ModulePropertiesRecord`s matching given bundle symbolic name.

Parameters

- **bundle** – the bundle symbolic name

Returns list of `ModulePropertiesRecord`s

findByBundleAndFileName

List<ModulePropertiesRecord> **findByBundleAndFileName** (String *bundle*, String *filename*)

Returns a list of `ModulePropertiesRecord`s matching given bundle symbolic name and file name.

Parameters

- **bundle** – the bundle symbolic name
- **filename** – the name of the file

Returns list of `ModulePropertiesRecord`s

12.23.2 ConfigurationService

public interface **ConfigurationService**

Central configuration service that monitors and manages configurations.

Fields

SETTINGS_CACHE_NAME

String **SETTINGS_CACHE_NAME**

Methods

addOrUpdate

void **addOrUpdate** (File *file*)

Saves both property and raw configurations in FILE mode only. Files are classified as either raw config or properties based on the extension of the file.

Parameters

- **file** – File to read configuration from.

addOrUpdateBundleRecord

void **addOrUpdateBundleRecord** (ModulePropertiesRecord *record*)

A convenient method for adding or updating the properties, which determines on its own whether the record should be added or updated

Parameters

- **record** – a record to store

addOrUpdateBundleRecords

void **addOrUpdateBundleRecords** (List<ModulePropertiesRecord> *records*)

Bulk add or update method for the Bundle Properties records. Iterates through the passed records and either adds them, if they are not present, or updates otherwise.

Parameters

- **records** – a list of properties records

addOrUpdateProperties

void **addOrUpdateProperties** (String *bundle*, String *version*, String *filename*, Properties *newProperties*, Properties *defaultProperties*)

Depending on the config source, it will either store properties in the DB or file. Only properties that are different from the default ones are stored. If the properties database record or file doesn't exist yet for the given bundle, it will be created.

Parameters

- **bundle** – Symbolic name of updated bundle
- **version** – Version of updated bundle
- **filename** – Resource filename
- **newProperties** – New properties to store
- **defaultProperties** – Default properties of the bundle

Throws

- **IOException** – if bundle properties cannot be retrieved from file

createZipWithConfigFiles

FileInputStream **createZipWithConfigFiles** (String *propertyFile*, String *fileName*)

Uses current configuration and default one to find changed properties and then connects them with annotations. Moreover creates file with non default configurations and packs is into the zip file.

Parameters

- **propertyFile** – name of exported file

Throws

- **IOException** –

Returns FileInputStream that contains zip file

deleteByBundle

void **deleteByBundle**(String *bundle*)

Deletes the db records corresponding to the bundle with given bundle symbolic name.

deleteByBundleAndFileName

void **deleteByBundleAndFileName**(String *bundle*, String *filename*)

Deletes the db record corresponding to the bundle and filename.

evictMotechSettingsCache

void **evictMotechSettingsCache**()

Removes all cached MOTECH settings.

getAllBundleProperties

Map<String, Properties> **getAllBundleProperties**(String *bundle*, Map<String, Properties> *defaultProperties*)

Retrieves all the bundle properties and returns them as Map, where key is the filename.

Parameters

- **bundle** – The bundle we wish to retrieve properties for
- **defaultProperties** – Default properties of the bundle

Throws

- **IOException** – if any of the bundle properties file cannot be read

Returns Properties mapped by filename

getBundleProperties

Properties **getBundleProperties**(String *bundle*, String *filename*, Properties *defaultProperties*)

Retrieves merged properties, given default set. Depending on the ConfigSource, it will either merge default properties with the properties from DB or get properties from file.

Parameters

- **bundle** – The bundle we wish to retrieve properties for
- **filename** – Resource filename
- **defaultProperties** – Default properties of the bundle

Throws

- **IOException** – if bundle properties cannot be read from file

Returns Merged properties of the certain bundle

getConfigSource

ConfigSource **getConfigSource**()

This method allows to check whether MOTECH is currently running in the FILE or UI mode

Returns Current Config Source

getPlatformSettings

MotechSettings **getPlatformSettings**()

getRawConfig

InputStream **getRawConfig**(String *bundle*, String *filename*, Resource *resource*)

Allows to retrieve raw JSON data either from the database or file, depending on the specified ConfigSource mode.

Parameters

- **bundle** – Bundle we wish to retrieve raw data for
- **filename** – Resource filename
- **resource** – Resource file containing default rawConfig, in case no other has been found

Throws

- **IOException** –

Returns Raw JSON data as InputStream

listRawConfigNames

List<String> **listRawConfigNames**(String *bundle*)

Depending on the selected ConfigSource mode, this method looks for all registered raw data properties within the specified bundle.

Parameters

- **bundle** – Bundle we wish to perform look for

Returns List of filenames that register raw config for specified bundle

loadBootstrapConfig

BootstrapConfig **loadBootstrapConfig**()

Loads bootstrap config that is used to start up the Motech server.

The bootstrap configuration is loaded in the following order:

1. Load the configuration from `bootstrap.properties` from the config directory specified by the environment variable `MOTECH_CONFIG_DIR`. `bootstrap.properties` contains the following properties:

```
sql.url (Mandatory)
sql.driver (Mandatory)
sql.username (If required)
sql.password (If required)
tenant.id (Optional. Defaults to 'DEFAULT')
config.source (Optional. Defaults to 'UI')
```

An example `bootstrap.properties` is given below:

```
sql.url=jdbc:mysql://localhost:3306/
sql.driver=com.mysql.jdbc.Driver
sql.username=motech
sql.password=motech
tenant.id=MotherChildCare
config.source=FILE
```

2. If `MOTECH_CONFIG_DIR` environment variable is **not** set, load the specific configuration values from the following environment variables:

```
MOTECH_SQL_URL (Mandatory)
MOTECH_SQL_DRIVER (Mandatory)
MOTECH_SQL_USERNAME (If required)
MOTECH_SQL_PASSWORD (If required)
MOTECH_TENANT_ID (Optional. Defaults to 'DEFAULT')
MOTECH_CONFIG_SOURCE (Optional. Defaults to 'UI')
```

3. If `MOTECH_DB_URL` environment is not set, load the configuration from `bootstrap.properties` from the default MOTECH config directory specified in the file `config-locations.properties`.

Throws

- **org.motechproject.config.core.MotechConfigurationException** – if bootstrap configuration cannot be loaded.

Returns Bootstrap configuration

loadConfig

SettingsRecord **`loadConfig`**()

Loads current MOTECH configuration

Returns current MOTECH settings

loadDefaultConfig

SettingsRecord **`loadDefaultConfig`**()

Loads the default config for MOTECH from the resource file.

Returns default settings

processExistingConfigs

void **`processExistingConfigs`**(List<File> *files*)

Adds, updates, or deletes configurations in FILE mode only. Files are classified as either raw config or properties based on the extension of the file.s

Parameters

- **files** – Files to read configuration from.

rawConfigExists

boolean **rawConfigExists** (String *bundle*, String *filename*)

Allows to check if raw data has been registered for specified bundle

Parameters

- **bundle** – Bundle symbolic name
- **filename** – Resource filename

Returns True if raw data exists for given parameters, false otherwise

registersProperties

boolean **registersProperties** (String *bundle*, String *filename*)

Checks if given bundle registers certain property file

Parameters

- **bundle** – Bundle we wish to perform check for
- **filename** – Resource filename

Returns True if properties exist, false otherwise

removeAllBundleProperties

void **removeAllBundleProperties** (String *bundle*)

Removes properties for given bundle.

Parameters

- **bundle** – The bundle we wish to remove properties for

removeBundleRecords

void **removeBundleRecords** (List<ModulePropertiesRecord> *records*)

Removes given bundle properties records

Parameters

- **records** – a list of properties records to remove

requiresConfigurationFiles

boolean **requiresConfigurationFiles** ()

Checks whether set MOTECH configuration requires the configuraton files to be present

Returns true if files are required, false otherwise

retrieveRegisteredBundleNames

List<String> **retrieveRegisteredBundleNames**()

Depending on the selected ConfigSource mode, this method looks for registered bundle properties and returns a list of files it has found

Returns List of files with registered properties

save

void **save**(BootstrapConfig *bootstrapConfig*)

Saves the given `BootstrapConfig` in the `bootstrap.properties` file located in default MOTECH config location. The default motech config location is specified in the file `config-locations.properties`.

Parameters

- **bootstrapConfig** – Bootstrap configuration.

Throws

- **org.motechproject.config.core.MotechConfigurationException** – if bootstrap configuration cannot be saved.

savePlatformSettings

void **savePlatformSettings**(Properties *settings*)

Saves given platform settings to the settings service. Available platform settings are language, login mode, provider name, provider URL, server URL, status message timeout, and upload size.

Parameters

- **settings** – the settings to be saved

savePlatformSettings

void **savePlatformSettings**(MotechSettings *settings*)

Saves given MOTECH settings to the settings service.

Parameters

- **settings** – the settings to be saved

saveRawConfig

void **saveRawConfig**(String *bundle*, String *version*, String *filename*, InputStream *rawData*)

Allows persisting of raw json properties either in the database or file, depending on the selected ConfigSource mode.

Parameters

- **bundle** – Bundle we wish to save properties for
- **filename** – Resource filename
- **rawData** – Raw JSON data to persist

Throws

- **IOException** –

setPlatformSetting

void **setPlatformSetting** (String *key*, String *value*)

Sets given value for the platform setting with given key.

Parameters

- **key** – the setting name
- **value** – the value to be set

updateConfigLocation

void **updateConfigLocation** (String *newConfigLocation*)

Adds a new config location and restarts the monitor.

Parameters

- **newConfigLocation** – New config location

updatePropertiesAfterReinstallation

void **updatePropertiesAfterReinstallation** (String *bundle*, String *version*, String *filename*, Properties *defaultProperties*, Properties *newProperties*)

Works similar to `addOrUpdateProperties` but instead of just adding / updating properties checks database for any deprecated properties and removes to ensure that only current ones are available

Parameters

- **bundle** – Symbolic name of updated bundle
- **version** – Version of updated bundle
- **filename** – Resource filename
- **newProperties** – New properties to store
- **defaultProperties** – Default properties of the bundle

Throws

- **IOException** – if bundle properties cannot be retrieved from file

12.24 org.motechproject.email.builder

12.24.1 EmailRecordSearchCriteria

public class **EmailRecordSearchCriteria**

The `EmailRecordSearchCriteria` class represents search criteria that may be used for searching `org.motechproject.email.domain.EmailRecord` entities in Motech Data Services. A consumer of this class may create search criteria to query on multiple fields by calling several of the with* methods. To perform the search, use `org.motechproject.email.service.EmailAuditService.findEmailRecords(EmailRecordSearchCr`

Methods

getDeliveryStatuses

public Set<DeliveryStatus> **getDeliveryStatuses**()

Gets the delivery statuses criterion.

Returns the delivery statuses criterion for this search criteria

getDeliveryTimeRange

public Range<DateTime> **getDeliveryTimeRange**()

Gets the delivery time range criterion.

Returns the deliveryTimeRange criterion for this search criteria

getFromAddress

public String **getFromAddress**()

Gets the from address criterion.

Returns the fromAddress criterion for this search criteria

getMessage

public String **getMessage**()

Gets the message body criterion.

Returns the message criterion for this search criteria

getQueryParams

public QueryParams **getQueryParams**()

Gets the query paramaters that are used for controlling order and size of the query results for this search criteria.

Returns the query params for this search criteria

getSubject

public String **getSubject**()

Gets the subject criterion.

Returns the subject criterion for this search criteria

getToAddress

public String **getToAddress**()

Gets the to address criterion.

Returns the toAddress criterion for this search criteria

withDeliveryStatuses

public EmailRecordSearchCriteria **withDeliveryStatuses** (Set<DeliveryStatus> *deliveryStatuses*)

Sets the delivery statuses criterion to the set specified

Parameters

- **deliveryStatuses** – the delivery statuses on which to search

Returns this `EmailRecordSearchCriteria` with its deliveryStatuses criterion set to the provided statuses

withDeliveryStatuses

public EmailRecordSearchCriteria **withDeliveryStatuses** (DeliveryStatus... *deliveryStatuses*)

Sets the delivery statuses criterion to the set specified

Parameters

- **deliveryStatuses** – the delivery statuses on which to search

Returns this `EmailRecordSearchCriteria` with its deliveryStatuses criterion set to the provided statuses

withFromAddress

public EmailRecordSearchCriteria **withFromAddress** (String *fromAddress*)

Sets the fromAddress criterion to the address specified

Parameters

- **fromAddress** – the sender email address on which to search

Returns this `EmailRecordSearchCriteria` with its fromAddress criterion set to the provided address

withMessage

public EmailRecordSearchCriteria **withMessage** (String *message*)

Sets the message criterion to the message specified

Parameters

- **message** – the email message body on which to search

Returns this `EmailRecordSearchCriteria` with its message criterion set to the provided message

withMessageTime

public EmailRecordSearchCriteria **withMessageTime** (DateTime *deliveryTimeRange*)

Sets the send time criterion to the time specified. Use this method to search on a specific date/time; if a range is needed, use `withMessageTimeRange` instead.

Parameters

- **deliveryTimeRange** – the specific time on which to search

Returns this `EmailRecordSearchCriteria` with its deliveryTimeRange criterion set to the specified date/time

withMessageTimeRange

public EmailRecordSearchCriteria **`withMessageTimeRange`** (Range<DateTime> *deliveryTimeRange*)

Sets the sent time criterion to the range specified. Use this method to search on a time range; if searching on a specific date/time is needed, use `withMessageTime` instead.

Parameters

- **deliveryTimeRange** – the date/time range on which to search

Returns this `EmailRecordSearchCriteria` with its deliveryTimeRange criterion set to the specified date/time range

withQueryParams

public EmailRecordSearchCriteria **`withQueryParams`** (QueryParams *queryParams*)

Sets the queryParams of the search criteria to the parameters specified. Use this method when it is necessary to specify order and size of query results. This is used mainly for paging/ordering queries from the UI.

Parameters

- **queryParams** – the query parameters to include with the search criteria

Returns this `EmailRecordSearchCriteria` with its queryParams set to the provided parameters

withSubject

public EmailRecordSearchCriteria **`withSubject`** (String *subject*)

Sets the subject criterion to the subject specified

Parameters

- **subject** – the subject on which to search

Returns this `EmailRecordSearchCriteria` with its subject criterion set to the provided subject

withToAddress

public EmailRecordSearchCriteria **`withToAddress`** (String *toAddress*)

Sets the toAddress criterion to the address specified

Parameters

- **toAddress** – the recipient email address on which to search

Returns this `EmailRecordSearchCriteria` with its toAddress criterion set to the provided address

12.25 org.motechproject.email.contract

12.25.1 Mail

public class **Mail**

The `Mail` class represents an email message.

Constructors

Mail

public **Mail** (String *fromAddress*, String *toAddress*, String *subject*, String *message*)

Creates a new instance of `Mail`, with all fields set to the values specified in the parameters.

Parameters

- **fromAddress** – the email address of the sender
- **toAddress** – the email address of the recipient
- **subject** – the subject of the email
- **message** – the body of the email

Methods

equals

public boolean **equals** (Object *obj*)

Indicates whether some other object is "equal to" this one. Returns true if this Mail and the object to compare have reference equality or their field values are all equal.

Parameters

- **obj** – The reference object with which to compare

Returns true if this object is the same as the obj argument; false otherwise.

getFromAddress

public String **getFromAddress** ()

Gets the email address of the sender.

Returns the sender of the message

getMessage

public String **getMessage** ()

Gets the message body.

Returns the body of the message

getSubject

public String **getSubject**()
Gets the message subject.

Returns the subject of the message

getText

public String **getText**()

getToAddress

public String **getToAddress**()
Gets the email address of the recipient.

Returns the recipient of the message

hashCode

public int **hashCode**()
Returns a hash code value for this `Mail` object.

Returns a hash code value for this `Mail` object

toString

public String **toString**()
Returns a string representation of this `Mail` object.

Returns a string representation of this `Mail` object

12.26 org.motechproject.email.domain

12.26.1 DeliveryStatus

public enum **DeliveryStatus**
The `DeliveryStatus` Enum contains the possible delivery states for an email message.

Enum Constants

ERROR

public static final DeliveryStatus **ERROR**
There was an error sending the message.

RECEIVED

public static final DeliveryStatus **RECEIVED**
The message was received.

SENT

public static final DeliveryStatus **SENT**
The message was sent.

12.26.2 EmailRecord

public class **EmailRecord**
The `EmailRecord` class represents a record of a sent Email. This class is exposed as an `org.motechproject.mds.annotations.Entity` through Motech Data Services.

See also: `org.motechproject.mds.annotations`

Constructors

EmailRecord

public **EmailRecord**()
Creates a new instance of `EmailRecord`, with all fields set to null.

EmailRecord

public **EmailRecord**(String *fromAddress*, String *toAddress*, String *subject*, String *message*, DateTime *deliveryTime*, DeliveryStatus *deliveryStatus*)
Creates a new instance of `EmailRecord`, with all fields set to the values specified in the parameters.

Parameters

- **fromAddress** – the email address of the sender
- **toAddress** – the email address of the recipient
- **subject** – the subject of the email
- **message** – the body of the email
- **deliveryTime** – the date and time that the email was sent
- **deliveryStatus** – the delivery status of the email

Methods

equals

public boolean **equals**(Object *obj*)
Indicates whether some other object is "equal to" this one. Returns true if this EmailRecord and the object to compare have reference equality or their field values are all equal.

Parameters

- **obj** – The reference object with which to compare.

Returns true if this object is the same as the obj argument; false otherwise.

getDeliveryStatus

public DeliveryStatus **getDeliveryStatus**()
Gets the delivery status.

Returns the delivery status of the message

getDeliveryTime

public DateTime **getDeliveryTime**()
Gets the delivery time.

Returns the time that the email was sent

getFromAddress

public String **getFromAddress**()
Gets the email address of the sender.

Returns the sender of the message

getId

public Long **getId**()

getMessage

public String **getMessage**()
Gets the message body.

Returns the body of the message

getSubject

public String **getSubject**()
Gets the message subject.

Returns the subject of the message

getToAddress

public String **getToAddress**()
Gets the email address of the recipient.

Returns the recipient of the message

hashCode

public int **hashCode**()

Returns a hash code value for this `EmailRecord` object.

Returns a hash code value for this `EmailRecord` object

setFromAddress

public void **setFromAddress**(String *fromAddress*)

Sets the email address of the sender.

Parameters

- **fromAddress** – the sender of the message

setId

public void **setId**(Long *id*)

setMessage

public void **setMessage**(String *message*)

Sets the message body.

Parameters

- **message** – the body of the message

setSubject

public void **setSubject**(String *subject*)

Sets the message subject.

Parameters

- **subject** – the subject of the message

setToAddress

public void **setToAddress**(String *toAddress*)

Sets the email address of the recipient.

Parameters

- **toAddress** – the recipient of the message

toString

public String **toString**()

Returns a string representation of this `EmailRecord` object.

Returns a string representation of this `EmailRecord` object

12.26.3 EmailRecordComparator

public class **EmailRecordComparator** implements Comparator<EmailRecord>

The `EmailRecordComparator` class is an implementation of the Comparator interface, that allows callers to compare `org.motechproject.email.domain.EmailRecord` objects by a single field.

Constructors

EmailRecordComparator

public **EmailRecordComparator** (Boolean *ascending*, String *compareField*)

Creates a new `EmailRecordComparator` that supports comparison based on the specified field.

Parameters

- **ascending** – boolean indicating whether comparisons should be ascending or descending
- **compareField** – the field for which comparisons should be performed

Methods

compare

public int **compare** (EmailRecord *o1*, EmailRecord *o2*)

Compares its two arguments for order. If ascending is `true`, returns a negative integer, zero, or a positive integer as the first argument is less than, equal to, or greater than the second. If ascending is `false`, returns a positive integer, zero, or negative integer as the first argument is less than, equal to, or greater than the second.

Parameters

- **o1** – the first `EmailRecord` to be compared
- **o2** – the second `EmailRecord` to be compared

Returns a positive integer, zero, or negative integer indicating the result of comparing the objects

12.26.4 EmailRecords

public class **EmailRecords**<T>

The `EmailRecords` class wraps the `EmailRecord` list and stores the current item count.

Constructors

EmailRecords

public **EmailRecords** ()

Creates a new instance of `EmailRecords`, which contains no records.

EmailRecords

public **EmailRecords** (Integer *totalRecords*, Integer *page*, Integer *totalPages*, List<T> *allRecords*)

Creates a new instance of `EmailRecords`, with all fields set to the values specified in the parameters. The `page` and `totalPages` parameters are for the purposes of paginating the list of records in the UI.

Parameters

- **totalRecords** – the total number of records
- **page** – the current page
- **totalPages** – the total number of pages
- **allRecords** – the list of records

Methods

getPage

public Integer **getPage** ()

Gets the current page.

Returns the current page

getRecords

public Integer **getRecords** ()

Gets the total number of records.

Returns the total number of records

getRows

public List<T> **getRows** ()

Gets the list of records.

Returns the list of records

getTotal

public Integer **getTotal** ()

Gets the total number of pages.

Returns the total number of pages

setRows

public void **setRows** (List<T> *rows*)

Sets the list of records.

Parameters

- **rows** – the list of records

setTotal

public void **setTotal** (Integer *total*)

Sets the total number of records.

Parameters

- **total** – the total number of records

toString

public String **toString** ()

Returns a string representation of this `EmailRecords` object.

Returns a string representation of this `EmailRecords` object

12.27 org.motechproject.email.service

12.27.1 EmailAuditService

public interface **EmailAuditService**

The `EmailAuditService` interface provides methods for logging email activity, as well as searching and deleting the email logs.

Methods

countEmailRecords

long **countEmailRecords** (EmailRecordSearchCriteria *criteria*)

Returns the count of `EmailRecord` entries matching the specified search criteria.

Returns the count of email records matching the provided criteria

delete

void **delete** (EmailRecord *emailRecord*)

Deletes the specified `EmailRecord` entry from the email log.

findAllEmailRecords

List<EmailRecord> **findAllEmailRecords** ()

Finds and returns all `EmailRecord` entries in the email log.

Returns all email records in the email log

findById

EmailRecord **findById** (long *id*)

Finds an `EmailRecord` in the log by ID.

Parameters

- **id** – the identifier of the record to find

Returns the email record that matches the provided identifier, or null if no matching record exists

findEmailRecords

List<EmailRecord> **findEmailRecords** (EmailRecordSearchCriteria *criteria*)

Finds and returns all `EmailRecord` entries matching the specified search criteria.

Returns all email records matching the provided criteria

12.27.2 EmailRecordService

public interface **EmailRecordService** extends MotechDataService<EmailRecord>

This service provides data access for `org.motechproject.email.domain.EmailRecord`. The implementation is generated by Motech Data Services and published as an OSGi service.

Methods

countFind

long **countFind** (String *fromAddress*, String *toAddress*, String *subject*, String *message*, Range<DateTime> *deliveryTimeRange*, Set<DeliveryStatus> *deliveryStatuses*)

Returns the count of all `EmailRecord` entries matching the specified search parameters.

Parameters

- **fromAddress** – the sender address on which to search
- **toAddress** – the recipient address on which to search
- **subject** – the subject on which to search
- **message** – the message body on which to search
- **deliveryTimeRange** – the delivery time range on which to search
- **deliveryStatuses** – the delivery statuses on which to search

Returns the count of `EmailRecord` entries that match the specified criteria

find

List<EmailRecord> **find** (String *fromAddress*, String *toAddress*, String *subject*, String *message*, Range<DateTime> *deliveryTimeRange*, Set<DeliveryStatus> *deliveryStatuses*, QueryParams *queryParams*)

Finds and returns all `EmailRecord` entries matching the specified search parameters. This method is exposed as a `org.motechproject.mds.annotations.Lookup` through Motech Data Services.

Parameters

- **fromAddress** – the sender address on which to search
- **toAddress** – the recipient address on which to search
- **subject** – the subject on which to search
- **message** – the message body on which to search
- **deliveryTimeRange** – the delivery time range on which to search
- **deliveryStatuses** – the delivery statuses on which to search
- **queryParams** – the query parameters to include with the search criteria

Returns the list of `EmailRecord` entries that match the specified criteria

See also: `org.motechproject.mds.annotations`

findByRecipientAddress

List<EmailRecord> **`findByRecipientAddress`** (String *recipientAddress*)

Finds and returns all `EmailRecord` entries for the specified recipient address. This method is exposed as a `org.motechproject.mds.annotations.Lookup` through Motech Data Services.

Parameters

- **recipientAddress** – the recipient address on which to search

Returns the list of `EmailRecord` entries that match the specified address

See also: `org.motechproject.mds.annotations`

12.27.3 EmailSenderService

public interface **`EmailSenderService`**

The `EmailSenderService` interface provides a method for sending email.

Methods

send

void **`send`** (Mail *message*)

Attempts to send the supplied email message. Adds an `org.motechproject.email.domain.EmailRecord` entry to the log with the details of the activity.

Parameters

- **message** – the message to send

12.28 org.motechproject.event

12.28.1 MotechEvent

public class **`MotechEvent`** implements Serializable

Motech Scheduled Event data carrier class. It contains a subject, to which listeners can subscribe and a payload

in the form of a map of parameters. Instance of this class with event specific data will be sent by Motech Scheduler when a scheduled event is fired.

This class is immutable

Fields

EVENT_TYPE_KEY_NAME

public static final String **EVENT_TYPE_KEY_NAME**

PARAM_DISCARDED_MOTECH_EVENT

public static final String **PARAM_DISCARDED_MOTECH_EVENT**

PARAM_INVALID_MOTECH_EVENT

public static final String **PARAM_INVALID_MOTECH_EVENT**

PARAM_REDELIVERY_COUNT

public static final String **PARAM_REDELIVERY_COUNT**

Constructors

MotechEvent

public **MotechEvent** ()

MotechEvent

public **MotechEvent** (String *subject*)

Constructs a MotechEvent with the given subject.

Parameters

- **subject** – the subject of the event

Throws

- **IllegalArgumentException** – if the subject is null or contains `'*', '..'`

MotechEvent

public **MotechEvent** (String *subject*, Map<String, Object> *parameters*)

Constructs a MotechEvent with the given subject and parameters.

Parameters

- **subject** – the subject of the event

- **parameters** – the map of additional parameters

Throws

- **IllegalArgumentException** – if the subject is null or contains `'*', '..'`

Methods

equals

public boolean **`equals`** (Object *o*)

getId

public UUID **`getId`** ()

Returns the universally unique identifier

Returns the id

getMessageRedeliveryCount

public int **`getMessageRedeliveryCount`** ()

Returns the `motechEventRedeliveryCount` from the parameters. This is incremented by the event system if the delivery fails, so it is equal to the number of failed deliveries. Any exception from the handler is counted as failure in this context. It cannot be larger than `org.motechproject.event.messaging.MotechEventConfig.messageMaxRedeliveryCount`

Returns the number of message redeliveries

getParameters

public Map<String, Object> **`getParameters`** ()

Returns the parameters, if null returns empty `HashMap`.

Returns the map of the parameters

getSubject

public String **`getSubject`** ()

Returns the name of the subject.

Returns the subject

hashCode

public int **`hashCode`** ()

incrementMessageRedeliveryCount

public void **incrementMessageRedeliveryCount**()

Increments the `motechEventRedeliveryCount` from the parameters. It is invoked by the event system if the delivery fails. If it is null, sets the value to 0.

setId

public void **setId**(UUID *id*)

Sets the id.

Parameters

- **id** – the universally unique identifier

toString

public String **toString**()

12.29 org.motechproject.event.listener

12.29.1 EventListener

public interface **EventListener**

Provides the base model interface for event listeners. In case of listeners using annotations, proxies implementing this interface are created, so there is no actual need to implement this interface when creating listeners.

Methods

getIdentifier

String **getIdentifier**()

Returns the unique identifier/key for the given listener object. The identifier is used when messages are destined for this specific listener type.

Returns the unique listener identifier/key

handle

void **handle**(MotechEvent *event*)

Handles the particular event that has been received

Parameters

- **event** – the event that occurred.

12.29.2 EventListenerRegistryService

public interface **EventListenerRegistryService**

Gives access to the registry of listeners for Motech events. This interface is necessary for OSGi service publication. One can register themselves to listen for a specific set of event's subject.

Methods

clearListenersForBean

void **clearListenersForBean** (String *beanName*)

Removes all listeners registered in the bean. This is necessary when bundles are stopped in some fashion so that the listener does not persist.

Parameters

- **beanName** – the name of the bean

getListenerCount

int **getListenerCount** (String *subject*)

Returns the number of event listeners for the event with the subject.

Parameters

- **subject** – the subject of the event

Returns the number of matching listeners

getListeners

Set<EventListener> **getListeners** (String *subject*)

Returns all the event listeners registered for the event with the given subject. If there are no listeners, an empty list is returned.

Parameters

- **subject** – the subject of the event

Returns the matching event listeners

hasListener

boolean **hasListener** (String *subject*)

Returns `true` if the event with the subject has any listeners.

Parameters

- **subject** – the subject of the event

Returns `true` if the subject has any listeners; `false` otherwise

registerListener

void **registerListener** (EventListener *listener*, List<String> *subjects*)

Registers the event listener to be notified when events with the matching subject are received via the Server JMS Event Queue.

Parameters

- **listener** – the listener to be registered
- **subjects** – the list of subjects the listener subscribes to, wildcards are allowed

registerListener

void **registerListener** (EventListener *listener*, String *subject*)

Registers the event listener to be notified when the event's subjects are received via the Server JMS Event Queue.

Parameters

- **listener** – the listener to be registered
- **subject** – the subject the listener subscribes to, wildcards are allowed

12.29.3 EventRelay

public interface **EventRelay**

The `EventRelay` interface provides methods that allow sending `org.motechproject.event.MotechEvent` via ActiveMQ, either to the queue (ActiveMQ selects the subscriber that will handle the event) or to the topic (event is sent to every registered subscriber).

Methods

broadcastEventMessage

void **broadcastEventMessage** (MotechEvent *motechEvent*)

Publishes the event message in a topic. The message will only go to ActiveMQ if there are listeners registered for the subject (in this instance). Meaning if you have clustered Motech instances, you must ensure they both have the listeners registered. The message is then handled by all Motech instances that are subscribed to a topic, by calling `org.motechproject.event.listener.impl.ServerEventRelay.relayTopicEvent(org.motechproje` service method. This method should only be used when it is desired to broadcast an event to all Motech instances.

Parameters

- **motechEvent** – the event to be broadcast

sendEventMessage

void **sendEventMessage** (MotechEvent *motechEvent*)

Publishes the event message in a queue. The message will only go to ActiveMQ if there are listeners registered for the subject (in this instance). Meaning if you have clustered Motech instances, you must ensure they both have the listeners registered. The message is then handled by exactly one Motech instance, by calling

`org.motechproject.event.listener.impl.ServerEventRelay.relayQueueEvent(org.motechproje`
service method.

Parameters

- **motechEvent** – the event to be sent

12.30 org.motechproject.event.listener.annotations

12.30.1 MotechListener

public @interface **`MotechListener`**

The `MotechListener` annotation is used by developers to specify which method should be invoked when an event with a particular subject will be fired.

This annotation is processed by `org.motechproject.event.listener.proxy.EventAnnotationBeanPostProc`

Author yyonkov

See also: `org.motechproject.event.listener.proxy.EventAnnotationBeanPostProcessor`

12.30.2 MotechListenerAbstractProxy

public abstract class **`MotechListenerAbstractProxy`** implements EventListener

Represents a `MotechListener` proxy, providing access to the listener's name, bean, method. Constructed for listeners defined using annotations.

Author yyonkov

Constructors

MotechListenerAbstractProxy

public **`MotechListenerAbstractProxy`** (String *name*, Object *bean*, Method *method*)

Parameters

- **name** – the unique listener identifier/key
- **bean** – the bean where handler exists
- **method** – the method which will be invoked when the particular event will be fired

Methods

callHandler

public abstract void **`callHandler`** (MotechEvent *event*)

Calls handler for the concrete proxy.

Parameters

- **event** – the event which will be handled

getBean

public Object **getBean**()

Returns the bean where handler exists.

Returns the bean where handler exists

getIdentifier

public String **getIdentifier**()

getMethod

public Method **getMethod**()

Returns the handler of a event.

Returns the method which handles a event.

handle

public void **handle**(MotechEvent *event*)

12.30.3 MotechListenerEventProxy

public class **MotechListenerEventProxy** extends MotechListenerAbstractProxy

Represents the type of `MotechListener` proxy where handler is a method with the `MotechEvent` parameter.

Author yyonkov

Constructors

MotechListenerEventProxy

public **MotechListenerEventProxy**(String *name*, Object *bean*, Method *method*)

See also: `org.motechproject.event.listener.annotations.MotechListenerAbstractProxy.MotechI`

Methods

callHandler

public void **callHandler**(MotechEvent *event*)

12.30.4 MotechListenerNamedParametersProxy

public class **MotechListenerNamedParametersProxy** extends MotechListenerAbstractProxy

Represents the type of `MotechListener` proxy where handler is a method with parameters defined by the `org.motechproject.event.listener.annotations.MotechParam` annotation.

Author yyonkov

Constructors

MotechListenerNamedParametersProxy

public **MotechListenerNamedParametersProxy** (String *name*, Object *bean*, Method *method*)

See also: `org.motechproject.event.listener.annotations.MotechListenerAbstractProxy.MotechI`

Methods

callHandler

public void **callHandler** (MotechEvent *event*)

12.30.5 MotechListenerType

public enum **MotechListenerType**

The enum defining types of `MotechListener` proxies.

Author yyonkov

Enum Constants

MOTECH_EVENT

public static final MotechListenerType **MOTECH_EVENT**

NAMED_PARAMETERS

public static final MotechListenerType **NAMED_PARAMETERS**

12.30.6 MotechParam

public @interface **MotechParam**

The `MotechParam` annotation is used by developers to specify parameters in a method which handles event. The parameters are used only in the `org.motechproject.event.listener.annotations.MotechListenerNamedParametersProxy` type of listener.

This annotation is processed by `org.motechproject.event.listener.annotations.MotechListenerNamedP`

Author yyonkov

See also: `org.motechproject.event.listener.annotations.MotechListenerNamedParametersProxy`

12.31 org.motechproject.event.messaging

12.31.1 MotechCachingConnectionFactory

public class **MotechCachingConnectionFactory** extends CachingConnectionFactory

Represents an extension of the CachingConnectionFactory that adds username and password support, in case the JMS broker is secured.

Methods

doCreateConnection

protected Connection **doCreateConnection** ()

Creates a connection with the username and password if both not blank, otherwise without them.

setBrokerUrl

public void **setBrokerUrl** (String *brokerURL*)

Sets the brokerURL of the ActiveMQ only when the `TargetConnectionFactory` is instanceof ActiveMQ-ConnectionFactory.

Parameters

- **brokerURL** – the brokerURL of the ActiveMQ

setPassword

public void **setPassword** (String *password*)

Sets the password.

Parameters

- **password** – the password of ActiveMQ

setUsername

public void **setUsername** (String *username*)

Sets the username.

Parameters

- **username** – the name of an user

12.31.2 MotechEventConfig

public class **MotechEventConfig**

Accesses the `MotechEventConfig` variables.

Methods

getMessageMaxRedeliveryCount

public int **getMessageMaxRedeliveryCount** ()

Returns maximum number of times a message would be re-delivered in case of any exception.

Returns the maximum number of message redelivery

getMessageRedeliveryDelay

public long **getMessageRedeliveryDelay** ()

Returns delay (in seconds) between successive re-deliveries of messages in case of any exception. If delay=d and first exception was raised at time=t, then successive redelivery times are t+d, t+(d*2), t+(d*4), t+(d*8), t+(d*16), t+(d*32), and so on, till maximum redelivery count is reached.

Returns the message redelivery delay

12.31.3 MotechEventHeaderMapper

public class **MotechEventHeaderMapper** extends DefaultJmsHeaderMapper

Sets the `AMQ_SCHEDULED_DELAY` header of the JMS message being sent based on the `MotechEventConfig`. For the delay to work, set attribute schedulerSupport="true" in the broker element of the activemq.xml Ref: http://activemq.apache.org/delay-and-schedule-message-delivery.html

Methods

fromHeaders

public void **fromHeaders** (MessageHeaders *messageHeaders*, Message *message*)

{@inheritDoc}. Additionally sets `AMQ_SCHEDULED_DELAY` using `MotechEventConfig` variables.

12.31.4 MotechEventTransformer

public class **MotechEventTransformer**

Transforms `MotechEvent` by settings its UUID.

Methods

transform

public MotechEvent **transform** (MotechEvent *motechEvent*)

Updates the motechEvent's `UUID` with a random value if it is null, otherwise it does not change it.

Parameters

- **motechEvent** – the motechEvent to be updated

Returns the motechEvent after being updated

See also: `java.util.UUID.randomUUID()`

12.31.5 OutboundEventGateway

public interface **OutboundEventGateway**

Sends `MotechEvent` to the ActiveMQ broker, the implementation is generated by Spring Integration.

Methods

broadcastEventMessage

void **broadcastEventMessage** (MotechEvent *motechEvent*)

Broadcast the motechEvent's message as a payload to the message channel defined in the Spring Integration configuration file. The channel is connected with a message topic, meaning all Motech instances will receive this event.

Parameters

- **motechEvent** – the event to be broadcast

sendEventMessage

void **sendEventMessage** (MotechEvent *motechEvent*)

Sends the motechEvent's message as a payload to the message channel defined in the Spring Integration configuration file. The channel is connected with a message queue, meaning only one Motech instance will receive this event.

Parameters

- **motechEvent** – the event to be sent

12.32 org.motechproject.mds.annotations

12.32.1 Access

public @interface **Access**

The `Access` annotation is used to specify security options of an entity. The discovery logic for this annotation is done in `org.motechproject.mds.annotations.internal.EntityProcessor`

See also: `org.motechproject.mds.annotations.internal.EntityProcessor`

12.32.2 Cascade

public @interface **Cascade**

The `Cascade` annotation is used to set correct cascade properties for the given field that is a relationship.

See also: `org.motechproject.mds.annotations.internal.FieldProcessor`

12.32.3 CrudEvents

public @interface **CrudEvents**

The `CrudEvents` annotation is used to specify which CRUD operations should send Motech events. CrudEvents value is an array of one or more values specified in

org.motechproject.mds.event.CrudEventType enum, that is: CREATE, UPDATE, DELETE. There are also two special values - ALL, NONE. When provided, all CRUD operations are enabled/disabled for entity, regardless of presence of other values.

This annotation is processed by org.motechproject.mds.annotations.internal.CrudEventsProcessor and can be applied only to class which is also annotated with Entity. It has no effect otherwise.

See also: org.motechproject.mds.event.CrudEventType, org.motechproject.mds.annotations.inte

12.32.4 Entity

public @interface **Entity**

The Entity annotation is used to point classes, that should be mapped as Motech Dataservices Entities. The discovery logic for this annotation is done in org.motechproject.mds.annotations.internal.EntityProcessor

See also: org.motechproject.mds.annotations.internal.EntityProcessor

12.32.5 Field

public @interface **Field**

The Field annotation is used to point fields, that should be mapped as entity fields. The discovery logic for this annotation is done in org.motechproject.mds.annotations.internal.FieldProcessor.

Only fields, 'getter' or 'setter' methods can have this annotation for other methods this annotation is omitted.

See also: org.motechproject.mds.annotations.internal.FieldProcessor

12.32.6 Ignore

public @interface **Ignore**

Thanks to this annotation, developers can point class fields that should not be included in entity schema definition. To work properly, it is required that either a field or a getter is marked with this annotation. By default, all public fields of an object are included. The discovery logic for this annotation is done in the FieldProcessor and partially in MdsIgnoreAnnotationHandler.

See also: org.motechproject.mds.annotations.internal.FieldProcessor, org.motechproject.mds.jdo.MdsIgnoreAnnotationHandler

12.32.7 InSet

public @interface **InSet**

The annotated element must have value that will be in defined set.

Supported types are:

- Integer
- Double
- int, double

12.32.8 InstanceLifecycleListener

public @interface **InstanceLifecycleListener**

The `InstanceLifecycleListener` annotation is used to point methods from the services exposed by OSGi that should listen to persistence events. The InstanceLifecycleListenerType value is an array of one or more values specified in `InstanceLifecycleListenerType` enum, that is: POST_CREATE, PRE_DELETE, POST_DELETE, POST_LOAD, PRE_STORE, POST_STORE. The annotated methods must have only one parameter. If no package is specified, the parameter type is a persistable class. Otherwise it has to be of type Object.

This annotation is processed by `org.motechproject.mds.annotations.internal.InstanceLifecycleListe`

See also: `org.motechproject.mds.annotations.internal.InstanceLifecycleListenerProcessor`, `InstanceLifecycleListenerType`

12.32.9 InstanceLifecycleListenerType

public enum **InstanceLifecycleListenerType**

The `InstanceLifecycleListenerType` enum represents persistence event types.

See also: `org.motechproject.mds.annotations.InstanceLifecycleListener`

Enum Constants

POST_CREATE

public static final InstanceLifecycleListenerType **POST_CREATE**

Represents a point in time, right after an instance is made persistent.

POST_DELETE

public static final InstanceLifecycleListenerType **POST_DELETE**

Represents a point in time, right after an instance is deleted.

POST_LOAD

public static final InstanceLifecycleListenerType **POST_LOAD**

Represents a point in time, right after loading instance from datastore.

POST_STORE

public static final InstanceLifecycleListenerType **POST_STORE**

Represents a point in time, right after an instance is stored (eg. due to commit or flush)

PRE_DELETE

public static final InstanceLifecycleListenerType **PRE_DELETE**

Represents a point in time, right before an instance is deleted.

PRE_STORE

public static final InstanceLifecycleListenerType **PRE_STORE**

Represents a point in time, right before an instance is stored (eg. due to commit or flush)

12.32.10 InstanceLifecycleListeners

public @interface **InstanceLifecycleListeners**

The `InstanceLifecycleListeners` annotation is used to point entities for which there may exist instance lifecycle listeners. The only valid case for using this option is when listeners are registered programmatically using the `org.motechproject.mds.service.JdoListenerRegistryService.registerListener(org.motechpro` service method. There is no need to bother with this annotation when listeners are registered with `InstanceLifecycleListener`.

See also: `org.motechproject.mds.annotations.internal.InstanceLifecycleListenersProcessor`

12.32.11 Lookup

public @interface **Lookup**

The `Lookup` annotation is used to point methods, in classes that implements `org.motechproject.mds.service.MotechDataService`, that should be mapped as MDS lookups. The discovery logic for this annotation is done in the `org.motechproject.mds.annotations.internal.LookupProcessor`

See also: `org.motechproject.mds.annotations.internal.LookupProcessor`

12.32.12 LookupField

public @interface **LookupField**

The `LookupField` annotation allows to point fields in Lookup method, that should be mapped as lookup fields for Developer Defined Lookup. The discovery logic for this annotation is done in the `org.motechproject.mds.annotations.internal.LookupProcessor`

See also: `org.motechproject.mds.annotations.internal.LookupProcessor`

12.32.13 NonEditable

public @interface **NonEditable**

The `NonEditable` annotation is used to mark a field non-editable via UI. The discovery logic for this annotation is done in `org.motechproject.mds.annotations.internal.NonEditableProcessor`. Only fields, 'getter' or 'setter' methods can have this annotation for other methods this annotation is omitted.

See also: `org.motechproject.mds.annotations.internal.NonEditableProcessor`

12.32.14 NotInSet

public @interface **NotInSet**

The annotated element must not have value that will be in defined set.

Supported types are:

- Integer
- Double
- `int`, `double`

12.32.15 ReadAccess

public @interface **ReadAccess**

The `ReadAccess` annotation is used to specify security options for read-only access to an entity. The discovery logic for this annotation is done in `org.motechproject.mds.annotations.internal.EntityProcessor`

See also: `org.motechproject.mds.annotations.internal.EntityProcessor`

12.32.16 RestExposed

public @interface **RestExposed**

The `RestExposed` annotation is used by developers to mark lookups that should be exposed via REST. It is being processed by `org.motechproject.mds.annotations.internal.LookupProcessor`.

See also: `org.motechproject.mds.annotations.internal.LookupProcessor`

12.32.17 RestIgnore

public @interface **RestIgnore**

The `RestIgnore` annotation is used to mark a field of entity as not exposed via REST. By default all fields (including auto-generated ones) are exposed. To ignore one of the auto-generated fields, it have to be declared in child entity and marked with this annotation.

This annotation is processed by `org.motechproject.mds.annotations.internal.RestIgnoreProcessor`

See also: `org.motechproject.mds.annotations.internal.RestIgnoreProcessor`

12.32.18 RestOperation

public enum **RestOperation**

The `RestOperation` enum represents CRUD operations that can be enabled for entities.

See also: `org.motechproject.mds.annotations.RestOperations`

Enum Constants

ALL

public static final RestOperation **ALL**

CREATE

public static final RestOperation **CREATE**

DELETE

public static final RestOperation **DELETE**

READ

public static final RestOperation **READ**

UPDATE

public static final RestOperation **UPDATE**

12.32.19 RestOperations

public @interface **RestOperations**

The `RestOperations` annotation is used to specify which CRUD operations should be enabled for entity. RestOperations value is an array of one or more values specified in `RestOperation` enum, that is: CREATE, READ, UPDATE, DELETE. There is also one special value - ALL. When provided, all CRUD operations are enabled for entity, regardless of presence of other values.

This annotation is processed by `org.motechproject.mds.annotations.internal.RestOperationsProcesso` and can be applied only to class which is also annotated with `org.motechproject.mds.annotations.Entity`. It has no effect otherwise.

See also: `RestOperation`, `org.motechproject.mds.annotations.internal.RestOperationsProcessor`

12.32.20 UIDisplayable

public @interface **UIDisplayable**

The `UIDisplayable` annotation is used to mark a field as being in the default display for a listing of objects. If no field is marked with this annotation, all of them will be treated as displayable. The discovery logic for this annotation is done in `org.motechproject.mds.annotations.internal.UIDisplayableProcessor`.

Only fields, 'getter' or 'setter' methods can have this annotation for other methods this annotation is omitted.

See also: `org.motechproject.mds.annotations.internal.UIDisplayableProcessor`

12.32.21 UIFilterable

public @interface **UIFilterable**

The `UIFilterable` annotation is used to mark a field that allows users to filter a list of objects by the values in the field. The discovery logic for this annotation is done in `org.motechproject.mds.annotations.internal.UIFilterableProcessor`.

Only fields, 'getter' or 'setter' methods can have this annotation for other methods this annotation is omitted. Also this annotation is permitted on fields of type: Date, DateTime, LocalDate, Boolean or List.

See also: `org.motechproject.mds.annotations.internal.UIFilterableProcessor`

12.33 org.motechproject.mds.builder

12.33.1 EntityBuilder

public interface **EntityBuilder**

An entity builder is responsible for building the entity class from an Entity schema.

Methods

build

ClassData **build** (Entity *entity*)

Builds a class definition for a given entity. The class is not registered with any classloader.

Parameters

- **entity** – the entity schema

Returns bytes of the newly constructed class

buildDDE

ClassData **buildDDE** (Entity *entity*, Bundle *bundle*)

Builds Developer Defined Entity. The main difference between regular build method and this one, is that this method fetches the class definition from the given bundle, and injects members to the constructed class from there, if possible, rather than building everything from scratch.

Parameters

- **entity** – the entity schema
- **bundle** – the bundle to fetch class definition from

Returns bytes of the constructed class

buildHistory

ClassData **buildHistory** (Entity *entity*)

Builds History class definition for the given entity. The history class definition contains the same members as the entity class, plus some fields history-exclusive, like schema version.

Parameters

- **entity** – the entity schema

Returns bytes of the constructed class

buildTrash

ClassData **buildTrash** (Entity *entity*)

Builds Trash class definition for the given entity. The trash class definition contains the same members as the entity class, plus some fields trash-exclusive.

Parameters

- **entity** – the entity schema

Returns bytes of the constructed class

prepareHistoryClass

void **prepareHistoryClass** (Entity *entity*)

Builds empty history class definition for the given entity and adds it to the class pool.

Parameters

- **entity** – the entity schema

prepareTrashClass

void **prepareTrashClass** (Entity *entity*)

Builds empty trash class definition for the given entity and adds it to the class pool.

Parameters

- **entity** – the entity schema

12.33.2 EntityInfrastructureBuilder

public interface **EntityInfrastructureBuilder**

The `EntityInfrastructureBuilder` is responsible for building infrastructure for a given entity: repository, interface and service classes.

Methods

buildHistoryInfrastructure

List<ClassData> **buildHistoryInfrastructure** (String *className*)

Builds the repository, interface and implementation of this interface classes for the given history class. The names for classes are generated by `org.motechproject.mds.util.ClassName.getRepositoryName(String)`, `org.motechproject.mds.util.ClassName.getInterfaceName(String)`, `org.motechproject.mds.util.ClassName.getServiceName(String)`, respectively.

Parameters

- **className** – a name of history class.

Returns a list of classes that represents infrastructure for the history class.

buildInfrastructure

List<ClassData> **buildInfrastructure** (Entity *entity*)

Builds the repository, interface and implementation of this interface classes for the given entity. The names for classes are generated by `org.motechproject.mds.util.ClassName.getRepositoryName(String)`, `org.motechproject.mds.util.ClassName.getInterfaceName(String)`, `org.motechproject.mds.util.ClassName.getServiceName(String)`, respectively.

Parameters

- **entity** – an instance of `org.motechproject.mds.domain.Entity`

Returns a list of classes that represents infrastructure for the given entity.

12.33.3 EntityMetadataBuilder

public interface **`EntityMetadataBuilder`**

The `EntityMetadataBuilderImpl` class is responsible for building jdo metadata for an entity class.

Methods

addBaseMetadata

void **`addBaseMetadata`**(JDOMetadata *jdoMetadata*, ClassData *classData*, EntityType *entityType*, Class<?> *definition*)

Adds base information about package and class name to a `javax.jdo.metadata.JDOMetadata` instance.

Parameters

- **jdoMetadata** – an empty instance of `javax.jdo.metadata.JDOMetadata`.
- **classData** – an instance of `org.motechproject.mds.domain.ClassData`
- **entityType** – type of the entity(regular, history or trash)
- **definition** – the definition of the parent class

addEntityMetadata

void **`addEntityMetadata`**(JDOMetadata *jdoMetadata*, Entity *entity*, Class<?> *definition*)

Adds information about package and class name to a `javax.jdo.metadata.JDOMetadata` instance.

Parameters

- **jdoMetadata** – a empty instance of `javax.jdo.metadata.JDOMetadata`.
- **entity** – a instance of `org.motechproject.mds.domain.Entity`
- **definition** – the definition of the class

addHelperClassMetadata

void **`addHelperClassMetadata`**(JDOMetadata *jdoMetadata*, ClassData *classData*, Entity *entity*, EntityType *entityType*, Class<?> *definition*)

Creates metadata with basic information about package and class name to the `javax.jdo.metadata.JDOMetadata` instance. Additionally, fetches fields from passed entities and adds metadata for fields, if it's necessary. If entity is null, it will work just like `addBaseMetadata(JDOMetadata, ClassData)` and won't add any metadata for fields.

Parameters

- **jdoMetadata** – an empty instance of `javax.jdo.metadata.JDOMetadata`.
- **classData** – an instance of `org.motechproject.mds.domain.ClassData`

- **entity** – an entity to fetch fields from
- **entityType** – type of the entity(history or trash)
- **definition** – the definition of the parent class

fixEnhancerIssuesInMetadata

void **fixEnhancerIssuesInMetadata** (JDOMetadata *jdoMetadata*)

This updates the metadata after enhancement. Nucleus makes some "corrections" which do not work with the runtime enhancer.

12.33.4 EnumBuilder

public interface **EnumBuilder**

An enum builder is responsible for building the enum class with the same values as those are defined in the field.

Methods

build

ClassData **build** (ComboboxHolder *holder*)

Builds an Enum, based on the passed `org.motechproject.mds.domain.ComboboxHolder` object. If not specified otherwise by the field metadata, the enum will be named after the entity and field name.

Parameters

- **holder** – A helper object, representing MDS Combobox type.

Returns Bytecode representation of the enum.

12.33.5 MDSConstructor

public interface **MDSConstructor**

This interface provides methods to create a class for the given entity. The implementation of this interface should also construct other classes like repository, service interface and implementation for this service interface.

Methods

constructEntities

boolean **constructEntities** ()

Creates a class definition and inserts it into the MDS class loader, based on data from database. The implementation of this method should also create a repository, interface (when it's necessary) and implementation of this interface.

After executing this method, it should be possible to create an instance of the given class definition and save it to the database by `javax.jdo.PersistenceManager` provided by DataNucleus.

An interface related with class definition should be created only for entities from outside bundles and if the bundle does not define its own interface.

Parameters

- **buildDDE** – `true` if class definitions for entities from outside bundles should also be created; otherwise `false`.

Returns `true` if there were entities for which class definitions should be created; otherwise `false`.

updateFields

void **`updateFields`** (Long *entityId*, Map<String, String> *fieldNameChanges*)

Updates the field names of an entity. This method alters the database schema by changing column names to the new value. This is done for the entity instances, history instances and trash instances.

Parameters

- **entityId** – The ID of an entity to update
- **fieldNameChanges** – A map, indexed by current field names and values being updated field names.

12.33.6 MDSDataProviderBuilder

public interface **`MDSDataProviderBuilder`**

The `MDSDataProviderBuilder` class is responsible for building the MDS Data Provider JSON, required to register a data provider in the Tasks module.

Methods

generateDataProvider

String **`generateDataProvider`**()

Generates JSON String, containing information about all entities with lookups. It contains all the information required by the task module to register a data provider.

Returns JSON-formatted String, containing information about MDS Data Provider

12.34 org.motechproject.mds.config

12.34.1 DeleteMode

public enum **`DeleteMode`**

The `DeleteMode` presents what should happen with objects when they are deleted. They can be deleted permanently `DELETE` or moved to the trash `TRASH`. This enum is related with the property `org.motechproject.mds.util.Constants.Config.MDS_DELETE_MODE`.

The `UNKNOWN` value should not be used in code as appropriate value. It was added to ensure that the `fromString(String)` method will not return {@value null} value.

Enum Constants

DELETE

public static final DeleteMode **`DELETE`**

TRASH

public static final DeleteMode **TRASH**

UNKNOWN

public static final DeleteMode **UNKNOWN**

12.34.2 MdsConfig

public class **MdsConfig**

Class responsible for handling MDS configuration. Since MDS does not use Server Config, everything connected to the MDS configuration needs to be handled by the module itself.

Constructors

MdsConfig

public **MdsConfig**()

Methods

asProperties

public Properties **asProperties**()

getDataNucleusProperties

public Properties **getDataNucleusProperties**()

getFlywayLocations

public String[] **getFlywayLocations**()

getFlywayMigrationDirectory

public File **getFlywayMigrationDirectory**()

getProperties

public Properties **getProperties**(String *filename*)

getResourceFileName

public String **getResourceFileName**(Resource *resource*)

init

public void **init** ()

setConfig

public void **setConfig** (List<Resource> *resources*)

setCoreConfigurationService

public void **setCoreConfigurationService** (CoreConfigurationService *coreConfigurationService*)

setMdsSqlProperties

public void **setMdsSqlProperties** (Properties *mdsSqlProperties*)

setSqlDBManager

public void **setSqlDBManager** (SqlDBManager *sqlDBManager*)

12.34.3 ModuleSettings

public class **ModuleSettings** extends Properties

The `ModuleSettings` contains the base module settings which are inside the `org.motechproject.mds.util.Constants.Config.MODULE_FILE`. The getters and setters inside this class always checks the given property and if it is incorrect then the default value of the given property will be returned.

Fields

DEFAULT_DELETE_MODE

public static final DeleteMode **DEFAULT_DELETE_MODE**

DEFAULT_EMPTY_TRASH

public static final Boolean **DEFAULT_EMPTY_TRASH**

DEFAULT_TIME_UNIT

public static final TimeUnit **DEFAULT_TIME_UNIT**

DEFAULT_TIME_VALUE

public static final Integer **DEFAULT_TIME_VALUE**

Methods

equals

public boolean **equals** (Object *obj*)

getDeleteMode

public DeleteMode **getDeleteMode** ()

getTimeUnit

public TimeUnit **getTimeUnit** ()

getTimeValue

public Integer **getTimeValue** ()

hashCode

public int **hashCode** ()

isEmptyTrash

public Boolean **isEmptyTrash** ()

setDeleteMode

public void **setDeleteMode** (String *deleteMode*)

setDeleteMode

public void **setDeleteMode** (DeleteMode *deleteMode*)

setEmptyTrash

public void **setEmptyTrash** (String *emptyTrash*)

setEmptyTrash

public void **setEmptyTrash** (Boolean *emptyTrash*)

setTimeUnit

public void **setTimeUnit** (String *timeUnit*)

setTimeUnit

public void **setTimeUnit** (TimeUnit *timeUnit*)

setTimeValue

public void **setTimeValue** (String *timeValue*)

setTimeValue

public void **setTimeValue** (Integer *timeValue*)

toString

public String **toString** ()

12.34.4 ModuleSettings.Deserializer

public static final class **Deserializer** extends JsonDeserializer<ModuleSettings>

Methods

deserialize

public ModuleSettings **deserialize** (JsonParser *jp*, DeserializationContext *ctxt*)

12.34.5 ModuleSettings.Serializer

public static final class **Serializer** extends JsonSerializer<ModuleSettings>

Methods

serialize

public void **serialize** (ModuleSettings *value*, JsonGenerator *jgen*, SerializerProvider *provider*)

12.34.6 SettingsService

public interface **SettingsService**

The `SettingsService` is a generic class, allowing access to all MDS settings, as well as providing an ability to easily change these settings.

Methods

getDeleteMode

DeleteMode **getDeleteMode**()

Returns current setting of the Delete mode. Depending on its setting, deleting an MDS instance either moves it to trash, or removes it permanently.

Returns current delete mode setting

getModuleSettings

ModuleSettings **getModuleSettings**()

Retrieves all MDS settings.

Returns MDS settings

getProperties

Properties **getProperties**()

Retrieves all MDS settings as `java.util.Properties`.

Returns MDS settings

getTimeUnit

TimeUnit **getTimeUnit**()

Together with `getTimeValue()` specifies frequency of the automatic removal of the instances.

Returns selected unit of time

getTimeValue

Integer **getTimeValue**()

Together with `getTimeUnit()` specifies frequency of the automatic removal of the instances.

Returns value as an integer

isEmptyTrash

Boolean **isEmptyTrash**()

Returns current setting of the Empty trash, which informs whether automatic removal of instances is enabled.

Returns true, if setting is enabled, false otherwise

saveModuleSettings

void **saveModuleSettings**(ModuleSettings *settings*)

Updates all MDS settings and performs necessary actions if required (eg. scheduling jobs, that remove instances from trash).

Parameters

- **settings** – settings to save

12.34.7 TimeUnit

public enum **`TimeUnit`**

The `TimeUnit` specifies what time unit should be used to specify time when the module trash should be cleaned. This enum is related with the property `org.motechproject.mds.util.Constants.Config.MDS_TIME_UNIT`.

Each value from this enum can be converted to long value that presents time interval in milliseconds. For example the `HOURS` value is equal to {@value 3.6E6}.

The `UNKNOWN` value should not be used in code as appropriate value. It was added to ensure that the `fromString(String)` method will not return {@value null} value.

Enum Constants

DAYS

public static final TimeUnit **`DAYS`**

HOURS

public static final TimeUnit **`HOURS`**

MONTHS

public static final TimeUnit **`MONTHS`**

UNKNOWN

public static final TimeUnit **`UNKNOWN`**

WEEKS

public static final TimeUnit **`WEEKS`**

YEARS

public static final TimeUnit **`YEARS`**

12.35 org.motechproject.mds.domain

12.35.1 BrowsingSettings

public class **BrowsingSettings**

The `BrowsingSettings` contains information about fields that will be visible on UI and could be used as filter on UI.

This class is read only (the data are not saved to database) and its main purpose is to provide methods that help developer to get displayed and filterable fields.

Constructors

BrowsingSettings

public **BrowsingSettings** (Entity *entity*)

Methods

getDisplayedFields

public List<Field> **getDisplayedFields** ()

getEntity

public Entity **getEntity** ()

getFilterableFields

public List<Field> **getFilterableFields** ()

setEntity

public void **setEntity** (Entity *entity*)

toDto

public BrowsingSettingsDto **toDto** ()

12.35.2 ClassData

public class **ClassData**

Represents a class name and its byte code.

Constructors

ClassData

public **ClassData** (String *className*, byte[] *bytecode*)

ClassData

public **ClassData** (String *className*, byte[] *bytecode*, boolean *interfaceClass*)

ClassData

public **ClassData** (Entity *entity*, byte[] *bytecode*)

ClassData

public **ClassData** (Entity *entity*, byte[] *bytecode*, boolean *interfaceClass*)

ClassData

public **ClassData** (String *className*, String *module*, String *namespace*, byte[] *bytecode*)

ClassData

public **ClassData** (String *className*, String *module*, String *namespace*, byte[] *bytecode*, EntityType *type*)

ClassData

public **ClassData** (String *className*, String *module*, String *namespace*, byte[] *bytecode*, boolean *interfaceClass*)

ClassData

public **ClassData** (String *className*, String *module*, String *namespace*, byte[] *bytecode*, boolean *interfaceClass*, EntityType *type*, boolean *enumClassData*)

Methods

getBytecode

public byte[] **getBytecode** ()

getClassName

public String **getClassName** ()

getModule

public String **getModule**()

getNamespace

public String **getNamespace**()

getType

public EntityType **getType**()

isDDE

public boolean **isDDE**()

isEnumClassData

public boolean **isEnumClassData**()

isInterfaceClass

public boolean **isInterfaceClass**()

toString

public String **toString**()

12.35.3 ComboboxHolder

public class **ComboboxHolder** extends FieldHolder

The main purpose of this class is to make it easier to find out what kind of type should be used when the field is added to the class definition.

Constructors

ComboboxHolder

public **ComboboxHolder**(Field *field*)

ComboboxHolder

public **ComboboxHolder**(Entity *entity*, Field *field*)

ComboboxHolder

public **ComboboxHolder** (EntityDto *entity*, FieldDto *field*)

ComboboxHolder

public **ComboboxHolder** (Class<?> *entityClass*, FieldDto *field*)

ComboboxHolder

public **ComboboxHolder** (List<? extends Pair<String, String>> *metadata*, List<? extends Pair<String, ?>> *settings*, String *className*, String *fieldName*)

Methods

getEnumName

public String **getEnumName** ()

Returns enum name, specified in the field metadata, or default name, if not explicitly provided

getTypeClassName

public String **getTypeClassName** ()

Returns fully qualified class name, that handles this combobox in the backend

getUnderlyingType

public String **getUnderlyingType** ()

Returns fully qualified class name, of the actual java type of this combobox field

getValues

public String[] **getValues** ()

Returns an array of possible values for this combobox

isAllowMultipleSelections

public boolean **isAllowMultipleSelections** ()

Returns true, if this combobox allows selecting multiple values; false otherwise

isAllowUserSupplied

public boolean **isAllowUserSupplied**()

Returns true, if this combobox allows user supplied values; false otherwise

isCollection

public boolean **isCollection**()

Returns true, if this combobox is handled by a collection type in the backend; false otherwise

isEnum

public boolean **isEnum**()

Returns true, if this combobox does not allow user supplied values and allows selecting only single value; false otherwise

isEnumCollection

public boolean **isEnumCollection**()

Returns true, if this combobox does not allow user supplied values and allows selecting multiple values; false otherwise

isString

public boolean **isString**()

Returns true, if this combobox allows user supplied values and allows selecting only single value; false otherwise

isStringCollection

public boolean **isStringCollection**()

Returns true, if this combobox allows user supplied values and allows selecting multiple values; false otherwise

12.35.4 ConfigSettings

public class **ConfigSettings**

The `ConfigSettings` class represents Data Services settings, that can be adjusted by users via UI.

Constructors

ConfigSettings

public **ConfigSettings**()

ConfigSettings

public **ConfigSettings** (DeleteMode *deleteMode*, boolean *emptyTrash*, int *afterTimeValue*, TimeUnit *afterTimeUnit*)

Methods

getAfterTimeUnit

public TimeUnit **getAfterTimeUnit** ()

getAfterTimeValue

public int **getAfterTimeValue** ()

getDeleteMode

public DeleteMode **getDeleteMode** ()

getEmptyTrash

public boolean **getEmptyTrash** ()

getId

public Long **getId** ()

setAfterTimeUnit

public void **setAfterTimeUnit** (TimeUnit *afterTimeUnit*)

setAfterTimeValue

public void **setAfterTimeValue** (int *afterTimeValue*)

setDeleteMode

public void **setDeleteMode** (DeleteMode *deleteMode*)

setEmptyTrash

public void **setEmptyTrash** (boolean *emptyTrash*)

setId

public void **setId** (Long *id*)

12.35.5 Entity

public class **Entity**

The `Entity` class contains information about an entity. It also contains information about advanced settings related with the entity.

Constructors

Entity

public **Entity** ()

Entity

public **Entity** (String *className*)

Entity

public **Entity** (String *className*, String *module*, String *namespace*, SecurityMode *securityMode*)

Entity

public **Entity** (String *className*, String *name*, String *module*, String *namespace*, SecurityMode *securityMode*, Set<String> *securityMembers*, SecurityMode *readOnlySecurityMode*, Set<String> *readOnlySecurityMembers*)

Methods

addField

public void **addField** (Field *field*)

addLookup

public void **addLookup** (Lookup *lookup*)

advancedSettingsDto

public AdvancedSettingsDto **advancedSettingsDto** ()

getBrowsingSettings

public BrowsingSettings **getBrowsingSettings**()

getClassName

public String **getClassName**()

getComboboxFields

public List<Field> **getComboboxFields**()

getDrafts

public List<EntityDraft> **getDrafts**()

getEntityVersion

public Long **getEntityVersion**()

getField

public Field **getField**(Long *id*)

getField

public Field **getField**(String *name*)

getFieldDtos

public List<FieldDto> **getFieldDtos**()

getFields

public List<Field> **getFields**()

getFieldsExposedByRest

public List<Field> **getFieldsExposedByRest**()

getId

public Long **getId**()

getLookupById

public Lookup **getLookupById** (Long *lookupId*)

getLookupByName

public Lookup **getLookupByName** (String *lookupName*)

getLookupDtos

public List<LookupDto> **getLookupDtos** ()

getLookups

public List<Lookup> **getLookups** ()

getLookupsExposedByRest

public List<Lookup> **getLookupsExposedByRest** ()

getMaxFetchDepth

public Integer **getMaxFetchDepth** ()

getModule

public String **getModule** ()

getName

public String **getName** ()

getNamespace

public String **getNamespace** ()

getReadOnlySecurityMembers

public Set<String> **getReadOnlySecurityMembers** ()

getReadOnlySecurityMode

public SecurityMode **getReadOnlySecurityMode** ()

getRestOptions

public RestOptions **getRestOptions**()

getSecurityMembers

public Set<String> **getSecurityMembers**()

getSecurityMode

public SecurityMode **getSecurityMode**()

getStringComboboxFields

public List<Field> **getStringComboboxFields**()

getSuperClass

public String **getSuperClass**()

getTableName

public String **getTableName**()

getTracking

public Tracking **getTracking**()

incrementVersion

public void **incrementVersion**()

isAbstractClass

public boolean **isAbstractClass**()

isActualEntity

public boolean **isActualEntity**()

isAllowCreateEvent

public boolean **isAllowCreateEvent**()

isAllowDeleteEvent

public boolean **isAllowDeleteEvent** ()

isAllowUpdateEvent

public boolean **isAllowUpdateEvent** ()

isBaseEntity

public boolean **isBaseEntity** ()

isDDE

public boolean **isDDE** ()

isDraft

public boolean **isDraft** ()

isRecordHistory

public boolean **isRecordHistory** ()

isSecurityOptionsModified

public boolean **isSecurityOptionsModified** ()

isSubClassOfMdsEntity

public boolean **isSubClassOfMdsEntity** ()

removeField

public void **removeField** (Long *fieldId*)

removeLookup

public void **removeLookup** (Long *lookupId*)

setAbstractClass

public void **setAbstractClass** (boolean *abstractClass*)

setClassName

public void **setClassName** (String *className*)

setDrafts

public void **setDrafts** (List<EntityDraft> *drafts*)

setEntityVersion

public void **setEntityVersion** (Long *entityVersion*)

setFields

public void **setFields** (List<Field> *fields*)

setId

public void **setId** (Long *id*)

setLookups

public void **setLookups** (List<Lookup> *lookups*)

setMaxFetchDepth

public void **setMaxFetchDepth** (Integer *maxFetchDepth*)

setModule

public void **setModule** (String *module*)

setName

public final void **setName** (String *name*)

setNamespace

public void **setNamespace** (String *namespace*)

setReadOnlySecurity

public void **setReadOnlySecurity** (SecurityMode *readOnlySecurityMode*, List<String> *readOnlySecurityMembersList*)

setReadOnlySecurityMembers

public void **setReadOnlySecurityMembers** (Set<String> *readOnlySecurityMembers*)

setReadOnlySecurityMode

public void **setReadOnlySecurityMode** (SecurityMode *readOnlySecurityMode*)

setRestOptions

public void **setRestOptions** (RestOptions *restOptions*)

setSecurity

public void **setSecurity** (SecurityMode *securityMode*, List<String> *securityMembersList*)

setSecurityMembers

public void **setSecurityMembers** (Set<String> *securityMembers*)

setSecurityMode

public void **setSecurityMode** (SecurityMode *securityMode*)

setSecurityOptionsModified

public void **setSecurityOptionsModified** (boolean *securityOptionsModified*)

setSuperClass

public void **setSuperClass** (String *superClass*)

setTableName

public void **setTableName** (String *tableName*)

setTracking

public void **setTracking** (Tracking *tracking*)

supportsAnyRestOperations

public boolean **supportsAnyRestOperations** ()

toDto

public EntityDto **toDto**()

updateAdvancedSetting

public void **updateAdvancedSetting**(AdvancedSettingsDto *advancedSettings*)

updateFromDraft

public void **updateFromDraft**(EntityDraft *draft*)

updateIndexes

public void **updateIndexes**(List<LookupDto> *indexes*)

updateRestOptions

public void **updateRestOptions**(RestOptionsDto *restOptionsDto*)

updateTracking

public void **updateTracking**(TrackingDto *trackingDto*)

12.35.6 EntityAudit

public class **EntityAudit** extends Entity

This class represents a single historical commit of an Entity.

Methods

getModificationDate

public DateTime **getModificationDate**()

getOwnerUsername

public String **getOwnerUsername**()

getVersion

public Long **getVersion**()

isActualEntity

public boolean **isActualEntity**()

setModificationDate

public void **setModificationDate**(DateTime *modificationDate*)

setOwnerUsername

public void **setOwnerUsername**(String *ownerUsername*)

setVersion

public void **setVersion**(Long *version*)

12.35.7 EntityDefinitionType

public enum **EntityDefinitionType**

Represents entity origin.

Enum Constants

DDE

public static final EntityDefinitionType **DDE**

Developer Defined Entity. Entity, that has been created by developer, using MDS annotations.

EUDE

public static final EntityDefinitionType **EUDE**

End User Defined Entity. Entity, that has been created by user, using MDS User Interface.

12.35.8 EntityDraft

public class **EntityDraft** extends Entity

This class represents user drafts of an Entity. A draft is user's work in progress from the UI. This shares the table with its superclass, `Entity`.

Constructors

EntityDraft

public **EntityDraft**()

Methods

getDraftOwnerUsername

public String **getDraftOwnerUsername**()

getFieldNameChanges

public Map<String, String> **getFieldNameChanges**()

getLastModificationDate

public DateTime **getLastModificationDate**()

getParentEntity

public Entity **getParentEntity**()

getParentVersion

public Long **getParentVersion**()

isActualEntity

public boolean **isActualEntity**()

isChangesMade

public boolean **isChangesMade**()

isDraft

public boolean **isDraft**()

isOutdated

public boolean **isOutdated**()

setChangesMade

public void **setChangesMade**(boolean *changesMade*)

setDraftOwnerUsername

public void **setDraftOwnerUsername**(String *draftOwnerUsername*)

setFieldNameChanges

public void **setFieldNameChanges** (Map<String, String> *fieldNameChanges*)

setLastModificationDate

public void **setLastModificationDate** (DateTime *lastModificationDate*)

setParentEntity

public void **setParentEntity** (Entity *parentEntity*)

setParentVersion

public void **setParentVersion** (Long *parentVersion*)

toDto

public EntityDto **toDto** ()

12.35.9 EntityInfo

public class **EntityInfo**

The `EntityInfo` class contains base information about the given entity, like its class name or infrastructure class names.

See also: `org.motechproject.mds.service.JarGeneratorService`

Methods

entitiesWithAnyCRUDAction

public static Collection<EntityInfo> **entitiesWithAnyCRUDAction** (Collection<EntityInfo> *entityInfos*)

getClassName

public String **getClassName** ()

getEntityName

public String **getEntityName** ()

getFieldsInfo

public List<FieldInfo> **getFieldsInfo** ()

getInfrastructure

public String[] **getInfrastructure**()

getInterfaceName

public String **getInterfaceName**()

getModule

public String **getModule**()

getName

public String **getName**()

getNamespace

public String **getNamespace**()

getNonAutoFieldInfos

public List<FieldInfo> **getNonAutoFieldInfos**()

getRepository

public String **getRepository**()

getRestFieldInfos

public List<FieldInfo> **getRestFieldInfos**()

getRestId

public String **getRestId**()

getServiceName

public String **getServiceName**()

isCreateEventFired

public boolean **isCreateEventFired**()

isDeleteEventFired

public boolean **isDeleteEventFired**()

isRestCreateEnabled

public boolean **isRestCreateEnabled**()

isRestDeleteEnabled

public boolean **isRestDeleteEnabled**()

isRestReadEnabled

public boolean **isRestReadEnabled**()

isRestUpdateEnabled

public boolean **isRestUpdateEnabled**()

isUpdateEventFired

public boolean **isUpdateEventFired**()

setClassName

public void **setClassName**(String *className*)

setCreateEventFired

public void **setCreateEventFired**(boolean *createEventFired*)

setDeleteEventFired

public void **setDeleteEventFired**(boolean *deleteEventFired*)

setEntityName

public void **setEntityName**(String *entityName*)

setFieldsInfo

public void **setFieldsInfo**(List<FieldInfo> *fieldsInfo*)

setInterfaceName

public void **setInterfaceName** (String *interfaceName*)

setModule

public void **setModule** (String *module*)

setNamespace

public void **setNamespace** (String *namespace*)

setRepository

public void **setRepository** (String *repository*)

setRestCreateEnabled

public void **setRestCreateEnabled** (boolean *restCreateEnabled*)

setRestDeleteEnabled

public void **setRestDeleteEnabled** (boolean *restDeleteEnabled*)

setRestReadEnabled

public void **setRestReadEnabled** (boolean *restReadEnabled*)

setRestUpdateEnabled

public void **setRestUpdateEnabled** (boolean *restUpdateEnabled*)

setServiceName

public void **setServiceName** (String *serviceName*)

setUpdateEventFired

public void **setUpdateEventFired** (boolean *updateEventFired*)

supportAnyRestAccess

public boolean **supportAnyRestAccess** ()

12.35.10 EntityType

public enum **EntityType**
Represents the type of an entity and their associated class names.

Enum Constants

HISTORY

public static final EntityType **HISTORY**
Entity representing a historical revision.

STANDARD

public static final EntityType **STANDARD**
Regular entity.

TRASH

public static final EntityType **TRASH**
Entity representing an instance in trash.

12.35.11 Field

public class **Field**
The `Field` class contains information about a single field.

Constructors

Field

public **Field**()

Field

public **Field**(Entity *entity*, String *name*, String *displayName*)

Field

public **Field**(Entity *entity*, String *name*, String *displayName*, Type *type*)

Field

public **Field**(Entity *entity*, String *name*, String *displayName*, Set<Lookup> *lookups*)

Field

public **Field** (Entity *entity*, String *name*, String *displayName*, Type *type*, boolean *required*, boolean *readOnly*)

Field

public **Field** (Entity *entity*, String *name*, String *displayName*, boolean *required*, boolean *readOnly*, boolean *nonEditable*, boolean *nonDisplayable*, String *defaultValue*, String *tooltip*, String *placeholder*, Set<Lookup> *lookups*)

Field

public **Field** (Entity *entity*, String *name*, String *displayName*, boolean *required*, boolean *readOnly*, boolean *nonEditable*, boolean *nonDisplayable*, boolean *uiChanged*, String *defaultValue*, String *tooltip*, String *placeholder*, Set<Lookup> *lookups*)

Methods

addMetadata

public void **addMetadata** (FieldMetadata *metadata*)

addSetting

public void **addSetting** (FieldSetting *setting*)

addValidation

public void **addValidation** (FieldValidation *validation*)

copy

public Field **copy** ()

getDefaultValue

public String **getDefaultValue** ()

getDisplayName

public String **getDisplayName** ()

getEntity

public Entity **getEntity** ()

getId

public Long **getId**()

getLookups

public Set<Lookup> **getLookups**()

getMetadata

public List<FieldMetadata> **getMetadata**()

getMetadata

public FieldMetadata **getMetadata**(String *key*)

getMetadataValue

public String **getMetadataValue**(String *key*)

getName

public String **getName**()

getPlaceholder

public String **getPlaceholder**()

getSettingByName

public FieldSetting **getSettingByName**(String *name*)

getSettings

public List<FieldSetting> **getSettings**()

getTooltip

public String **getTooltip**()

getType

public Type **getType**()

getUIDisplayPosition

public Long **getUIDisplayPosition**()

getValidationByName

public FieldValidation **getValidationByName**(String *name*)

getValidations

public List<FieldValidation> **getValidations**()

hasMetadata

public boolean **hasMetadata**(String *key*)

isAutoGenerated

public boolean **isAutoGenerated**()

isExposedViaRest

public boolean **isExposedViaRest**()

isMultiSelectCombobox

public boolean **isMultiSelectCombobox**()

isNonDisplayable

public boolean **isNonDisplayable**()

isNonEditable

public boolean **isNonEditable**()

isReadOnly

public boolean **isReadOnly**()

isRequired

public boolean **isRequired**()

isUIDisplayable

public boolean **isUIDisplayable**()

isUIFilterable

public boolean **isUIFilterable**()

isUiChanged

public boolean **isUiChanged**()

setDefaultValue

public void **setDefaultValue**(String *defaultValue*)

setDisplayName

public void **setDisplayName**(String *displayName*)

setEntity

public void **setEntity**(Entity *entity*)

setExposedViaRest

public void **setExposedViaRest**(boolean *exposedViaRest*)

setId

public void **setId**(Long *id*)

setLookups

public void **setLookups**(Set<Lookup> *lookups*)

setMetadata

public void **setMetadata**(List<FieldMetadata> *metadata*)

setMetadataValue

public void **setMetadataValue**(String *key*, String *value*)

setName

public final void **setName** (String *name*)

setNonDisplayable

public void **setNonDisplayable** (boolean *nonDisplayable*)

setNonEditable

public void **setNonEditable** (boolean *nonEditable*)

setPlaceholder

public void **setPlaceholder** (String *placeholder*)

setReadOnly

public void **setReadOnly** (boolean *readOnly*)

setRequired

public void **setRequired** (boolean *required*)

setSettings

public void **setSettings** (List<FieldSetting> *settings*)

setTooltip

public void **setTooltip** (String *tooltip*)

setType

public void **setType** (Type *type*)

setUIDisplayPosition

public void **setUIDisplayPosition** (Long *uiDisplayPosition*)

setUIDisplayable

public void **setUIDisplayable** (boolean *uiDisplayable*)

setUIFilterable

public void **setUIFilterable** (boolean *uiFilterable*)

setUiChanged

public void **setUiChanged** (boolean *uiChanged*)

setValidations

public void **setValidations** (List<FieldValidation> *validations*)

settingsToDto

public List<SettingDto> **settingsToDto** ()

toDto

public FieldDto **toDto** ()

update

public Field **update** (FieldDto *field*)

12.35.12 FieldHolder

public class **FieldHolder**

The main purpose of this class is to provide an easy way to access values inside metadata and settings related with the given field.

Constructors

FieldHolder

public **FieldHolder** (Field *field*)

FieldHolder

public **FieldHolder** (FieldDto *field*)

FieldHolder

protected **FieldHolder** (List<? extends Pair<String, String>> *metadata*, List<? extends Pair<String, ?>> *settings*)

Methods

getMetadata

public String **getMetadata** (String *name*)

Retrieves metadata value of the given name from this field.

Parameters

- **name** – metadata key

Returns value of the metadata entry

getMetadata

public String **getMetadata** (String *name*, String *defaultValue*)

Retrieves metadata value of the given name from this field. If there's no metadata entry of the given name, a default value is returned.

Parameters

- **name** – metadata key
- **defaultValue** – default value to use, in case metadata entry is not present

Returns value of the metadata entry, or default value

getSetting

public String **getSetting** (String *name*)

Retrieves value of the setting, with the given name.

Parameters

- **name** – setting name

Returns value of the setting

getSetting

public String **getSetting** (String *name*, String *defaultValue*)

Retrieves value of the setting, with the given name. If there's no setting with the given name, a default value is returned.

Parameters

- **name** – setting name
- **defaultValue** – default value to use, in case given setting is not present

Returns value of the setting

getSettingAsArray

public String[] **getSettingAsArray** (String *name*)

Retrieves value of the setting, with the given name and parses the result into an array of Strings. Comma mark (,) will be treated as a separator of the elements in the setting value.

Parameters

- **name** – setting name

Returns value of the setting, in form of an array of Strings

getSettingAsBoolean

public boolean **getSettingAsBoolean** (String *name*)

Retrieves value of the setting, with the given name and parses the result to `boolean`.

Parameters

- **name** – setting name

Returns value of the setting; true or false only

12.35.13 FieldInfo

public class **FieldInfo**

The `FieldInfo` class contains base information about the given entity field like its name or type.

See also: `org.motechproject.mds.service.JarGeneratorService`

Methods

getDisplayName

public String **getDisplayName** ()

getName

public String **getName** ()

getTaskType

public String **getTaskType** ()

getType

public String **getType** ()

getTypeInfo

public TypeInfo **getTypeInfo** ()

isAutoGenerated

public boolean **isAutoGenerated** ()

isRequired

public boolean **isRequired**()

isRestExposed

public boolean **isRestExposed**()

setAutoGenerated

public void **setAutoGenerated**(boolean *autoGenerated*)

setDisplayName

public void **setDisplayName**(String *displayName*)

setName

public void **setName**(String *name*)

setRequired

public void **setRequired**(boolean *required*)

setRestExposed

public void **setRestExposed**(boolean *restExposed*)

setTypeInfo

public void **setTypeInfo**(TypeInfo *typeInfo*)

12.35.14 FieldInfo.TypeInfo

public class **TypeInfo**

Methods

getItems

public List<String> **getItems**()

getTaskType

public String **getTaskType**()

getType

public String **getType**()

isAllowUserSupplied

public boolean **isAllowUserSupplied**()

isAllowsMultipleSelection

public boolean **isAllowsMultipleSelection**()

isCombobox

public boolean **isCombobox**()

setAllowUserSupplied

public void **setAllowUserSupplied**(boolean *allowUserSupplied*)

setAllowsMultipleSelection

public void **setAllowsMultipleSelection**(boolean *allowsMultipleSelection*)

setCombobox

public void **setCombobox**(boolean *isCombobox*)

setItems

public void **setItems**(List<String> *items*)

setTaskType

public void **setTaskType**(String *taskType*)

setType

public void **setType**(String *type*)

12.35.15 FieldMetadata

public class **FieldMetadata** implements Pair<String, String>

The `FieldMetadata` class contains information about a single metadata added into a field.

Constructors

FieldMetadata

public **FieldMetadata**()

FieldMetadata

public **FieldMetadata**(Field *field*, String *key*)

FieldMetadata

public **FieldMetadata**(MetadataDto *metadata*)

FieldMetadata

public **FieldMetadata**(Field *field*, String *key*, String *value*)

Methods

copy

public FieldMetadata **copy**()

getField

public Field **getField**()

getId

public Long **getId**()

getKey

public String **getKey**()

getValue

public String **getValue**()

setField

public void **setField**(Field *field*)

setId

public void **setId** (Long *id*)

setKey

public void **setKey** (String *key*)

setValue

public void **setValue** (String *value*)

toDto

public MetadataDto **toDto** ()

update

public final void **update** (MetadataDto *metadata*)

12.35.16 FieldSetting

public class **FieldSetting** implements Pair<String, String>
Represents settings of a single field.

Constructors

FieldSetting

public **FieldSetting** ()

FieldSetting

public **FieldSetting** (Field *field*, TypeSetting *details*)

FieldSetting

public **FieldSetting** (Field *field*, TypeSetting *details*, String *value*)

Methods

copy

public FieldSetting **copy** ()

getDetails

public TypeSetting **getDetails**()

getField

public Field **getField**()

getId

public Long **getId**()

getKey

public String **getKey**()

getValue

public String **getValue**()

setDetails

public void **setDetails**(TypeSetting *details*)

setField

public void **setField**(Field *field*)

setId

public void **setId**(Long *id*)

setValue

public void **setValue**(String *value*)

toDto

public SettingDto **toDto**()

12.35.17 FieldValidation

public class **FieldValidation**

The `FieldValidation` class contains the value that is related with the correct type validation and information about that whether the given validation is enabled or not.

Constructors

FieldValidation

public **FieldValidation**()

FieldValidation

public **FieldValidation**(Field *field*, TypeValidation *details*)

FieldValidation

public **FieldValidation**(Field *field*, TypeValidation *details*, String *value*, boolean *enabled*)

Methods

copy

public FieldValidation **copy**()

getDetails

public TypeValidation **getDetails**()

getField

public Field **getField**()

getId

public Long **getId**()

getValue

public String **getValue**()

isEnabled

public boolean **isEnabled**()

setDetails

public void **setDetails**(TypeValidation *details*)

setEnabled

public void **setEnabled** (boolean *enabled*)

setField

public void **setField** (Field *field*)

setId

public void **setId** (Long *id*)

setValue

public void **setValue** (String *value*)

toDto

public ValidationCriterionDto **toDto** ()

12.35.18 ImportExportBlueprint

public class **ImportExportBlueprint** extends ArrayList<ImportExportBlueprint.Record>

The `ImportExportBlueprint` represents MDS import or export plan, specifying which entities and which parts of those entities should be included in either import or export.

See also: `org.motechproject.mds.service.ImportExportService`

Methods

includeEntityData

public void **includeEntityData** (String *entityName*, boolean *includeData*)

includeEntitySchema

public void **includeEntitySchema** (String *entityName*, boolean *includeSchema*)

isIncludeEntityData

public boolean **isIncludeEntityData** (String *entityName*)

isIncludeEntitySchema

public boolean **isIncludeEntitySchema** (String *entityName*)

12.35.19 ImportExportBlueprint.Record

public static class **Record**

Methods

getEntityName

public String **getEntityName**()

isIncludeData

public boolean **isIncludeData**()

isIncludeSchema

public boolean **isIncludeSchema**()

setEntityName

public void **setEntityName**(String *entityName*)

setIncludeData

public void **setIncludeData**(boolean *includeData*)

setIncludeSchema

public void **setIncludeSchema**(boolean *includeSchema*)

12.35.20 ImportManifest

public class **ImportManifest**

The `ImportManifest` holds components available for import that are contained in a single MDS import file.

See also: `org.motechproject.mds.service.ImportExportService`

Constructors

ImportManifest

public **ImportManifest**()

Methods

addRecord

public Record **addRecord** (String *entityName*, String *moduleName*)

getImportId

public String **getImportId** ()

getRecords

public List<Record> **getRecords** ()

setImportId

public void **setImportId** (String *importId*)

12.35.21 ImportManifest.Record

public static class **Record**

Methods

getEntityName

public String **getEntityName** ()

getModuleName

public String **getModuleName** ()

isCanIncludeData

public boolean **isCanIncludeData** ()

isCanIncludeSchema

public boolean **isCanIncludeSchema** ()

setCanIncludeData

public void **setCanIncludeData** (boolean *canIncludeData*)

setCanIncludeSchema

public void **setCanIncludeSchema** (boolean *canIncludeSchema*)

setEntityName

public void **setEntityName** (String *entityName*)

setModuleName

public void **setModuleName** (String *moduleName*)

12.35.22 JsonLookup

public class **JsonLookup**
Contains information about single lookup added via JSON file.

Methods

getEntityClassName

public String **getEntityClassName** ()

getOriginLookupName

public String **getOriginLookupName** ()

setEntityClassName

public void **setEntityClassName** (String *entityClassName*)

setOriginLookupName

public void **setOriginLookupName** (String *originLookupName*)

12.35.23 Lookup

public class **Lookup**
The `Lookup` class contains information about single lookup

Constructors

Lookup

public **Lookup** ()

Lookup

public **Lookup** (String *lookupName*, boolean *singleObjectReturn*, boolean *exposedViaRest*, List<Field> *fields*)

Lookup

public **Lookup** (String *lookupName*, boolean *singleObjectReturn*, boolean *exposedViaRest*, List<Field> *fields*, boolean *readOnly*, String *methodName*)

Lookup

public **Lookup** (String *lookupName*, boolean *singleObjectReturn*, boolean *exposedViaRest*, List<Field> *fields*, boolean *readOnly*, String *methodName*, List<String> *rangeLookupFields*, List<String> *setLookupFields*, Map<String, String> *customOperators*)

Lookup

public **Lookup** (String *lookupName*, boolean *singleObjectReturn*, boolean *exposedViaRest*, List<Field> *fields*, boolean *readOnly*, String *methodName*, List<String> *rangeLookupFields*, List<String> *setLookupFields*, Map<String, String> *customOperators*, Map<String, Boolean> *useGenericParams*, List<String> *fieldsOrder*)

Lookup

public **Lookup** (String *lookupName*, boolean *singleObjectReturn*, boolean *exposedViaRest*, List<Field> *fields*, Entity *entity*)

Lookup

public **Lookup** (LookupDto *lookupDto*, List<Field> *lookupFields*)

Methods

copy

public Lookup **copy** (List<Field> *fields*)

getCustomOperators

public Map<String, String> **getCustomOperators** ()

getEntity

public Entity **getEntity** ()

getFields

public List<Field> **getFields**()

getFieldsOrder

public List<String> **getFieldsOrder**()

getId

public Long **getId**()

getLookupFieldById

public final Field **getLookupFieldById**(Long *id*)

getLookupFieldByName

public final Field **getLookupFieldByName**(String *name*)

getLookupFieldType

public LookupFieldType **getLookupFieldType**(String *fieldName*)

getLookupName

public String **getLookupName**()

getMethodName

public String **getMethodName**()

getRangeLookupFields

public final List<String> **getRangeLookupFields**()

getSetLookupFields

public final List<String> **getSetLookupFields**()

getUseGenericParams

public Map<String, Boolean> **getUseGenericParams**()

isExposedViaRest

public boolean **isExposedViaRest** ( )

isRangeParam

public boolean **isRangeParam** (String *field*)

isReadOnly

public boolean **isReadOnly** ( )

isSetParam

public boolean **isSetParam** (String *field*)

isSingleObjectReturn

public boolean **isSingleObjectReturn** ( )

setCustomOperators

public void **setCustomOperators** (Map<String, String> *customOperators*)

setEntity

public void **setEntity** (Entity *entity*)

setExposedViaRest

public void **setExposedViaRest** (boolean *exposedViaRest*)

setFields

public void **setFields** (List<Field> *fields*)

setFieldsOrder

public void **setFieldsOrder** (List<String> *fieldsOrder*)

setId

public void **setId** (Long *id*)

setLookupName

public final void **setLookupName** (String *lookupName*)

setMethodName

public void **setMethodName** (String *methodName*)

setRangeLookupFields

public void **setRangeLookupFields** (List<String> *rangeLookupFields*)

setReadOnly

public void **setReadOnly** (boolean *readOnly*)

setSetLookupFields

public void **setSetLookupFields** (List<String> *setLookupFields*)

setSingleObjectReturn

public void **setSingleObjectReturn** (boolean *singleObjectReturn*)

setUseGenericParams

public void **setUseGenericParams** (Map<String, Boolean> *useGenericParams*)

toDto

public LookupDto **toDto** ()

update

public final void **update** (LookupDto *lookupDto*, List<Field> *lookupFields*)

12.35.24 ManyToManyRelationship

public class **ManyToManyRelationship** extends Relationship

A specialization of the `Relationship` class. Represents a many-to-many relationship.

Methods

getFieldType

public String **getFieldType** (Field *field*, EntityType *type*)

getGenericSignature

public String **getGenericSignature** (Field *field*, EntityType *type*)

12.35.25 ManyToOneRelationship

public class **ManyToOneRelationship** extends Relationship

A specialization of the `org.motechproject.mds.domain.Relationship` class. Represents a many-to-one relationship.

Methods

getFieldType

public String **getFieldType** (Field *field*, EntityType *type*)

getGenericSignature

public String **getGenericSignature** (Field *field*, EntityType *type*)

12.35.26 MdsEntity

public abstract class **MdsEntity**

The `MdsEntity` is an optional class for all domain classes acting as Motech data services Entities. This class stores and allows access to all default fields like id, creator or modification date. All classes annotated `org.motechproject.mds.annotations.Entity` can extend this base class.

Methods

getCreationDate

public DateTime **getCreationDate** ()

getCreator

public String **getCreator** ()

getId

public Long **getId** ()

getModificationDate

public DateTime **getModificationDate**()

getModifiedBy

public String **getModifiedBy**()

getOwner

public String **getOwner**()

setCreationDate

public void **setCreationDate**(DateTime *creationDate*)

setCreator

public void **setCreator**(String *creator*)

setId

public void **setId**(Long *id*)

setModificationDate

public void **setModificationDate**(DateTime *modificationDate*)

setModifiedBy

public void **setModifiedBy**(String *modifiedBy*)

setOwner

public void **setOwner**(String *owner*)

12.35.27 MigrationMapping

public class **MigrationMapping**

The `MigrationMapping` class contains information about flyway migrations from modules(It maps module migration version to the flyway migration version).

Constructors

MigrationMapping

public **MigrationMapping**()

MigrationMapping

public **MigrationMapping**(String *moduleName*, Integer *moduleMigrationVersion*)

Methods

getFlywayMigrationVersion

public Integer **getFlywayMigrationVersion**()

getModuleMigrationVersion

public Integer **getModuleMigrationVersion**()

getModuleName

public String **getModuleName**()

setFlywayMigrationVersion

public void **setFlywayMigrationVersion**(Integer *flywayMigrationVersion*)

setModuleMigrationVersion

public void **setModuleMigrationVersion**(Integer *moduleMigrationVersion*)

setModuleName

public void **setModuleName**(String *moduleName*)

12.35.28 OneToManyRelationship

public class **OneToManyRelationship** extends Relationship

A specialization of the `Relationship` class. Represents a one-to-many relationship.

Methods

getFieldType

public String **getFieldType** (Field *field*, EntityType *type*)

getGenericSignature

public String **getGenericSignature** (Field *field*, EntityType *type*)

12.35.29 OneToOneRelationship

public class **OneToOneRelationship** extends Relationship

A specialization of the `Relationship` class. Represents a one-to-one relationship.

Methods

getFieldType

public String **getFieldType** (Field *field*, EntityType *type*)

getGenericSignature

public String **getGenericSignature** (Field *field*, EntityType *type*)

12.35.30 Relationship

public class **Relationship**

A class representing a relationship type. This class is inherited by different types of relationships. This class only represents the field type and provides some utility methods. It is not used in entities themselves.

Methods

getFieldType

public String **getFieldType** (Field *field*, EntityType *type*)

getGenericSignature

public String **getGenericSignature** (Field *field*, EntityType *type*)

getRelatedClassName

protected String **getRelatedClassName** (Field *field*, EntityType *type*)

12.35.31 RelationshipHolder

public class **RelationshipHolder** extends FieldHolder

The main purpose of this class is to find out how cascade should be used for the given field with relationship type.

Constructors

RelationshipHolder

public **RelationshipHolder** (Field *field*)

RelationshipHolder

public **RelationshipHolder** (ClassData *data*, Field *field*)

Methods

getCollectionClassName

public String **getCollectionClassName** ()

getRelatedClass

public String **getRelatedClass** ()

getRelatedField

public String **getRelatedField** ()

hasUnresolvedRelation

public boolean **hasUnresolvedRelation** ()

If this returns true, it means that either: the relation is uni-directional or the relation is bi-directional, and we should expect related class to define which fields are related

Returns true if relation is uni-directional or bi-directional without defined related field; false otherwise

isBiDirectional

public boolean **isBiDirectional** ()

isCascadeDelete

public boolean **isCascadeDelete** ()

isCascadePersist

public boolean **isCascadePersist** ( )

isCascadeUpdate

public boolean **isCascadeUpdate** ( )

isManyToMany

public boolean **isManyToMany** ( )

isManyToOne

public boolean **isManyToOne** ( )

isOneToMany

public boolean **isOneToMany** ( )

isOneToOne

public boolean **isOneToOne** ( )

isOwningSide

public boolean **isOwningSide** ( )

12.35.32 RestDocs

public class **RestDocs**

Domain class for persisting the generated Rest documentation.

Methods

getDocumentation

public String **getDocumentation** ( )

getId

public Long **getId** ( )

setDocumentation

public void **setDocumentation** (String *documentation*)

setId

public void **setId** (Long *id*)

12.35.33 RestOptions

public class **RestOptions**

The `RestOptions` class represents rest options of an entity. This class is related with table in database with the same name.

Constructors

RestOptions

public **RestOptions** ()

RestOptions

public **RestOptions** (Entity *entity*)

Methods

copy

public RestOptions **copy** ()

equals

public boolean **equals** (Object *obj*)

getEntity

public Entity **getEntity** ()

getFields

public List<Field> **getFields** ()

getId

public Long **getId** ()

getLookups

public List<Lookup> **getLookups** ()

hashCode

public int **hashCode** ()

isAllowCreate

public boolean **isAllowCreate** ()

isAllowDelete

public boolean **isAllowDelete** ()

isAllowRead

public boolean **isAllowRead** ()

isAllowUpdate

public boolean **isAllowUpdate** ()

isModifiedByUser

public boolean **isModifiedByUser** ()

setAllowCreate

public void **setAllowCreate** (boolean *allowCreate*)

setAllowDelete

public void **setAllowDelete** (boolean *allowDelete*)

setAllowRead

public void **setAllowRead** (boolean *allowRead*)

setAllowUpdate

public void **setAllowUpdate** (boolean *allowUpdate*)

setEntity

public void **setEntity** (Entity *entity*)

setId

public void **setId** (Long *id*)

setModifiedByUser

public void **setModifiedByUser** (boolean *modifiedByUser*)

supportsAnyOperation

public boolean **supportsAnyOperation** ()

toDto

public RestOptionsDto **toDto** ()

update

public final void **update** (RestOptionsDto *restOptionsDto*)

12.35.34 SchemaChangeLock

public class **SchemaChangeLock**
The object used for locking schema change access.

Fields

LOCK_ID

public static final long **LOCK_ID**

Methods

getComment

public String **getComment** ()

getId

public Long **getId** ()

getLockId

public long **getLockId**()

getTimestamp

public DateTime **getTimestamp**()

setComment

public void **setComment**(String *comment*)

setId

public void **setId**(Long *id*)

setLockId

public void **setLockId**(long *lockId*)

setTimestamp

public void **setTimestamp**(DateTime *timestamp*)

12.35.35 Tracking

public class **Tracking**

The `Tracking` contains properties that describe the audit settings of an Entity, such as whether to record history or publish CRUD events for a given Entity. This class is related with table in database with the same name.

Constructors

Tracking

public **Tracking**()

Tracking

public **Tracking**(Entity *entity*)

Methods

copy

public Tracking **copy**()

equals

public boolean **equals** (Object *obj*)

getEntity

public Entity **getEntity** ()

getId

public Long **getId** ()

hashCode

public int **hashCode** ()

isAllowCreateEvent

public boolean **isAllowCreateEvent** ()

isAllowDeleteEvent

public boolean **isAllowDeleteEvent** ()

isAllowUpdateEvent

public boolean **isAllowUpdateEvent** ()

isModifiedByUser

public boolean **isModifiedByUser** ()

isNonEditable

public boolean **isNonEditable** ()

isRecordHistory

public boolean **isRecordHistory** ()

setAllowCreateEvent

public void **setAllowCreateEvent** (boolean *allowCreateEvent*)

setAllowDeleteEvent

public void **setAllowDeleteEvent** (boolean *allowDeleteEvent*)

setAllowUpdateEvent

public void **setAllowUpdateEvent** (boolean *allowUpdateEvent*)

setEntity

public void **setEntity** (Entity *entity*)

setId

public void **setId** (Long *id*)

setModifiedByUser

public void **setModifiedByUser** (boolean *modifiedByUser*)

setNonEditable

public void **setNonEditable** (boolean *nonEditable*)

setRecordHistory

public void **setRecordHistory** (boolean *recordHistory*)

toDto

public TrackingDto **toDto** ()

update

public void **update** (TrackingDto *trackingDto*)

12.35.36 Type

public class **Type**

The `Type` class contains information about a single type in MDS. The MDS type can have settings and validations that can be assigned to field with the same type.

Constructors

Type

public **Type** ( )

Type

public **Type** (Class<?> *typeClass*)

Type

public **Type** (String *displayName*, String *description*, Class<?> *typeClass*)

Methods

getDefaultName

public String **getDefaultName** ( )

getDescription

public String **getDescription** ( )

getDisplayName

public String **getDisplayName** ( )

getId

public Long **getId** ( )

getSettings

public List<TypeSetting> **getSettings** ( )

getTypeClass

public Class<?> **getTypeClass** ( )

getTypeClassName

public String **getTypeClassName** ( )

getValidations

public List<TypeValidation> **getValidations**()

hasSettings

public boolean **hasSettings**()

hasValidation

public boolean **hasValidation**()

isBlob

public boolean **isBlob**()

isCombobox

public boolean **isCombobox**()

isMap

public boolean **isMap**()

isRelationship

public boolean **isRelationship**()

isTextArea

public boolean **isTextArea**()

parse

public Object **parse**(String *str*)

setDefaultName

public void **setDefaultName**(String *defaultName*)

setDescription

public void **setDescription**(String *description*)

setDisplayName

public void **setDisplayName** (String *displayName*)

setId

public void **setId** (Long *id*)

setSettings

public void **setSettings** (List<TypeSetting> *settings*)

setTypeClass

public void **setTypeClass** (Class<?> *typeClass*)

setValidations

public void **setValidations** (List<TypeValidation> *validations*)

toDto

public TypeDto **toDto** ()

12.35.37 TypeSetting

public class **TypeSetting**

The `TypeSetting` contains settings for the given MDS type. This class is related with table in database with the same name.

Constructors

TypeSetting

public **TypeSetting** ()

TypeSetting

public **TypeSetting** (String *name*)

Methods

getDefaultValue

public String **getDefaultValue** ()

getId

public Long **getId**()

getName

public String **getName**()

getTypeSettingOptions

public List<TypeSettingOption> **getTypeSettingOptions**()

getValueType

public Type **getValueType**()

setDefaultValue

public void **setDefaultValue**(String *defaultValue*)

setId

public void **setId**(Long *id*)

setName

public void **setName**(String *name*)

setTypeSettingOptions

public void **setTypeSettingOptions**(List<TypeSettingOption> *typeSettingOptions*)

setValueType

public void **setValueType**(Type *valueType*)

12.35.38 TypeSettingOption

public class **TypeSettingOption**

The `TypeSettingOption` contains a single setting option for the given type setting. This class is related with table in database with the same name.

Constructors

TypeSettingOption

public **TypeSettingOption** (String *name*)

Methods

getId

public Long **getId** ()

getName

public String **getName** ()

setId

public void **setId** (Long *id*)

setName

public void **setName** (String *name*)

12.35.39 TypeValidation

public class **TypeValidation**

The `TypeValidation` contains a single validation option for the given type. This class is related with table in database with the same name.

Constructors

TypeValidation

public **TypeValidation** ()

TypeValidation

public **TypeValidation** (String *displayName*, Type *valueType*)

Methods

getAnnotations

public List<Class<? extends Annotation>> **getAnnotations** ()

getDisplayName

public String **getDisplayName**()

getId

public Long **getId**()

getValueType

public Type **getValueType**()

setAnnotations

public void **setAnnotations**(List<Class<? extends Annotation>> *annotations*)

setDisplayName

public void **setDisplayName**(String *displayName*)

setId

public void **setId**(Long *id*)

setValueType

public void **setValueType**(Type *valueType*)

toString

public String **toString**()

12.35.40 UIDisplayFieldComparator

public class **UIDisplayFieldComparator** implements Comparator<Field>

`UIDisplayFieldComparator` compares positions added in UIDisplayable annotation. Fields without annotation are placed at the end.

Methods

compare

public int **compare**(Field *o1*, Field *o2*)

12.36 org.motechproject.mds.dto

12.36.1 AdvancedSettingsDto

public class **AdvancedSettingsDto**

The `AdvancedSettingsDto` contains information about advanced settings of an entity.

Methods

addNewIndex

public void **addNewIndex** (String *lookupName*)

equals

public boolean **equals** (Object *obj*)

{@inheritDoc}

getBrowsing

public BrowsingSettingsDto **getBrowsing** ()

getEntityId

public Long **getEntityId** ()

getId

public Long **getId** ()

getIndexes

public List<LookupDto> **getIndexes** ()

getRestOptions

public RestOptionsDto **getRestOptions** ()

getTracking

public TrackingDto **getTracking** ()

hashCode

public int **hashCode** ( )
{@inheritDoc}

removeIndex

public void **removeIndex** (Integer *idx*)

setBrowsing

public void **setBrowsing** (BrowsingSettingsDto *browsing*)

setEntityId

public void **setEntityId** (Long *entityId*)

setId

public void **setId** (Long *id*)

setIndexes

public void **setIndexes** (List<LookupDto> *indexes*)

setRestOptions

public void **setRestOptions** (RestOptionsDto *restOptions*)

setTracking

public void **setTracking** (TrackingDto *tracking*)

toString

public String **toString** ( )
{@inheritDoc}

12.36.2 BrowsingSettingsDto

public class **BrowsingSettingsDto**
The `BrowsingSettingsDto` contains information about filled browsing settings

Methods

addDisplayedField

public void **addDisplayedField** (Number *id*)

addFilterableField

public void **addFilterableField** (Number *id*)

containsDisplayedField

public boolean **containsDisplayedField** (Long *number*)

containsFilterableField

public boolean **containsFilterableField** (Number *id*)

equals

public boolean **equals** (Object *obj*)
{@inheritDoc}

getDisplayedFields

public List<Number> **getDisplayedFields** ()

getFilterableFields

public List<Number> **getFilterableFields** ()

hashCode

public int **hashCode** ()
{@inheritDoc}

indexOfDisplayedField

public long **indexOfDisplayedField** (Long *id*)

removeFilterableField

public void **removeFilterableField** (Number *id*)

setDisplayedFields

public void **setDisplayedFields** (List<Number> *displayedFields*)

setFilterableFields

public void **setFilterableFields** (List<Number> *filterableFields*)

toString

public String **toString** ()
{@inheritDoc}

12.36.3 CsvImportResults

public class **CsvImportResults** implements Serializable
This class holds results from a CSV import - IDs of updated and created instances.

Constructors

CsvImportResults

public **CsvImportResults** (EntityDto *entity*, List<Long> *newInstanceIDs*, List<Long> *updatedInstanceIDs*)

Parameters

- **entity** – entity for which this import was performed
- **newInstanceIDs** – a list of IDs for instances that were newly created during import
- **updatedInstanceIDs** – a list of IDs for instances that were updated during import

Methods

getEntityClassName

public String **getEntityClassName** ()

Returns the class name of the entity for which this import was performed

getEntityModule

public String **getEntityModule** ()

Returns the name of the module of the entity for which this import was performed

getEntityName

public String **getEntityName**()

Returns the name of the entity for which this import was performed

getEntityNamespace

public String **getEntityNamespace**()

Returns the namespace of the entity for which this import was performed

getNewInstanceIDs

public List<Long> **getNewInstanceIDs**()

Returns a list of IDs for instances that were newly created during import

getUpdatedInstanceIDs

public List<Long> **getUpdatedInstanceIDs**()

Returns a list of IDs for instances that were updated during import

newInstanceCount

public int **newInstanceCount**()

Returns the total number of instances that were newly created during import

totalNumberOfImportedInstances

public int **totalNumberOfImportedInstances**()

Returns the total number of imported instances. The total number of imported instances is the sum of the number of updated instances and the total number of newly created instances. In other words this is the number of affected instances.

Returns total number of imported instances

updatedInstanceCount

public int **updatedInstanceCount**()

Returns the total number of instances that were updated during import

12.36.4 DraftData

public class **DraftData**

The `DraftData` contains information that are used for creating temporary changes in a field.

Fields

ADD_NEW_INDEX

public static final String **ADD_NEW_INDEX**

ADVANCED

public static final String **ADVANCED**

DISPLAY_NAME

public static final String **DISPLAY_NAME**

FIELD

public static final String **FIELD**

FIELD_ID

public static final String **FIELD_ID**

NAME

public static final String **NAME**

PATH

public static final String **PATH**

REMOVE_INDEX

public static final String **REMOVE_INDEX**

SECURITY

public static final String **SECURITY**

TYPE_CLASS

public static final String **TYPE_CLASS**

VALUE

public static final String **VALUE**

Methods

getPath

public String **getPath**()

getValue

public Object **getValue**(String *key*)

getValues

public Map<String, Object> **getValues**()

isCreate

public boolean **isCreate**()

isEdit

public boolean **isEdit**()

isForAdvanced

public boolean **isForAdvanced**()

isForField

public boolean **isForField**()

isForSecurity

public boolean **isForSecurity**()

isRemove

public boolean **isRemove**()

setCreate

public void **setCreate**(boolean *create*)

setEdit

public void **setEdit**(boolean *edit*)

setRemove

public void **setRemove** (boolean *remove*)

setValues

public void **setValues** (Map<String, Object> *values*)

12.36.5 DraftResult

public class **DraftResult** implements Serializable

After users do draft changes an instance of this class is returned. It contains information about the draft state.

Constructors

DraftResult

public **DraftResult** (boolean *changesMade*, boolean *outdated*)

Methods

isChangesMade

public boolean **isChangesMade** ()

isOutdated

public boolean **isOutdated** ()

setChangesMade

public void **setChangesMade** (boolean *changesMade*)

setOutdated

public void **setOutdated** (boolean *outdated*)

12.36.6 DtoHelper

public final class **DtoHelper**

Utility class for managing dto collections.

Methods

asFieldMapById

public static Map<Long, FieldDto> **asFieldMapById** (Collection<FieldDto> *fields*)

Stores fields in a map using id as the key for faster lookup

Parameters

- **fields** – the field collection

Returns a map with field ids being the keys and fields being the values

asFieldMapByName

public static Map<String, FieldDto> **asFieldMapByName** (Collection<FieldDto> *fields*)

Stores fields in a map using name as the key for faster lookup

Parameters

- **fields** – the field collection

Returns a map with field names being the keys and fields being the values

findById

public static FieldDto **findById** (Collection<FieldDto> *fields*, Long *id*)

Looks through a collection of fields, in order to find a field of given id.

Parameters

- **fields** – the field collection
- **id** – id of the field to find

Returns field of the given id or null, if field of given id was not found

findByName

public static FieldDto **findByName** (Collection<FieldDto> *fields*, String *name*)

Looks through a collection of fields, in order to find a field of given name.

Parameters

- **fields** – the field collection
- **name** – name of the field to find

Returns field of the given name or null, if field of given name was not found

12.36.7 EntityDto

public class **EntityDto**

The `EntityDto` class contains only basic information about an entity like id, name, module and namespace.

Constructors

EntityDto

public **EntityDto** ( )

EntityDto

public **EntityDto** (String *className*)

EntityDto

public **EntityDto** (Long *id*, String *className*)

EntityDto

public **EntityDto** (String *className*, SecurityMode *securityMode*, Set<String> *securityMembers*)

EntityDto

public **EntityDto** (Long *id*, String *className*, SecurityMode *securityMode*, Set<String> *securityMembers*)

EntityDto

public **EntityDto** (Long *id*, String *className*, String *module*, SecurityMode *securityMode*, Set<String> *securityMembers*)

EntityDto

public **EntityDto** (Long *id*, String *className*, String *module*, String *namespace*, SecurityMode *securityMode*, Set<String> *securityMembers*)

EntityDto

public **EntityDto** (String *className*, String *name*, String *module*, String *namespace*, SecurityMode *securityMode*, Set<String> *securityMembers*)

EntityDto

public **EntityDto** (Long *id*, String *className*, String *name*, String *module*, String *namespace*, SecurityMode *securityMode*, Set<String> *securityMembers*)

EntityDto

public **EntityDto** (Long *id*, String *className*, String *name*, String *module*, String *namespace*, SecurityMode *securityMode*, Set<String> *securityMembers*, String *superClass*)

EntityDto

public **EntityDto** (Long *id*, String *className*, String *name*, String *module*, String *namespace*, String *tableName*, boolean *recordHistory*, SecurityMode *securityMode*, Set<String> *securityMembers*, SecurityMode *readOnlySecurityMode*, Set<String> *readOnlySecurityMembers*, String *superClass*, boolean *abstractClass*, boolean *securityOptionsModified*)

Methods

checkIfUserHasOnlyReadAccessAuthorization

public boolean **checkIfUserHasOnlyReadAccessAuthorization** ()

equals

public boolean **equals** (Object *obj*)
{@inheritDoc}

getClassName

public String **getClassName** ()

getId

public Long **getId** ()

getMaxFetchDepth

public Integer **getMaxFetchDepth** ()

getModule

public String **getModule** ()

getName

public String **getName** ()

getNamespace

public String **getNamespace** ()

getReadOnlySecurityMembers

public Set<String> **getReadOnlySecurityMembers** ()

getReadOnlySecurityMode

public SecurityMode **getReadOnlySecurityMode**()

getSecurityMembers

public Set<String> **getSecurityMembers**()

getSecurityMode

public SecurityMode **getSecurityMode**()

getSuperClass

public String **getSuperClass**()

getTableName

public String **getTableName**()

hasAccessToEntityFromSecurityMode

public boolean **hasAccessToEntityFromSecurityMode**(SecurityMode *mode*, Set<String> *members*)

hashCode

public int **hashCode**()
{@inheritDoc}

isAbstractClass

public boolean **isAbstractClass**()

isDDE

public boolean **isDDE**()

isModified

public boolean **isModified**()

isNonEditable

public boolean **isNonEditable**()

isOutdated

public boolean **isOutdated**()

isReadOnly

public boolean **isReadOnly**()

isReadOnlyAccess

public boolean **isReadOnlyAccess**()

isRecordHistory

public boolean **isRecordHistory**()

isSecurityOptionsModified

public boolean **isSecurityOptionsModified**()

setAbstractClass

public void **setAbstractClass**(boolean *abstractClass*)

setClassName

public void **setClassName**(String *className*)

setId

public void **setId**(Long *id*)

setMaxFetchDepth

public void **setMaxFetchDepth**(Integer *maxFetchDepth*)

setModified

public void **setModified**(boolean *modified*)

setModule

public void **setModule**(String *module*)

setName

public void **setName** (String *name*)

setNamespace

public void **setNamespace** (String *namespace*)

setNonEditable

public void **setNonEditable** (boolean *nonEditable*)

setOutdated

public void **setOutdated** (boolean *outdated*)

setReadOnly

public void **setReadOnly** (boolean *readOnly*)

setReadOnlyAccess

public void **setReadOnlyAccess** (boolean *readOnlyAccess*)

setReadOnlySecurityMembers

public void **setReadOnlySecurityMembers** (Set<String> *readOnlySecurityMembers*)

setReadOnlySecurityMode

public void **setReadOnlySecurityMode** (SecurityMode *readOnlySecurityMode*)

setRecordHistory

public void **setRecordHistory** (boolean *recordHistory*)

setSecurityMembers

public void **setSecurityMembers** (Set<String> *securityMembers*)

setSecurityMode

public void **setSecurityMode** (SecurityMode *securityMode*)

setSecurityOptionsModified

public void **setSecurityOptionsModified**(boolean *securityOptionsModified*)

setSuperClass

public void **setSuperClass**(String *superClass*)

setTableName

public void **setTableName**(String *tableName*)

toString

public String **toString**()
{@inheritDoc}

12.36.8 FieldBasicDto

public class **FieldBasicDto**
The `FieldBasicDto` contains basic information about a field.

Constructors

FieldBasicDto

public **FieldBasicDto**()

FieldBasicDto

public **FieldBasicDto**(String *displayName*, String *name*)

FieldBasicDto

public **FieldBasicDto**(String *displayName*, String *name*, boolean *required*)

FieldBasicDto

public **FieldBasicDto**(String *displayName*, String *name*, boolean *required*, Object *defaultValue*, String *tooltip*, String *placeholder*)

Methods

equals

public boolean **equals** (Object *obj*)
{@inheritDoc}

getDefaultValue

public Object **getDefaultValue** ()

getDisplayName

public String **getDisplayName** ()

getName

public String **getName** ()

getPlaceholder

public String **getPlaceholder** ()

getTooltip

public String **getTooltip** ()

hashCode

public int **hashCode** ()
{@inheritDoc}

isRequired

public boolean **isRequired** ()

setDefaultValue

public void **setDefaultValue** (Object *defaultValue*)

setDisplayName

public void **setDisplayName** (String *displayName*)

setName

public void **setName** (String *name*)

setPlaceholder

public void **setPlaceholder** (String *placeholder*)

setRequired

public void **setRequired** (boolean *required*)

setTooltip

public void **setTooltip** (String *tooltip*)

toString

public String **toString** ()
{@inheritDoc}

12.36.9 FieldDto

public class **FieldDto**
The `FieldDto` class contains information about an existing field in an entity.

Constructors

FieldDto

public **FieldDto** ()

FieldDto

public **FieldDto** (String *name*, String *displayName*, TypeDto *type*)

FieldDto

public **FieldDto** (String *name*, String *displayName*, TypeDto *type*, boolean *required*)

FieldDto

public **FieldDto** (String *name*, String *displayName*, TypeDto *type*, boolean *required*, Object *defaultValue*)

FieldDto

public **FieldDto** (String *name*, String *displayName*, TypeDto *type*, boolean *required*, Object *defaultValue*, String *tooltip*, String *placeholder*)

FieldDto

public **FieldDto** (Long *id*, Long *entityId*, TypeDto *type*, FieldBasicDto *basic*, boolean *readOnly*, List<MetadataDto> *metadata*, FieldValidationDto *validation*, List<SettingDto> *settings*, List<LookupDto> *lookups*)

FieldDto

public **FieldDto** (Long *id*, Long *entityId*, TypeDto *type*, FieldBasicDto *basic*, boolean *readOnly*, boolean *nonEditable*, boolean *nonDisplayable*, List<MetadataDto> *metadata*, FieldValidationDto *validation*, List<SettingDto> *settings*, List<LookupDto> *lookups*)

FieldDto

public **FieldDto** (Long *id*, Long *entityId*, TypeDto *type*, FieldBasicDto *basic*, boolean *readOnly*, boolean *nonEditable*, boolean *nonDisplayable*, boolean *uiChanged*, List<MetadataDto> *metadata*, FieldValidationDto *validation*, List<SettingDto> *settings*, List<LookupDto> *lookups*)

FieldDto

public **FieldDto** (Long *id*, Long *entityId*, TypeDto *type*, FieldBasicDto *basic*, boolean *readOnly*, FieldValidationDto *validation*)

Methods

addEmptyMetadata

public void **addEmptyMetadata** ()

addMetadata

public void **addMetadata** (MetadataDto *metadata*)

equals

public boolean **equals** (Object *obj*)
{@inheritDoc}

getBasic

public FieldBasicDto **getBasic** ()

getEntityId

public Long **getEntityId**()

getId

public Long **getId**()

getLookups

public List<LookupDto> **getLookups**()

getMetadata

public List<MetadataDto> **getMetadata**()

getMetadata

public MetadataDto **getMetadata**(String *key*)

getSetting

public SettingDto **getSetting**(String *name*)

getSettings

public List<SettingDto> **getSettings**()

getSettingsValueAsString

public String **getSettingsValueAsString**(String *name*)

getType

public TypeDto **getType**()

getValidation

public FieldValidationDto **getValidation**()

hashCode

public int **hashCode**()
{@inheritDoc}

isNonDisplayable

public boolean **isNonDisplayable**()

isNonEditable

public boolean **isNonEditable**()

isReadOnly

public boolean **isReadOnly**()

isUiChanged

public boolean **isUiChanged**()

multiSelect

public boolean **multiSelect**()

removeMetadata

public void **removeMetadata**(Integer *idx*)

setBasic

public void **setBasic**(FieldBasicDto *basic*)

setEntityId

public void **setEntityId**(Long *entityId*)

setId

public void **setId**(Long *id*)

setLookups

public void **setLookups**(List<LookupDto> *lookups*)

setMetadata

public void **setMetadata**(List<MetadataDto> *metadata*)

setNonDisplayable

public void **setNonDisplayable** (boolean *nonDisplayable*)

setNonEditable

public void **setNonEditable** (boolean *nonEditable*)

setReadOnly

public void **setReadOnly** (boolean *readOnly*)

setSettings

public void **setSettings** (List<SettingDto> *settings*)

setType

public void **setType** (TypeDto *type*)

setUiChanged

public void **setUiChanged** (boolean *uiChanged*)

setValidation

public void **setValidation** (FieldValidationDto *validation*)

toString

public String **toString** ()
{@inheritDoc}

12.36.10 FieldInstanceDto

public class **FieldInstanceDto**
The `FieldInstanceDto` class contains information about an existing field in an instance.

Constructors

FieldInstanceDto

public **FieldInstanceDto** ()

FieldInstanceDto

public **FieldInstanceDto** (Long *id*, Long *instanceId*, FieldBasicDto *basic*)

Methods

equals

public boolean **equals** (Object *obj*)
{@inheritDoc}

getBasic

public FieldBasicDto **getBasic** ()

getId

public Long **getId** ()

getInstanceId

public Long **getInstanceId** ()

hashCode

public int **hashCode** ()
{@inheritDoc}

setBasic

public void **setBasic** (FieldBasicDto *basic*)

setId

public void **setId** (Long *id*)

setInstanceId

public void **setInstanceId** (Long *instanceId*)

toString

public String **toString** ()
{@inheritDoc}

12.36.11 FieldValidationDto

public class **FieldValidationDto**
The `FieldValidationDto` class contains information about validation criteria for field.

Fields

DOUBLE

public static final FieldValidationDto **DOUBLE**
Constant `DOUBLE` contains validation criteria for double type.

INTEGER

public static final FieldValidationDto **INTEGER**
Constant `INTEGER` contains validation criteria for integer type.

STRING

public static final FieldValidationDto **STRING**
Constant `STRING` contains validation criteria for string type.

Constructors

FieldValidationDto

public **FieldValidationDto**()

FieldValidationDto

public **FieldValidationDto**(ValidationCriterionDto... *criteria*)

Methods

addCriterion

public void **addCriterion**(ValidationCriterionDto *criterion*)

equals

public boolean **equals**(Object *obj*)
{@inheritDoc}

getCriteria

public List<ValidationCriterionDto> **getCriteria**()

getCriterion

public ValidationCriterionDto **getCriterion** (String *displayName*)

hashCode

public int **hashCode** ()
{@inheritDoc}

setCriteria

public void **setCriteria** (List<ValidationCriterionDto> *criteria*)

toString

public String **toString** ()
{@inheritDoc}

12.36.12 JsonLookupDto

public class **JsonLookupDto**
Contains information about single lookup added via JSON file.

Constructors

JsonLookupDto

public **JsonLookupDto** (String *entityClassName*, String *originLookupName*)

Methods

getEntityClassName

public String **getEntityClassName** ()

getOriginLookupName

public String **getOriginLookupName** ()

setEntityClassName

public void **setEntityClassName** (String *entityClassName*)

setOriginLookupName

public void **setOriginLookupName** (String *originLookupName*)

12.36.13 LookupDto

public class **LookupDto**

The `LookupDto` class contains information about single lookup defined by user

Constructors

LookupDto

public **LookupDto** ()

LookupDto

public **LookupDto** (String *lookupName*, boolean *singleObjectReturn*, boolean *exposedViaRest*)

LookupDto

public **LookupDto** (String *lookupName*, boolean *singleObjectReturn*, boolean *exposedViaRest*, List<LookupFieldDto> *lookupFields*)

LookupDto

public **LookupDto** (String *lookupName*, boolean *singleObjectReturn*, boolean *exposedViaRest*, List<LookupFieldDto> *lookupFields*, boolean *readOnly*)

LookupDto

public **LookupDto** (String *lookupName*, boolean *singleObjectReturn*, boolean *exposedViaRest*, List<LookupFieldDto> *lookupFields*, boolean *readOnly*, String *methodName*, List<String> *fieldsOrder*)

LookupDto

public **LookupDto** (Long *id*, String *lookupName*, boolean *singleObjectReturn*, boolean *exposedViaRest*, List<LookupFieldDto> *lookupFields*, boolean *readOnly*, String *methodName*, List<String> *fieldsOrder*)

Methods

addField

public void **addField** (Long *field*)

addField

public void **addField** (Integer *field*)

equals

public boolean **equals** (Object *o*)
{@inheritDoc}

getFieldsOrder

public List<String> **getFieldsOrder** ()

getId

public Long **getId** ()

getLookupFields

public final List<LookupFieldDto> **getLookupFields** ()

getLookupName

public String **getLookupName** ()

getMethodName

public String **getMethodName** ()

hashCode

public int **hashCode** ()
{@inheritDoc}

insertField

public void **insertField** (Integer *idx*, Integer *fieldId*, String *relatedFieldName*)

insertField

public void **insertField** (Integer *idx*, Long *fieldId*, String *relatedFieldName*)

insertField

public void **insertField** (Integer *idx*, Integer *fieldId*, String *lookupFieldType*, String *relatedFieldName*)

insertField

public void **insertField** (Integer *idx*, Long *fieldId*, String *lookupFieldType*, String *relatedFieldName*)

isExposedViaRest

public boolean **isExposedViaRest** ()

isReadOnly

public boolean **isReadOnly** ()

isReferenced

public boolean **isReferenced** ()

isSingleObjectReturn

public boolean **isSingleObjectReturn** ()

removeField

public void **removeField** (String *name*)

removeField

public void **removeField** (Integer *idx*)

setExposedViaRest

public void **setExposedViaRest** (boolean *isExposedViaRest*)

setFieldsOrder

public void **setFieldsOrder** (List<String> *fieldsOrder*)

setId

public void **setId** (Long *id*)

setLookupFields

public void **setLookupFields** (List<LookupFieldDto> *lookupFields*)

setLookupName

public void **setLookupName** (String *lookupName*)

setMethodName

public void **setMethodName** (String *methodName*)

setReadOnly

public void **setReadOnly** (boolean *readOnly*)

setReferenced

public void **setReferenced** (boolean *referenced*)

setSingleObjectReturn

public void **setSingleObjectReturn** (boolean *singleObjectReturn*)

toString

public String **toString** ()
{@inheritDoc}

updateCustomOperatorForLookupField

public void **updateCustomOperatorForLookupField** (Integer *idx*, String *customOperator*)

updateFieldRelatedName

public void **updateFieldRelatedName** (Integer *idx*, String *relatedName*)

updateTypeForLookupField

public void **updateTypeForLookupField** (Integer *idx*, String *lookupFieldType*)

12.36.14 LookupFieldDto

public class **LookupFieldDto**
Represents a field added to a lookup. The lookup using a given field can be done using multiple lookup types.

Constructors

LookupFieldDto

public **LookupFieldDto** ( )

LookupFieldDto

public **LookupFieldDto** (String *name*, LookupFieldType *type*)

LookupFieldDto

public **LookupFieldDto** (Long *id*, String *name*, LookupFieldType *type*)

LookupFieldDto

public **LookupFieldDto** (String *name*, LookupFieldType *type*, String *customOperator*)

LookupFieldDto

public **LookupFieldDto** (Long *id*, String *name*, LookupFieldType *type*, String *customOperator*)

LookupFieldDto

public **LookupFieldDto** (Long *id*, String *name*, LookupFieldType *type*, String *customOperator*, boolean *useGenericParam*, String *relatedName*)

Methods

equals

public boolean **equals** (Object *o*)

getClassName

public String **getClassName** ( )

getCustomOperator

public String **getCustomOperator** ( )

getDisplayName

public String **getDisplayName** ( )

getId

public Long **getId**()

getLookupFieldName

public String **getLookupFieldName**()

getName

public String **getName**()

getRelatedName

public String **getRelatedName**()

getSettings

public List<SettingDto> **getSettings**()

getType

public LookupFieldType **getType**()

hashCode

public int **hashCode**()

isUseGenericParam

public boolean **isUseGenericParam**()

setClassName

public void **setClassName**(String *className*)

setCustomOperator

public void **setCustomOperator**(String *customOperator*)

setDisplayName

public void **setDisplayName**(String *displayName*)

setId

public void **setId** (Long *id*)

setName

public void **setName** (String *name*)

setRelatedName

public void **setRelatedName** (String *relatedName*)

setSettings

public void **setSettings** (List<SettingDto> *settings*)

setType

public void **setType** (LookupFieldType *type*)

setUseGenericParam

public void **setUseGenericParam** (boolean *useGenericParam*)

12.36.15 LookupFieldType

public enum **LookupFieldType**

The lookup type represents whether the lookup will be done by comparing to a single field, matching values to a range, or matching to a set of values.

Enum Constants

RANGE

public static final LookupFieldType **RANGE**

Lookup field that accepts a range of values, specified by a minimum and maximum values.

SET

public static final LookupFieldType **SET**

Lookup field that accepts a collection of values.

VALUE

public static final LookupFieldType **VALUE**

Single value lookup field.

12.36.16 MetadataDto

public class **MetadataDto** implements Pair<String, String>

The `MetadataDto` contains key and value of a single field metadata.

Constructors

MetadataDto

public **MetadataDto**()

MetadataDto

public **MetadataDto**(String *key*, String *value*)

MetadataDto

public **MetadataDto**(Long *id*, String *key*, String *value*)

Methods

equals

public boolean **equals**(Object *obj*)

{@inheritDoc}

getId

public Long **getId**()

getKey

public String **getKey**()

getValue

public String **getValue**()

hashCode

public int **hashCode**()

{@inheritDoc}

setId

public void **setId**(Long *id*)

setKey

public void **setKey** (String *key*)

setValue

public void **setValue** (String *value*)

toString

public String **toString** ()
{@inheritDoc}

12.36.17 RestOptionsDto

public class **RestOptionsDto**
Class representing rest options of given entity.

Constructors

RestOptionsDto

public **RestOptionsDto** ()

RestOptionsDto

public **RestOptionsDto** (boolean *create*, boolean *read*, boolean *update*, boolean *delete*, boolean *modifiedByUser*)

Methods

addField

public void **addField** (String *name*)

addLookup

public void **addLookup** (String *name*)

containsField

public boolean **containsField** (String *name*)

containsLookup

public boolean **containsLookup** (String *name*)

equals

public boolean **equals** (Object *obj*)
{@inheritDoc}

getFieldNames

public List<String> **getFieldNames** ()

getId

public Long **getId** ()

getLookupNames

public List<String> **getLookupNames** ()

hashCode

public int **hashCode** ()
{@inheritDoc}

isCreate

public boolean **isCreate** ()

isDelete

public boolean **isDelete** ()

isModifiedByUser

public boolean **isModifiedByUser** ()

isRead

public boolean **isRead** ()

isUpdate

public boolean **isUpdate** ()

removeField

public void **removeField** (String *name*)

removeLookup

public void **removeLookup** (String *name*)

setCreate

public void **setCreate** (boolean *create*)

setDelete

public void **setDelete** (boolean *delete*)

setFieldNames

public void **setFieldNames** (List<String> *fieldNames*)

setId

public void **setId** (Long *id*)

setLookupNames

public void **setLookupNames** (List<String> *lookupNames*)

setModifiedByUser

public void **setModifiedByUser** (boolean *modifiedByUser*)

setRead

public void **setRead** (boolean *read*)

setUpdate

public void **setUpdate** (boolean *update*)

supportsAnyOperation

public boolean **supportsAnyOperation** ()

toString

public String **toString**()
{@inheritDoc}

12.36.18 SettingDto

public class **SettingDto** implements Pair<String, Object>
The `SettingDto` contains information about a single setting inside a field.

Constructors

SettingDto

public **SettingDto**()

SettingDto

public **SettingDto**(String *name*, Object *value*)

SettingDto

public **SettingDto**(String *name*, Object *value*, TypeDto *type*, SettingOptions... *options*)

Methods

copy

public SettingDto **copy**()

equals

public boolean **equals**(Object *obj*)
{@inheritDoc}

getKey

public String **getKey**()

getName

public String **getName**()

getOptions

public List<SettingOptions> **getOptions**()

getType

public TypeDto **getType**()

getValue

public Object **getValue**()

getValueAsString

public String **getValueAsString**()

hashCode

public int **hashCode**()
{@inheritDoc}

multiSelect

public boolean **multiSelect**()

setName

public void **setName**(String *name*)

setOptions

public void **setOptions**(List<SettingOptions> *options*)

setType

public void **setType**(TypeDto *type*)

setValue

public void **setValue**(Object *value*)

toString

public String **toString**()
{@inheritDoc}

12.36.19 SettingOptions

public enum **SettingOptions**

The `SettingOptions` contains available options that can be added to field setting.

Enum Constants

POSITIVE

public static final SettingOptions **POSITIVE**

Ensure that a value in a given setting is a number and it has a positive value.

REQUIRE

public static final SettingOptions **REQUIRE**

Force setting a value for a given setting.

12.36.20 TrackingDto

public class **TrackingDto**

The `TrackingDto` contains properties that describe the audit settings of an Entity, such as whether to record history or publish CRUD events for a given Entity.

Constructors

TrackingDto

public **TrackingDto**()

TrackingDto

public **TrackingDto**(boolean *recordHistory*, boolean *allowCreateEvent*, boolean *allowUpdateEvent*, boolean *allowDeleteEvent*, boolean *modifiedByUser*, boolean *nonEditable*)

Methods

equals

public boolean **equals**(Object *obj*)

{@inheritDoc}

hashCode

public int **hashCode**()

{@inheritDoc}

isAllowCreateEvent

public boolean **isAllowCreateEvent** ( )

isAllowDeleteEvent

public boolean **isAllowDeleteEvent** ( )

isAllowUpdateEvent

public boolean **isAllowUpdateEvent** ( )

isModifiedByUser

public boolean **isModifiedByUser** ( )

isNonEditable

public boolean **isNonEditable** ( )

isRecordHistory

public boolean **isRecordHistory** ( )

setAllEvents

public void **setAllEvents** (boolean *value*)

setAllowCreateEvent

public void **setAllowCreateEvent** (boolean *allowCreateEvent*)

setAllowDeleteEvent

public void **setAllowDeleteEvent** (boolean *allowDeleteEvent*)

setAllowUpdateEvent

public void **setAllowUpdateEvent** (boolean *allowUpdateEvent*)

setModifiedByUser

public void **setModifiedByUser** (boolean *modifiedByUser*)

setNonEditable

public void **setNonEditable** (boolean *nonEditable*)

setRecordHistory

public void **setRecordHistory** (boolean *recordHistory*)

toString

public String **toString** ()
{@inheritDoc}

12.36.21 TypeDto

public class **TypeDto**
The `TypeDto` class contains information about an available field in an entity.

Fields

BLOB

public static final TypeDto **BLOB**
Constant `BLOB` is a representation of the MDS BLOB type.

BOOLEAN

public static final TypeDto **BOOLEAN**
Constant `BOOLEAN` is a representation of the MDS Boolean type.

COLLECTION

public static final TypeDto **COLLECTION**
Constant `LIST` is a representation of the MDS Combobox type.

DATE

public static final TypeDto **DATE**
Constant `DATE` is a representation of the MDS Date type.

DATETIME

public static final TypeDto **DATETIME**
Constant `DATETIME` is a representation of the MDS DateTime type.

DOUBLE

public static final TypeDto **DOUBLE**
Constant `DOUBLE` is a representation of the MDS Decimal type.

INTEGER

public static final TypeDto **INTEGER**
Constant `INTEGER` is a representation of the MDS Integer type.

LOCAL_DATE

public static final TypeDto **LOCAL_DATE**
Constant `LOCAL_DATE` is a representation of the `org.joda.time.LocalDate` type.

LONG

public static final TypeDto **LONG**
Constant `LONG` is a representation of the MDS Long type.

MANY_TO_MANY_RELATIONSHIP

public static final TypeDto **MANY_TO_MANY_RELATIONSHIP**

MANY_TO_ONE_RELATIONSHIP

public static final TypeDto **MANY_TO_ONE_RELATIONSHIP**

MAP

public static final TypeDto **MAP**
Constant `MAP` is a representation of the MDS Map type.

ONE_TO_MANY_RELATIONSHIP

public static final TypeDto **ONE_TO_MANY_RELATIONSHIP**

ONE_TO_ONE_RELATIONSHIP

public static final TypeDto **ONE_TO_ONE_RELATIONSHIP**

PERIOD

public static final TypeDto **PERIOD**
Constant `PERIOD` is a representation of the MDS Period type.

STRING

public static final TypeDto **STRING**
Constant `STRING` is a representation of the MDS String type.

TIME

public static final TypeDto **TIME**
Constant `TIME` is a representation of the MDS Time type.

Constructors

TypeDto

public **TypeDto**()

TypeDto

public **TypeDto**(String *displayName*, String *description*, String *defaultName*, String *typeClass*)

TypeDto

public **TypeDto**(Long *id*, String *displayName*, String *description*, String *defaultName*, String *typeClass*)

Methods

equals

public boolean **equals**(Object *obj*)
{@inheritDoc}

getDefaultName

public String **getDefaultName**()

getDescription

public String **getDescription**()

getDisplayName

public String **getDisplayName**()

getId

public Long **getId**()

getTypeClass

public String **getTypeClass**()

hashCode

public int **hashCode**()
{@inheritDoc}

isBlob

public boolean **isBlob**()

isCombobox

public boolean **isCombobox**()

isRelationship

public boolean **isRelationship**()

isTextArea

public boolean **isTextArea**()

setDefaultName

public void **setDefaultName**(String *defaultName*)

setDescription

public void **setDescription**(String *description*)

setDisplayName

public void **setDisplayName**(String *displayName*)

setId

public void **setId**(Long *id*)

setTypeClass

public void **setTypeClass** (String *typeClass*)

toString

public String **toString** ()
{@inheritDoc}

12.36.22 ValidationCriterionDto

public class **ValidationCriterionDto**
The `ValidationCriterionDto` contains information about single criterion for field validation.

Constructors

ValidationCriterionDto

public **ValidationCriterionDto** ()

ValidationCriterionDto

public **ValidationCriterionDto** (String *displayName*, TypeDto *type*)

ValidationCriterionDto

public **ValidationCriterionDto** (String *displayName*, TypeDto *type*, Object *value*, boolean *enabled*)

Methods

equals

public boolean **equals** (Object *obj*)
{@inheritDoc}

getDisplayName

public String **getDisplayName** ()

getType

public TypeDto **getType** ()

getValue

public Object **getValue**()

hashCode

public int **hashCode**()
{@inheritDoc}

isEnabled

public boolean **isEnabled**()

setDisplayName

public void **setDisplayName**(String *displayName*)

setEnabled

public void **setEnabled**(boolean *enabled*)

setType

public void **setType**(TypeDto *type*)

setValue

public void **setValue**(Object *value*)

toString

public String **toString**()
{@inheritDoc}

valueAsString

public String **valueAsString**()

12.37 org.motechproject.mds.enhancer

12.37.1 MdsJDOEnhancer

public class **MdsJDOEnhancer** extends JDOEnhancer

The `MdsJDOEnhancer` class is a wrapper for `org.datanucleus.api.jdo.JDOEnhancer` class. Its task is to add the missing information into created entity class.

Constructors

MdsJDOEnhancer

public **MdsJDOEnhancer** (Properties *config*, ClassLoader *classLoader*)

Methods

addClass

public void **addClass** (ClassData *classData*)

12.38 org.motechproject.mds.event

12.38.1 CrudEventBuilder

public final class **CrudEventBuilder**

The `MDSCrudEvents` class is responsible for creating MDS CRUD events.

Methods

buildEventParams

public static Map<String, Object> **buildEventParams** (String *module*, String *namespace*, String *entity*, String *entityClassName*, Long *id*)

Builds parameters for a Motech CRUD event.

Parameters

- **module** – module name of an entity
- **namespace** – namespace of an entity
- **entity** – entity name
- **entityClassName** – entity class name
- **action** – an action that took place
- **id** – id of the affected instance

Returns constructed parameters for the events

createSubject

public static String **createSubject** (EntityInfo *entity*, String *action*)

Creates subject for a Motech event, sent upon encounter of a CRUD event in MDS.

Parameters

- **entity** – entity information
- **action** – String representation of a CRUD event type

Returns Constructed subject for the event

createSubject

public static String **createSubject** (String *module*, String *namespace*, String *entity*, CrudEventType *action*)

Creates subject for a Motech Event, sent upon encounter of a CRUD event in MDS.

Parameters

- **module** – module name of an entity
- **namespace** – namespace of an entity
- **entity** – entity name
- **action** – CRUD event type

Returns Constructed subject for the Motech Event

createSubject

public static String **createSubject** (String *module*, String *namespace*, String *entity*, String *action*)

Creates subject for a Motech Event, sent upon encounter of a CRUD event in MDS.

Parameters

- **module** – module name of an entity
- **namespace** – namespace of an entity
- **entity** – entity name
- **action** – String representation of a CRUD event type

Returns Constructed subject for the Motech Event

setEntityData

public static void **setEntityData** (Map<String, Object> *params*, String *module*, String *namespace*, String *entityName*, String *entityClassName*)

Sets properties in the given `java.util.Map`.

Parameters

- **params** – a `java.util.Map` to write properties in
- **module** – module name of an entity
- **namespace** – namespace of an entity
- **entityName** – entity name
- **entityClassName** – entity class name

12.38.2 CrudEventType

public enum **CrudEventType**

The `MDSEventsAction` enum represents CRUD operations which send events, this option can be enabled only for entities.

See also: `org.motechproject.mds.annotations.CrudEvents`

Enum Constants

ALL

public static final CrudEventType **ALL**
Represents all CRUD event types.

CREATE

public static final CrudEventType **CREATE**
One of the CRUD event types, representing creating an instance.

DELETE

public static final CrudEventType **DELETE**
One of the CRUD event types, representing deleting an instance.

NONE

public static final CrudEventType **NONE**
Represents zero CRUD event types.

UPDATE

public static final CrudEventType **UPDATE**
One of the CRUD event types, representing updating an instance.

12.39 org.motechproject.mds.ex

12.39.1 JdoListenerInvocationException

public class **JdoListenerInvocationException** extends MdsException
Exception, that signalizes problems invoking method by the JDO lifecycle listener.

Constructors

JdoListenerInvocationException

public **JdoListenerInvocationException** (String *message*)

JdoListenerInvocationException

public **JdoListenerInvocationException** (String *message*, Throwable *cause*)

12.39.2 MdsEntityWireException

public class **MdsEntityWireException** extends MdsException

Exception, that informs about a problem when an entity is outside OSGi exported package. It contains message with problem description and possible solutions.

Fields

SOLUTION_MESSAGE

public static final String **SOLUTION_MESSAGE**

Constructors

MdsEntityWireException

public **MdsEntityWireException** (Throwable *cause*)

12.39.3 MdsException

public class **MdsException** extends RuntimeException

The `MdsException` exception is a basic class for all other exceptions defined in the mds module. It contains information about a message key which will be used on UI to present a message in appropriate language.

Constructors

MdsException

public **MdsException** (String *message*)

MdsException

public **MdsException** (String *message*, Throwable *cause*)

Constructs a new mds exception with the specified message key and params.

Parameters

- **message** – the error message for the logs
- **cause** – the cause of the exception

MdsException

public **MdsException** (String *message*, Throwable *cause*, String *messageKey*)

Constructs a new mds exception with the specified message key and params.

Parameters

- **message** – the error message for the logs
- **cause** – the cause of the exception

- **messageKey** – the message key used later to display message in appropriate language on UI.

MdsException

public **MdsException** (String *message*, Throwable *cause*, String *messageKey*, String *params*)

Constructs a new mds exception with the specified message key and params.

Parameters

- **message** – the error message for the logs
- **cause** – the cause of the exception
- **messageKey** – the message key used later to display message in appropriate language on UI.
- **params** – the params used later to change placeholders in the message

MdsException

public **MdsException** (String *message*, Throwable *cause*, String *messageKey*, String... *params*)

Constructs a new mds exception with the specified message key and params.

Parameters

- **message** – the error message for the logs
- **cause** – the cause of the exception
- **messageKey** – the message key used later to display message in appropriate language on UI.
- **params** – the params used later to change placeholders in the message

Methods

getMessageKey

public String **getMessageKey** ()

Returns the message key used later to display message in appropriate language on UI

getParams

public String **getParams** ()

Returns the params used later to change placeholders in the message

12.39.4 MdsInitializationException

public class **MdsInitializationException** extends MdsException

This exception singals an issue with starting MDS.

Constructors

MdsInitializationException

public **MdsInitializationException** (String *message*, Throwable *cause*)

12.39.5 UserSuppliedComboboxValuesUsedException

public class **UserSuppliedComboboxValuesUsedException** extends MdsException
Exception indicating that user supplied value is used in an instance.

Constructors

UserSuppliedComboboxValuesUsedException

public **UserSuppliedComboboxValuesUsedException** (String *comboboxName*, String *value*)

12.40 org.motechproject.mds.ex.csv

12.40.1 CsvImportException

public class **CsvImportException** extends MdsException
Signals that CSV import failed.

Constructors

CsvImportException

public **CsvImportException** (String *message*)

CsvImportException

public **CsvImportException** (String *message*, Throwable *cause*)

12.40.2 DataExportException

public class **DataExportException** extends MdsException
Signals an error when exporting tabular data.

Constructors

DataExportException

public **DataExportException** (String *message*)

DataExportException

public **DataExportException** (String *message*, Throwable *cause*)

12.41 org.motechproject.mds.ex.entity

12.41.1 DataMigrationFailedException

public class **DataMigrationFailedException** extends MdsException

Thrown when there were some error during combobox data migration.

Constructors

DataMigrationFailedException

public **DataMigrationFailedException** (String *message*, Throwable *cause*)

12.41.2 EntityAlreadyExistException

public class **EntityAlreadyExistException** extends MdsException

The `EntityAlreadyExistException` exception signals a situation in which a user wants to create a new entity with a name that already exists in database.

Constructors

EntityAlreadyExistException

public **EntityAlreadyExistException** (String *entityName*)

Constructs a new EntityAlreadyExistException with *mds.error.entityAlreadyExist* as a message key.

12.41.3 EntityChangedException

public class **EntityChangedException** extends MdsException

This exception signals that an Entity was changed (presumably by another user).

Constructors

EntityChangedException

public **EntityChangedException** ()

12.41.4 EntityCreationException

public class **EntityCreationException** extends MdsException

The `EntityCreationException` exception signals a situation when there were problems with creating new entity class.

Constructors

EntityCreationException

public **`EntityCreationException`** (String *message*)
Constructs a new EntityCreationException with *mds.error.entityBuilderFailure* as a message key.

Parameters

- **message** – the message for the logs

EntityCreationException

public **`EntityCreationException`** (String *message*, Throwable *cause*)
Constructs a new EntityCreationException with *mds.error.entityBuilderFailure* as a message key.

Parameters

- **message** – the message for the logs
- **cause** – the cause of exception.

12.41.5 EntityInfrastructureException

public class **`EntityInfrastructureException`** extends MdsException
The `EntityInfrastructureException` exception signals a situation when there were problems with creating repository/service interface/service class for entity.

Constructors

EntityInfrastructureException

public **`EntityInfrastructureException`** (String *className*, Throwable *cause*)
Constructs a new EntityInfrastructureException with *mds.error.entityInfrastructureFailure* as a message key.

Parameters

- **className** – name of the class building which caused the issue
- **cause** – the cause of exception.

12.41.6 EntityInstancesNonEditableException

public class **`EntityInstancesNonEditableException`** extends RuntimeException
The `EntityInstancesNonEditableException` exception signals a situation in which an user try to edit an instance from nonEditable Entity.

12.41.7 EntityNotFoundException

public class **`EntityNotFoundException`** extends MdsException
The `EntityNotFoundException` exception signals a situation in which an entity with a given id does not exist in database.

Constructors

EntityNotFoundException

public **`EntityNotFoundException`** (String *entityName*)

Constructs a new EntityNotFoundException with *mds.error.entityNotFound* as a message key.

Parameters

- **entityName** – the name of entity not found

EntityNotFoundException

public **`EntityNotFoundException`** (Long *id*)

Constructs a new EntityNotFoundException with *mds.error.entityNotFound* as a message key.

Parameters

- **id** – the id of entity not found

12.41.8 EntityReadOnlyException

public class **`EntityReadOnlyException`** extends MdsException

The `EntityReadOnlyException` exception signals a situation in which a user wants to make changes on an entity which is read only (it was created by a module).

Constructors

EntityReadOnlyException

public **`EntityReadOnlyException`** (String *entityName*)

Constructs a new EntityReadOnlyException with *mds.error.entityIsReadOnly* as a message key.

Parameters

- **entityName** – name of the entity

12.41.9 EntitySchemaMismatchException

public class **`EntitySchemaMismatchException`** extends MdsException

The `EntitySchemaMismatch` exception signals a situation in which a user wants to revert their instance to a version on a different schema version.

Constructors

EntitySchemaMismatchException

public **`EntitySchemaMismatchException`** (String *entityName*)

Constructs a new EntitySchemaMismatch with *mds.error.entitySchemaMismatch* as a message key.

Parameters

- **entityName** – name of the entity

12.41.10 IncompatibleComboboxFieldException

public class **`IncompatibleComboboxFieldException`** extends MdsException

Throw when one of the combobox fields is not compatible (some instances are using multiple values for that field) with single-select.

Constructors

IncompatibleComboboxFieldException

public **`IncompatibleComboboxFieldException`** (String *entityName*, String *fieldName*)

Constructs the exception with *mds.error.comboboxIncompatible* as the message key.

Parameters

- **entityName** – name of the entity
- **fieldName** – name of the field that caused the issue

12.41.11 InvalidEntitySettingsException

public class **`InvalidEntitySettingsException`** extends MdsException

Signals that there were problems with the relation in the data model, due to incorrect entity settings.

Constructors

InvalidEntitySettingsException

public **`InvalidEntitySettingsException`** (String *className*, String *relatedClassName*)

Parameters

- **className** – class name of the entity
- **relatedClassName** – the class name of the related entity

12.41.12 InvalidRelationshipException

public class **`InvalidRelationshipException`** extends MdsException

Signals that there were problems with the relations in the data model.

Constructors

InvalidRelationshipException

public **`InvalidRelationshipException`** (String *relatedClass*, String *fieldClassName*)

Parameters

- **relatedClass** – the name of the related class
- **fieldClassName** – the class name of the field

12.41.13 ReservedKeywordException

public class **ReservedKeywordException** extends MdsException

Signals that field/lookup name is invalid because it is a java keyword.

Constructors

ReservedKeywordException

public **ReservedKeywordException** (String *keyword*)

12.41.14 ServiceNotFoundException

public class **ServiceNotFoundException** extends MdsException

Signals that service for a corresponding entity was not found. This most likely signals an issue with entities bundle.

Constructors

ServiceNotFoundException

public **ServiceNotFoundException** (String *serviceClassName*)

Parameters

- **serviceClassName** – class name of the service that was not found

12.42 org.motechproject.mds.ex.field

12.42.1 FieldNotFoundException

public class **FieldNotFoundException** extends MdsException

This exception signals that a given field was not found for the Entity.

Constructors

FieldNotFoundException

public **FieldNotFoundException** (String *entityClassName*, String *fieldName*)

Parameters

- **entityClassName** – class name of the entity
- **fieldName** – name of the field

FieldNotFoundException

public **`FieldNotFoundException`** (String *entityClassName*, Long *fieldId*)

Parameters

- **entityClassName** – class name of the entity
- **fieldId** – the id of the field

12.42.2 FieldReadOnlyException

public class **`FieldReadOnlyException`** extends MdsException

The `FieldReadOnlyException` exception signals an attempt to edit read only field.

Constructors

FieldReadOnlyException

public **`FieldReadOnlyException`** (String *entityName*, String *fieldName*)

Parameters

- **entityName** – name of the entity
- **fieldName** – name of the readonly field

12.42.3 FieldUsedInLookupException

public class **`FieldUsedInLookupException`** extends MdsException

Exception indicating that a field cannot be removed, since it is used in a lookup.

Constructors

FieldUsedInLookupException

public **`FieldUsedInLookupException`** (String *fieldName*, String *lookupNames*)

Parameters

- **fieldName** – the name of the field
- **lookupNames** – names of the lookups the field is used

12.43 org.motechproject.mds.ex.lookup

12.43.1 CollectionResultFromLookupExpectedException

public class **`CollectionResultFromLookupExpectedException`** extends IllegalLookupReturnTypeException

Signals that a collection result was expected, but the lookup returned a single result.

Constructors

CollectionResultFromLookupExpectedException

public **`CollectionResultFromLookupExpectedException`** (String *lookupName*)

Parameters

- **lookupName** – name of the lookup

12.43.2 IllegalLookupException

public class **`IllegalLookupException`** extends MdsException

Signals that the user defined an illegal lookup.

Constructors

IllegalLookupException

public **`IllegalLookupException`** (String *message*)

12.43.3 IllegalLookupReturnTypeException

public abstract class **`IllegalLookupReturnTypeException`** extends MdsException

Signals that the exception returns one object but we expected a list(or vice-versa).

Constructors

IllegalLookupReturnTypeException

public **`IllegalLookupReturnTypeException`** (String *message*)

12.43.4 LookupExecutionException

public class **`LookupExecutionException`** extends MdsException

Signals that it was not possible to execute a lookup for a given entity.

Constructors

LookupExecutionException

public **`LookupExecutionException`** (Throwable *cause*)

12.43.5 LookupExecutorException

public class **`LookupExecutorException`** extends MdsException

Signals that an error occurred during lookup execution.

Constructors

LookupExecutorException

public **`LookupExecutorException`** (String *message*, Throwable *cause*)

12.43.6 LookupNotFoundException

public class **`LookupNotFoundException`** extends MdsException

The `LookupNotFoundException` exception signals a situation in which a lookup with given id does not exist in database.

Constructors

LookupNotFoundException

public **`LookupNotFoundException`** (String *entityName*, String *lookupName*)

Constructs a new LookupNotFoundException with *mds.error.lookupNotFound* as a message key.

Parameters

- **entityName** – the name of the entity
- **lookupName** – the name of the lookup

LookupNotFoundException

public **`LookupNotFoundException`** (Long *entityId*, String *lookupName*)

Constructs a new LookupNotFoundException with *mds.error.lookupNotFound* as a message key.

Parameters

- **entityId** – the id of the entity
- **lookupName** – the name of the lookup

12.43.7 LookupReadOnlyException

public class **`LookupReadOnlyException`** extends MdsException

The `LookupReadOnlyException` exception signals an attempt to edit read only lookup.

Constructors

LookupReadOnlyException

public **`LookupReadOnlyException`** (String *message*)

12.43.8 LookupReferencedException

public class **`LookupReferencedException`** extends MdsException

The `LookupReferencedException` exception signals a situation in which a lookup is used somewhere

Constructors

LookupReferencedException

public **`LookupReferencedException`** (String *entity*, String *lookups*)

12.43.9 LookupWrongParameterTypeException

public class **`LookupWrongParameterTypeException`** extends MdsException
Signals wrong type of lookup parameter

Constructors

LookupWrongParameterTypeException

public **`LookupWrongParameterTypeException`** (String *message*)

12.43.10 SingleResultFromLookupExpectedException

public class **`SingleResultFromLookupExpectedException`** extends IllegalLookupReturnTypeException
Signals that the lookup returned a Collection, while a single result was expected.

Constructors

SingleResultFromLookupExpectedException

public **`SingleResultFromLookupExpectedException`** (String *lookupName*)

Parameters

- **lookupName** – name of the lookup

12.44 org.motechproject.mds.ex.object

12.44.1 ObjectCreateException

public class **`ObjectCreateException`** extends MdsException
Signals that it was not possible to update object instance from the provided data.

Constructors

ObjectCreateException

public **`ObjectCreateException`** (String *entityName*, Throwable *cause*)

12.44.2 ObjectNotFoundException

public class **`ObjectNotFoundException`** extends MdsException
Signals that the expected object was not found in the database.

Constructors

ObjectNotFoundException

public **`ObjectNotFoundException`** (String *entityName*, Long *id*)

Parameters

- **entityName** – name of the entity
- **id** – id of the object we were unable to retrieve

12.44.3 ObjectReadException

public class **`ObjectReadException`** extends MdsException
Signals that it was not possible to parse the object coming from the database.

Constructors

ObjectReadException

public **`ObjectReadException`** (String *entityName*, Throwable *cause*)

Parameters

- **entityName** – the name of the entity
- **cause** – the cause of the error

ObjectReadException

public **`ObjectReadException`** (Long *entityId*, Throwable *cause*)

Parameters

- **entityId** – the id of the entity
- **cause** – the cause of the error

12.44.4 ObjectUpdateException

public class **`ObjectUpdateException`** extends MdsException
Signals that it was not possible to update object instance from the provided data.

Constructors

ObjectUpdateException

public **ObjectUpdateException** (String *entityName*, Long *id*, Throwable *cause*)

12.44.5 PropertyCopyException

public class **PropertyCopyException** extends MdsException

Thrown when there was a problem with property creation.

Constructors

PropertyCopyException

public **PropertyCopyException** (String *message*, Throwable *cause*)

12.44.6 PropertyCreationException

public class **PropertyCreationException** extends MdsException

Thrown when there was a problem with property creation.

Constructors

PropertyCreationException

public **PropertyCreationException** (String *message*, Throwable *cause*)

12.44.7 RevertFromTrashException

public class **RevertFromTrashException** extends MdsException

Signals an error when reverting an instance from trash.

Constructors

RevertFromTrashException

public **RevertFromTrashException** (String *entityName*, Long *instanceId*, Throwable *cause*)

12.44.8 SecurityException

public class **SecurityException** extends MdsException

The `SecurityException` exception signals a situation in which user wants to perform an operation on objects, they don't have access to.

Constructors

SecurityException

public **SecurityException**()
Constructs a new SecurityException with *mds.error.securityError* as a message key.

12.45 org.motechproject.mds.ex.rest

12.45.1 RestBadBodyFormatException

public class **RestBadBodyFormatException** extends MdsException
Signals that there were errors parsing the class from the provided body.

Constructors

RestBadBodyFormatException

public **RestBadBodyFormatException**(String *message*)

RestBadBodyFormatException

public **RestBadBodyFormatException**(String *message*, Throwable *cause*)

12.45.2 RestEntityNotFoundException

public class **RestEntityNotFoundException** extends MdsException
The `RestEntityNotFoundException` exception signals a situation in which an entity with a given id does not exist in database.

Constructors

RestEntityNotFoundException

public **RestEntityNotFoundException**(String *field*, String *value*)

12.45.3 RestInternalException

public class **RestInternalException** extends MdsException
Signals an internal issue with the REST support.

Constructors

RestInternalException

public **RestInternalException**(String *message*)

RestInternalException

public **`RestInternalException`** (String *message*, Throwable *cause*)

12.45.4 RestLookupExecutionForbiddenException

public class **`RestLookupExecutionForbiddenException`** extends MdsException

Signals that the lookup can not be executed through REST since it is not exposed. Thrown only for existing lookups that are not exposed.

Constructors

RestLookupExecutionForbiddenException

public **`RestLookupExecutionForbiddenException`** (String *lookupName*)

Parameters

- **lookupName** – the name of the lookup

12.45.5 RestLookupNotFoundException

public class **`RestLookupNotFoundException`** extends MdsException

Signals that the lookup requested by REST does not exist.

Constructors

RestLookupNotFoundException

public **`RestLookupNotFoundException`** (String *lookupName*)

Parameters

- **lookupName** – name of the lookup

12.45.6 RestNoLookupResultException

public class **`RestNoLookupResultException`** extends MdsException

Thrown when there was no result for a single-value lookup.

Constructors

RestNoLookupResultException

public **`RestNoLookupResultException`** (String *message*)

12.45.7 RestNotSupportedException

public class **RestNotSupportedException** extends MdsException
Signals that the entity does not support rest.

Constructors

RestNotSupportedException

public **RestNotSupportedException** (String *entityName*, String *moduleName*, String *namespace*)

Methods

getMessage

public String **getMessage** ()

12.45.8 RestOperationNotSupportedException

public class **RestOperationNotSupportedException** extends MdsException
Signals that the given operation is not supported by the given entity.

Constructors

RestOperationNotSupportedException

public **RestOperationNotSupportedException** (String *message*)

12.46 org.motechproject.mds.filter

12.46.1 BooleanFilterValue

public class **BooleanFilterValue** extends FilterValue
Represents boolean values (YES/NO) for filtering data in MDS Data Browser. Provides proper value, param and operator for value.

Constructors

BooleanFilterValue

public **BooleanFilterValue** (String *value*)

Methods

operatorForQueryFilter

public List<String> **operatorForQueryFilter**()

paramTypeForQuery

public String **paramTypeForQuery**()

valueForQuery

public Object **valueForQuery**()

12.46.2 ComboboxFilterValue

public class **ComboboxFilterValue** extends FilterValue

Represents Combobox values used for filtering in MDS Data Browser. Those values are defined by user when new entity is created. Provides proper value, param and operator for value.

Constructors

ComboboxFilterValue

public **ComboboxFilterValue**(String *value*)

Methods

operatorForQueryFilter

public List<String> **operatorForQueryFilter**()

paramTypeForQuery

public String **paramTypeForQuery**()

setMultiSelect

public void **setMultiSelect**()

valueForQuery

public Object **valueForQuery**()

12.46.3 DateFilterValue

public class **DateFilterValue** extends FilterValue

Represents Date values used for filtering data in MDS Data Browser. Provides proper value, param and operator for value.

Constructors

DateFilterValue

public **DateFilterValue** (String *value*)

Methods

operatorForQueryFilter

public List<String> **operatorForQueryFilter** ()

paramTypeForQuery

public String **paramTypeForQuery** ()

valueForQuery

public Object **valueForQuery** ()

12.46.4 Filter

public class **Filter** implements Serializable

Represents a filter on a field.

Constructors

Filter

public **Filter** ()

Filter

public **Filter** (String *field*, FilterValue[] *values*)

Filter

public **Filter** (String *field*, String *value*)

Methods

filterForQuery

public String **filterForQuery**()

filterForQueryAsList

public List<List<String>> **filterForQueryAsList**()

getField

public String **getField**()

getValues

public List<FilterValue> **getValues**()

paramsDeclarationForQuery

public String **paramsDeclarationForQuery**()

paramsDeclarationForQueryAsList

public List<String> **paramsDeclarationForQueryAsList**()

requiresFiltering

public boolean **requiresFiltering**()

setField

public void **setField**(String *field*)

setMultiSelect

public void **setMultiSelect**()

setValues

public void **setValues**(FilterValue[] *type*)

valuesForQuery

public Object[] **valuesForQuery**()

12.46.5 FilterValue

public abstract class **`FilterValue`**
Represents a method of filtering.

Fields

ALL

public static final String **`ALL`**

BOOLEAN_FILTER_VALUES

public static final List<String> **`BOOLEAN_FILTER_VALUES`**

DATE_FILTER_VALUES

public static final List<String> **`DATE_FILTER_VALUES`**

NO

public static final String **`NO`**

PAST_7_DAYS

public static final String **`PAST_7_DAYS`**

THIS_MONTH

public static final String **`THIS_MONTH`**

THIS_YEAR

public static final String **`THIS_YEAR`**

TODAY

public static final String **`TODAY`**

YES

public static final String **`YES`**

Methods

fromString

public static FilterValue **fromString** (String *str*)

getValue

public String **getValue** ()

operatorForQueryFilter

public abstract List<String> **operatorForQueryFilter** ()

paramTypeForQuery

public abstract String **paramTypeForQuery** ()

setValue

public void **setValue** (String *value*)

valueForQuery

public abstract Object **valueForQuery** ()

12.46.6 Filters

public class **Filters**

Represents multiple filters for multiple fields. Responsible for collecting and joining queries, parameters and values.

Constructors

Filters

public **Filters** (Filter[] *filters*)

Filters

public **Filters** (Filter *filter*)

Methods

filterForQuery

public String **filterForQuery**()

paramsDeclarationForQuery

public String **paramsDeclarationForQuery**()

requiresFiltering

public boolean **requiresFiltering**()

setMultiselect

public void **setMultiselect**(List<FieldDto> *fields*)

valuesForQuery

public Object[] **valuesForQuery**()

12.47 org.motechproject.mds.helper

12.47.1 ActionParameterTypeResolver

public final class **ActionParameterTypeResolver**

The `ActionParameterTypeResolver` utility class provides a method that resolves tasks parameter type name based on entity field type.

See also: `org.motechproject.mds.domain.Field`, `org.motechproject.mds.domain.Type`

Methods

resolveType

public static String **resolveType**(Field *field*)

Resolves correct task parameter type, based on the MDS field type.

Parameters

- **field** – MDS field

Returns matching task parameter type

12.47.2 ClassTableName

public final class **ClassTableName**

Util class, that provides methods connected to the table name generation.

Methods

getTableName

public static String **getTableName** (String *table*, String *suffix*)

Builds table name for the underlying database, based on the provided values. Replaces all occurences of hyphen ("-") and space (" ") with the underscore ("_") and makes all characters uppercase.

Parameters

- **table** – the base table name
- **suffix** – suffix to use, after the base name

Returns parsed table name

getTableName

public static String **getTableName** (Entity *entity*)

Builds table name for the underlying database, based on the given entity. Replaces all occurences of hyphen ("-") and space (" ") with the underscore ("_") and makes all characters uppercase.

Parameters

- **entity** – entity to build table name for

Returns parsed table name

getTableName

public static String **getTableName** (Entity *entity*, EntityType *type*)

Builds table name for the underlying database, based on the given entity. Replaces all occurences of hyphen ("-") and space (" ") with the underscore ("_") and makes all characters uppercase.

Parameters

- **entity** – entity to build table name for
- **type** – the type of an entity; will be added to the end of the name, if other than "STANDARD"

Returns parsed table name

getTableName

public static String **getTableName** (String *className*, String *module*, String *namespace*, String *tableName*, EntityType *entityType*)

Builds table name for the underlying database, based on the provided values. Replaces all occurences of hyphen ("-") and space (" ") with the underscore ("_") and makes all characters uppercase.

Parameters

- **className** – fully qualified or simple name of the class
- **module** – entity module (defaults to "MDS")
- **namespace** – namespace of the entity
- **tableName** – base table name; if not empty, this method will simply append the type of an entity to the base name
- **entityType** – the type of the entity; will be added to the end of the name, if other than "STANDARD"

Returns parsed table name

12.47.3 ComboboxDataMigrationHelper

public class **`ComboboxDataMigrationHelper`**

Responsible for migrating data of Combobox fields between correct tables. Transfers data from entity table to Combobox table if selecting multiple values has been allowed. Also migrated data back to entity table if that option has been disallowed.

Methods

migrateComboboxDataIfNecessary

public void **`migrateComboboxDataIfNecessary`**(Entity *parent*, Entity *draft*)

Compares given entity with it's draft and migrates data to proper table if multiple selections were allowed or disallowed.

Parameters

- **parent** – the parent entity
- **draft** – the draft of the parent entity

setPersistenceManagerFactory

public void **`setPersistenceManagerFactory`**(PersistenceManagerFactory *persistenceManagerFactory*)

setSettingsService

public void **`setSettingsService`**(SettingsService *settingsService*)

setSqlDBManager

public void **`setSqlDBManager`**(SqlDBManager *sqlDBManager*)

12.47.4 ComboboxHelper

public final class **`ComboboxHelper`**

Helper class for listing selection type changes.

Methods

comboboxesWithChangedSelectionType

public static Map<String, Boolean> **comboboxesWithChangedSelectionType** (List<Field> *oldFields*, List<Field> *newFields*)

Returns map of fields with changed selection type. Key is fields name and value defines whether field will be using multi-select or not.

Parameters

- **oldFields** – the definitions of the fields before change
- **newFields** – the definitions of the fields after change

Returns the map of fields with changed selection type

12.47.5 DataServiceHelper

public final class **DataServiceHelper**

The `DataServiceHelper` is a helper class that simplifies retrieving Data Service for a given entity.

See also: `org.motechproject.mds.service.MotechDataService`, `org.motechproject.mds.domain.Entity`

Methods

getDataService

public static MotechDataService **getDataService** (BundleContext *bundleContext*, String *entityClass*)

Retrieves `org.motechproject.mds.service.MotechDataService` implementation for the given entity class. It will throw `org.motechproject.mds.ex.entity.ServiceNotFoundException`, in case a service for the given entity class cannot be found.

Parameters

- **bundleContext** – context of a bundle
- **entityClass** – fully qualified class name of an entity

Returns generated `org.motechproject.mds.service.MotechDataService` implementation

getDataService

public static MotechDataService **getDataService** (BundleContext *bundleContext*, Entity *entity*)

Retrieves `org.motechproject.mds.service.MotechDataService` implementation for the given entity. It will throw `org.motechproject.mds.ex.entity.ServiceNotFoundException`, in case a service for the given entity class cannot be found.

Parameters

- **bundleContext** – context of a bundle
- **entity** – entity representation, to retrieve its service for

Returns generated `org.motechproject.mds.service.MotechDataService` implementation

12.47.6 EntityDefaultFieldsHelper

public final class **`EntityDefaultFieldsHelper`**

Helper class, responsible for generating default fields, that are a part of each base entity.

Methods

defaultFields

public static List<FieldDto> **`defaultFields`** (TypeService *typeService*)

12.47.7 EntityHelper

public final class **`EntityHelper`**

The `EntityHelper` class contains useful methods that helps managing entities.

See also: `org.motechproject.mds.domain.Entity`

Methods

addDefaultFields

public static void **`addDefaultFields`** (Entity *entity*, AllTypes *allTypes*)

Adds default fields to entity

Parameters

- **entity** – entity to add fields to
- **allTypes** – types repository

getRelatedEntityClasses

public static List<String> **`getRelatedEntityClasses`** (Entity *entity*)

Retrieves all entity classes that are referenced from entity.

Parameters

- **entity** – entity for which related classes will be retrieved

Returns related entity classes names

removeAdditionalFieldsAndLookups

public static void **`removeAdditionalFieldsAndLookups`** (Entity *entity*)

In case of DDE, removes all fields and lookups that are not part of the original schema (was added later). In case of EUDE, removes all fields and lookups.

Parameters

- **entity** – entity to remove fields and lookups from

12.47.8 EntitySorter

public final class **`EntitySorter`**

The `EntitySorter` is a helper class that allows to sort and validate entities.

Methods

sortByHasARelation

public static List<Entity> **`sortByHasARelation`** (List<Entity> *list*)

Takes a list of entities and sorts them, according to relationships they have. The entities that have uni-directional relationship with another entity, will be moved to the position behind the entity they are related with. The bi-directional relationships are not sorted, moreover if invalid bi-directional relationship is found, an exception is thrown.

Parameters

- **list** – Initial list of entities to sort

Returns List of entities, sorted by relationship

sortByInheritance

public static List<Entity> **`sortByInheritance`** (List<Entity> *list*)

Takes a list of entities and sorts them by the inheritance tree. The entities that extend the Object class or MdsEntity class will be moved to the beggining of the list. After that, the entites that are already present on the list will be added, up the inheritance tree.

Parameters

- **list** – Initial list of entities to sort

Returns List of entities, sorted by inheritance tree

12.47.9 EnumHelper

public final class **`EnumHelper`**

This is a helper class, used while generating enums in MDS.

Methods

prefixEnumValue

public static String **`prefixEnumValue`** (String *value*)

Prefixes enum values, if they start with illegal character (other than letter, dollar sign or underscore

Parameters

- **value** – an enum value

Returns either the same value, if the value is legal, or value prefixed with underscore, if the value is illegal

prefixEnumValues

public static Collection<String> **prefixEnumValues** (Collection<String> *values*)

Prefixes a collection of enum values. For each value in the collection, `prefixEnumValue(java.lang.String)` is called.

Parameters

- **values** – a collection of enum values

Returns a collection of prefixed values

12.47.10 FieldHelper

public final class **FieldHelper**

Utility class handling dynamic setting of field values

Methods

addMetadataForRelationship

public static void **addMetadataForRelationship** (String *typeClass*, Field *field*)

addOrUpdateMetadataForCombobox

public static void **addOrUpdateMetadataForCombobox** (Field *field*)

createMetadataForManyToManyRelationship

public static void **createMetadataForManyToManyRelationship** (Field *field*, String *relatedClass*, String *collectionType*, String *relatedField*, boolean *isOwningSide*)

fieldMapByName

public static Map<String, Field> **fieldMapByName** (Collection<Field> *fields*)

setField

public static void **setField** (Object *current*, String *path*, List *value*)

setMetadataForManyToManyRelationship

public static void **setMetadataForManyToManyRelationship** (Field *field*, boolean *isOwningSide*)

12.47.11 JavassistBuilder

public final class **JavassistBuilder**

Builder class for javassist related tasks. Helps with building appropriate elements of class e.g. fields, getters, field initializer

Methods

createCollectionInitializer

public static CtField.Initializer **createCollectionInitializer**(String *genericType*, Object *defaultValue*)

Creates a collection initializer for the given generic type and default value.

Parameters

- **genericType** – the generic type
- **defaultValue** – the default value

Returns initializer for collections

createEnumInitializer

public static CtField.Initializer **createEnumInitializer**(String *enumType*, String *defaultValue*)

Makes an initializer for enums.

Parameters

- **enumType** – the enum type
- **defaultValue** – the default value

Returns enum initializer

createField

public static CtField **createField**(CtClass *declaring*, CtClass *type*, String *name*, String *genericSignature*)

Creates class field with the given name for the given class declaration and type.

Parameters

- **declaring** – the class to which the field will be added
- **type** – the field type
- **name** – the field name
- **genericSignature** – the generic signature

Throws

- **CannotCompileException** – when bytecode transformation has failed

Returns An instance of `javassist.CtField` represents a field

createGetter

public static CtMethod **`createGetter`** (String *fieldName*, CtClass *declaring*, CtField *field*)

Creates a public getter method with the given field name for the given class declaration and type.

Parameters

- **fieldName** – the field name
- **declaring** – the class to which the getter will be added
- **field** – the field declaration

Throws

- **CannotCompileException** – when bytecode transformation has failed

Returns An instance of `javassist.CtMethod` represents a getter method

createInitializer

public static CtField.Initializer **`createInitializer`** (String *typeClass*, String *defaultValueAsString*)

Creates a field initializer for the given type and default value.

Parameters

- **typeClass** – the field type
- **defaultValueAsString** – the default value for field as string

Returns field initializer

createListInitializer

public static CtField.Initializer **`createListInitializer`** (String *genericType*, Object *defaultValue*)

Creates a list initializer for the given generic type and default value.

Parameters

- **genericType** – the generic type
- **defaultValue** – the default value

Returns initializer for lists

createLocaleInitializer

public static CtField.Initializer **`createLocaleInitializer`** (String *defaultValue*)

Makes an initializer for `java.util.Locale` class.

Parameters

- **defaultValue** – the default value

Returns `java.util.Locale` initializer

createSetInitializer

public static CtField.Initializer **`createSetInitializer`** (String *genericType*, Object *defaultValue*)

Creates a set initializer for the given generic type and default value.

Parameters

- **genericType** – the generic type
- **defaultValue** – the default value

Returns initializer for sets

createSetter

public static CtMethod **`createSetter`** (String *fieldName*, CtField *field*)

Creates a public setter method with the given field name for the given class declaration and type.

Parameters

- **fieldName** – the field name
- **field** – the field declaration

Throws

- **CannotCompileException** – when bytecode transformation has failed

Returns An instance of `javassist.CtMethod` represents a setter method

createSimpleInitializer

public static CtField.Initializer **`createSimpleInitializer`** (String *type*, Object *defaultValue*)

Makes a simple initializer for the given type and default value.

Parameters

- **type** – the field type
- **defaultValue** – the default value

Returns simple initializer

createSimpleInitializer

public static CtField.Initializer **`createSimpleInitializer`** (String *type*, String *defaultValue*)

Makes a simple initializer for the given type and default value.

Parameters

- **type** – the field type
- **defaultValue** – the default value as string

Returns simple initializer

12.47.12 MdsBundleHelper

public final class **MdsBundleHelper**

Helper class, that provides utility methods for MDS OSGi bundles.

Methods

findMdsBundle

public static Bundle **findMdsBundle** (BundleContext *bundleContext*)

findMdsEntitiesBundle

public static Bundle **findMdsEntitiesBundle** (BundleContext *bundleContext*)

isBundleMdsDependent

public static boolean **isBundleMdsDependent** (Bundle *bundle*)

isFrameworkBundle

public static boolean **isFrameworkBundle** (Bundle *bundle*)

isMdsBundle

public static boolean **isMdsBundle** (Bundle *bundle*)

isMdsClassLoader

public static boolean **isMdsClassLoader** (ClassLoader *classLoader*)

isMdsEntitiesBundle

public static boolean **isMdsEntitiesBundle** (Bundle *bundle*)

unregisterBundleJDOClasses

public static void **unregisterBundleJDOClasses** (Bundle *bundle*)

Unregisters all entity classes registered to JDO that are accessible from bundle class loader. This method should be called after bundle that registers MDS entities gets unresolved, so that they are removed from JDO cache. Not doing this might produce hard to track exception when refreshing MDS Entities Bundle after bundle removal.

Parameters

- **bundle** – the bundle for which entity classes are to be unregistered

12.47.13 RelationshipResolver

public class **RelationshipResolver**

The `RelationshipResolver` class provides a method that removes unresolved entities from a set. Entity is considered unresolved if at least one of its dependencies is not contained in provided set or does not exist in database.

See also: `org.motechproject.mds.domain.Entity`

Methods

removeUnresolvedEntities

public List<Entity> **removeUnresolvedEntities** (Set<Entity> *entities*)

setAllEntities

public void **setAllEntities** (AllEntities *allEntities*)

12.47.14 RelationshipSorter

public class **RelationshipSorter**

The `RelationshipSorter` class provides method that sorts given entities list using dependency ordering. It means, that if entity A depends on entity B, B will be before A in sorted list. In case of any circular dependencies, entities contained in the cycle are considered equal, thus their mutual positions are unspecified.

Methods

sort

public void **sort** (List<Entity> *entities*)

12.48 org.motechproject.mds.javassist

12.48.1 JavassistLoader

public class **JavassistLoader** extends Loader<ClassData>

The `JavassistLoader` is a implementation of the `org.motechproject.mds.util.Loader` interface. It takes class information from instance of `org.motechproject.mds.domain.ClassData` and the missing classes are taken from `org.motechproject.mds.javassist.MotechClassPool`

See also: `org.motechproject.mds.util.Loader`, `org.motechproject.mds.domain.ClassData`, `org.motechproject.mds.javassist.MotechClassPool`

Constructors

JavassistLoader

public **JavassistLoader** (MDSClassLoader *classLoader*)

Methods

doWhenClassNotFound

public void **doWhenClassNotFound** (String *name*)

getClassDefinition

public Class<?> **getClassDefinition** (ClassData *data*)

loadClass

public Class<?> **loadClass** (ClassData *arg*)

12.48.2 MotechClassPool

public final class **MotechClassPool**

This class holds the javasisst classpool, enriched by motech classes. All predefined additions to the ClassPool should take place here. The classpool should also be retrieved using this class, in order to be sure that the a initialization took place.

Methods

clearEnhancedData

public static void **clearEnhancedData** ()

getDefault

public static ClassPool **getDefault** ()

getEnhancedClassData

public static ClassData **getEnhancedClassData** (String *className*)

getEnhancedClasses

public static Collection<ClassData> **getEnhancedClasses** (boolean *includeInerfaces*)

getHistoryClassData

public static ClassData **getHistoryClassData** (String *className*)

getInterfaceName

public static String **getInterfaceName** (String *className*)

getRepositoryName

public static String **getRepositoryName** (String *className*)

getServiceImplName

public static String **getServiceImplName** (String *className*)

getTrashClassData

public static ClassData **getTrashClassData** (String *className*)

isDDEReady

public static boolean **isDDEReady** (String *className*)

isServiceInterfaceRegistered

public static boolean **isServiceInterfaceRegistered** (String *className*)

registerDDE

public static void **registerDDE** (String *className*)

registerEnhancedClassData

public static void **registerEnhancedClassData** (ClassData *enhancedClassData*)

registerEnum

public static void **registerEnum** (String *enumName*)

registerHistoryClassData

public static void **registerHistoryClassData** (ClassData *cData*)

registerServiceInterface

public static void **registerServiceInterface** (String *className*, String *interfaceName*)

registerTrashClassData

public static void **registerTrashClassData** (ClassData *cData*)

registeredEnums

public static Collection<String> **registeredEnums** ()

registeredInterfaces

public static Collection<String> **registeredInterfaces** ()

unregisterEnhancedData

public static void **unregisterEnhancedData** (String *className*)

12.49 org.motechproject.mds.jdo

12.49.1 AbstractObjectValueGenerator

public abstract class **AbstractObjectValueGenerator**<T> implements ObjectValueGenerator

Base class for other generator classes. It takes value of property (see `getPropertName()` method) from object and modify it depending on the implementation. If the modified value is null then the `java.lang.IllegalStateException` is thrown.

Parameters

- **<T>** – type of property

Methods

generate

public Object **generate** (ExecutionContext *ec*, Object *obj*, ExtensionMetaData[] *extensions*)

getPropertName

protected abstract String **getPropertName** ()

modify

protected abstract T **modify** (T *value*)

Modifies the given value. This method cannot return null value.

Parameters

- **value** – the given value related with property.

Returns modified value.

12.49.2 CreationDateValueGenerator

public class **CreationDateValueGenerator** extends DateTimeValueGenerator

The `CreationDateValueGenerator` class is responsible for generating value for `org.motechproject.mds.util.Constants.Util.CREATION_DATE_FIELD_NAME` field.

Methods

getPropertName

protected String **getPropertName** ()

12.49.3 CreatorValueGenerator

public class **CreatorValueGenerator** extends UsernameValueGenerator

The `CreatorValueGenerator` class is responsible for generating value for `org.motechproject.mds.util.Constants.Util.CREATOR_FIELD_NAME` field.

Methods

getPropertName

protected String **getPropertName** ()

12.49.4 DateTimeValueGenerator

public abstract class **DateTimeValueGenerator** extends AbstractObjectValueGenerator<DateTime>

The `DateTimeValueGenerator` class modifies properties with `org.joda.time.DateTime` type. If the given value is null then the current time is returned; otherwise the given value is returned.

Methods

modify

protected DateTime **modify** (DateTime *value*)

12.49.5 MDSClassLoaderResolver

public class **MDSClassLoaderResolver** implements ClassLoaderResolver

This is a wrapper for `org.motechproject.mds.jdo.MDSClassLoaderResolverImpl`. All calls for the `org.datanucleus.ClassLoaderResolver` interface are passed to the current instance of the ClassLoaderResolver implementation. When we hit a NullPointerException originating in Felix, we can determine it is due to a synchronization bug after bundle updates - as a result of this DataNucleus has passed us ClassLoaders from the former Bundle version. In that case we reload the instance passing it the ClassLoaders from the new bundle.

Constructors

MDSClassLoaderResolver

public **MDSClassLoaderResolver** ()

MDSClassLoaderResolver

public **MDSClassLoaderResolver** (ClassLoader *pmLoader*)

Methods

classForName

public Class **classForName** (String *name*, ClassLoader *primary*)

classForName

public Class **classForName** (String *name*, ClassLoader *primary*, boolean *initialize*)

classForName

public Class **classForName** (String *name*)

classForName

public Class **classForName** (String *name*, boolean *initialize*)

getResource

public URL **getResource** (String *resourceName*, ClassLoader *primary*)

getResources

public Enumeration<URL> **getResources** (String *resourceName*, ClassLoader *primary*)

isAssignableFrom

public boolean **isAssignableFrom**(String *className*, Class *clazz*)

isAssignableFrom

public boolean **isAssignableFrom**(Class *clazz*, String *className*)

isAssignableFrom

public boolean **isAssignableFrom**(String *className1*, String *className2*)

registerUserClassLoader

public void **registerUserClassLoader**(ClassLoader *loader*)

setPrimary

public void **setPrimary**(ClassLoader *primary*)

setRuntimeClassLoader

public void **setRuntimeClassLoader**(ClassLoader *loader*)

unsetPrimary

public void **unsetPrimary**()

12.49.6 MDSClassLoaderResolverImpl

class **MDSClassLoaderResolverImpl** extends ClassLoaderResolverImpl

The main purpose of the `MDSClassLoaderResolverImpl` class is to avoid situation in which standard datanucleus class loader resolver does not see classes that are saved in database. This is the main implementation that extends the standard ClassLoaderResolverImpl from datanucleus. Due to a synchronization bug in Felix, there are cases when we will instantiate this more then once (after we hit the bug).

Constructors

MDSClassLoaderResolverImpl

public **MDSClassLoaderResolverImpl**()

MDSClassLoaderResolverImpl

public **MDSClassLoaderResolverImpl**(ClassLoader *pmLoader*)

Methods

classForName

public Class **classForName** (String *name*, ClassLoader *primary*)

12.49.7 MdsIgnoreAnnotationHandler

public class **MdsIgnoreAnnotationHandler** implements MemberAnnotationHandler

The `MdsIgnoreAnnotationHandler` provides a mechanism to handle entity fields annotated with @Ignore at the DataNucleus level, so that there is no database column created for that field.

See also: `org.motechproject.mds.annotations.Ignore`

Methods

processMemberAnnotation

public void **processMemberAnnotation** (AnnotationObject *annotation*, AbstractMemberMetaData *mmd*, ClassLoaderResolver *clr*)

12.49.8 MdsJdoAnnotationReader

public class **MdsJdoAnnotationReader** extends JDOAnnotationReader

MDS JDO annotation reader, extends the regular `org.datanucleus.api.jdo.metadata.JDOAnnotationReader` This class was introduced because `org.datanucleus.api.jdo.metadata.JDOAnnotationReader` would not read field annotations for metadata if there was no class level JDO annotations. This extension will recognize the `org.motechproject.mds.annotations.Entity` annotation as an annotation indicating that the class is persistence capable.

Constructors

MdsJdoAnnotationReader

public **MdsJdoAnnotationReader** (MetaDataManager *mgr*)

Methods

isClassPersistable

protected AnnotationObject **isClassPersistable** (Class *cls*)

12.49.9 MdsJdoDialect

public class **MdsJdoDialect** extends DefaultJdoDialect

This is an extensions of Springs default JDO dialect that allows controlling the transaction serialization level per transaction with Spring transactions. This was fixed in newer versions of Spring and this class should get removed once we upgrade to a newer Spring version.

Constructors

MdsJdoDialect

public **MdsJdoDialect** (Object *connectionFactory*)

Methods

beginTransaction

public Object **beginTransaction** (Transaction *transaction*, TransactionDefinition *definition*)

getJdoIsolationLevel

protected String **getJdoIsolationLevel** (TransactionDefinition *definition*)

12.49.10 MdsLongVarBinaryRDBMSMapping

public class **MdsLongVarBinaryRDBMSMapping** extends LongVarBinaryRDBMSMapping

Mapping of a LONGVARBINARY RDBMS type. This implementation will try use context class loader, joda -time bundle class loader and mds-entities bundle class loader for resolving classes during object deserialisation when errors occur when using the default implementation.

Constructors

MdsLongVarBinaryRDBMSMapping

public **MdsLongVarBinaryRDBMSMapping** (JavaTypeMapping *mapping*, RDBMSStoreManager *storeMgr*, Column *col*)

Methods

getObjectForBytes

protected Object **getObjectForBytes** (byte[] *bytes*, int *param*)

12.49.11 MdsTransactionManager

public class **MdsTransactionManager** extends JdoTransactionManager

We override springs transaction for classloader control. We store context classloaders as thread local variables, and switch them with the bundle classloader for the transaction. Since we only allow operations in transactions, this entry point for classloader switching is enough.

Methods

doBegin

protected void **doBegin** (Object *transaction*, TransactionDefinition *definition*)

doCleanupAfterCompletion

protected void **doCleanupAfterCompletion** (Object *transaction*)

getBundleClassLoader

public ClassLoader **getBundleClassLoader** ()

setBundleClassLoader

public void **setBundleClassLoader** (ClassLoader *bundleClassLoader*)

setBundleContext

public void **setBundleContext** (BundleContext *bundleContext*)

12.49.12 ModificationDateValueGenerator

public class **ModificationDateValueGenerator** extends DateTimeValueGenerator

The `ModificationDateValueGenerator` class is responsible for generating value for `org.motechproject.mds.util.Constants.Util.MODIFICATION_DATE_FIELD_NAME` field.

Methods

getPropertName

protected String **getPropertName** ()

12.49.13 ModifiedByValueGenerator

public class **ModifiedByValueGenerator** extends UsernameValueGenerator

The `ModifiedByValueGenerator` class is responsible for generating value for `org.motechproject.mds.util.Constants.Util.MODIFIED_BY_FIELD_NAME` field.

Methods

getPropertName

protected String **getPropertName** ()

12.49.14 OwnerValueGenerator

public class **OwnerValueGenerator** extends UsernameValueGenerator

The `OwnerValueGenerator` class is responsible for generating value for `org.motechproject.mds.util.Constants.Util.OWNER_FIELD_NAME` field.

Methods

getPropertName

protected String **getPropertName**()

12.49.15 SchemaGenerator

public class **SchemaGenerator** implements InitializingBean

The schema generator class is responsible for generating the table schema for entities and for running entities migrations upon start. Schema for all entity classes has to be generated, otherwise issues might arise in foreign key generation for example. This code runs in the generated entities bundle.

Fields

CONNECTION_DRIVER_KEY

public static final String **CONNECTION_DRIVER_KEY**

CONNECTION_URL_KEY

public static final String **CONNECTION_URL_KEY**

CONNECTION_USER_NAME_KEY

public static final String **CONNECTION_USER_NAME_KEY**

CONNECTION_USER_PASSWORD_KEY

public static final String **CONNECTION_USER_PASSWORD_KEY**

Constructors

SchemaGenerator

public **SchemaGenerator**(JDOPersistenceManagerFactory *persistenceManagerFactory*)

Methods

afterPropertiesSet

public void **afterPropertiesSet** ()

generateSchema

public void **generateSchema** ()

runMigrations

public void **runMigrations** ()

12.49.16 TimeTypeConverter

public class **TimeTypeConverter** implements TypeConverter<Time, String>

This is datanucleus type converter we plug in. It is responsible for converting `org.motechproject.commons.date.model.Time` instances to Strings which are persisted.

Methods

toDatastoreType

public String **toDatastoreType** (Time *memberValue*)

toMemberType

public Time **toMemberType** (String *datastoreValue*)

12.49.17 UsernameValueGenerator

public abstract class **UsernameValueGenerator** extends AbstractObjectValueGenerator<String>

The `UsernameValueGenerator` class modifies properties with `java.lang.String` type. The given value is returned without any change if it is not blank. Otherise the class tries to get current logged user name. If the user exists and name is not blank then this name is returned otherwise the empty string is returned.

Methods

modify

protected String **modify** (String *value*)

12.50 org.motechproject.mds.listener

12.50.1 MotechLifecycleListener

public class **MotechLifecycleListener**

Contains essential information about InstanceLifecycleListeners, which are provided by `org.motechproject.mds.annotations.InstanceLifecycleListener` annotation.

See also: `org.motechproject.mds.annotations.InstanceLifecycleListener`

Constructors

MotechLifecycleListener

public **MotechLifecycleListener** (Class *service*, String *methodName*, String *parameterType*, String *packageName*, InstanceLifecycleListenerType[] *types*, List<String> *entityNames*)

Methods

addMethod

public void **addMethod** (Map<InstanceLifecycleListenerType, Set<String>> *method*)

equals

public boolean **equals** (Object *obj*)

getEntityNames

public List<String> **getEntityNames** ()

getMethodsByType

public Map<InstanceLifecycleListenerType, Set<String>> **getMethodsByType** ()

getPackageName

public String **getPackageName** ()

getParameterType

public String **getParameterType** ()

getService

public Class **getService** ()

hashCode

public int **hashCode** ()

setEntityNames

public void **setEntityNames** (List<String> *entityNames*)

setPackageName

public void **setPackageName** (String *packageName*)

12.51 org.motechproject.mds.listener.proxy

12.51.1 ProxyJdoListener

public class **ProxyJdoListener** implements CreateLifecycleListener, StoreLifecycleListener, DeleteLifecycleListener, LoadLifecy

The `ProxyJdoListener` is a listener for persistence events. This listener is for a set of defined classes which are registered in DataNucleus by `org.motechproject.mds.listener.register.JdoListenerRegister`. It is responsible for invoking methods, which are annotated by `org.motechproject.mds.annotations.InstanceLifecycleListene`

See also: `org.motechproject.mds.listener.register.JdoListenerRegister`, `org.motechproject.mds.annotations.InstanceLifecycleListener`

Constructors

ProxyJdoListener

public **ProxyJdoListener** ()

Methods

postCreate

public void **postCreate** (InstanceLifecycleEvent *event*)

postDelete

public void **postDelete** (InstanceLifecycleEvent *event*)

postLoad

public void **postLoad** (InstanceLifecycleEvent *event*)

postStore

public void **postStore** (InstanceLifecycleEvent *event*)

preDelete

public void **preDelete** (InstanceLifecycleEvent *event*)

preStore

public void **preStore** (InstanceLifecycleEvent *event*)

12.52 org.motechproject.mds.listener.register

12.52.1 EntitiesClassListLoader

public final class **EntitiesClassListLoader**

Utility classes used in the MDS Entities Bundle for reading the txt files containing class names.

Methods

entities

public static Set<String> **entities** ()

entitiesStr

public static String **entitiesStr** ()

entitiesWithHistory

public static Set<String> **entitiesWithHistory** ()

entitiesWithHistoryStr

public static String **entitiesWithHistoryStr** ()

entitiesWithListener

public static Set<String> **entitiesWithListener** ()

entitiesWithListenerStr

public static String **entitiesWithListenerStr** ( )

12.52.2 JdoListenerRegister

public class **JdoListenerRegister**

This class adds listener entries to the properties passed to DataNucleus. The classes for which listeners will be called are read from the entities file.

Methods

addJdoListener

public Properties **addJdoListener** (Properties *properties*)

12.53 org.motechproject.mds.lookup

12.53.1 EntityLookups

public class **EntityLookups**

Stores information about lookups and classname of the related entity.

Methods

getEntityClassName

public String **getEntityClassName** ( )

getLookups

public List<LookupDto> **getLookups** ( )

setEntityClassName

public void **setEntityClassName** (String *entityClassName*)

setLookups

public void **setLookups** (List<LookupDto> *lookups*)

12.53.2 LookupExecutor

public class **LookupExecutor**

This class allows executing lookups by providing the lookup name as a string and the lookup params in name-value map. Used both by the REST api and the Databrowser UI for executing lookups based on only metadata. The dataservice and metadata must be provided during construction.

Constructors

LookupExecutor

public **LookupExecutor** (MotechDataService *dataService*, LookupDto *lookup*, Map<String, FieldDto> *fieldsByName*)

Methods

execute

public Object **execute** (Map<String, ?> *lookupMap*)

execute

public Object **execute** (Map<String, ?> *lookupMap*, QueryParams *queryParams*)

executeCount

public long **executeCount** (Map<String, ?> *lookupMap*)

12.54 org.motechproject.mds.performance.domain

12.54.1 Sample

public class **Sample**

Constructors

Sample

public **Sample** ()

Sample

public **Sample** (Integer *testInt*, String *testString*)

Methods

getTestInt

public Integer **getTestInt**()

getTestString

public String **getTestString**()

setTestInt

public void **setTestInt**(Integer *testInt*)

setTestString

public void **setTestString**(String *testString*)

12.55 org.motechproject.mds.performance.service

12.55.1 MdsDummyDataGenerator

public interface **MdsDummyDataGenerator**

Methods

clearEntities

void **clearEntities**()

generateDummyEntities

void **generateDummyEntities**(int *numberOfEntities*, boolean *regenerateBundle*)

Generate a given amount of empty entities

Parameters

- **numberOfEntities** – How many entities should be generated
- **regenerateBundle** – Should the data bundle be regenerated after entities are created

Throws

- **IOException** –

generateDummyEntities

void **generateDummyEntities**(int *numberOfEntities*, int *fieldsPerEntity*, int *lookupsPerEntity*, boolean *regenerateBundle*)

Generate a given amount of entities

Parameters

- **numberOfEntities** – How many entities should be generated
- **fieldsPerEntity** – How many fields should each entity have - type of field is assigned randomly
- **lookupsPerEntity** – How many lookups should each entity have
- **regenerateBundle** – Should the data bundle be regenerated after entities are created

Throws

- **IOException** –

generateDummyInstances

void **generateDummyInstances**(Long *entityId*, int *numberOfInstances*)

Generates a given amount of instances. For most common field types, values will be automatically assigned.

Parameters

- **entityId** – Id of an entity, for which we want to generate instances
- **numberOfInstances** – How many instances should be generated

Throws

- **IllegalAccessException** –
- **ClassNotFoundException** –
- **InstantiationException** –

generateDummyInstances

void **generateDummyInstances**(Long *entityId*, int *numberOfInstances*, int *numberOfHistoricalRevisions*, int *numberOfTrashInstances*)

Generates a given amount of instances, historical revisions and deleted instances. For most common field types, values will be automatically assigned.

Parameters

- **entityId** – Id of an entity, for which we want to generate instances
- **numberOfInstances** – How many instances should be generated
- **numberOfHistoricalRevisions** – How many historical revisions should be generated
- **numberOfTrashInstances** – How many instances in trash should be generated

Throws

- **IllegalAccessException** –
- **ClassNotFoundException** –
- **InstantiationException** –

getEntityPrefix

String **getEntityPrefix**()

getFieldPrefix

String **getFieldPrefix**()

getLookupPrefix

String **getLookupPrefix**()

getService

MotechDataService **getService**(BundleContext *bundleContext*, String *className*)

makeDummyInstance

Object **makeDummyInstance**(Long *entityId*)

Makes a dummy instance for a given entity. It won't insert the created instance into the database, so it can be used to prepare a large dataset of random instances (for example, to insert them manually and measure time)

Parameters

- **entityId** – Id of an entity, for which we want to generate instance

Throws

- **IllegalAccessException** –
- **ClassNotFoundException** –
- **InstantiationException** –

Returns Generated instance, with randomly assigned fields

setEntityPrefix

void **setEntityPrefix**(String *entityPrefix*)

Sets a prefix for the generated entities. Defaults to "Entity"

Parameters

- **entityPrefix** – The prefix for generated entities

setFieldPrefix

void **setFieldPrefix**(String *fieldPrefix*)

Sets the prefix for the generated fields. Defaults to "field"

Parameters

- **fieldPrefix** – The prefix for generated fields

setLookupPrefix

void **setLookupPrefix** (String *lookupPrefix*)

Sets a prefix for the generated lookups. Defaults to "Lookup"

Parameters

- **lookupPrefix** – The prefix for generated lookups

12.55.2 SampleService

public interface **SampleService** extends MotechDataService<Sample>

12.56 org.motechproject.mds.performance.service.impl

12.56.1 MdsDummyDataGeneratorImpl

public class **MdsDummyDataGeneratorImpl** implements MdsDummyDataGenerator

The `MdsDummyDataGenerator` class, allows developers and testers to create any amount of dummy entities and instances in MDS.

Constructors

MdsDummyDataGeneratorImpl

public **MdsDummyDataGeneratorImpl** (EntityService *entityService*, JarGeneratorService *jarGeneratorService*, BundleContext *bundleContext*)

Methods

clearEntities

public void **clearEntities** ()

generateDummyEntities

public void **generateDummyEntities** (int *number*, boolean *regenerateBundle*)

generateDummyEntities

public void **generateDummyEntities** (int *numberOfEntities*, int *fieldsPerEntity*, int *lookupsPerEntity*, boolean *regenerateBundle*)

generateDummyInstances

public void **generateDummyInstances** (Long *entityId*, int *numberOfInstances*)

generateDummyInstances

public void **generateDummyInstances** (Long *entityId*, int *numberOfInstances*, int *numberOfHistoricalRevisions*, int *numberOfTrashInstances*)

getEntityPrefix

public String **getEntityPrefix** ()

getFieldPrefix

public String **getFieldPrefix** ()

getLookupPrefix

public String **getLookupPrefix** ()

getService

public MotechDataService **getService** (BundleContext *bundleContext*, String *className*)

makeDummyInstance

public Object **makeDummyInstance** (Long *entityId*)

setEntityPrefix

public void **setEntityPrefix** (String *entityPrefix*)

setFieldPrefix

public void **setFieldPrefix** (String *fieldPrefix*)

setLookupPrefix

public void **setLookupPrefix** (String *lookupPrefix*)

12.57 org.motechproject.mds.query

12.57.1 AbstractCollectionBasedProperty

public abstract class **AbstractCollectionBasedProperty**<T extends Collection> extends Property<T>

Base for collection based properties

Parameters

- **<T>** – type of the underlying collection

Constructors

AbstractCollectionBasedProperty

protected **`AbstractCollectionBasedProperty`**(String *name*, T *value*, String *type*)

AbstractCollectionBasedProperty

protected **`AbstractCollectionBasedProperty`**(String *jdoVariableName*, String *name*, T *value*, String *type*)

Methods

generateDeclareParameter

public CharSequence **`generateDeclareParameter`**(int *idx*)

shouldIgnoreThisProperty

protected boolean **`shouldIgnoreThisProperty`**()

unwrap

public Collection **`unwrap`**()

12.57.2 CollectionProperty

public class **`CollectionProperty`** extends AbstractCollectionBasedProperty<Collection>

The `CollectionProperty` class represent a property that will be used in JDO query and it has to have the given value(s).

Constructors

CollectionProperty

public **`CollectionProperty`**(String *name*, Object *value*, String *type*)

CollectionProperty

public **`CollectionProperty`**(String *name*, Object *value*, Collection *collectionType*, String *type*)

CollectionProperty

public **CollectionProperty**(String *jdoVariable*, String *name*, Object *value*, Collection *collectionType*, String *type*)

Methods

generateFilter

public CharSequence **generateFilter**(int *idx*)

12.57.3 CustomOperatorProperty

public class **CustomOperatorProperty**<T> extends Property<T>

The `CustomOperatorProperty` class represents a property that will be used in JDO query. This class allows inserting a custom operator, such as `>, <=, matches(), etc.`

Parameters

- **<T>** – type of the passed value

Constructors

CustomOperatorProperty

public **CustomOperatorProperty**(String *name*, T *value*, String *type*, String *operator*)

CustomOperatorProperty

public **CustomOperatorProperty**(String *jdoVariableName*, String *name*, T *value*, String *type*, String *operator*)

Methods

generateFilter

public CharSequence **generateFilter**(int *idx*)

generateFilterForRelation

public CharSequence **generateFilterForRelation**(int *idx*)

getOperator

public String **getOperator**()

isOperatorAMethod

public boolean **isOperatorAMethod**()

12.57.4 EqualProperty

public class **EqualProperty**<T> extends Property<T>

The `EqualProperty` class represents a property that will be used in JDO query and it has to be equal to the given value.

Parameters

- **<T>** – type of the passed value

Constructors

EqualProperty

public **EqualProperty**(String *name*, T *value*, String *type*)

EqualProperty

public **EqualProperty**(String *jdoVariableName*, String *name*, T *value*, String *type*)

Methods

generateFilter

public CharSequence **generateFilter**(int *idx*)

12.57.5 MatchesCaseInsensitiveProperty

public class **MatchesCaseInsensitiveProperty** extends CustomOperatorProperty<String>

A convenience extension of the `CustomOperatorProperty`. The custom operator is "matches()" with an added case insensitivity flag added, the underlying type is String. The value passed will be wrapped inside (?i).*.* for matching purposes.

Constructors

MatchesCaseInsensitiveProperty

public **MatchesCaseInsensitiveProperty**(String *name*, String *value*)

Methods

shouldIgnoreThisProperty

protected boolean **shouldIgnoreThisProperty**()

12.57.6 MatchesProperty

public class **MatchesProperty** extends CustomOperatorProperty<String>

A convenience extension of the `org.motechproject.mds.query.CustomOperatorProperty`. The custom operator is "matches()" and the underlying type is String. The value passed will be wrapped inside .*.* for matching purposes.

Constructors

MatchesProperty

public **MatchesProperty** (String *name*, String *value*)

MatchesProperty

public **MatchesProperty** (String *jdoVariableName*, String *name*, String *value*)

Methods

shouldIgnoreThisProperty

protected boolean **shouldIgnoreThisProperty** ()

12.57.7 Property

public abstract class **Property**<T>

The `Property` class represents a property that will be used in JDO query. Classes that extend this class should define how that property should be used in WHERE section in JDO query.

Parameters

- **<T>** – type of the passed value

Constructors

Property

protected **Property** (String *name*, T *value*, String *type*)

Property

protected **Property** (String *jdoVariableName*, String *name*, T *value*, String *type*)

Methods

asDeclareParameter

public CharSequence **asDeclareParameter** (int *idx*)

asFilter

public CharSequence **asFilter** (int *idx*)

generateDeclareParameter

protected CharSequence **generateDeclareParameter** (int *idx*)

generateFilter

protected abstract CharSequence **generateFilter** (int *idx*)

getJdoVariableName

public String **getJdoVariableName** ()

getName

public String **getName** ()

getType

public String **getType** ()

getValue

public T **getValue** ()

isForRelation

public boolean **isForRelation** ()

shouldIgnoreThisProperty

protected boolean **shouldIgnoreThisProperty** ()

unwrap

public Collection **unwrap** ()

12.57.8 PropertyBuilder

public final class **PropertyBuilder**

The `PropertyBuilder` class is a util class that helps create appropriate property class based on passed name and value.

Methods

create

public static Property **create** (Field *field*, Object *value*)

create

public static Property **create** (String *name*, Object *value*, Class *type*)

create

public static Property **create** (String *name*, Object *value*, String *type*)

create

public static Property **create** (String *name*, Object *value*, String *type*, String *operator*)

createRelationProperty

public static Property **createRelationProperty** (String *jdoVariableName*, String *name*, Object *value*, String *type*)

createRelationProperty

public static Property **createRelationProperty** (String *jdoVariableName*, String *name*, Object *value*, String *type*, String *operator*)

createRelationPropertyForComboboxCollection

public static CollectionProperty **createRelationPropertyForComboboxCollection** (String *jdoVariable*, String *name*, Object *value*, String *type*)

12.57.9 QueryExecution

public interface **QueryExecution**<T>

Allows users to execute custom queries through Motech Data Services. Implementations need only to implement the execute method, which can operate directly on the `javax.jdo.Query` object. The return value type is left to the implementation.

Parameters

- **<T>** – the type that will be returned from this query

Methods

execute

T **execute** (Query *query*, InstanceSecurityRestriction *restriction*)

12.57.10 QueryExecutor

public final class **QueryExecutor**

The `QueryExecutor` util class provides methods that help execute a JDO query.

Methods

execute

public static Object **execute** (Query *query*, InstanceSecurityRestriction *restriction*)

execute

public static Object **execute** (Query *query*, Object *value*, InstanceSecurityRestriction *restriction*)

executeDelete

public static long **executeDelete** (Query *query*, Object *value*, InstanceSecurityRestriction *restriction*)

executeDelete

public static long **executeDelete** (Query *query*, Object[] *values*, InstanceSecurityRestriction *restriction*)

executeWithArray

public static Object **executeWithArray** (Query *query*, Object[] *values*, InstanceSecurityRestriction *restriction*)

executeWithArray

public static Object **executeWithArray** (Query *query*, List<Property> *properties*)

executeWithFilters

public static Object **executeWithFilters** (Query *query*, Filters *filters*, InstanceSecurityRestriction *restriction*)

12.57.11 QueryParams

public class **QueryParams** implements Serializable

Utility class containing parameters which control order and size of query results. Used mainly for paging/ordering queries from the UI.

Fields

ORDER_ID_ASC

public static final QueryParams **ORDER_ID_ASC**

Constant query parameter, that orders records ascending by ID.

Constructors

QueryParams

public **QueryParams** (Integer *page*, Integer *pageSize*)

Creates query parameters.

Parameters

- **page** – number of page
- **pageSize** – amount of entries to include, per page

QueryParams

public **QueryParams** (Order *order*)

Creates query parameters.

Parameters

- **order** – specifies order of the records

QueryParams

public **QueryParams** (Integer *page*, Integer *pageSize*, Order *order*)

Creates query parameters.

Parameters

- **page** – number of page
- **pageSize** – amount of entries to include, per page
- **order** – specifies order of the records

Methods

ascOrder

public static QueryParams **ascOrder** (String *field*)

Creates query parameter that sorts records ascending, by the given field.

Parameters

- **field** – field to sort records by

Returns query parameter, ordering records ascending

descOrder

public static QueryParams **descOrder** (String *field*)

Creates query parameter that sorts records descending, by the given field.

Parameters

- **field** – field to sort records by

Returns query parameter, ordering records descending

getOrder

public Order **getOrder** ()

getPage

public Integer **getPage** ()

getPageSize

public Integer **getPageSize** ()

isOrderSet

public boolean **isOrderSet** ()

isPagingSet

public boolean **isPagingSet** ()

12.57.12 QueryUtil

public final class **QueryUtil**

The `QueryUtil` util class provides methods that help developer to create a JDO query.

See also: `javax.jdo.Query`

Methods

asMatchesPattern

public static String **asMatchesPattern** (String *string*)

Returns the string in a form a pattern used for searching with the matches operator.

Parameters

- **string** – The string to search for.

Returns The string in format `.*<string>.*` or the string unchanged if it is empty.

setCountResult

public static void **setCountResult** (Query *query*)

setQueryParams

public static void **setQueryParams** (Query *query*, QueryParams *queryParams*)

useFilter

public static void **useFilter** (Query *query*, String[] *properties*, Object[] *values*, Map<String, String> *fieldTypeMap*)

useFilter

public static void **useFilter** (Query *query*, String[] *properties*, Object[] *values*, Map<String, String> *fieldTypeMap*, InstanceSecurityRestriction *restriction*)

useFilter

public static void **useFilter** (Query *query*, List<Property> *properties*)

useFilter

public static void **useFilter** (Query *query*, List<Property> *properties*, InstanceSecurityRestriction *restriction*)

useFilterFromPattern

public static void **useFilterFromPattern** (Query *query*, String *pattern*, List<Property> *properties*)

useFilters

public static void **useFilters** (Query *query*, Filters *filters*)

12.57.13 RangeProperty

public class **RangeProperty**<T> extends Property<Range<T>>

The `RangeProperty` class represents a property that will be used in JDO query and it has to be inside the given range.

Parameters

- **<T>** – type used in range.

Constructors

RangeProperty

public **RangeProperty** (String *name*, Range<T> *value*, String *type*)

RangeProperty

public **RangeProperty** (String *jdoVariableName*, String *name*, Range<T> *value*, String *type*)

Methods

generateDeclareParameter

public CharSequence **generateDeclareParameter** (int *idx*)

generateFilter

public CharSequence **generateFilter** (int *idx*)

generateFilterForRelation

public CharSequence **generateFilterForRelation** (int *idx*)

shouldIgnoreThisProperty

protected boolean **shouldIgnoreThisProperty** ()

unwrap

public Collection **unwrap** ()

12.57.14 RestrictionProperty

public class **RestrictionProperty** extends EqualProperty<String>

The `RestrictionProperty` class represents a property that will be used in JDO query and depends on restriction criteria the `creator` or `owner` field in an instance has to have the appropriate user name.

Constructors

RestrictionProperty

public **RestrictionProperty** (InstanceSecurityRestriction *restriction*, String *value*)

Methods

shouldIgnoreThisProperty

protected boolean **shouldIgnoreThisProperty** ()

12.57.15 SetProperty

public class **SetProperty**<T> extends AbstractCollectionBasedProperty<Set<T>>

The `SetProperty` class represents a property that will be used in JDO query and it has to have one of the value from the given set.

Parameters

- **<T>** – type used in set.

Constructors

SetProperty

public **SetProperty** (String *name*, Set<T> *value*, String *type*)

SetProperty

public **SetProperty** (String *jdoVariableName*, String *name*, Set<T> *value*, String *type*)

Methods

generateFilter

public CharSequence **generateFilter** (int *idx*)

unwrap

public Collection **unwrap** ()

12.57.16 SqlQueryExecution

public interface **SqlQueryExecution**<T>

Allows users to execute custom SQL queries through Motech Data Services. Implementations need to implement the execute method, which can operate directly on the `javax.jdo.Query` object and `getSqlQuery()` should return the sql query that will be executed. The return value type is left to the implementation. It is not advised to rely on raw SQL, however some use cases may require it.

Parameters

- **<T>** – the type that will be returned from this query

Methods

execute

T **execute**(Query *query*)

getSqlQuery

String **getSqlQuery**()

12.58 org.motechproject.mds.repository

12.58.1 AllConfigSettings

public class **AllConfigSettings** extends MotechDataRepository<ConfigSettings>

`AllConfigSettings` is responsible for communication with database for MDS configuration.

Constructors

AllConfigSettings

public **AllConfigSettings**()

Methods

addOrUpdate

public void **addOrUpdate**(ConfigSettings *record*)

12.58.2 AllEntities

public class **AllEntities** extends MotechDataRepository<Entity>

The `AllEntities` class is a repository class that operates on instances of `org.motechproject.mds.domain.Entity`.

Constructors

AllEntities

public **AllEntities** ( )

Methods

contains

public boolean **contains** (String *className*)

create

public Entity **create** (EntityDto *dto*)

delete

public void **delete** (Long *id*)

retrieveByClassName

public Entity **retrieveByClassName** (String *className*)

retrieveById

public Entity **retrieveById** (Long *id*)

updateAndIncrementVersion

public Entity **updateAndIncrementVersion** (Entity *entity*)

12.58.3 AllEntityAudits

public class **AllEntityAudits** extends MotechDataRepository<EntityAudit>

This is a repository for persisting entity audits. It provides methods which create audits, by cloning the given entity.

Constructors

AllEntityAudits

public **AllEntityAudits** ( )

Methods

createAudit

public EntityAudit **`createAudit`** (Entity *entity*, String *username*)

12.58.4 AllEntityDrafts

public class **`AllEntityDrafts`** extends MotechDataRepository<EntityDraft>

This a repository for persisting entity drafts. It provides methods which create drafts, by cloning the given entity.

Constructors

AllEntityDrafts

public **`AllEntityDrafts`** ()

Methods

create

public EntityDraft **`create`** (Entity *entity*, String *username*)

deleteAll

public void **`deleteAll`** (Entity *entity*)

retrieve

public EntityDraft **`retrieve`** (Entity *entity*, String *username*)

retrieveAll

public List<EntityDraft> **`retrieveAll`** (String *username*)

retrieveAll

public List<EntityDraft> **`retrieveAll`** (Entity *entity*)

setProperties

public void **`setProperties`** (EntityDraft *draft*, Entity *entity*)

setProperties

public void **setProperties** (EntityDraft *draft*, Entity *entity*, String *username*)

update

public EntityDraft **update** (EntityDraft *draft*)

12.58.5 AllJsonLookups

public class **AllJsonLookups** extends MotechDataRepository<JsonLookup>

Constructors

AllJsonLookups

public **AllJsonLookups** ()

Methods

create

public JsonLookup **create** (JsonLookupDto *dto*)

getByOriginName

public JsonLookup **getByOriginName** (String *entityClassName*, String *originName*)

12.58.6 AllMigrationMappings

public class **AllMigrationMappings** extends MotechDataRepository<MigrationMapping>

The `AllMigrationMappings` class is a repository class that operates on instances of `org.motechproject.mds.domain.MigrationMapping`.

Constructors

AllMigrationMappings

public **AllMigrationMappings** ()

Methods

retrieveByModuleAndMigrationVersion

public MigrationMapping **retrieveByModuleAndMigrationVersion** (String *moduleName*, Integer *version*)

12.58.7 AllTypeSettings

public class **AllTypeSettings** extends MotechDataRepository<TypeSetting>

The `AllTypeSettings` class is a repository class that operates on instances of `org.motechproject.mds.domain.TypeSetting`.

Constructors

AllTypeSettings

public **AllTypeSettings**()

12.58.8 AllTypeValidations

public class **AllTypeValidations** extends MotechDataRepository<TypeValidation>

The `AllTypeValidations` class is a repository class that operates on instances of `org.motechproject.mds.domain.TypeValidation`.

Constructors

AllTypeValidations

public **AllTypeValidations**()

12.58.9 AllTypes

public class **AllTypes** extends MotechDataRepository<Type>

The `AllTypes` repository class allows persistence and retrieving of Field Types in Data Services database.

Constructors

AllTypes

public **AllTypes**()

Methods

retrieveByClassName

public Type **retrieveByClassName**(String *className*)

retrieveByDisplayName

public Type **retrieveByDisplayName**(String *displayName*)

12.58.10 MetadataHolder

public class **MetadataHolder**

Holds the current JDO metadata for MDS. Allows reloading the metadata and retrieval for modifications.

Methods

getJdoMetadata

public JDOMetadata **getJdoMetadata** ( )

reloadMetadata

public JDOMetadata **reloadMetadata** ( )

setPersistenceManagerFactory

public void **setPersistenceManagerFactory** (PersistenceManagerFactory *persistenceManagerFactory*)

12.58.11 MotechDataRepository

public abstract class **MotechDataRepository**<T>

This is a basic repository class with standard CRUD operations. It should be used by other repositories inside this package.

This class is also used as super class to create a repository related with the given entity schema in `org.motechproject.mds.builder.EntityInfrastructureBuilder`.

Parameters

- **<T>** – the type of class

Constructors

MotechDataRepository

protected **MotechDataRepository** (Class<T> *classType*)

MotechDataRepository

protected **MotechDataRepository** (Class<T> *classType*, int *fetchDepth*)

Methods

count

public long **count** (InstanceSecurityRestriction *restriction*)

count

public long **count** (String[] *properties*, Object[] *values*, InstanceSecurityRestriction *restriction*)

count

public long **count** (List<Property> *properties*, InstanceSecurityRestriction *restriction*)

countForFilters

public long **countForFilters** (Filters *filters*, InstanceSecurityRestriction *restriction*)

create

public T **create** (T *object*)

delete

public void **delete** (T *object*)

delete

public long **delete** (String *property*, Object *value*)

delete

public long **delete** (String *property*, Object *value*, InstanceSecurityRestriction *restriction*)

delete

public long **delete** (String[] *properties*, Object[] *values*, InstanceSecurityRestriction *restriction*)

exists

public boolean **exists** (String *property*, Object *value*)

exists

public boolean **exists** (String[] *properties*, Object[] *values*)

filter

public List<T> **filter** (Filters *filters*, QueryParams *queryParams*, InstanceSecurityRestriction *restriction*)

getClassType

public Class<T> **getClassType**()

getDetachedField

public Object **getDetachedField**(T *instance*, String *field*)

getPersistenceManager

public PersistenceManager **getPersistenceManager**()

retrieve

public T **retrieve**(Object *key*)

retrieve

public T **retrieve**(String *property*, Object *value*)

retrieve

public T **retrieve**(String *property*, Object *value*, InstanceSecurityRestriction *restriction*)

retrieve

public T **retrieve**(String[] *properties*, Object[] *values*)

retrieve

public T **retrieve**(String[] *properties*, Object[] *values*, InstanceSecurityRestriction *restriction*)

retrieveAll

public List<T> **retrieveAll**()

retrieveAll

public List<T> **retrieveAll**(InstanceSecurityRestriction *restriction*)

retrieveAll

public List<T> **retrieveAll**(String *property*, Object *value*)

retrieveAll

public List<T> **retrieveAll** (String *property*, Object *value*, InstanceSecurityRestriction *restriction*)

retrieveAll

public List<T> **retrieveAll** (String[] *properties*, Object[] *values*, InstanceSecurityRestriction *restriction*)

retrieveAll

public List<T> **retrieveAll** (String[] *properties*, Object[] *values*, QueryParams *queryParams*, InstanceSecurityRestriction *restriction*)

retrieveAll

public List<T> **retrieveAll** (QueryParams *queryParams*, InstanceSecurityRestriction *restriction*)

retrieveAll

public List<T> **retrieveAll** (List<Property> *properties*, InstanceSecurityRestriction *restriction*)

retrieveAll

public List<T> **retrieveAll** (List<Property> *properties*, QueryParams *queryParams*, InstanceSecurityRestriction *restriction*)

retrieveUnique

public T **retrieveUnique** (List<Property> *properties*, InstanceSecurityRestriction *restriction*)

setFieldTypeMap

public void **setFieldTypeMap** (Map<String, String> *fieldTypeMap*)

setPersistenceManagerFactory

public void **setPersistenceManagerFactory** (PersistenceManagerFactory *persistenceManagerFactory*)

update

public T **update** (T *object*)

12.58.12 SchemaChangeLockManager

public class **SchemaChangeLockManager**

Manager for obtaining the schema change lock.

Fields

LOCK_TIMEOUT_SECONDS

public static final int **LOCK_TIMEOUT_SECONDS**

Methods

acquireLock

public void **acquireLock** (String *comment*)

releaseLock

public void **releaseLock** (String *comment*)

12.59 org.motechproject.mds.rest

12.59.1 MdsRestFacade

public interface **MdsRestFacade**<T>

Interface called by the REST controller REST operations. Should be exposed as an OSGi service for each MDS Entity. If rest is not supported, it throws a `org.motechproject.mds.ex.rest.RestNotSupportedException`.

Parameters

- **<T>** – the entity class.

Methods

create

RestProjection **create** (InputStream *instanceBody*)

Creates an instance in MDS, reading it from the input stream. Only fields that are visible via REST will be set in the created instance. It will fail, if data read from input stream contains fields that do not exists or if field values do not match their type. It also throws `org.motechproject.mds.ex.rest.RestOperationNotSupportedException` if the entity settings do not permit CREATE access via REST.

Parameters

- **instanceBody** – input stream, containing instance representation in JSON

Returns created instance, in form of a map with field names and their respective values

delete

void **delete** (Long *id*)

Deletes an instance by id. This works exactly like deleting an instance in any other way, but will throw `org.motechproject.mds.ex.rest.RestOperationNotSupportedException` if the entity settings do not permit DELETE access via REST.

Parameters

- **id** – id of the instance

executeLookup

Object **executeLookup** (String *lookupName*, Map<String, String> *lookupMap*, QueryParams *queryParams*, boolean *includeBlob*)

Executes a lookup for REST, given the lookup name, lookup parameters and query parameters. The result will only contain fields that are visible for REST. If requested lookup is not available via REST, this will throw `RestLookupExecutionForbiddenException`. If a lookup of given name does not exist, it throws `org.motechproject.mds.ex.rest.RestLookupNotFoundException`.

Parameters

- **lookupName** – name of the lookup
- **lookupMap** – map containing field names and their respective values
- **queryParams** – query parameters to use retrieving instances
- **includeBlob** – set to true, if you wish to retrieve value for binary object fields

Returns lookup result, that can be either a single instance or a collection of instances. Response contains also metadata.

get

RestResponse **get** (QueryParams *queryParams*, boolean *includeBlob*)

Retrieves entity instances for REST. This will only include fields that are visible for REST. It throws `org.motechproject.mds.ex.rest.RestOperationNotSupportedException` if the entity settings do not permit READ access via REST.

Parameters

- **queryParams** – query parameters to use retrieving instances
- **includeBlob** – set to true, if you wish to retrieve value for binary object fields

Returns a response that contains metadata and list of instances, in form of a map with field names and their respective values

get

RestResponse **get** (Long *id*, boolean *includeBlob*)

Retrieves a single instance for REST. This will only include fields that are visible for REST. It throws `org.motechproject.mds.ex.rest.RestOperationNotSupportedException` if the entity settings do not permit READ access via REST.

Parameters

- **id** – id of the instance
- **includeBlob** – set to true, if you wish to retrieve value for binary object fields

Returns a response that contains metadata and instance

update

RestProjection **update** (InputStream *instanceBody*)

Updates an instance in MDS, reading it from the input stream. Only fields that are visible via REST will be set in the updated instance. It will fail, if data read from input stream contains fields that do not exists or if field values do not match their type. It also throws `org.motechproject.mds.ex.rest.RestOperationNotSupportedException` if the entity settings do not permit UPDATE access via REST.

Parameters

- **instanceBody** – input stream, containing instance representation in JSON

Returns updated instance, in form of a map with field names and their respective values

12.59.2 MdsRestFacadeImpl

public class **MdsRestFacadeImpl**<T> implements MdsRestFacade<T>

This `org.motechproject.mds.rest.MdsRestFacade` implementation retrieves REST related metadata on initialization. It uses an instance of `org.motechproject.mds.service.MotechDataService` for operations and the jackson JSON library for parsing InputStreams.

Parameters

- **<T>** –

Methods

create

public RestProjection **create** (InputStream *instanceBody*)

delete

public void **delete** (Long *id*)

executeLookup

public Object **executeLookup** (String *lookupName*, Map<String, String> *lookupMap*, QueryParams *queryParams*, boolean *includeBlob*)

get

public RestResponse **get** (QueryParams *queryParams*, boolean *includeBlob*)

get

public RestResponse **get** (Long *id*, boolean *includeBlob*)

init

public void **init** ()

setAllEntities

public void **setAllEntities** (AllEntities *allEntities*)

setDataService

public void **setDataService** (MotechDataService<T> *dataService*)

update

public RestProjection **update** (InputStream *instanceBody*)

12.59.3 RestMetadata

public class **RestMetadata**

The `RestResponse` class represents metadata of retrieved instances over REST. It contains entity name, entity class name, module name, namespace and pagination information

See also: `org.motechproject.mds.rest.MdsRestFacade`, `org.motechproject.mds.rest.RestProjecti` `org.motechproject.mds.rest.RestMetadata`

Constructors

RestMetadata

public **RestMetadata** ()

Default constructor.

RestMetadata

public **RestMetadata** (String *entity*, String *className*, String *moduleName*, String *namespace*, Long *totalCount*, QueryParams *queryParams*)

Constructor.

Parameters

- **entity** – the entity name
- **className** – the name of the entity class
- **moduleName** – the module name

- **namespace** – the namespace in which the entity is defined
- **totalCount** – the total number of instances that match the search conditions
- **queryParams** – the query params used to retrieve instances

Methods

getClassName

public String **getClassName**()

Returns the class name of the entity

getEntity

public String **getEntity**()

Returns the entity name

getModule

public String **getModule**()

Returns the module name

getNamespace

public String **getNamespace**()

Returns the namespace in which the entity is defined

getPage

public int **getPage**()

Returns the page number

getPageSize

public int **getPageSize**()

Returns the page size

getTotalCount

public long **getTotalCount**()

Returns the total count of instances that match the search conditions

setClassName

public void **setClassName** (String *className*)

Parameters

- **className** – the class name of the entity

setEntity

public void **setEntity** (String *entity*)

Parameters

- **entity** – the entity name

setModule

public void **setModule** (String *module*)

Parameters

- **module** – the module name

setNamespace

public void **setNamespace** (String *namespace*)

Parameters

- **namespace** – the namespace in which the entity is defined

setPage

public void **setPage** (int *page*)

Parameters

- **page** – the page number

setPageSize

public void **setPageSize** (int *pageSize*)

Parameters

- **pageSize** – the page size

setTotalCount

public void **setTotalCount** (long *totalCount*)

Parameters

- **totalCount** – the total count of instances that match the search conditions

12.59.4 RestProjection

public class **RestProjection** extends LinkedHashMap<String, Object>

The `RestProjection` class represents entity fields projection onto entity fields exposed over REST. Because of its Map interface inheritance, we can leave output handling to other components. Additionally, LinkedHashMap ensures unchanged fields order.

See also: `org.motechproject.mds.rest.MdsRestFacade`

Methods

createProjection

public static <T> RestProjection **createProjection**(T *element*, List<String> *fields*, List<String> *blobFields*)

createProjectionCollection

public static <T> List<RestProjection> **createProjectionCollection**(Collection<T> *collection*, List<String> *fields*, List<String> *blobFields*)

12.59.5 RestResponse

public class **RestResponse**

The `RestResponse` class represents data retrieved over REST. It contains metadata and data.

See also: `org.motechproject.mds.rest.MdsRestFacade`, `org.motechproject.mds.rest.RestProjecti` `org.motechproject.mds.rest.RestMetadata`

Constructors

RestResponse

public **RestResponse**()

Default constructor.

RestResponse

public **RestResponse**(String *entity*, String *className*, String *moduleName*, String *namespace*, Long *totalSize*, QueryParams *queryParams*, List<RestProjection> *data*)

Constructor.

Parameters

- **entity** – the entity name
- **className** – the name of the entity class
- **moduleName** – the module name
- **namespace** – the namespace in which the entity is defined

- **totalSize** – the total number of data that match the search conditions
- **queryParams** – the query params used to retrieve data
- **data** – the list of the data

RestResponse

public **RestResponse** (String *entity*, String *className*, String *moduleName*, String *namespace*, Long *totalSize*, QueryParams *queryParams*, RestProjection *data*)

Constructor.

Parameters

- **entity** – the entity name
- **className** – the name of the entity class
- **moduleName** – the module name
- **namespace** – the namespace in which the entity is defined
- **totalSize** – the total number of data that match the search conditions
- **queryParams** – the query params used to retrieve data
- **data** – the record

Methods

getData

public List<RestProjection> **getData** ()

Returns the list of the data

getMetadata

public RestMetadata **getMetadata** ()

Returns the metadata for the response

setData

public void **setData** (List<RestProjection> *data*)

Parameters

- **data** – the list of the data

setMetadata

public void **setMetadata** (RestMetadata *metadata*)

Parameters

- **metadata** – the metadata for the response

12.60 org.motechproject.mds.service

12.60.1 ActionHandlerService

public interface **`ActionHandlerService`**

The `ActionHandlerService` interface provides methods for handling tasks actions events related with MDS CRUD operations.

Methods

create

void **`create`** (Map<String, Object> *parameters*)

Creates an instance of the entity, based on the provided parameters. The parameters should contain the entity class name and field values.

Parameters

- **parameters** – a map of parameters

Throws

- **ActionHandlerException** – if the instance of the entity could not get created due to missing class name, lack of constructor, or any other reasons

delete

void **`delete`** (Map<String, Object> *parameters*)

Deletes an instance of the entity, based on the provided parameters. The parameters should contain the entity class name and instance id.

Parameters

- **parameters** – a map of parameters

Throws

- **ActionHandlerException** – if the instance could not get deleted due to missing class name, missing id of the instance, missing instance of the given id, or any other reasons.

update

void **`update`** (Map<String, Object> *parameters*)

Updates an instance of the entity, based on the provided parameters. The parameters should contain the entity class name, id of the instance and field values to update.

Parameters

- **parameters** – a map of parameters

Throws

- **ActionHandlerException** – if the instance could not get updated due to missing class name, missing id of the instance, missing instance of the given id, problems setting instance properties, or any other reasons

12.60.2 BundleWatcherSuspensionService

public interface **BundleWatcherSuspensionService**

An OSGi service, allowing to temporarily disable bundle processing by MDS. When processing gets suspended, MDS will still listen to bundle events, but instead of processing them, they will be queued and processed after the processing gets restored. This allows, besides others, to install/uninstall a larger amount of bundles at one time, without facing annotation processing problems.

Methods

restoreBundleProcessing

void **restoreBundleProcessing**()

Restores suspended bundle processing. All bundle events, that happened while processing was suspended, will get processed.

suspendBundleProcessing

void **suspendBundleProcessing**()

Temporarily suspends bundle processing. MDS will queue bundle events and process them once the processing gets restored.

12.60.3 CsvExportCustomizer

public interface **CsvExportCustomizer**

The `CsvExportCustomizer` interface allows to provide custom method to format related instances during csv import.

Methods

columnOrderComparator

Comparator<Field> **columnOrderComparator**()

Allows the customizer to change the ordering of columns in the exporter file. The comparator returned by this method will be used for ordering fields. Note that the comparator might be requested to order fields that were not selected for export - it will be used to order the entire collection of fields from the entity.

Returns the comparator that will be used for determining the column order

formatRelationship

String **formatRelationship**(Object *object*)

Formats related instance into a csv value

Parameters

- **object** – the related instance (or collection of instances)

Returns formatted string

12.60.4 CsvImportCustomizer

public interface **CsvImportCustomizer**

The `CsvImportCustomizer` interface allows to provide custom methods for finding, creating and updating an instance during csv import.

See also: `org.motechproject.mds.domain.Entity`

Methods

doCreate

Object **doCreate** (Object *instance*, MotechDataService *dataService*)

Creates an instance using given dataService

Parameters

- **instance** – the instance to create
- **dataService** – the data service of an entity

Returns the created instance

doUpdate

Object **doUpdate** (Object *instance*, MotechDataService *dataService*)

Updates an instance using given dataService

Parameters

- **instance** – the instance to update
- **dataService** – the data service of an entity

Returns the updated instance

findExistingInstance

Object **findExistingInstance** (Map<String, String> *row*, MotechDataService *dataService*)

Retrieves an instance based on the fields imported from csv

Parameters

- **row** – the imported row containing fields of an instance
- **dataService** – the data service of an entity

Returns single instance or null if none is found

12.60.5 CsvImportExportService

public interface **CsvImportExportService**

Service for exporting and importing entity data in CSV format. The format is the same for both import and export. Columns are separated by the ',' character. The top row(header row) consists of names of the fields represented by the columns.

Methods

exportCsv

long **exportCsv**(long *entityId*, Writer *writer*)

Exports entity instances to a CSV file.

Parameters

- **entityId** – id of the entity for which the instances will be exported
- **writer** – the writer that will be used for output

Returns number of exported instances

exportCsv

long **exportCsv**(String *entityClassName*, Writer *writer*)

Exports entity instances to a CSV file.

Parameters

- **entityClassName** – class name of the entity for which the instances will be exported
- **writer** – the writer that will be used for output

Returns number of exported instances

exportCsv

long **exportCsv**(long *entityId*, Writer *writer*, CsvExportCustomizer *exportCustomizer*)

Exports entity instances to a CSV file.

Parameters

- **entityId** – id of the entity for which the instances will be exported
- **writer** – the writer that will be used for output
- **exportCustomizer** – the customizer that will be used during export

Returns number of exported instances

exportCsv

long **exportCsv**(String *entityClassName*, Writer *writer*, CsvExportCustomizer *exportCustomizer*)

Exports entity instances to a CSV file.

Parameters

- **entityClassName** – class name of the entity for which the instances will be exported
- **writer** – the writer that will be used for output
- **exportCustomizer** – the customizer that will be used during export

Returns number of exported instances

exportCsv

long **exportCsv**(long *entityId*, Writer *writer*, String *lookupName*, QueryParams *params*, List<String> *headers*, Map<String, Object> *lookupFields*)

Exports entity instances to a CSV file.

Parameters

- **entityId** – id of the entity for which the instances will be exported
- **writer** – the writer that will be used for output
- **lookupName** – the name of lookup
- **params** – query parameters to be used retrieving instances
- **headers** – the headers of exported file
- **lookupFields** – the lookupFields used in the lookup

Returns number of exported instances

exportCsv

long **exportCsv**(String *entityClassName*, Writer *writer*, String *lookupName*, QueryParams *params*, List<String> *headers*, Map<String, Object> *lookupFields*)

Exports entity instances to a CSV file.

Parameters

- **entityClassName** – class name of the entity for which the instances will be exported
- **writer** – the writer that will be used for output
- **lookupName** – the name of lookup
- **params** – query parameters to be used retrieving instances
- **headers** – the headers of exported file
- **lookupFields** – the lookupFields used in the lookup

Returns number of exported instances

exportCsv

long **exportCsv**(long *entityId*, Writer *writer*, String *lookupName*, QueryParams *params*, List<String> *headers*, Map<String, Object> *lookupFields*, CsvExportCustomizer *exportCustomizer*)

Exports entity instances to a CSV file.

Parameters

- **entityId** – id of the entity for which the instances will be exported
- **writer** – the writer that will be used for output
- **lookupName** – the name of lookup
- **params** – query parameters to be used retrieving instances
- **headers** – the headers of exported file
- **lookupFields** – the lookupFields used in the lookup

Returns number of exported instances

exportCsv

long **exportCsv**(String *entityClassName*, Writer *writer*, String *lookupName*, QueryParams *params*, List<String> *headers*, Map<String, Object> *lookupFields*, CsvExportCustomizer *exportCustomizer*)

Exports entity instances to a CSV file.

Parameters

- **entityClassName** – class name of the entity for which the instances will be exported
- **writer** – the writer that will be used for output
- **lookupName** – the name of lookup
- **params** – query parameters to be used retrieving instances
- **headers** – the headers of exported file
- **lookupFields** – the lookupFields used in the lookup
- **exportCustomizer** – the customizer that will be used during export

Returns number of exported instances

exportPdf

long **exportPdf**(long *entityId*, OutputStream *outputStream*)

Exports entity instances to a PDF file.

Parameters

- **entityId** – id of the entity for which the instances will be exported
- **outputStream** – the stream to write the PDF to

Returns number of exported instances

exportPdf

long **exportPdf**(String *entityClassName*, OutputStream *outputStream*)

Exports entity instances to a PDF file.

Parameters

- **entityClassName** – class name of the entity for which the instances will be exported
- **outputStream** – the stream to write the PDF to

Returns number of exported instances

exportPdf

long **exportPdf**(long *entityId*, OutputStream *outputStream*, CsvExportCustomizer *exportCustomizer*)

Exports entity instances to a PDF file.

Parameters

- **entityId** – id of the entity for which the instances will be exported
- **outputStream** – the stream to write the PDF to
- **exportCustomizer** – the customizer that will be used during export

Returns number of exported instances

exportPdf

long **exportPdf**(String *entityClassName*, OutputStream *outputStream*, CsvExportCustomizer *exportCustomizer*)

Exports entity instances to a PDF file.

Parameters

- **entityClassName** – class name of the entity for which the instances will be exported
- **outputStream** – the writer that will be used for output
- **exportCustomizer** – the customizer that will be used during export

Returns number of exported instances

exportPdf

long **exportPdf**(long *entityId*, OutputStream *outputStream*, String *lookupName*, QueryParams *params*, List<String> *headers*, Map<String, Object> *lookupFields*)

Exports entity instances to a PDF file.

Parameters

- **entityId** – id of the entity for which the instances will be exported
- **outputStream** – the stream to write the PDF to
- **lookupName** – the name of lookup
- **params** – query parameters to be used retrieving instances
- **headers** – the headers of exported file
- **lookupFields** – the lookupFields used in the lookup

Returns number of exported instances

exportPdf

long **exportPdf**(String *entityClassName*, OutputStream *outputStream*, String *lookupName*, QueryParams *params*, List<String> *headers*, Map<String, Object> *lookupFields*)

Exports entity instances to a PDF file.

Parameters

- **entityClassName** – class name of the entity for which the instances will be exported
- **outputStream** – the stream to write the PDF to
- **lookupName** – the name of lookup
- **params** – query parameters to be used retrieving instances
- **headers** – the headers of exported file

- **lookupFields** – the lookupFields used in the lookup

Returns number of exported instances

exportPdf

long **exportPdf**(long *entityId*, OutputStream *outputStream*, String *lookupName*, QueryParams *params*, List<String> *headers*, Map<String, Object> *lookupFields*, CsvExportCustomizer *exportCustomizer*)

Exports entity instances to a PDF file.

Parameters

- **entityId** – id of the entity for which the instances will be exported
- **outputStream** – the stream to write the PDF to
- **lookupName** – the name of lookup
- **params** – query parameters to be used retrieving instances
- **headers** – the headers of exported file
- **lookupFields** – the lookupFields used in the lookup

Returns number of exported instances

exportPdf

long **exportPdf**(String *entityClassName*, OutputStream *outputStream*, String *lookupName*, QueryParams *params*, List<String> *headers*, Map<String, Object> *lookupFields*, CsvExportCustomizer *exportCustomizer*)

Exports entity instances to a PDF file.

Parameters

- **entityClassName** – class name of the entity for which the instances will be exported
- **outputStream** – the stream to write the PDF to
- **lookupName** – the name of lookup
- **params** – query parameters to be used retrieving instances
- **headers** – the headers of exported file
- **lookupFields** – the lookupFields used in the lookup
- **exportCustomizer** – the customizer that will be used during export

Returns number of exported instances

importCsv

CsvImportResults **importCsv**(long *entityId*, Reader *reader*, String *fileName*)

Import instances from a CSV file

Parameters

- **entityId** – id of the entity for which the instances will be imported
- **reader** – the reader that will be used for reading the file contents

- **fileName** – the name of the CSV file

Returns IDs of instances updated/added during import

importCsv

CsvImportResults **importCsv**(long *entityId*, Reader *reader*, String *fileName*, CsvImportCustomizer *importCustomizer*)

Import instances from a CSV file

Parameters

- **entityId** – id of the entity for which the instances will be imported
- **reader** – the reader that will be used for reading the file contents
- **fileName** – the name of the CSV file
- **importCustomizer** – the customizer that will be used during import

Returns IDs of instances updated/added during import

importCsv

CsvImportResults **importCsv**(String *entityClassName*, Reader *reader*, String *fileName*)

Import instances from a CSV file

Parameters

- **entityClassName** – class name of the entity for which the instances will be imported
- **reader** – the reader that will be used for reading the file contents
- **fileName** – the name of the CSV file

Returns IDs of instances updated/added during import

12.60.6 DefaultCsvExportCustomizer

public class **DefaultCsvExportCustomizer** implements CsvExportCustomizer

This is a basic implementation of `org.motechproject.mds.service.CsvExportCustomizer`.

Methods

columnOrderComparator

public Comparator<Field> **columnOrderComparator**()

formatRelationship

public String **formatRelationship**(Object *object*)

12.60.7 DefaultCsvImportCustomizer

public class **DefaultCsvImportCustomizer** implements CsvImportCustomizer

This is a basic implementation of `org.motechproject.mds.service.CsvImportCustomizer`.

Methods

doCreate

public Object **doCreate** (Object *instance*, MotechDataService *dataService*)

doUpdate

public Object **doUpdate** (Object *instance*, MotechDataService *dataService*)

findExistingInstance

public Object **findExistingInstance** (Map<String, String> *row*, MotechDataService *dataService*)

12.60.8 DefaultMotechDataService

public abstract class **DefaultMotechDataService**<T> implements MotechDataService<T>

This is a basic implementation of `org.motechproject.mds.service.MotechDataService`. Mainly it is used as super class to create a service related with the given entity schema in `org.motechproject.mds.builder.EntityInfrastructureBuilder` but it can be also used by other services inside this package.

Parameters

- **<T>** – the type of entity schema.

Methods

count

public long **count** ()

count

protected long **count** (List<Property> *properties*)

countForFilters

public long **countForFilters** (Filters *filters*)

create

public T **create** (T *object*)

createOrUpdate

public T **`createOrUpdate`** (T *object*)

delete

public void **`delete`** (T *object*)

delete

public void **`delete`** (String *primaryKeyName*, Object *value*)

deleteAll

public void **`deleteAll`** ()

deleteById

public void **`deleteById`** (long *id*)

doInTransaction

public <R> R **`doInTransaction`** (TransactionCallback<R> *transactionCallback*)

executeQuery

public <R> R **`executeQuery`** (QueryExecution<R> *queryExecution*)

executeSQLQuery

public <R> R **`executeSQLQuery`** (SqlQueryExecution<R> *queryExecution*)

filter

public List<T> **`filter`** (Filters *filters*, QueryParams *queryParams*)

findById

public T **`findById`** (Long *id*)

findTrashInstanceById

public T **`findTrashInstanceById`** (Object *instanceId*, Object *entityId*)

getClassType

public Class<T> **getClassType**()

getDetachedField

public Object **getDetachedField**(T *instance*, String *fieldName*)

getId

protected Object **getId**(T *instance*)

getLogger

protected Logger **getLogger**()

getRepository

protected MotechDataRepository<T> **getRepository**()

initializeSecurityState

public void **initializeSecurityState**()

retrieve

public T **retrieve**(String *primaryKeyName*, Object *value*)

retrieveAll

public List<T> **retrieveAll**()

retrieveAll

public List<T> **retrieveAll**(QueryParams *queryParams*)

retrieveAll

protected List<T> **retrieveAll**(List<Property> *properties*)

retrieveAll

protected List<T> **retrieveAll**(List<Property> *properties*, QueryParams *queryParams*)

revertFromTrash

public void **revertFromTrash** (Object *newInstance*, Object *trash*)

setAllEntities

public void **setAllEntities** (AllEntities *allEntities*)

setEntityService

public void **setEntityService** (EntityService *entityService*)

setHistoryService

public void **setHistoryService** (HistoryService *historyService*)

setOsgiEventProxy

public void **setOsgiEventProxy** (OsgiEventProxy *osgiEventProxy*)

setRepository

public void **setRepository** (MotechDataRepository<T> *repository*)

setTransactionManager

public void **setTransactionManager** (JdoTransactionManager *transactionManager*)

setTrashService

public void **setTrashService** (TrashService *trashService*)

update

public T **update** (T *object*)

updateFromTransient

public T **updateFromTransient** (T *transientObject*)

updateFromTransient

public T **updateFromTransient** (T *transientObject*, Set<String> *fieldsToUpdate*)

validateCredentials

protected InstanceSecurityRestriction **validateCredentials**()

validateCredentials

protected InstanceSecurityRestriction **validateCredentials**(T *instance*)

12.60.9 EntityService

public interface **EntityService**

This interface provides methods related with executing actions on an entity.

See also: `org.motechproject.mds.domain.Entity`

Methods

abandonChanges

void **abandonChanges**(Long *entityId*)

Removes the draft data permanently.

Parameters

- **entityId** – id of the draft entity

addDisplayedFields

void **addDisplayedFields**(EntityDto *entityDto*, Map<String, Long> *positions*)

Adds ability to point fields that should be displayed on the data browser by default and allows to set their position on the UI. If not invoked on any field and no field has the `org.motechproject.mds.annotations.UIDisplayable` annotation, all the fields, except auto-generated ones will be displayed. If invoked on at least one field, all other fields will get hidden by default.

Parameters

- **entityDto** – entity representation
- **positions** – a map of field names and their positions. If position is irrelevant, place -1 as entry value

addFields

void **addFields**(EntityDto *entity*, FieldDto... *fields*)

Adds fields to the given entity. If the field of identical name already exists in the entity definition, it will be updated. If the entity does not exist, it throws `org.motechproject.mds.ex.entity.EntityNotFoundException`

Parameters

- **entity** – the entity to add fields to
- **fields** – fields to add or update

addFields

void **addFields** (EntityDto *entity*, Collection<FieldDto> *fields*)

Adds fields to the given entity. If the field of identical name already exists in the entity definition, it will be updated. If the entity does not exist, it throws `org.motechproject.mds.ex.entity.EntityNotFoundException`

Parameters

- **entity** – the entity to add fields to
- **fields** – fields to add or update

addFields

void **addFields** (Long *entityId*, FieldDto... *fields*)

Adds fields to the given entity. If the field of identical name already exists in the entity definition, it will be updated. If the entity does not exist, it throws `org.motechproject.mds.ex.entity.EntityNotFoundException`

Parameters

- **entityId** – id of the entity to add fields to
- **fields** – fields to add or update

addFields

void **addFields** (Long *entityId*, Collection<FieldDto> *fields*)

Adds fields to the given entity. If the field of identical name already exists in the entity definition, it will be updated. If the entity does not exist, it throws `org.motechproject.mds.ex.entity.EntityNotFoundException`

Parameters

- **entityId** – id of the entity to add fields to
- **fields** – fields to add or update

addFilterableFields

void **addFilterableFields** (EntityDto *entityDto*, Collection<String> *fieldNames*)

Provides ability to point fields, for which UI should provide the ability to filter through. Note, that only several field types support filtering via UI. If a field of not supported type is marked as filterable, this will have no effect.

Parameters

- **entityDto** – entity representation
- **fieldNames** – the names of the fields, that should be marked filterable

addLookups

void **addLookups** (EntityDto *entityDto*, LookupDto... *lookups*)

Adds or updates lookups for the given entity.

Parameters

- **entityDto** – entity representation
- **lookups** – lookups to add or update

addLookups

void **addLookups** (EntityDto *entityDto*, Collection<LookupDto> *lookups*)

Adds or updates lookups for the given entity.

Parameters

- **entityDto** – entity representation
- **lookups** – lookups to add or update

addLookups

void **addLookups** (Long *entityId*, LookupDto... *lookups*)

Adds or updates lookups for the given entity.

Parameters

- **entityId** – id of an entity
- **lookups** – lookups to add or update

addLookups

void **addLookups** (Long *entityId*, Collection<LookupDto> *lookups*)

Adds or updates lookups for the given entity.

Parameters

- **entityId** – id of an entity
- **lookups** – lookups to add or update

addNonEditableFields

void **addNonEditableFields** (EntityDto *entityDto*, Map<String, Boolean> *nonEditableFields*)

Provides ability to point fields that should be non-editable via UI.

Parameters

- **entityDto** – entity representation
- **nonEditableFields** – a map of the non-editable field names and their display values.

commitChanges

List<String> **commitChanges** (Long *entityId*)

Retrieves a draft and attempts to update actual entity, according to the changes present in the draft. The username, for which the draft should be retrieved, will be determined on the current security context. If the draft is outdated, which means that somebody else has already updated the entity, the

`org.motechproject.mds.ex.entity.EntityChangedException` will be thrown. If the draft is not outdated, a validation will be performed to determine whether the changes are valid and can be applied, and if so the changes will be made and the draft will get deleted.

Parameters

- **entityId** – id of the draft or actual entity

Returns a list of modules, affected by the commit

commitChanges

List<String> **commitChanges** (Long *entityId*, String *changesOwner*)

Retrieves a draft for a given user and attempts to update actual entity, according to the changes present in the draft. If the draft is outdated, which means that somebody else has already updated the entity, the `org.motechproject.mds.ex.entity.EntityChangedException` will be thrown. If the draft is not outdated, a validation will be performed to determine whether the changes are valid and can be applied, and if so the changes will be made and the draft will get deleted.

Parameters

- **entityId** – id of the draft or actual entity

Returns a list of modules, affected by the commit

createEntity

EntityDto **createEntity** (EntityDto *entityDto*)

Creates an entity and adds default fields, provided the entity does not contain them already (from inheritance). It will throw `org.motechproject.mds.ex.entity.EntityAlreadyExistException` if an entity of identical class name already exists.

Parameters

- **entityDto** – representation of an entity to construct from.

Returns representation of a created entity.

deleteEntity

void **deleteEntity** (Long *entityId*)

Removes the entity with the given id. This will also remove all drafts associated with the deleted entity.

Parameters

- **entityId** – The id of an entity

findEntitiesByPackage

List<EntityDto> **findEntitiesByPackage** (String *packageName*)

Retrieves entities for the given package

Parameters

- **packageName** – the package name

Returns A list of entities

findEntityFieldByName

FieldDto **findEntityFieldByName** (Long *entityId*, String *name*)

Retrieves a field by name. This will not include draft fields.

Parameters

- **entityId** – id of an entity
- **name** – name of the field

Returns Field of the given name for given entity id

findFieldByName

FieldDto **findFieldByName** (Long *entityId*, String *name*)

Retrieves a field by name. This will be able to find any draft fields, that the current user has added, deleted or modified in any way.

Parameters

- **entityId** – id of an entity
- **name** – name of the field

Returns Actual or draft field of the given name for given entity id

getAdvancedSettings

AdvancedSettingsDto **getAdvancedSettings** (Long *entityId*)

Retrieves advanced settings for an entity. This will include any draft changes that the current user has made to the entity.

Parameters

- **entityId** – id of an entity

Returns advanced settings for the entity

getAdvancedSettings

AdvancedSettingsDto **getAdvancedSettings** (Long *entityId*, boolean *committed*)

Retrieves advanced settings for an entity.

Parameters

- **entityId** – id of an entity
- **committed** – a flag indicating whether the settings should come from actual entity or a draft

Returns advanced settings for the entity

getCurrentSchemaVersion

Long **getCurrentSchemaVersion** (String *entityClassName*)

Retrieves current version of the entity schema. The version gets incremented each time the entity gets updated. It throws `org.motechproject.mds.ex.entity.EntityNotFoundException` if entity of given class name does not exist.

Parameters

- **entityClassName** – fully qualified class name of the entity

Returns schema version for the entity

getDisplayFields

List<FieldDto> **getDisplayFields** (Long *entityId*)

Retrieves all fields of an entity, that are marked as displayable. By default, these are all the fields that aren't auto-generated by the MDS. The displayable fields can be adjusted using annotations or `addDisplayedFields(org.motechproject.mds.dto.EntityDto, java.util.Map)` method. If entity of given id does not exist, it throws `org.motechproject.mds.ex.entity.EntityNotFoundExcept`

Parameters

- **entityId** – id of an entity

Returns All fields of the entity, that are marked as displayable

getEntity

EntityDto **getEntity** (Long *entityId*)

Returns an entity of the given id. If an entity with given id does not exist, it will return null.

Parameters

- **entityId** – The id of an entity.

Returns Entity with given id or null.

getEntityByClassName

EntityDto **getEntityByClassName** (String *className*)

Retrieves entity by the className parameter

Parameters

- **className** – the className of an entity

Returns Entity with the given className

getEntityDraft

EntityDraft **getEntityDraft** (Long *entityId*)

Retrieves the entity draft. The user, for which the draft should be obtained, will be determined on the current security context.

Parameters

- **entityId** – id of the draft or actual entity

Returns Entity draft for the user

getEntityDraft

EntityDraft **getEntityDraft** (Long *entityId*, String *username*)

Retrieves the entity draft for the given user.

Parameters

- **entityId** – id of the draft or actual entity

Returns Entity draft for the user

getEntityFieldById

FieldDto **getEntityFieldById** (Long *entityId*, Long *fieldId*)

Retrieves a field by id. This will not include draft fields.

Parameters

- **entityId** – id of an entity
- **fieldId** – id of the field

Returns Field of the given name for given entity id

getEntityFields

List<FieldDto> **getEntityFields** (Long *entityId*)

Retrieves a list of all fields for the given entity. This will not include any draft fields.

Parameters

- **entityId** – id of an entity

Returns a list of fields for an entity

getEntityFieldsByClassName

List<FieldDto> **getEntityFieldsByClassName** (String *className*)

Retrieves a list of all fields for the given entity class name. This will not include any draft fields.

Parameters

- **className** – the entity class name

Returns a list of fields for an entity

getEntityForEdit

EntityDto **getEntityForEdit** (Long *entityId*)

Retrieves a draft entity representation, connected to the currently logged in user.

Parameters

- **entityId** – id of an entity to retrieve

Returns draft entity representation

getEntityLookups

List<LookupDto> **getEntityLookups** (Long *entityId*)

Retrieves a list of all lookups for the given entity. This will not include draft lookups.

Parameters

- **entityId** – id of an entity

Returns a list of lookups for an entity

getFields

List<FieldDto> **getFields** (Long *entityId*)

Retrieves a list of all fields for the given entity. This will include draft fields, that the current user has added, deleted or modified in any way.

Parameters

- **entityId** – id of an entity

Returns a list of fields for an entity

getLookupByName

LookupDto **getLookupByName** (Long *entityId*, String *lookupName*)

Retrieves lookup representation by entity id and lookup name. If entity of given id does not exists, it throws `org.motechproject.mds.ex.entity.EntityNotFoundException`. If there is no lookup of such name in the entity, it returns `null`.

Parameters

- **entityId** – id of an entity
- **lookupName** – name of a lookup to retrieve

Returns Lookup representation, or null if a lookup of given name does not exist

getLookupFieldsMapping

Map<String, FieldDto> **getLookupFieldsMapping** (Long *entityId*, String *lookupName*)

Returns a map which contains lookup fields. Fields may come from related entities. Map keys represents lookup fields name which can contains a dot operator(for example relationshipField.id).

Parameters

- **entityId** – The id of an entity
- **lookupName** – name of a lookup

Returns a map of lookup fields.

listEntities

List<EntityDto> **listEntities** ()

Returns all entities, that are currently stored in the database. This will not return Entity drafts and Entity audit. It will also not perform any security checks on the entities.

Returns A list of all entities.

listEntities

List<EntityDto> **listEntities** (boolean *withSecurityCheck*)

Returns all entities, that are currently stored in the database. Allows to filter out entities, that the current user does not have access to. This will not return Entity drafts and Entity audit.

Parameters

- **withSecurityCheck** – set to true, if you wish to filter out entities, that current user does not have access to

Returns A list of all entities, that currently logged user has got access to.

listWorkInProgress

List<EntityDto> **listWorkInProgress** ()

Retrieves a list of all entities, that currently authenticated user has applied changes to.

Returns A list of entities, modified by the user

safeGetAdvancedSettingsCommitted

AdvancedSettingsDto **safeGetAdvancedSettingsCommitted** (String *entityClassName*)

Returns the advanced settings for the entity with the given class name. This method is safe, meaning that it will return null for non-existent entities.

Parameters

- **entityClassName** – the class name of the entity

Returns the advanced settings of the entity, or null if the entity does not exist

saveDraftEntityChanges

DraftResult **saveDraftEntityChanges** (Long *entityId*, DraftData *draftData*, String *username*)

Creates, updates or removes draft data for the user. If there's no draft for given user, it will be created. If a draft already exists, the existing draft will get updated.

Parameters

- **entityId** – id of an actual entity
- **draftData** – data representing changes to the entity
- **username** – the username to whom draft will be assigned

Returns The result, indicating whether changes have been made and whether a draft is outdated

saveDraftEntityChanges

DraftResult **saveDraftEntityChanges** (Long *entityId*, DraftData *draftData*)

Creates, updates or removes draft data. The username will be retrieved from the existing security context. If there's no draft for given user, it will be created. If a draft already exists, the existing draft will get updated.

Parameters

- **entityId** – id of an actual entity
- **draftData** – data representing changes to the entity

Returns The result, indicating whether changes have been made and whether a draft is outdated

updateComboboxValues

void **updateComboboxValues** (Long *entityId*, Map<String, Collection> *fieldValuesToUpdate*)

Updates available combobox choices. Allows users to extend available values, for comboboxes that allow user-supplied values.

Parameters

- **entityId** – id of an entity
- **fieldValuesToUpdate** – a map of the field names and values to add to these fields

Throws

- **java.lang.IllegalArgumentException** – when field passed to the method is not of the combobox type
- **org.motechproject.mds.ex.entity.EntityNotFoundException** – when entity of the given id does not exist
- **org.motechproject.mds.ex.field.FieldNotFoundException** – when field of the passed name does not exist

updateDraft

EntityDto **updateDraft** (Long *entityId*)

Updates draft entity for the user, determined on the current security context. The update changes the parent entity of the draft to the latest version, which may happen if another user commits changes to the entity.

Parameters

- **entityId** – id of an entity

Returns updated draft entity

updateMaxFetchDepth

void **updateMaxFetchDepth** (Long *entityId*, Integer *maxFetchDepth*)

Updated the max fetch depth for a given entity. That fetch depth will be passed to the fetch plan of the persistence manager for that entity.

Parameters

- **entityId** – the id of the entity to update
- **maxFetchDepth** – the new maximum fetch depth

updateRestOptions

void **updateRestOptions** (Long *entityId*, RestOptionsDto *restOptionsDto*)

Updates rest options for the given entity. If entity of the given id does not exist, it throws `org.motechproject.mds.ex.entity.EntityNotFoundException`

Parameters

- **entityId** – id of an entity
- **restOptionsDto** – new rest options

updateSecurityOptions

void **updateSecurityOptions** (Long *entityId*, SecurityMode *securityMode*, Set<String> *securityMembers*, SecurityMode *readOnlySecurityMode*, Set<String> *readOnlySecurityMembers*)

Updates security options for the given entity. If entity of the given id does not exist, it throws `org.motechproject.mds.ex.entity.EntityNotFoundException`

Parameters

- **entityId** – id of an entity
- **securityMode** – new security mode
- **securityMembers** – set of user or role names
- **readOnlySecurityMode** – new read only security mode
- **readOnlySecurityMembers** – set of user or role names for read only security mode

updateTracking

void **updateTracking** (Long *entityId*, TrackingDto *trackingDto*)

Updates audit settings for the given entity. If entity of the given id does not exist, it throws `org.motechproject.mds.ex.entity.EntityNotFoundException`

Parameters

- **entityId** – id of an entity
- **trackingDto** – new audit settings

12.60.10 HistoryService

public interface **HistoryService**

The `HistoryService` provides methods related with processing historical changes on the given instance of entity.

Methods

countHistoryRecords

long **countHistoryRecords** (Object *instance*)

Returns a number of historical revisions for the given instance.

Parameters

- **instance** – an instance to count historical revisions for

Returns a number of historical revisions of the instance

getHistoryForInstance

List **getHistoryForInstance**(Object *instance*, QueryParams *queryParams*)

Returns the historical data for the given instance. This method return historical data only for objects that are not in the MDS trash. For trash instances the return value will be incorrect.

Parameters

- **instance** – an instance created from the given entity definition.
- **queryParams** – Query parameters such as page number, size of page and sort direction. If null method will return all history records.

Returns a list of historical data related with the given instance.

getSingleHistoryInstance

Object **getSingleHistoryInstance**(Object *instance*, Long *historyId*)

Returns a single historical revision for the given instance.

Parameters

- **instance** – an instance to retrieve historical revisions from
- **historyId** – id of a historical revision

Returns Historical revision or null if not found

record

void **record**(Object *instance*)

Records changes made on the given instance of entity. The first historical data should be equal to data inside the given instance. Two instance of historical data should be connected using appropriate fields (defined in history class definition). This method should be used only for instances that are not in the MDS trash.

Parameters

- **instance** – an instance created from the given entity definition.

remove

void **remove**(Object *instance*)

Removes all historical data with information what changes were made on the given instance of entity.

Parameters

- **instance** – an instance created from the given entity definition.

setTrashFlag

void **setTrashFlag** (Object *instance*, Object *trash*, boolean *flag*)

Sets the trash flag for historical data related with the given instance object.

Parameters

- **instance** – an instance created from the given entity definition.
- **flag** – true if instance was moved to trash; otherwise false.

12.60.11 ImportExportService

public interface **ImportExportService**

The `ImportExportService` interface provides methods for importing and exporting entities schema and data in json format.

See also: `org.motechproject.mds.domain.ImportExportBlueprint`

Methods

exportEntities

void **exportEntities** (ImportExportBlueprint *blueprint*, Writer *writer*)

Exports entities schema and/or instances based on provided blueprint.

Parameters

- **blueprint** – export blueprint containing information which entities to export and what to include
- **writer** – the writer used for output

importEntities

void **importEntities** (String *importId*, ImportExportBlueprint *blueprint*)

Imports entities schema and/or instances from previously stored import file.

Parameters

- **importId** – previously saved import file id
- **blueprint** – import blueprint containing information which entities to import and what to include

saveImportFileAndExtractManifest

ImportManifest **saveImportFileAndExtractManifest** (byte[] *bytes*)

Saves uploaded import file to temporary location, validates it and extracts `org.motechproject.mds.domain.ImportManifest` from it.

Parameters

- **bytes** – binary file representation

Returns import manifest extracted from saved file

12.60.12 JarGeneratorService

public interface **JarGeneratorService**

This interface provides methods to create a bundle jar with all entities defined in MDS module.

Fields

BLUEPRINT_TEMPLATE

String **BLUEPRINT_TEMPLATE**

BLUEPRINT_XML

String **BLUEPRINT_XML**

BUNDLE_IMPORTS

String **BUNDLE_IMPORTS**

DATANUCLEUS_PROPERTIES

String **DATANUCLEUS_PROPERTIES**

ENTITY_LIST_FILE

String **ENTITY_LIST_FILE**

HISTORY_LIST_FILE

String **HISTORY_LIST_FILE**

LISTENER_LIST_FILE

String **LISTENER_LIST_FILE**

MDS_CHANNEL_TEMPLATE

String **MDS_CHANNEL_TEMPLATE**

MDS_COMMON_CONTEXT

String **MDS_COMMON_CONTEXT**

MDS_ENTITIES_CONTEXT

String **MDS_ENTITIES_CONTEXT**

MDS_ENTITIES_CONTEXT_TEMPLATE

String **MDS_ENTITIES_CONTEXT_TEMPLATE**

MOTECH_MDS_PROPERTIES

String **MOTECH_MDS_PROPERTIES**

PACKAGE_JDO

String **PACKAGE_JDO**

TASK_CHANNEL_JSON

String **TASK_CHANNEL_JSON**

VALIDATION_PROVIDER

String **VALIDATION_PROVIDER**

Methods

generate

File **generate**()

Generates a jar file that contains entity class definitions, repositories, interfaces, implementations of these interfaces. The jar should also contains class related with historical data and trash.

Throws

- **IOException** – if an I/O error occurs while creating the jar file.

Returns file that points to the entities bundle jar.

regenerateMdsDataBundle

void **regenerateMdsDataBundle**()

Constructs entities, builds and starts the entities bundle jar

See also: `.generate()`

regenerateMdsDataBundle

void **regenerateMdsDataBundle** (boolean *startBundle*)

Constructs entities, builds the entities bundle jar. The generated bundle will start only if the **startBundle** will be set to `true`.

Parameters

- **startBundle** – `true` if the generated bundle should start; otherwise `false`.

See also: `.generate()`

regenerateMdsDataBundleAfterDdeEnhancement

void **regenerateMdsDataBundleAfterDdeEnhancement** (String... *moduleNames*)

Constructs entities, builds and starts the entities bundle jar. This method should be used after DDE enhancement. It will build all DDE classes and refresh modules from which the DDE being enhanced comes from.

Parameters

- **moduleNames** – modules names of the entities from which the enhanced DDE comes from

See also: `.generate()`

12.60.13 JdoListenerRegistryService

public interface **JdoListenerRegistryService**

Gives access to the registry of listeners for persistence events.

Methods

getEntitiesListenerStr

String **getEntitiesListenerStr** ()

Gets entities from listeners in one string, where every entity name is in a new line.

Returns the entities from listeners

getListeners

List<MotechLifecycleListener> **getListeners** ()

Gets the listeners from the registry.

Returns the list of listeners

getListeners

List<MotechLifecycleListener> **getListeners** (String *entity*, InstanceLifecycleListenerType *type*)

Gets the list of listeners for the given entity and type of persistence event.

Parameters

- **entity** – the class name of persistence object

- **type** – the type of persistence event

Returns the list of listeners

getMethods

Set<String> **getMethods** (MotechLifecycleListener *listener*, InstanceLifecycleListenerType *type*)

Gets the list of methods from the listener for the given type of persistence event.

Parameters

- **listener** – the listener for persistence object
- **type** – the type of persistence event

Returns the list of methods

registerEntityWithListeners

void **registerEntityWithListeners** (String *entity*)

Adds the given entity to the list of entities for which there may exist instance lifecycle listeners.

Parameters

- **entity** – the class name of the entity to add

registerListener

void **registerListener** (MotechLifecycleListener *listener*)

Registers the listener. If the registry already has listener for this type of persistence event, the methods from the given listener will be added to the existed one.

Parameters

- **listener** – the listener to be registered

removeInactiveListeners

void **removeInactiveListeners** (String *entitiesNames*)

Removes inactive listeners in the registry. This method checks if entities in listeners are still persistable classes.

Parameters

- **entitiesNames** – the names of all active entities

removeListener

void **removeListener** (MotechLifecycleListener *listener*)

Removes the listener from the registry.

Parameters

- **listener** – the listener to be removed

updateEntityNames

void **updateEntityNames** ()
Updates entity names for package listeners

12.60.14 JsonLookupService

public interface **JsonLookupService**
Service for managing lookups coming from JSON files.

Methods

createJsonLookup

void **createJsonLookup** (JsonLookupDto *jsonLookup*)
Stores the given `jsonLookup` in the database.

Parameters

- **jsonLookup** – the lookup to be stored.

exists

boolean **exists** (String *entityClassName*, String *originLookupName*)
Checks if a lookup with the given `originLookupName` was already added for the entity with the given `entityClassName`.

Parameters

- **entityClassName** – the class name of the entity
- **originLookupName** – the origin name of the lookup

Returns true if the lookup was added, false otherwise

12.60.15 MDSLookupService

public interface **MDSLookupService**
This service allows executing lookups on entities given their classes or class names and lookup names as Strings. Allows generic access to any entity in MDS. This is just a facade and all data access goes through the underlying data service. EUDE can be identified either by their fully qualified class name (eg: "org.motechproject.mds.entity.Patient") or by their entity name (eg: "Patient")

Methods

count

long **count** (Class *entityClass*, String *lookupName*, Map<String, ?> *lookupParams*)
Retrieves a total number of instances, that match the specified lookup parameters, for the given lookup and entity. This will fail if specified lookup parameters do not match the lookup definition or if the lookup of given name is not specified for the given entity.

Parameters

- **entityClass** – entity class
- **lookupName** – name of the lookup from entity
- **lookupParams** – parameters to use, when executing the lookup

Returns number of instances

count

long **count** (String *entityClassName*, String *lookupName*, Map<String, ?> *lookupParams*)

Retrieves a total number of instances, that match the specified lookup parameters, for the given lookup and entity class name. This will fail if specified lookup parameters do not match the lookup definition or if the lookup of given name is not specified for the given entity.

Parameters

- **entityClassName** – entity class name
- **lookupName** – name of the lookup from entity
- **lookupParams** – parameters to use, when executing the lookup

Returns number of instances

countAll

long **countAll** (Class *entityClass*)

Retrieves a total number of instances, for the given entity class.

Parameters

- **entityClass** – entity class

Returns number of instances

countAll

long **countAll** (String *entityClassName*)

Retrieves a total number of instances, for the given entity class name.

Parameters

- **entityClassName** – entity class name

Returns number of instances

findMany

<T> List<T> **findMany** (Class<T> *entityClass*, String *lookupName*, Map<String, ?> *lookupParams*)

Retrieves and executes multi-return lookup for the given entity class, lookup name and parameters. It will fail, if lookup parameters do not match the parameters specified in the lookup or if the lookup of given name does not exist for the retrieved entity.

Parameters

- **entityClass** – entity class

- **lookupName** – name of the lookup from entity
- **lookupParams** – parameters to use, when executing the lookup
- **<T>** – entity class

Returns collection of instances, retrieved using given lookup criteria

findMany

<T> List<T> **findMany**(String *entityClassName*, String *lookupName*, Map<String, ?> *lookupParams*)

Retrieves and executes multi-return lookup for the given entity class name, lookup name and parameters. It will fail, if lookup parameters do not match the parameters specified in the lookup or if the lookup of given name does not exist for the retrieved entity.

Parameters

- **entityClassName** – entity class name
- **lookupName** – name of the lookup from entity
- **lookupParams** – parameters to use, when executing the lookup
- **<T>** – entity class

Returns collection of instances, retrieved using given lookup criteria

findMany

<T> List<T> **findMany**(Class<T> *entityClass*, String *lookupName*, Map<String, ?> *lookupParams*, QueryParams *queryParams*)

Retrieves and executes multi-return lookup for the given entity class, lookup name and parameters. It will fail, if lookup parameters do not match the parameters specified in the lookup or if the lookup of given name does not exist for the retrieved entity. This version additionally allows to use query parameters, to adjust retrieved instances (eg. limit their number).

Parameters

- **entityClass** – entity class
- **lookupName** – name of the lookup from entity
- **lookupParams** – parameters to use, when executing the lookup
- **queryParams** – parameters to use, retrieving the instances
- **<T>** – entity class

Returns collection of instances, retrieved using given lookup criteria

findMany

<T> List<T> **findMany**(String *entityClassName*, String *lookupName*, Map<String, ?> *lookupParams*, QueryParams *queryParams*)

Retrieves and executes multi-return lookup for the given entity class name, lookup name and parameters. It will fail, if lookup parameters do not match the parameters specified in the lookup or if the lookup of given name does not exist for the retrieved entity. This version additionally allows to use query parameters, to adjust retrieved instances (eg. limit their number).

Parameters

- **entityClassName** – entity class name
- **lookupName** – name of the lookup from entity
- **lookupParams** – parameters to use, when executing the lookup
- **queryParams** – parameters to use, retrieving the instances
- **<T>** – entity class

Returns collection of instances, retrieved using given lookup criteria

findOne

<T> T **findOne** (Class<T> *entityClass*, String *lookupName*, Map<String, ?> *lookupParams*)

Retrieves and executes single-return lookup for the given entity class, lookup name and parameters. It will fail, if lookup parameters do not match the parameters specified in the lookup or if the lookup of given name does not exist for the retrieved entity. It will also throw `org.motechproject.mds.ex.lookup.SingleResultFromLookupExpectedException` in case lookup returns a collection of instances, rather than single instance.

Parameters

- **entityClass** – entity class
- **lookupName** – name of the lookup from entity
- **lookupParams** – parameters to use, when executing the lookup
- **<T>** – entity class

Returns Single instance, retrieved using given lookup criteria

findOne

<T> T **findOne** (String *entityClassName*, String *lookupName*, Map<String, ?> *lookupParams*)

Retrieves and executes single-return lookup for the given entity class name, lookup name and parameters. It will fail, if lookup parameters do not match the parameters specified in the lookup or if the lookup of given name does not exist for the retrieved entity. It will also throw `org.motechproject.mds.ex.lookup.SingleResultFromLookupExpectedException` in case lookup returns a collection of instances, rather than single instance.

Parameters

- **entityClassName** – entity class name
- **lookupName** – name of the lookup from entity
- **lookupParams** – parameters to use, when executing the lookup
- **<T>** – entity class

Returns Single instance, retrieved using given lookup criteria

retrieveAll

<T> List<T> **retrieveAll** (Class<T> *entityClass*)

Retrieves all instances for the given entity class.

Parameters

- **entityClass** – entity class
- **<T>** – entity class

Returns a list of all instances for the given entity

retrieveAll

<T> List<T> **retrieveAll** (String *entityClassName*)

Retrieves all instances for the given entity class name.

Parameters

- **entityClassName** – entity class name
- **<T>** – entity class

Returns a list of all instances for the given entity

retrieveAll

<T> List<T> **retrieveAll** (Class<T> *entityClass*, QueryParams *queryParams*)

Retrieves all instances for the given entity class name. This version additionally allows to use query parameters, to adjust retrieved instances (eg. limit their number).

Parameters

- **entityClass** – entity class
- **<T>** – entity class

Returns a list of all instances for the given entity

retrieveAll

<T> List<T> **retrieveAll** (String *entityClassName*, QueryParams *queryParams*)

Retrieves all instances for the given entity class name. This version additionally allows to use query parameters, to adjust retrieved instances (eg. limit their number).

Parameters

- **entityClassName** – entity class name
- **<T>** – entity class

Returns a list of all instances for the given entity

12.60.16 MdsBundleRegenerationService

public interface **MdsBundleRegenerationService**

The `MdsBundleRegenerationService` interface provides methods for regenerating MDS Entities Bundle and commands all Motech instances to do the same.

Fields

REGENERATE_MDS_DATA_BUNDLE

String **REGENERATE_MDS_DATA_BUNDLE**

REGENERATE_MDS_DATA_BUNDLE_AFTER_DDE_ENHANCEMENT

String **REGENERATE_MDS_DATA_BUNDLE_AFTER_DDE_ENHANCEMENT**

Methods

regenerateMdsDataBundle

void **regenerateMdsDataBundle**()

Constructs entities, builds and starts the MDS Entities Bundle, commands other Motech instances to regenerate their MDS Entities Bundle.

regenerateMdsDataBundleAfterDdeEnhancement

void **regenerateMdsDataBundleAfterDdeEnhancement**(String... *moduleNames*)

Constructs entities, builds and starts the MDS Entities Bundle, commands other Motech instances to regenerate their MDS Entities Bundle. This method should be used after DDE enhancement. It will build all DDE classes and refresh modules from which the DDE being enhanced comes from.

Parameters

- **moduleNames** – modules names of the entities from which the enhanced DDE comes from

12.60.17 MdsSchedulerService

public interface **MdsSchedulerService**

The `MdsSchedulerService` provides methods for scheduling and unscheduling jobs. We do not use the MOTECH scheduler, to avoid circular dependencies. This service only allows to schedule MDS-specific jobs.

Methods

scheduleRepeatingJob

void **scheduleRepeatingJob**(long *interval*)

Schedules a job, responsible for periodic emptying of the MDS trash. Throws `java.lang.IllegalArgumentException` if the passed interval is set to 0.

Parameters

- **interval** – interval between next fires, in milliseconds

unscheduleRepeatingJob

void **unscheduleRepeatingJob** ( )

Unschedules a job, responsible for periodic emptying of the MDS trash.

12.60.18 MigrationService

public interface **MigrationService**

This interface provides method for finding flyway migrations within bundle. Default search location is db/migration.

See also: `org.motechproject.mds.jdo.SchemaGenerator`, `org.motechproject.mds.domain.Migration`

Methods

processBundle

void **processBundle** (Bundle *bundle*)

Finds migration files in the given bundle and copy them to the .motech/migration directory. This method also updates migration mapping.

Parameters

- **bundle** – the bundle to process.

Throws

- **IOException** – if an I/O error occurs while copying migration files.

12.60.19 MotechDataService

public interface **MotechDataService**<T>

This is a basic service interface with CRUD operations. Mainly it is used as super interface to create service interface related with the given entity schema in `org.motechproject.mds.builder.EntityInfrastructureBuilder` but it can be also used by other service interfaces inside this package.

Parameters

- **<T>** – the type of entity schema.

Methods

count

long **count** ( )

Gets the total number of instances.

Returns number of instances

countForFilters

long **countForFilters** (Filters *filters*)

Gets a total number of instances, after being filtered by the given filter.

Parameters

- **filters** – filters to use

Returns number of filtered instances

create

T **create** (T *object*)

Creates the given instance in MDS.

Parameters

- **object** – instance to create

Returns created instance

createOrUpdate

T **createOrUpdate** (T *object*)

Updates the given instance in MDS if it exists (checks the presence of the instances id to verify that) or creates a new one if it doesn't.

Parameters

- **object** – instance to update or create

Returns updated or created instance

delete

void **delete** (T *object*)

Deletes given instance from MDS.

Parameters

- **object** – instance to delete

delete

void **delete** (String *primaryKeyName*, Object *value*)

Deletes instance from MDS, by its primary key value.

Parameters

- **primaryKeyName** – datastore primary key name
- **value** – value of the primary key

deleteAll

void **deleteAll**()
Removes all instances of type {@value T} from MDS.

deleteById

void **deleteById**(long *id*)
Deletes instance from MDS, by its id.

Parameters

- **id** – id of the instance to delete.

doInTransaction

<R> R **doInTransaction**(TransactionCallback<R> *transactionCallback*)
Allows to wrap several instructions into a single transaction. Developers should implement the `TransactionCallback` interface and override the `TransactionCallback.doInTransaction(org.springframework.transaction.TransactionStatus)` method with whatever should be done in the transaction.

Parameters

- **transactionCallback** – implementation of the `TransactionCallback`
- **<R>** – type that should be returned from the transaction

Returns anything of type {@value R}. Left to the developer, implementing the transaction

executeQuery

<R> R **executeQuery**(QueryExecution<R> *queryExecution*)
Allows to execute custom query in MDS. Users are supposed to implement the `QueryExecution` interface and override its `QueryExecution.execute(javax.jdo.Query,org.motechproject.mds.util.InstanceSe` method with their custom behaviour.

Parameters

- **queryExecution** – implementation of the `QueryExecution`, with custom behaviour
- **<R>** – type that should be returned from the custom query

Returns anything of type {@value R}. Left to the developer, implementing the custom query.

executeSQLQuery

<R> R **executeSQLQuery**(SqlQueryExecution<R> *queryExecution*)
Allows to execute custom SQL query in MDS. Users should implement the `SqlQueryExecution` interface and override its methods, defining their custom query.

Parameters

- **queryExecution** – implementation of the `SqlQueryExecution`
- **<R>** – type that should be returned by the custom sql query

Returns anything of type {@value R}, left to the developer, implementing the custom sql query.

filter

List<T> **filter** (Filters *filters*, QueryParams *queryParams*)

Retrieves all instances of type {@value T} from MDS, filtered using specified filters and query params.

Parameters

- **filters** – filters to use filtering instances
- **queryParams** – query parameters to use filtering instances

Returns a list of instances, filtered using specified parameters

findById

T **findById** (Long *id*)

Retrieves instance of type {@value T} and given id from MDS.

Parameters

- **id** – id of the instance

Returns instance with the given id

findTrashInstanceById

T **findTrashInstanceById** (Object *instanceId*, Object *entityId*)

Retrieves an instance, that has been moved to trash, by its id. These instances are not retrieved with other retrieve methods.

Parameters

- **instanceId** – id of the instance, that has been moved to trash
- **entityId** – id of the entity

Returns instance of the given id, from trash

getClassType

Class<T> **getClassType** ()

Returns class type assigned to this service.

Returns class type

getDetachedField

Object **getDetachedField** (T *instance*, String *fieldName*)

Makes instance persistent and retrieves field values from that persisted instance.

Parameters

- **instance** – instance to retrieve field value from
- **fieldName** – name of the field to retrieve

Returns value from the field

retrieve

T **retrieve** (String *primaryKeyName*, Object *value*)

Retrieves instance from MDS based on the value of the given primary key.

Parameters

- **primaryKeyName** – datastore primary key name
- **value** – value of the primary key

Returns instance with the given value for the specified primary key

retrieveAll

List<T> **retrieveAll** ()

Retrieves all instances of the {@value T} type.

Returns all instances

retrieveAll

List<T> **retrieveAll** (QueryParams *queryParams*)

Retrieves all instances of the {@value T} type, that match the provided parameters.

Parameters

- **queryParams** – query parameters to be used retrieving instances

Returns all isntances matching query parameters

revertFromTrash

void **revertFromTrash** (Object *newInstance*, Object *trash*)

Brings back instance from trash. This will be in fact a new instance, that has got exactly the same values as previous instance, except of its id.

Parameters

- **newInstance** – new instance representation
- **trash** – instance from the trash

update

T **update** (T *object*)

Updates the given instance in MDS.

Parameters

- **object** – instance to update

Returns updated instance

updateFromTransient

T **updateFromTransient** (T *transientObject*)

Returns the persistent instance, updated with the values from the transient instance. If there's no instance of the id from the transient instance, it will create one.

Parameters

- **transientObject** – transient object, from which an update will take place

Returns persistent instance, updated with the values from the transient instance

updateFromTransient

T **updateFromTransient** (T *transientObject*, Set<String> *fieldsToUpdate*)

Returns the persistent instance, updated with the values from the transient instance. If there's no instance of the id from the transient instance, it will create one. Only fields with the names passed to the method will be updated.

Parameters

- **transientObject** – transient object, from which an update will take place
- **fieldsToUpdate** – set of field names that should be updated

Returns persistent instance, updated with the values from the transient instance

12.60.20 RestDocumentationService

public interface **RestDocumentationService**

This service allows retrieval of dynamically generated MDS REST documentation. This is an OSGi service interface, it is used by the mds-web module to serve the documentation through HTTP. The documentation returned is a JSON representation of the API in Swagger json format.

Methods

retrieveDocumentation

void **retrieveDocumentation** (Writer *writer*, String *serverPrefix*, Locale *locale*)

Writes REST API documentation the documentation to the writer provided.

Parameters

- **writer** – the output for the documentation.
- **serverPrefix** – the prefix of the server, for example /motech-platform-server, will be used in the swagger spec
- **locale** – the locale to be used while generating REST documentation

12.60.21 TransactionalMotechDataService

public abstract class **TransactionalMotechDataService**<T> extends DefaultMotechDataService<T>

The main goal of the `TransactionalMotechDataService` class is to resolve problems with transaction annotations not working for generated lookups. We use the traditional transaction callback instead.

Parameters

- **<T>** – the type of entity schema.

Methods

count

protected long **count** (List<Property> *properties*)

retrieveAll

protected List<T> **retrieveAll** (List<Property> *properties*)

retrieveAll

protected List<T> **retrieveAll** (List<Property> *properties*, QueryParams *queryParams*)

retrieveUnique

protected T **retrieveUnique** (List<Property> *properties*, QueryParams *queryParams*)

retrieveUnique

protected T **retrieveUnique** (List<Property> *properties*)

12.60.22 TrashService

public interface **TrashService**

The `TrashService` provides methods related with the module trash mode (by default the mode is active and it can be turned off by the user).

Methods

countTrashRecords

long **countTrashRecords** (String *className*)

Gets a number of instances moved to trash, for entity with given class name. This will only consider the instances, that have been moved to trash on the current entity schema version.

Parameters

- **className** – fully qualified entity class name

Returns trash instances count

emptyTrash

void **emptyTrash**()

Cleans the module trash. All instances in trash should be removed permanently and if they contain any historical data they should also be removed permanently.

This method should only be executed by the job created in the `scheduleEmptyTrashJob()` method.

findTrashById

Object **findTrashById**(Object *instanceId*, Object *entityId*)

Return instance with given id from trash.

Parameters

- **instanceId** – id of instance
- **entityId** – id of instance entity

getInstancesFromTrash

Collection **getInstancesFromTrash**(String *entityName*, QueryParams *queryParams*)

Returns the collection of instances from trash of a certain entity. Returned collection contains only instances that are on the current schema version.

Parameters

- **entityName** – Instances of what entity should be looked for
- **queryParams** – Query parameters such as page number, size of page and sort direction. If null method will return all records in trash.

Returns Collection of instances on the current schema version in trash

isTrashMode

boolean **isTrashMode**()

Checks if trash mode is active. This method should be used before executing the `moveToTrash(Object,Long,boolean)` method to resolve whether the given instance should be moved to trash or removed permanently.

Returns true if delete mode is equal to `org.motechproject.mds.config.DeleteMode.TRASH`; false otherwise.

moveFromTrash

void **moveFromTrash**(Object *newInstance*, Object *trash*, boolean *recordHistory*)

Sets history for given trashed instance to match the new one and deletes trashed one from trash.

Parameters

- **newInstance** – instance to be returned from trash
- **trash** – trashed instance to be removed
- **recordHistory** – true if entity has active history recording ; otherwise false

moveToTrash

void **moveToTrash** (Object *instance*, Long *schemaVersion*, boolean *recordHistory*)

Moves the given instance to the trash. This method should only be executed, when the module trash mode is active.

Parameters

- **instance** – an instance created from the given entity definition.
- **recordHistory** – true if entity has active history recording ; otherwise false

See also: `.isTrashMode()`

scheduleEmptyTrashJob

void **scheduleEmptyTrashJob** ()

Sets the repeating schedule job that will be executed from time to time. Execution time depends on the value of time value and time unit (defined in `org.motechproject.mds.util.Constants.Config.MODULE_FILE`).

Before scheduling new job, the old one should be unscheduled to prevent the errors.

12.60.23 TypeService

public interface **TypeService**

The `TypeService` is an interface defining available methods to execute various actions on Field Types.

Methods

findType

TypeDto **findType** (Class<?> *clazz*)

Retrieves MDS type, based on the class that handles that type in the backend. Throws `org.motechproject.mds.ex.type.TypeNotFoundException` when the given class does not handle any MDS type.

Parameters

- **clazz** – handler class

Returns MDS type that is handled by the given class

findValidations

List<TypeValidation> **findValidations** (TypeDto *type*, Class<? extends Annotation> *aClass*)

Retrieves all MDS validations for the given type, that are triggered by the given annotation.

Parameters

- **type** – MDS type representation
- **aClass** – Annotation class type

Returns A list of validations that match the criteria or empty list, if none were found

See also: `org.motechproject.mds.domain.TypeValidation`

getAllTypes

List<TypeDto> **getAllTypes** ()
Retrieves all available MDS types.

Returns a list of types

getType

Type **getType** (TypeValidation *validation*)
Retrieves MDS Type, connected to the given validation.

Parameters

- **validation** – Validation representation

Returns MDS Type that is connected to this validation

See also: `org.motechproject.mds.domain.TypeValidation`

12.61 org.motechproject.mds.service.impl

12.61.1 ActionHandlerServiceImpl

public class **ActionHandlerServiceImpl** implements ActionHandlerService
Default implementation of ActionHandlerService interface

See also: `org.motechproject.mds.service.ActionHandlerService`

Methods

create

public void **create** (Map<String, Object> *parameters*)

delete

public void **delete** (Map<String, Object> *parameters*)

setAllEntities

public void **setAllEntities** (AllEntities *allEntities*)

setBundleContext

public void **setBundleContext** (BundleContext *bundleContext*)

update

public void **update** (Map<String, Object> *parameters*)

12.61.2 BundleWatcherSuspensionServiceImpl

public class **BundleWatcherSuspensionServiceImpl** implements BundleWatcherSuspensionService

Methods

restoreBundleProcessing

public void **restoreBundleProcessing** ()

setMdsBundleWatcher

public void **setMdsBundleWatcher** (MdsBundleWatcher *mdsBundleWatcher*)

suspendBundleProcessing

public void **suspendBundleProcessing** ()

12.61.3 EntityServiceImpl

public class **EntityServiceImpl** implements EntityService

Default implementation of `org.motechproject.mds.service.EntityService` interface.

Methods

abandonChanges

public void **abandonChanges** (Long *entityId*)

addDisplayedFields

public void **addDisplayedFields** (EntityDto *entityDto*, Map<String, Long> *positions*)

addFields

public void **addFields** (EntityDto *entity*, Collection<FieldDto> *fields*)

addFields

public void **addFields** (Long *entityId*, FieldDto... *fields*)

addFields

public void **addFields** (EntityDto *entity*, FieldDto... *fields*)

addFields

public void **addFields** (Long *entityId*, Collection<FieldDto> *fields*)

addFilterableFields

public void **addFilterableFields** (EntityDto *entityDto*, Collection<String> *fieldNames*)

addLookups

public void **addLookups** (EntityDto *entityDto*, LookupDto... *lookups*)

addLookups

public void **addLookups** (EntityDto *entityDto*, Collection<LookupDto> *lookups*)

addLookups

public void **addLookups** (Long *entityId*, LookupDto... *lookups*)

addLookups

public void **addLookups** (Long *entityId*, Collection<LookupDto> *lookups*)

addNonEditableFields

public void **addNonEditableFields** (EntityDto *entityDto*, Map<String, Boolean> *nonEditableFields*)

commitChanges

public List<String> **commitChanges** (Long *entityId*, String *changesOwner*)

commitChanges

public List<String> **commitChanges** (Long *entityId*)

createEntity

public EntityDto **createEntity** (EntityDto *entityDto*)

deleteEntity

public void **deleteEntity** (Long *entityId*)

findEntitiesByPackage

public List<EntityDto> **findEntitiesByPackage** (String *packageName*)

findEntityFieldByName

public FieldDto **findEntityFieldByName** (Long *entityId*, String *name*)

findFieldByName

public FieldDto **findFieldByName** (Long *entityId*, String *name*)

getAdvancedSettings

public AdvancedSettingsDto **getAdvancedSettings** (Long *entityId*)

getAdvancedSettings

public AdvancedSettingsDto **getAdvancedSettings** (Long *entityId*, boolean *committed*)

getCurrentSchemaVersion

public Long **getCurrentSchemaVersion** (String *className*)

getDisplayFields

public List<FieldDto> **getDisplayFields** (Long *entityId*)

getEntity

public EntityDto **getEntity** (Long *entityId*)

getEntityByClassName

public EntityDto **getEntityByClassName** (String *className*)

getEntityDraft

public EntityDraft **getEntityDraft** (Long *entityId*)

getEntityDraft

public EntityDraft **getEntityDraft** (Long *entityId*, String *username*)

getEntityFieldById

public FieldDto **getEntityFieldById** (Long *entityId*, Long *fieldId*)

getEntityFields

public List<FieldDto> **getEntityFields** (Long *entityId*)

getEntityFieldsByClassName

public List<FieldDto> **getEntityFieldsByClassName** (String *className*)

getEntityForEdit

public EntityDto **getEntityForEdit** (Long *entityId*)

getEntityLookups

public List<LookupDto> **getEntityLookups** (Long *entityId*)

getFields

public List<FieldDto> **getFields** (Long *entityId*)

getLookupByName

public LookupDto **getLookupByName** (Long *entityId*, String *lookupName*)

getLookupFieldsMapping

public Map<String, FieldDto> **getLookupFieldsMapping** (Long *entityId*, String *lookupName*)

listEntities

public List<EntityDto> **listEntities** ()

listEntities

public List<EntityDto> **listEntities** (boolean *withSecurityCheck*)

listWorkInProgress

public List<EntityDto> **listWorkInProgress**()

safeGetAdvancedSettingsCommitted

public AdvancedSettingsDto **safeGetAdvancedSettingsCommitted**(String *entityClassName*)

saveDraftEntityChanges

public DraftResult **saveDraftEntityChanges**(Long *entityId*, DraftData *draftData*, String *username*)

saveDraftEntityChanges

public DraftResult **saveDraftEntityChanges**(Long *entityId*, DraftData *draftData*)

setAllEntities

public void **setAllEntities**(AllEntities *allEntities*)

setAllEntityAudits

public void **setAllEntityAudits**(AllEntityAudits *allEntityAudits*)

setAllEntityDrafts

public void **setAllEntityDrafts**(AllEntityDrafts *allEntityDrafts*)

setAllTypes

public void **setAllTypes**(AllTypes *allTypes*)

setBundleContext

public void **setBundleContext**(BundleContext *bundleContext*)

setComboboxDataMigrationHelper

public void **setComboboxDataMigrationHelper**(ComboboxDataMigrationHelper *comboboxDataMigrationHelper*)

setEntityValidator

public void **setEntityValidator**(EntityValidator *entityValidator*)

setMDSConstructor

public void **setMDSConstructor** (MDSConstructor *mdsConstructor*)

updateComboboxValues

public void **updateComboboxValues** (Long *entityId*, Map<String, Collection> *fieldValuesToUpdate*)

updateDraft

public EntityDto **updateDraft** (Long *entityId*)

updateMaxFetchDepth

public void **updateMaxFetchDepth** (Long *entityId*, Integer *maxFetchDepth*)

updateRestOptions

public void **updateRestOptions** (Long *entityId*, RestOptionsDto *restOptionsDto*)

updateSecurityOptions

public void **updateSecurityOptions** (Long *entityId*, SecurityMode *securityMode*, Set<String> *securityMembers*, SecurityMode *readOnlySecurityMode*, Set<String> *readOnlySecurityMembers*)

updateTracking

public void **updateTracking** (Long *entityId*, TrackingDto *trackingDto*)

12.61.4 ImportExportServiceImpl

public class **ImportExportServiceImpl** implements ImportExportService

Implementation of `org.motechproject.mds.service.ImportExportService`.

See also: `org.motechproject.mds.domain.ImportExportBlueprint`, `org.motechproject.mds.json.EntityWriter`, `org.motechproject.mds.json.InstancesWriter`, `com.google.gson.stream.JsonWriter`

Methods

exportEntities

public void **exportEntities** (ImportExportBlueprint *blueprint*, Writer *writer*)

importEntities

public void **importEntities** (String *importId*, ImportExportBlueprint *blueprint*)

saveImportFileAndExtractManifest

public ImportManifest **saveImportFileAndExtractManifest** (byte[] *bytes*)

setAllEntities

public void **setAllEntities** (AllEntities *allEntities*)

setAllTypes

public void **setAllTypes** (AllTypes *allTypes*)

setBundleContext

public void **setBundleContext** (BundleContext *bundleContext*)

setMdsBundleRegenerationService

public void **setMdsBundleRegenerationService** (MdsBundleRegenerationService *mdsBundleRegenerationService*)

setRelationshipResolver

public void **setRelationshipResolver** (RelationshipResolver *relationshipResolver*)

setTransactionManager

public void **setTransactionManager** (JdoTransactionManager *transactionManager*)

12.61.5 JarGeneratorServiceImpl

public class **JarGeneratorServiceImpl** implements JarGeneratorService

Default implementation of `org.motechproject.mds.service.JarGeneratorService` interface.

Methods

generate

public File **generate** ()

regenerateMdsDataBundle

public synchronized void **regenerateMdsDataBundle** ( )

regenerateMdsDataBundle

public void **regenerateMdsDataBundle** (boolean *startBundle*)

regenerateMdsDataBundleAfterDdeEnhancement

public void **regenerateMdsDataBundleAfterDdeEnhancement** (String... *moduleNames*)

setAllEntities

public void **setAllEntities** (AllEntities *allEntities*)

setBundleContext

public void **setBundleContext** (BundleContext *bundleContext*)

setListenerRegistryService

public void **setListenerRegistryService** (JdoListenerRegistryService *jdoListenerRegistryService*)

setMdsConstructor

public void **setMdsConstructor** (MDSConstructor *mdsConstructor*)

setMdsDataProvider

public void **setMdsDataProvider** (MDSDataProvider *mdsDataProvider*)

setMetadataHolder

public void **setMetadataHolder** (MetadataHolder *metadataHolder*)

setMonitor

public void **setMonitor** (EntitiesBundleMonitor *monitor*)

setVelocityEngine

public void **setVelocityEngine** (VelocityEngine *velocityEngine*)

12.61.6 JdoListenerRegistryServiceImpl

public class **JdoListenerRegistryServiceImpl** implements JdoListenerRegistryService

Implementation of the `org.motechproject.mds.service.JdoListenerRegistryService`.

Methods

getEntitiesListenerStr

public String **getEntitiesListenerStr**()

getListeners

public List<MotechLifecycleListener> **getListeners**()

getListeners

public List<MotechLifecycleListener> **getListeners**(String *entity*, InstanceLifecycleListenerType *type*)

getMethods

public Set<String> **getMethods**(MotechLifecycleListener *listener*, InstanceLifecycleListenerType *type*)

registerEntityWithListeners

public void **registerEntityWithListeners**(String *entity*)

registerListener

public void **registerListener**(MotechLifecycleListener *listener*)

removeInactiveListeners

public void **removeInactiveListeners**(String *entitiesNames*)

removeListener

public void **removeListener**(MotechLifecycleListener *listener*)

setEntityService

public void **setEntityService**(EntityService *entityService*)

updateEntityNames

public void **updateEntityNames** ()

12.61.7 JsonLookupServiceImpl

public class **JsonLookupServiceImpl** implements JsonLookupService

Methods

createJsonLookup

public void **createJsonLookup** (JsonLookupDto *jsonLookup*)

exists

public boolean **exists** (String *entityClassName*, String *originLookupName*)

setAllJsonLookups

public void **setAllJsonLookups** (AllJsonLookups *allJsonLookups*)

12.61.8 MdsBundleRegenerationServiceImpl

public class **MdsBundleRegenerationServiceImpl** implements MdsBundleRegenerationService, EventHandler

Default implementation of the `MdsBundleRegenerationService` interface. It uses the `org.motechproject.mds.service.JarGeneratorService` to perform the MDS Entities Bundle regeneration and messages broadcasting for communication with other Motech instances. This class uses `OsgiEventProxy` to proxy Motech events though OSGi events, in order to avoid a dependency on the event module.

See also: `org.motechproject.mds.service.JarGeneratorService`

Methods

handleEvent

public void **handleEvent** (Event *event*)

regenerateMdsDataBundle

public void **regenerateMdsDataBundle** ()

regenerateMdsDataBundleAfterDdeEnhancement

public void **regenerateMdsDataBundleAfterDdeEnhancement** (String... *moduleNames*)

setJarGeneratorService

public void **setJarGeneratorService** (JarGeneratorService *jarGeneratorService*)

setOsgiEventProxy

public void **setOsgiEventProxy** (OsgiEventProxy *osgiEventProxy*)

12.61.9 MdsLookupServiceImpl

public class **MdsLookupServiceImpl** implements MDSLookupService

Implementation of the `org.motechproject.mds.service.MDSLookupService`. This runs in the MDS context(not entities context). All calls are delegated to the respective data service for the entity.

Methods

count

public long **count** (Class *entityClass*, String *lookupName*, Map<String, ?> *lookupParams*)

count

public long **count** (String *entityClassName*, String *lookupName*, Map<String, ?> *lookupParams*)

countAll

public long **countAll** (Class *entityClass*)

countAll

public long **countAll** (String *entityClassName*)

findMany

public <T> List<T> **findMany** (Class<T> *entityClass*, String *lookupName*, Map<String, ?> *lookupParams*)

findMany

public <T> List<T> **findMany** (String *entityClassName*, String *lookupName*, Map<String, ?> *lookupParams*)

findMany

public <T> List<T> **findMany** (Class<T> *entityClass*, String *lookupName*, Map<String, ?> *lookupParams*, QueryParams *queryParams*)

findMany

public <T> List<T> **findMany** (String *entityClassName*, String *lookupName*, Map<String, ?> *lookupParams*, QueryParams *queryParams*)

findOne

public <T> T **findOne** (Class<T> *entityClass*, String *lookupName*, Map<String, ?> *lookupParams*)

findOne

public <T> T **findOne** (String *entityClassName*, String *lookupName*, Map<String, ?> *lookupParams*)

retrieveAll

public <T> List<T> **retrieveAll** (Class<T> *entityClass*)

retrieveAll

public <T> List<T> **retrieveAll** (String *entityClassName*)

retrieveAll

public <T> List<T> **retrieveAll** (Class<T> *entityClass*, QueryParams *queryParams*)

retrieveAll

public <T> List<T> **retrieveAll** (String *entityClassName*, QueryParams *queryParams*)

12.61.10 MdsScheduledJob

public class **MdsScheduledJob** implements Job

Job responsible for emptying MDS trash.

Methods

execute

public void **execute** (JobExecutionContext *jobExecutionContext*)

12.61.11 MdsSchedulerServiceImpl

public class **MdsSchedulerServiceImpl** implements MdsSchedulerService

Default implementation of the `MdsSchedulerService`.

Fields

DEFAULT_WAIT_TIME

public static final int **DEFAULT_WAIT_TIME**

JOB_GROUP_NAME

public static final String **JOB_GROUP_NAME**

MAX_REPEAT_COUNT

public static final int **MAX_REPEAT_COUNT**

RETRIEVAL_RETRIES_COUNT

public static final int **RETRIEVAL_RETRIES_COUNT**

SCHEDULER_SYMBOLIC_NAME

public static final String **SCHEDULER_SYMBOLIC_NAME**

Constructors

MdsSchedulerServiceImpl

public **MdsSchedulerServiceImpl** (BundleContext *bundleContext*)

Methods

scheduleRepeatingJob

public void **scheduleRepeatingJob** (long *interval*)

unscheduleRepeatingJob

public void **unscheduleRepeatingJob** ()

12.61.12 MigrationServiceImpl

public class **MigrationServiceImpl** implements MigrationService

Default implementation of `org.motechproject.mds.service.MigrationService` interface.

Methods

processBundle

public void **processBundle** (Bundle *bundle*)

12.61.13 RestDocumentationServiceImpl

public class **RestDocumentationServiceImpl** implements RestDocumentationService

Implementation of `org.motechproject.mds.service.RestDocumentationService`

Methods

retrieveDocumentation

public void **retrieveDocumentation** (Writer *writer*, String *serverPrefix*, Locale *locale*)

12.61.14 TypeServiceImpl

public class **TypeServiceImpl** implements TypeService

Default implementation of `org.motechproject.mds.service.TypeService` interface

Methods

findType

public TypeDto **findType** (Class<?> *clazz*)

findValidations

public List<TypeValidation> **findValidations** (TypeDto *type*, Class<? extends Annotation> *aClass*)

getAllTypes

public List<TypeDto> **getAllTypes** ()

getType

public Type **getType** (TypeValidation *validation*)

setAllTypeValidations

public void **setAllTypeValidations** (AllTypeValidations *allTypeValidations*)

setAllTypes

public void **setAllTypes** (AllTypes *allTypes*)

12.62 org.motechproject.mds.service.impl.csv

12.62.1 AbstractMdsExporter

public abstract class **AbstractMdsExporter**

Base class used by classes responsible for exporting MDS Data in a tabular CSV-like form. Using the `TableWriter` class, implementing classes can provide their own ouput format.

Methods

exportData

protected long **exportData** (Entity *entity*, TableWriter *writer*)

exportData

protected long **exportData** (Entity *entity*, TableWriter *writer*, CsvExportCustomizer *exportCustomizer*)

exportData

protected long **exportData** (Entity *entity*, TableWriter *writer*, String *lookupName*, QueryParams *params*, List<String> *headers*, Map<String, Object> *lookupFields*, CsvExportCustomizer *exportCustomizer*)

getAllEntities

protected AllEntities **getAllEntities** ()

getBundleContext

protected BundleContext **getBundleContext** ()

getEntity

protected Entity **getEntity** (long *entityId*)

getEntity

protected Entity **getEntity** (String *entityClassName*)

getMdsLookupService

protected MDSLookupService **getMdsLookupService**()

orderHeaders

protected String[] **orderHeaders**(String[] *selectedHeaders*, List<Field> *entityFields*, CsvExportCustomizer *customizer*)

12.62.2 CsvImportExportServiceImpl

public class **CsvImportExportServiceImpl** implements CsvImportExportService

Implementation of the `org.motechproject.mds.service.CsvImportExportService`. Uses the SuperCSV library for handling CSV files. `CsvImporterExporter` is used for handling import/export logic. This service implementation also fires MOTECH events upon import completion or import failure. This bean lives in the context of the generated MDS entities bundle.

See also: `CsvImporterExporter`

Methods

exportCsv

public long **exportCsv**(long *entityId*, Writer *writer*)

exportCsv

public long **exportCsv**(long *entityId*, Writer *writer*, CsvExportCustomizer *exportCustomizer*)

exportCsv

public long **exportCsv**(long *entityId*, Writer *writer*, String *lookupName*, QueryParams *params*, List<String> *headers*, Map<String, Object> *lookupFields*)

exportCsv

public long **exportCsv**(long *entityId*, Writer *writer*, String *lookupName*, QueryParams *params*, List<String> *headers*, Map<String, Object> *lookupFields*, CsvExportCustomizer *exportCustomizer*)

exportCsv

public long **exportCsv**(String *entityClassName*, Writer *writer*)

exportCsv

public long **exportCsv**(String *entityClassName*, Writer *writer*, CsvExportCustomizer *exportCustomizer*)

exportCsv

public long **exportCsv** (String *entityClassName*, Writer *writer*, String *lookupName*, QueryParams *params*, List<String> *headers*, Map<String, Object> *lookupFields*)

exportCsv

public long **exportCsv** (String *entityClassName*, Writer *writer*, String *lookupName*, QueryParams *params*, List<String> *headers*, Map<String, Object> *lookupFields*, CsvExportCustomizer *exportCustomizer*)

exportPdf

public long **exportPdf** (long *entityId*, OutputStream *outputStream*)

exportPdf

public long **exportPdf** (String *entityClassName*, OutputStream *outputStream*)

exportPdf

public long **exportPdf** (long *entityId*, OutputStream *outputStream*, CsvExportCustomizer *exportCustomizer*)

exportPdf

public long **exportPdf** (String *entityClassName*, OutputStream *outputStream*, CsvExportCustomizer *exportCustomizer*)

exportPdf

public long **exportPdf** (long *entityId*, OutputStream *outputStream*, String *lookupName*, QueryParams *params*, List<String> *headers*, Map<String, Object> *lookupFields*)

exportPdf

public long **exportPdf** (String *entityClassName*, OutputStream *outputStream*, String *lookupName*, QueryParams *params*, List<String> *headers*, Map<String, Object> *lookupFields*)

exportPdf

public long **exportPdf** (long *entityId*, OutputStream *outputStream*, String *lookupName*, QueryParams *params*, List<String> *headers*, Map<String, Object> *lookupFields*, CsvExportCustomizer *exportCustomizer*)

exportPdf

public long **exportPdf** (String *entityClassName*, OutputStream *outputStream*, String *lookupName*, QueryParams *params*, List<String> *headers*, Map<String, Object> *lookupFields*, CsvExportCustomizer *exportCustomizer*)

importCsv

public CsvImportResults **importCsv** (long *entityId*, Reader *reader*, String *fileName*)

importCsv

public CsvImportResults **importCsv** (long *entityId*, Reader *reader*, String *fileName*, CsvImportCustomizer *importCustomizer*)

importCsv

public CsvImportResults **importCsv** (String *entityClassName*, Reader *reader*, String *fileName*)

12.62.3 CsvImporterExporter

public class **CsvImporterExporter** extends AbstractMdsExporter

Component used for importing CSV records to the database. The reason for separating import logic is keeping the db transaction and sending the MOTECH event at completion separate. This bean lives in the context of the generated MDS entities bundle.

Methods

exportCsv

public long **exportCsv** (long *entityId*, Writer *writer*)

Exports entity instances to a CSV file.

Parameters

- **entityId** – id of the entity for which the instances will be exported
- **writer** – the writer that will be used for output

Returns number of exported instances

exportCsv

public long **exportCsv** (String *entityClassName*, Writer *writer*)

Exports entity instances to a CSV file.

Parameters

- **entityClassName** – the class name of the entity for which instances will be exported
- **writer** – the writer that will be used for output

Returns number of exported instances

exportCsv

public long **exportCsv** (long *entityId*, Writer *writer*, CsvExportCustomizer *exportCustomizer*)

Exports entity instances to a CSV file.

Parameters

- **entityId** – id of the entity for which the instances will be exported
- **writer** – the writer that will be used for output
- **exportCustomizer** – the customizer that will be used during export

Returns number of exported instances

exportCsv

public long **exportCsv** (String *entityClassName*, Writer *writer*, CsvExportCustomizer *exportCustomizer*)

Exports entity instances to a CSV file.

Parameters

- **entityClassName** – the class name of the entity for which instances will be exported
- **writer** – the writer that will be used for output
- **exportCustomizer** – the customizer that will be used during export

Returns number of exported instances

exportCsv

public long **exportCsv** (long *entityId*, Writer *writer*, String *lookupName*, QueryParams *params*, List<String> *headers*, Map<String, Object> *lookupFields*)

Exports entity instances to a CSV file.

Parameters

- **entityId** – id of the entity for which the instances will be exported
- **writer** – the writer that will be used for output
- **lookupName** – the name of lookup
- **params** – query parameters to be used retrieving instances
- **headers** – the headers of exported file
- **lookupFields** – the lookupFields used in the lookup

Returns number of exported instances

exportCsv

public long **exportCsv** (String *entityClassName*, Writer *writer*, String *lookupName*, QueryParams *params*, List<String> *headers*, Map<String, Object> *lookupFields*)

Exports entity instances to a CSV file.

Parameters

- **entityClassName** – the class name of the entity for which instances will be exported
- **writer** – the writer that will be used for output
- **lookupName** – the name of lookup
- **params** – query parameters to be used retrieving instances
- **headers** – the headers of exported file
- **lookupFields** – the lookupFields used in the lookup

Returns number of exported instances

exportCsv

public long **exportCsv**(long *entityId*, Writer *writer*, String *lookupName*, QueryParams *params*, List<String> *headers*, Map<String, Object> *lookupFields*, CsvExportCustomizer *exportCustomizer*)

Exports entity instances to a CSV file.

Parameters

- **entityId** – id of the entity for which the instances will be exported
- **writer** – the writer that will be used for output
- **lookupName** – the name of lookup
- **params** – query parameters to be used retrieving instances
- **headers** – the headers of exported file
- **lookupFields** – the lookupFields used in the lookup
- **exportCustomizer** – the customizer that will be used during export

Returns number of exported instances

exportCsv

public long **exportCsv**(String *entityClassName*, Writer *writer*, String *lookupName*, QueryParams *params*, List<String> *headers*, Map<String, Object> *lookupFields*, CsvExportCustomizer *exportCustomizer*)

Exports entity instances to a CSV file.

Parameters

- **entityClassName** – the class name of the entity for which instances will be exported
- **writer** – the writer that will be used for output
- **lookupName** – the name of lookup
- **params** – query parameters to be used retrieving instances
- **headers** – the headers of exported file
- **lookupFields** – the lookupFields used in the lookup
- **exportCustomizer** – the customizer that will be used during export

Returns number of exported instances

importCsv

public CsvImportResults **importCsv** (long *entityId*, Reader *reader*)

Imports instances of the given entity to the database.

Parameters

- **entityId** – the ID of the entity for which instances will be imported
- **reader** – reader from which the csv file will be read

Returns IDs of instances updated/added during import

importCsv

public CsvImportResults **importCsv** (long *entityId*, Reader *reader*, CsvImportCustomizer *importCustomizer*)

Imports instances of the given entity to the database.

Parameters

- **entityId** – the ID of the entity for which instances will be imported
- **reader** – reader from which the csv file will be read
- **importCustomizer** – the customizer that will be used during instance import from rows

Returns IDs of instances updated/added during import

importCsv

public CsvImportResults **importCsv** (String *entityClassName*, Reader *reader*)

Imports instances of the given entity to the database.

Parameters

- **entityClassName** – the class name of the entity for which instances will be imported
- **reader** – reader from which the csv file will be read

Returns IDs of instances updated/added during import

12.62.4 PdfCsvExporter

public class **PdfCsvExporter** extends AbstractMdsExporter

A class exporting CSV-like tables in PDF format.

Methods

exportPdf

public long **exportPdf** (long *entityId*, OutputStream *outputStream*)

Exports entity instances to a PDF file.

Parameters

- **entityId** – id of the entity for which the instances will be exported

- **outputStream** – the output stream that will be used for writing the file

Returns number of exported instances

exportPdf

public long **exportPdf** (String *entityClassName*, OutputStream *outputStream*)

Exports entity instances to a PDF file.

Parameters

- **entityClassName** – the class name of the entity for which instances will be exported
- **outputStream** – the output stream that will be used for writing the file

Returns number of exported instances

exportPdf

public long **exportPdf** (long *entityId*, OutputStream *outputStream*, CsvExportCustomizer *exportCustomizer*)

Exports entity instances to a PDF file.

Parameters

- **entityId** – id of the entity for which the instances will be exported
- **outputStream** – the output stream that will be used for writing the file
- **exportCustomizer** – the customizer that will be used during export

Returns number of exported instances

exportPdf

public long **exportPdf** (String *entityClassName*, OutputStream *outputStream*, CsvExportCustomizer *exportCustomizer*)

Exports entity instances to a PDF file.

Parameters

- **entityClassName** – the class name of the entity for which instances will be exported
- **outputStream** – the output stream that will be used for writing the file
- **exportCustomizer** – the customizer that will be used during export

Returns number of exported instances

exportPdf

public long **exportPdf** (long *entityId*, OutputStream *outputStream*, String *lookupName*, QueryParams *params*, List<String> *headers*, Map<String, Object> *lookupFields*)

Exports entity instances to a PDF file.

Parameters

- **entityId** – id of the entity for which the instances will be exported
- **outputStream** – the output stream that will be used for writing the file

- **lookupName** – the name of lookup
- **params** – query parameters to be used retrieving instances
- **headers** – the headers of exported file
- **lookupFields** – the lookupFields used in the lookup

Returns number of exported instances

exportPdf

public long **exportPdf** (String *entityClassName*, OutputStream *outputStream*, String *lookupName*, QueryParams *params*, List<String> *headers*, Map<String, Object> *lookupFields*)

Exports entity instances to a PDF file.

Parameters

- **entityClassName** – the class name of the entity for which instances will be exported
- **outputStream** – the output stream that will be used for writing the file
- **lookupName** – the name of lookup
- **params** – query parameters to be used retrieving instances
- **headers** – the headers of exported file
- **lookupFields** – the lookupFields used in the lookup

Returns number of exported instances

exportPdf

public long **exportPdf** (long *entityId*, OutputStream *outputStream*, String *lookupName*, QueryParams *params*, List<String> *headers*, Map<String, Object> *lookupFields*, CsvExportCustomizer *exportCustomizer*)

Exports entity instances to a PDF file.

Parameters

- **entityId** – id of the entity for which the instances will be exported
- **outputStream** – the output stream that will be used for writing the file
- **lookupName** – the name of lookup
- **params** – query parameters to be used retrieving instances
- **headers** – the headers of exported file
- **lookupFields** – the lookupFields used in the lookup
- **exportCustomizer** – the customizer that will be used during export

Returns number of exported instances

exportPdf

public long **exportPdf** (String *entityClassName*, OutputStream *outputStream*, String *lookupName*, QueryParams *params*, List<String> *headers*, Map<String, Object> *lookupFields*, CsvExportCustomizer *exportCustomizer*)

Exports entity instances to a PDF file.

Parameters

- **entityClassName** – the class name of the entity for which instances will be exported
- **outputStream** – the output stream that will be used for writing the file
- **lookupName** – the name of lookup
- **params** – query parameters to be used retrieving instances
- **headers** – the headers of exported file
- **lookupFields** – the lookupFields used in the lookup
- **exportCustomizer** – the customizer that will be used during export

Returns number of exported instances

12.63 org.motechproject.mds.service.impl.csv.writer

12.63.1 CsvTableWriter

public class **CsvTableWriter** implements TableWriter

An implementation of the table writer that writes the table data in CSV format. Uses the SuperCSV library underneath.

Constructors

CsvTableWriter

public **CsvTableWriter** (Writer *writer*)

Methods

close

public void **close** ()

writeHeader

public void **writeHeader** (String[] *headers*)

writeRow

public void **writeRow** (Map<String, String> *row*, String[] *headers*)

12.63.2 PdfTableWriter

public class **PdfTableWriter** implements TableWriter

An implementation of the table writer that writes the table data in PDF format. Uses the iText PDF library underneath.

Constructors

PdfTableWriter

public **PdfTableWriter** (OutputStream *outputStream*)

Methods

close

public void **close** ()

writeHeader

public void **writeHeader** (String[] *headers*)

writeRow

public void **writeRow** (Map<String, String> *row*, String[] *headers*)

12.63.3 TableWriter

public interface **TableWriter** extends AutoCloseable

An interface for writing tabular data. A writer should be created for each supported format such as PDF or CSV.

Methods

close

void **close** ()

{@inheritDoc}

writeHeader

void **writeHeader** (String[] *headers*)

Writes the table header.

Parameters

- **headers** – an array of headers for the table

Throws

- **IOException** –

writeRow

void **writeRow** (Map<String, String> *row*, String[] *headers*)

Writes a row of data to the table.

Parameters

- **row** – the row data, keys are field names and values are their values in string form that should be directly written to the output
- **headers** – the array of headers for the table

Throws

- **IOException** –

12.64 org.motechproject.mds.service.impl.history

12.64.1 BasePersistenceService

public abstract class **BasePersistenceService**

The `BasePersistenceService` class provides utility methods for communication with the database for `HistoryServiceImpl` and `TrashServiceImpl`. It allows to create and retrieve instances, load proper classes and parse values.

Methods

create

protected <T> Object **create** (Class<T> *clazz*, Object *instance*, EntityType *type*, ValueGetter *valueGetter*)

create

protected <T> Object **create** (Class<T> *clazz*, Object *instance*, EntityType *type*, ValueGetter *valueGetter*, ObjectReferenceRepository *objectReferenceRepository*)

getAllEntities

protected AllEntities **getAllEntities** ()

getBundleContext

protected BundleContext **getBundleContext** ()

getCurrentSchemaVersion

protected Long **getCurrentSchemaVersion** (String *className*)

getEntities

protected List<Entity> **getEntities** ()

getEntity

protected Entity **getEntity** (Long *id*)

getEntitySchemaVersion

protected Long **getEntitySchemaVersion** (Object *src*)

getInstanceId

protected Long **getInstanceId** (Object *instance*)

getPersistenceManagerFactory

protected PersistenceManagerFactory **getPersistenceManagerFactory** ()

setAllEntities

public void **setAllEntities** (AllEntities *allEntities*)

setBundleContext

public void **setBundleContext** (BundleContext *bundleContext*)

setPersistenceManagerFactory

public void **setPersistenceManagerFactory** (PersistenceManagerFactory *persistenceManagerFactory*)

12.64.2 HistoryServiceImpl

public class **HistoryServiceImpl** extends BasePersistenceService implements HistoryService

Default implementation of `org.motechproject.mds.service.HistoryService` interface.

Methods

countHistoryRecords

public long **countHistoryRecords** (Object *instance*)

getHistoryForInstance

public List **getHistoryForInstance** (Object *instance*, QueryParams *queryParams*)

getSingleHistoryInstance

public Object **getSingleHistoryInstance** (Object *instance*, Long *historyId*)

record

public void **record** (Object *instance*)

remove

public void **remove** (Object *instance*)

setTrashFlag

public void **setTrashFlag** (Object *instance*, Object *trash*, boolean *flag*)

12.64.3 HistoryTrashClassHelper

public final class **HistoryTrashClassHelper**

Contains utility methods for dealing with history and trash clasess

Methods

currentVersion

public static String **currentVersion** (Class<?> *historyClass*)

getClass

public static Class<?> **getClass** (Object *src*, EntityType *type*, BundleContext *bundleContext*)

getClass

public static Class<?> **getClass** (String *srcClassName*, EntityType *type*, BundleContext *bundleContext*)

getInstanceClassName

public static String **getInstanceClassName** (Object *instance*)

getMethodParameterType

public static String **getMethodParameterType** (Type *type*, ComboboxHolder *holder*)

schemaVersion

public static String **schemaVersion** (Class<?> *historyClass*)

trashFlag

public static String **trashFlag** (Class<?> *historyClass*)

12.64.4 TrashServiceImpl

public class **TrashServiceImpl** extends BasePersistenceService implements TrashService

Default implementation of `org.motechproject.mds.service.TrashService` interface.

Methods

countTrashRecords

public long **countTrashRecords** (String *className*)

emptyTrash

public void **emptyTrash** ()

findTrashById

public Object **findTrashById** (Object *instanceId*, Object *entityId*)

getInstancesFromTrash

public Collection **getInstancesFromTrash** (String *className*, QueryParams *queryParams*)

init

public void **init** ()

isTrashMode

public boolean **isTrashMode** ()

moveFromTrash

public void **moveFromTrash** (Object *newInstance*, Object *trash*, boolean *recordHistory*)

moveToTrash

public void **moveToTrash** (Object *instance*, Long *entityVersion*, boolean *recordHistory*)

scheduleEmptyTrashJob

public void **scheduleEmptyTrashJob** ()

setHistoryService

public void **setHistoryService** (HistoryService *historyService*)

setMdsSchedulerService

public void **setMdsSchedulerService** (MdsSchedulerService *mdsSchedulerService*)

setSettingsService

public void **setSettingsService** (SettingsService *settingsService*)

12.64.5 ValueGetter

public class **ValueGetter**

This class is required for retrieving values from entities. Since the history implementation makes additional changes to records involved in relationships and sets additional field, this class gives it an easy way to override the default behaviour.

Constructors

ValueGetter

public **ValueGetter** (BasePersistenceService *persistenceService*, BundleContext *bundleContext*)

Methods

findService

protected MotechDataService **findService** (Class<?> *clazz*)

getValue

public Object **getValue** (Field *field*, Object *instance*, Object *recordInstance*, EntityType *type*, ObjectReferenceRepository *objectReferenceRepository*)

updateRecordFields

protected void **updateRecordFields** (Object *newHistoryRecord*, Object *realCurrentObj*)

Updates fields of a new record after it is created. Called for newly created relationship records as well.

Parameters

- **newHistoryRecord** – the newly created record
- **realCurrentObj** – the actual current object

12.65 org.motechproject.mds.util

12.65.1 BlobDeserializer

public class **BlobDeserializer** extends JsonDeserializer<Byte[]>

Class responsible for deserializing blob from Base64 to Byte[].

Methods

deserialize

public Byte[] **deserialize** (JsonParser *jp*, DeserializationContext *ctxt*)

12.65.2 ClassName

public final class **ClassName**

The `ClassName` util provides several methods which should help for example with getting class name or package from string representation of class. There is also methods related with creating names for repository, service interface and implementation of this service interface.

Methods

getEntityName

public static String **getEntityName** (String *className*)

Retrieves fully qualified entity class name, for the End User Defined Entity.

Parameters

- **className** – class name

Returns fully qualified class name

getEntityTypeSuffix

public static String **getEntityTypeSuffix** (String *name*)

Retrieves entity type suffix from the class name. If the class is neither a history type class, nor a trash type class, it returns empty String.

Parameters

- **name** – class name

Returns suffix of an entity type or empty String, if not applicable

getEnumPackage

public static String **getEnumPackage** (String *className*)

Returns the package name that contain enum added as a field to a class with a given name.

Parameters

- **className** – name of a class that contain enum field

Returns package name of the enum generated for the class

getHistoryClassName

public static String **getHistoryClassName** (String *className*)

Retrieves fully qualified history class name, for the given class.

Parameters

- **className** – class name

Returns fully qualified history class name

getInterfaceName

public static String **getInterfaceName** (String *className*)

Retrieves interface name for the End User Defined Entity or for Developer Defined Entity that do not define their own interface.

Parameters

- **className** – entity class name

Returns fully qualified interface name

getPackage

public static String **getPackage** (String *className*)

Returns package name from the fully qualified class name. If package cannot be resolved, an empty String will be returned.

Parameters

- **className** – fully qualified class name

Returns package name

getRepositoryName

public static String **getRepositoryName** (String *className*)

Retrieves repository class name for the given entity class name.

Parameters

- **className** – entity class name

Returns fully qualified repository class name

getServiceName

public static String **getServiceName** (String *className*)

Retrieves fully qualified class name of the `org.motechproject.mds.service.MotechDataService` service implementation.

Parameters

- **className** – entity class name

Returns fully qualified MDS service implementation name

getSimpleName

public static String **getSimpleName** (String *className*)

Returns simple name of the class (without the package prefix), from the given String. If the name is already simple, it will return that name.

Parameters

- **className** – class name

Returns simple class name

getTrashClassName

public static String **getTrashClassName** (String *className*)

Retrieves fully qualified trash class name, for the given class.

Parameters

- **className** – class name

Returns fully qualified trash class name

isHistoryClassName

public static boolean **isHistoryClassName** (String *className*)

Verifies whether the given class name matches history class naming pattern.

Parameters

- **className** – class name to verify

Returns true, if given class matches history class naming pattern; false otherwise

isTrashClassName

public static boolean **isTrashClassName** (String *className*)

Verifies whether the given class name matches trash class naming pattern.

Parameters

- **className** – class name to verify

Returns true, if given class matches trash class naming pattern; false otherwise

restId

public static String **restId** (String *entityName*, String *module*, String *namespace*)

Builds REST id, based on the entity name, module name and namespace.

Parameters

- **entityName** – name of the entity
- **module** – name of the module
- **namespace** – namespace

Returns REST id

restLookupUrl

public static String **restLookupUrl** (String *entityName*, String *entityModule*, String *entityNamespace*, String *lookupMethodName*)

Builds URL endpoint, to access lookups via REST, based on the entity name, module, namespace and lookup method name.

Parameters

- **entityName** – name of the entity
- **entityModule** – name of the module
- **entityNamespace** – namespace
- **lookupMethodName** – name of the lookup method

Returns URL endpoint for REST lookup

restUrl

public static String **restUrl** (String *entityName*, String *entityModule*, String *entityNamespace*)

Builds URL endpoint to access REST operations, based on the entity name, module name and namespace.

Parameters

- **entityName** – name of the entity
- **entityModule** – name of the module
- **entityNamespace** – namespace

Returns URL endpoint for REST

simplifiedModuleName

public static String **simplifiedModuleName** (String *moduleName*)

Returns simplified name of the module. It will drop the "motech" and "motechplatform" prefix from the module names. Also any blank spaces will be removed.

Parameters

- **moduleName** – module name to simplify

Returns simplified module name

trimTrashHistorySuffix

public static String **trimTrashHistorySuffix** (String *name*)

Removes entity type suffix from the class name. If the name does not contain entity type suffix, the passed name will be returned.

Parameters

- **name** – class name

Returns class name, trimmed from type suffix

12.65.3 Constants

public final class **Constants**

The `Constants` contains constant values used in MDS module. They are grouped by their role.

12.65.4 Constants.AnnotationFields

public static final class **AnnotationFields**

The `AnnotationFields` contains constant values related with attributes names in mds annotations.

See also: `org.motechproject.mds.annotations.Entity`, `org.motechproject.mds.annotations.Field` `org.motechproject.mds.annotations.Ignore`, `org.motechproject.mds.annotations.Lookup`, `org.motechproject.mds.annotations.LookupField`

Fields

CRUD_EVENTS

public static final String **CRUD_EVENTS**

Constant `CRUD_EVENTS` corresponding to the @Entity attribute named `crudEvents`

DELETE

public static final String **DELETE**

Constant `DELETE` corresponding to the attribute name `delete`

DISPLAY_NAME

public static final String **DISPLAY_NAME**
Constant `DISPLAY_NAME` corresponding to the primitive value `displayName`

FRACTION

public static final String **FRACTION**
Constant `FRACTION` corresponding to the primitive value `fraction`

HISTORY

public static final String **HISTORY**
Constant `HISTORY` corresponding to the @Entity attribute named `recordHistory`

INTEGER

public static final String **INTEGER**
Constant `INTEGER` corresponding to the primitive value `integer`

MAX

public static final String **MAX**
Constant `MAX` corresponding to the primitive value `max`

MAX_FETCH_DEPTH

public static final String **MAX_FETCH_DEPTH**
Constant `TABLE_NAME` corresponding to the @Entity attribute named `maxFetchDepth`

MIN

public static final String **MIN**
Constant `MIN` corresponding to the primitive value `min`

MODULE

public static final String **MODULE**
Constant `MODULE` corresponding to the @Entity attribute named `module`

NAME

public static final String **NAME**
Constant `NAME` corresponding to the @Entity attribute named `name`

NAMESPACE

public static final String **NAMESPACE**
Constant `NAMESPACE` corresponding to the @Entity attribute named `namespace`

NON_EDITABLE

public static final String **NON_EDITABLE**
Constant `NON_EDITABLE` corresponding to the @Entity attribute named `nonEditable`

PERSIST

public static final String **PERSIST**
Constant `PERSIST` corresponding to the attribute name `persist`

REGEXP

public static final String **REGEXP**
Constant `REGEXP` corresponding to the primitive value `regexp`

TABLE_NAME

public static final String **TABLE_NAME**
Constant `TABLE_NAME` corresponding to the @Entity attribute named `tableName`

TYPE

public static final String **TYPE**

UPDATE

public static final String **UPDATE**
Constant `UPDATE` corresponding to the attribute name `update`

VALUE

public static final String **VALUE**
Constant `VALUE` corresponding to the primitive value `value`

12.65.5 Constants.BundleNames

public static final class **BundleNames**
The names of the bundles.

Fields

MDS_BUNDLE_NAME

public static final String **MDS_BUNDLE_NAME**

MDS_BUNDLE_SYMBOLIC_NAME

public static final String **MDS_BUNDLE_SYMBOLIC_NAME**

MDS_ENTITIES_NAME

public static final String **MDS_ENTITIES_NAME**

MDS_ENTITIES_SYMBOLIC_NAME

public static final String **MDS_ENTITIES_SYMBOLIC_NAME**

MDS_MIGRATION_NAME

public static final String **MDS_MIGRATION_NAME**

MDS_MIGRATION_SYMBOLIC_NAME

public static final String **MDS_MIGRATION_SYMBOLIC_NAME**

SCHEDULER_MODULE

public static final String **SCHEDULER_MODULE**

SERVER_CONFIG_MODULE

public static final String **SERVER_CONFIG_MODULE**

SYMBOLIC_NAME_PREFIX

public static final String **SYMBOLIC_NAME_PREFIX**

WEB_SECURITY_MODULE

public static final String **WEB_SECURITY_MODULE**

12.65.6 Constants.Config

public static final class **Config**

The `Config` contains constant values related with properties inside files:

- datanucleus.properties
- motech-mds.properties

Fields

DATANUCLEUS_FILE

public static final String **DATANUCLEUS_FILE**

Constant `DATANUCLEUS_FILE` presents the file name with configuration for datanucleus.

EMPTY_TRASH_JOB

public static final String **EMPTY_TRASH_JOB**

Constant `EMPTY_TRASH_JOB` presents a name of job scheduled by scheduler module.

MDS_DELETE_MODE

public static final String **MDS_DELETE_MODE**

Constant `MDS_DELETE_MODE` presents what should happen with objects when there are deleted. They can be deleted permanently or moved to the trash.The following values are valid for this property:

- delete
- trash

MDS_EMPTY_TRASH

public static final String **MDS_EMPTY_TRASH**

The boolean property that specifies if the trash should be empty after some time.

See also: `.MDS_DELETE_MODE`, `.MDS_TIME_VALUE`, `.MDS_TIME_UNIT`

MDS_TIME_UNIT

public static final String **MDS_TIME_UNIT**

The property that specifies what time unit should be used to specify time when trash should be cleaned. The following values are valid for this property:

- Hours
- Days
- Weeks
- Months
- Years

See also: `.MDS_DELETE_MODE`, `.MDS_EMPTY_TRASH`, `.MDS_TIME_VALUE`

MDS_TIME_VALUE

public static final String **MDS_TIME_VALUE**

The integer property that specifies after what time (according with correct time unit) trash should be cleaned.

See also: `.MDS_DELETE_MODE`, `.MDS_EMPTY_TRASH`, `.MDS_TIME_UNIT`

MODULE_FILE

public static final String **MODULE_FILE**

Constant `MODULE_FILE` presents the file name with configuration for MDS module.

MYSQL_DRIVER_CLASSNAME

public static final String **MYSQL_DRIVER_CLASSNAME**

Constant `MYSQL_DRIVER_CLASSNAME` represents the name of MySql driver class. It is used in various places, to verify what driver class has been chosen by the user.

POSTGRES_DRIVER_CLASSNAME

public static final String **POSTGRES_DRIVER_CLASSNAME**

Constant `POSTGRES_DRIVER_CLASSNAME` represents the name of Postgres driver class. It is used in various places, to verify what driver vlass has been chosen by the user.

12.65.7 Constants.DisplayNames

public static final class **DisplayNames**

Fields

BLOB

public static final String **BLOB**

COMBOBOX

public static final String **COMBOBOX**

MAP

public static final String **MAP**

TEXT_AREA

public static final String **TEXT_AREA**

12.65.8 Constants.EntitiesMigration

public static final class **`EntitiesMigration`**
Constants corresponding to the entities migrations.

Fields

ENTITY_MIGRATIONS_PREFIX

public static final String **`ENTITY_MIGRATIONS_PREFIX`**

FILESYSTEM_PREFIX

public static final String **`FILESYSTEM_PREFIX`**

MIGRATION_DIRECTORY

public static final String **`MIGRATION_DIRECTORY`**

MIGRATION_FILE_NAME_PATTERN

public static final String **`MIGRATION_FILE_NAME_PATTERN`**

MIGRATION_VERSION_OFFSET

public static final int **`MIGRATION_VERSION_OFFSET`**

12.65.9 Constants.ExportFormat

public static final class **`ExportFormat`**
Formats for table data exported by MDS.

Fields

CSV

public static final String **`CSV`**

PDF

public static final String **`PDF`**

Methods

isValidFormat

public static boolean **isValidFormat** (String *format*)

12.65.10 Constants.FetchDepth

public static final class **FetchDepth**

Constants corresponding to the fetch depths when retrieving entities.

Fields

INFINITE

public static final int **INFINITE**

Represents greedy fetching - the infinite fetch depth.

MDS_DEFAULT

public static final int **MDS_DEFAULT**

Signals that default MDS value should be used. No custom fetch depth will be passed to the persistence manager.

12.65.11 Constants.MDSEvents

public static final class **MDSEvents**

The `MDSEvents` contains constant values related with MDS CRUD events.

Fields

BASE_SUBJECT

public static final String **BASE_SUBJECT**

CSV_IMPORT_CREATED_COUNT

public static final String **CSV_IMPORT_CREATED_COUNT**

CSV_IMPORT_CREATED_IDS

public static final String **CSV_IMPORT_CREATED_IDS**

CSV_IMPORT_FAILURE

public static final String **CSV_IMPORT_FAILURE**

CSV_IMPORT_FAILURE_MSG

public static final String **CSV_IMPORT_FAILURE_MSG**

CSV_IMPORT_FAILURE_STACKTRACE

public static final String **CSV_IMPORT_FAILURE_STACKTRACE**

CSV_IMPORT_FILENAME

public static final String **CSV_IMPORT_FILENAME**

CSV_IMPORT_SUCCESS

public static final String **CSV_IMPORT_SUCCESS**

CSV_IMPORT_TOTAL_COUNT

public static final String **CSV_IMPORT_TOTAL_COUNT**

CSV_IMPORT_UPDATED_COUNT

public static final String **CSV_IMPORT_UPDATED_COUNT**

CSV_IMPORT_UPDATED_IDS

public static final String **CSV_IMPORT_UPDATED_IDS**

ENTITY_CLASS

public static final String **ENTITY_CLASS**

ENTITY_NAME

public static final String **ENTITY_NAME**

MODULE_NAME

public static final String **MODULE_NAME**

NAMESPACE

public static final String **NAMESPACE**

OBJECT_ID

public static final String **OBJECT_ID**

12.65.12 Constants.Manifest

public static final class **Manifest**

The `Manifest` contains constant values related with attributes inside the motech-platform-dataservices-entities bundle manifest.

See also: `org.motechproject.mds.service.JarGeneratorService`, `org.motechproject.mds.service.impl.JarGeneratorServiceImpl`

Fields

BUNDLE_MANIFESTVERSION

public static final String **BUNDLE_MANIFESTVERSION**

Constant `BUNDLE_MANIFESTVERSION` presents a version of bundle manifest.

BUNDLE_NAME_SUFFIX

public static final String **BUNDLE_NAME_SUFFIX**

Constant `BUNDLE_NAME_SUFFIX` presents suffix of the name of bundle that will be created by implementation of `org.motechproject.mds.service.JarGeneratorService` interface.

MANIFEST_VERSION

public static final String **MANIFEST_VERSION**

Constant `MANIFEST_VERSION` presents a version of jar manifest.

SYMBOLIC_NAME_SUFFIX

public static final String **SYMBOLIC_NAME_SUFFIX**

Constant `SYMBOLIC_NAME_SUFFIX` presents suffix of the bundle symbolic name of bundle that will be created by implementation of `org.motechproject.mds.service.JarGeneratorService` interface.

12.65.13 Constants.MetadataKeys

public static final class **MetadataKeys**

The keys used in fields metadata

Fields

DATABASE_COLUMN_NAME

public static final String **DATABASE_COLUMN_NAME**

ENUM_CLASS_NAME

public static final String **ENUM_CLASS_NAME**

ENUM_COLLECTION_TYPE

public static final String **ENUM_COLLECTION_TYPE**

MAP_KEY_TYPE

public static final String **MAP_KEY_TYPE**

MAP_VALUE_TYPE

public static final String **MAP_VALUE_TYPE**

OWNING_SIDE

public static final String **OWNING_SIDE**

RELATED_CLASS

public static final String **RELATED_CLASS**

RELATED_FIELD

public static final String **RELATED_FIELD**

RELATIONSHIP_COLLECTION_TYPE

public static final String **RELATIONSHIP_COLLECTION_TYPE**

12.65.14 Constants.Operators

public static final class **Operators**

Operators that users can use in lookups.

Fields

ENDS_WITH

public static final String **ENDS_WITH**

EQ

public static final String **EQ**

EQ_IGNORE_CASE

public static final String **EQ_IGNORE_CASE**

GT

public static final String **GT**

GT_EQ

public static final String **GT_EQ**

LT

public static final String **LT**

LT_EQ

public static final String **LT_EQ**

MATCHES

public static final String **MATCHES**

MATCHES_CASE_INSENSITIVE

public static final String **MATCHES_CASE_INSENSITIVE**

NEQ

public static final String **NEQ**

STARTS_WITH

public static final String **STARTS_WITH**

12.65.15 Constants.Packages

public static final class **Packages**

The `Packages` contains constant values related with packages inside MDS module.

Fields

BASE

public static final String **BASE**

Constant `BASE` presents the base package for all pakcages inside MDS module.

ENTITY

public static final String **ENTITY**

Constant `ENTITY` presents a package for entity classes.

See also: `.BASE`

REPOSITORY

public static final String **REPOSITORY**

Constant `REPOSITORY` presents a package for repository classes.

See also: `.BASE`

SERVICE

public static final String **SERVICE**

Constant `SERVICE` presents a package for service interfaces.

See also: `.BASE`

SERVICE_IMPL

public static final String **SERVICE_IMPL**

Constant `SERVICE_IMPL` presents a package for implementation of interfaces defined in `SERVICE` package.

See also: `.BASE`, `.SERVICE`

12.65.16 Constants.PackagesGenerated

public static final class **PackagesGenerated**

Fields

ENTITY

public static final String **ENTITY**

Constant `ENTITY` presents a package for generated entity classes.

REPOSITORY

public static final String **REPOSITORY**

Constant `REPOSITORY` presents a package for generated repository classes.

See also: `.ENTITY`

SERVICE

public static final String **SERVICE**

Constant `SERVICE` presents a package for generated service interfaces.

See also: `.ENTITY`

SERVICE_IMPL

public static final String **SERVICE_IMPL**

Constant `SERVICE_IMPL` presents a package for generated implementation of interfaces defined in `SERVICE` package.

See also: `.SERVICE`

12.65.17 Constants.Roles

public static final class **Roles**

The `Roles` contains constant values related with security roles.

Fields

DATA_ACCESS

public static final String **DATA_ACCESS**

Users with 'Data Access' have the ability to view the Data Browser tab. From that tab then can search for objects within the system, view and modify the data stored in the system.

HAS_ANY_MDS_ROLE

public static final String **HAS_ANY_MDS_ROLE**

Spring security el expression to check if the given user has any of the MDS roles.

See also: `.SCHEMA_ACCESS`, `.SETTINGS_ACCESS`, `.DATA_ACCESS`

HAS_DATA_ACCESS

public static final String **HAS_DATA_ACCESS**

Spring security el expression to check if the given user has the 'Data Access' role.

See also: `.DATA_ACCESS`

HAS_DATA_OR_SCHEMA_ACCESS

public static final String **HAS_DATA_OR_SCHEMA_ACCESS**

Spring security el expression to check if the given user has the 'Schema Access' or 'Data Access' roles.

See also: `.SCHEMA_ACCESS`, `.DATA_ACCESS`

HAS_SCHEMA_ACCESS

public static final String **HAS_SCHEMA_ACCESS**

Spring security el expression to check if the given user has the 'Schema Access' role.

See also: `.SCHEMA_ACCESS`

HAS_SETTINGS_ACCESS

public static final String **HAS_SETTINGS_ACCESS**

Spring security el expression to check if the given user has the 'Settings Access' role.

See also: `.SETTINGS_ACCESS`

SCHEMA_ACCESS

public static final String **SCHEMA_ACCESS**

Users with 'Schema Access' have the ability to view the Schema Editor tab of the UI. Then can add new objects, delete existing objects and modify the fields on existing objects.

SETTINGS_ACCESS

public static final String **SETTINGS_ACCESS**

Users with 'Settings Access' have the ability to view the Settings tab. From that tab then can modify data retention policies as well as import and export schema and data.

12.65.18 Constants.Settings

public static final class **Settings**

Keys for entity settings.

Fields

ALLOW_MULTIPLE_SELECTIONS

public static final String **ALLOW_MULTIPLE_SELECTIONS**

ALLOW_USER_SUPPLIED

public static final String **ALLOW_USER_SUPPLIED**

CASCADE_DELETE

public static final String **CASCADE_DELETE**

CASCADE_PERSIST

public static final String **CASCADE_PERSIST**

CASCADE_UPDATE

public static final String **CASCADE_UPDATE**

COMBOBOX_VALUES

public static final String **COMBOBOX_VALUES**

STRING_MAX_LENGTH

public static final String **STRING_MAX_LENGTH**

STRING_TEXT_AREA

public static final String **STRING_TEXT_AREA**

TEXT_AREA_SQL_TYPE

public static final String **TEXT_AREA_SQL_TYPE**

12.65.19 Constants.Util

public static final class **Util**

The `Util` contains constant values to help avoid string literal repetition.

See also: pmd

Fields

AUTO_GENERATED

public static final String **AUTO_GENERATED**

AUTO_GENERATED_EDITABLE

public static final String **AUTO_GENERATED_EDITABLE**

CREATION_DATE_DISPLAY_FIELD_NAME

public static final String **CREATION_DATE_DISPLAY_FIELD_NAME**

CREATION_DATE_FIELD_NAME

public static final String **CREATION_DATE_FIELD_NAME**

CREATOR_DISPLAY_FIELD_NAME

public static final String **CREATOR_DISPLAY_FIELD_NAME**

CREATOR_FIELD_NAME

public static final String **CREATOR_FIELD_NAME**

DATANUCLEUS

public static final String **DATANUCLEUS**

DEFAULT_DATE_FORMAT

public static final DateFormat **DEFAULT_DATE_FORMAT**

Default `java.text.DateFormat` to be used to parse and format `java.util.Date`.

ENTITY

public static final String **ENTITY**

Constant `ENTITY` corresponding to the field name of the class that want to create a bidirectional connection with instane of `org.motechproject.mds.domain.Entity`

FALSE

public static final String **FALSE**

Constant `FALSE` corresponding to the primitive value `false`

GENERATED_FIELD_NAMES

public static final String[] **GENERATED_FIELD_NAMES**

ID_DISPLAY_FIELD_NAME

public static final String **ID_DISPLAY_FIELD_NAME**

ID_FIELD_NAME

public static final String **ID_FIELD_NAME**

MDS_DATABASE

public static final String **MDS_DATABASE**

MDS_TABLE_PREFIX

public static final String **MDS_TABLE_PREFIX**

MODIFICATION_DATE_DISPLAY_FIELD_NAME

public static final String **MODIFICATION_DATE_DISPLAY_FIELD_NAME**

MODIFICATION_DATE_FIELD_NAME

public static final String **MODIFICATION_DATE_FIELD_NAME**

MODIFIED_BY_DISPLAY_FIELD_NAME

public static final String **MODIFIED_BY_DISPLAY_FIELD_NAME**

MODIFIED_BY_FIELD_NAME

public static final String **MODIFIED_BY_FIELD_NAME**

OWNER_DISPLAY_FIELD_NAME

public static final String **OWNER_DISPLAY_FIELD_NAME**

OWNER_FIELD_NAME

public static final String **OWNER_FIELD_NAME**

SQL_QUERY

public static final String **SQL_QUERY**

TRUE

public static final String **TRUE**

Constant `TRUE` corresponding to the primitive value `true`

VALUE_GENERATOR

public static final String **VALUE_GENERATOR**

12.65.20 InstanceSecurityRestriction

public class **InstanceSecurityRestriction**

Represents a restriction on entity instances

Methods

isByCreator

public boolean **isByCreator** ( )

Returns true, if only creators of an instance should be able to access it; false otherwise

isByOwner

public boolean **isByOwner** ( )

Returns true, if only owners of an instance should be able to access it; false otherwise

isEmpty

public boolean **isEmpty** ( )

Returns true, if access to the instance is not limited to creator or owner; false otherwise

setByCreator

public void **setByCreator** (boolean *byCreator*)

setByOwner

public void **setByOwner** (boolean *byOwner*)

12.65.21 JavassistUtil

public final class **JavassistUtil**

Utils class for javassist related tasks. Helps with generic signature generation, plus methods related with analyzing and loading javassist class representations.

Methods

containsDeclaredField

public static boolean **containsDeclaredField** (CtClass *ctClass*, String *fieldName*)

containsDeclaredMethod

public static boolean **containsDeclaredMethod** (CtClass *ctClass*, String *methodName*)

containsField

public static boolean **containsField** (CtClass *ctClass*, String *fieldName*)

containsMethod

public static boolean **containsMethod** (CtClass *ctClass*, String *methodName*)

findDeclaredField

public static CtField **findDeclaredField** (CtClass *ctClass*, String *fieldName*)

findDeclaredMethod

public static CtMethod **findDeclaredMethod** (CtClass *ctClass*, String *methodName*)

findField

public static CtField **findField** (CtClass *ctClass*, String *fieldName*)

findMethod

public static CtMethod **findMethod** (CtClass *ctClass*, String *methodName*)

genericGetterSignature

public static String **genericGetterSignature** (String *genericFieldSignature*)

genericSetterSignature

public static String **genericSetterSignature** (String *genericFieldSignature*)

genericSignature

public static String **genericSignature** (Class<?> *typeClass*, Class<?> *genericParam*)

genericSignature

public static String **genericSignature** (Class<?> *typeClass*, String *genericParam*)

genericSignature

public static String **genericSignature** (String *typeClass*, String *genericParam*)

hasInterface

public static boolean **hasInterface** (CtClass *ctClass*, CtClass *ctInterface*)

inheritsFromCustomClass

public static boolean **inheritsFromCustomClass** (Class<?> *clazz*)

loadClass

public static CtClass **loadClass** (Bundle *bundle*, String *className*, ClassPool *classPool*)

removeDeclaredFieldIfExists

public static void **removeDeclaredFieldIfExists** (CtClass *ctClass*, String *fieldName*)

removeDeclaredMethodIfExists

public static void **removeDeclaredMethodIfExists** (CtClass *ctClass*, String *methodName*)

removeFieldIfExists

public static void **removeFieldIfExists** (CtClass *ctClass*, String *fieldName*)

removeMethodIfExists

public static void **removeMethodIfExists** (CtClass *ctClass*, String *methodName*)

toClassPath

public static String **toClassPath** (Class<?> *clazz*)

toClassPath

public static String **toClassPath** (String *clazz*)

toClassPath

public static String **toClassPath** (String *clazz*, boolean *extension*)

toGenericParam

public static String **toGenericParam** (Class<?> *clazz*)

toGenericParam

public static String **toGenericParam** (String *clazz*)

12.65.22 Loader

public abstract class **Loader**<T>

The `Loader` is an abstract class that checks if all class dependencies to the given class definition are resolved. If not then the missing class name is taken from exception and the `doWhenClassNotFound(String)` method is executed.

Parameters

- **<T>** – the type of argument data

Methods

doWhenClassNotFound

public abstract void **doWhenClassNotFound** (String *name*)

getClassDefinition

public abstract Class<?> **getClassDefinition** (T *arg*)

loadClass

public Class<?> **loadClass** (T *arg*)

loadFieldsAndMethodsOfClass

public void **loadFieldsAndMethodsOfClass** (Class<?> *definition*)

12.65.23 LookupName

public final class **LookupName**

Utility class for dealing with lookup names.

Methods

buildLookupFieldName

public static String **buildLookupFieldName** (String *fieldName*, String *relatedFieldName*)

Builds lookup field name which may contain information about the searching by relationship.

Parameters

- **fieldName** – The name of the field from entity
- **relatedFieldName** – The name of the field in related entity

Returns lookup field name

getFieldName

public static String **getFieldName** (String *lookupFieldName*)

Returns the name of the field from entity.

Parameters

- **lookupFieldName** – the lookup field name

Returns field name from entity

getRelatedFieldName

public static String **getRelatedFieldName** (String *lookupFieldName*)

Returns a field name in related entity.

Parameters

- **lookupFieldName** – the lookup field name

Returns field name from related entity

lookupCountMethod

public static String **lookupCountMethod** (String *lookupNameOrMethodName*)

Builds count lookup name. The resulting method name will be in form: `countXxxYyyZzz`.

Parameters

- **lookupNameOrMethodName** – name of the lookup or lookup method name

Returns count lookup method name

lookupMethod

public static String **lookupMethod** (String *lookupName*)

Builds lookup method name, based on the lookup name. The resulting method name is camelCase, based on the words in the lookup.

Parameters

- **lookupName** – name of the lookup

Returns camelCase lookup method name

12.65.24 MDSClassLoader

public class **MDSClassLoader** extends ClassLoader

The `MDSClassLoader` class is a mds wrapper for `ClassLoader`.

Constructors

MDSClassLoader

protected **MDSClassLoader** ( )

MDSClassLoader

protected **MDSClassLoader** (ClassLoader *parent*)

Methods

defineClass

public Class<?> **defineClass** (String *name*, byte[] *bytecode*)

getInstance

public static MDSClassLoader **getInstance** ( )

getStandaloneInstance

public static MDSClassLoader **getStandaloneInstance** ( )

getStandaloneInstance

public static MDSClassLoader **getStandaloneInstance** (ClassLoader *parent*)

reloadClassLoader

public static void **reloadClassLoader** ( )

safeDefineClass

public Class<?> **safeDefineClass** (String *name*, byte[] *bytecode*)

12.65.25 MemberUtil

public final class **MemberUtil**

Util class that provides convenient methods connected with the class members.

Fields

BOOLEAN_GETTER_PREFIX

public static final String **BOOLEAN_GETTER_PREFIX**

GETTER_PREFIX

public static final String **GETTER_PREFIX**

GET_OR_SET_END_INDEX

public static final int **GET_OR_SET_END_INDEX**

IS_END_INDEX

public static final int **IS_END_INDEX**

SETTER_PREFIX

public static final String **SETTER_PREFIX**

Methods

getCorrectType

public static Class<?> **getCorrectType** (AnnotatedElement *object*)

Gets annotated element type. If this element is not a member of the class, it returns null. Otherwise, it will try to resolve the type by checking it directly, or via getter/setter methods. If member is neither a field or getter/setter method, it returns null.

Parameters

- **object** – annotated element to retrieve type from

Returns type of the element, or null if not applicable

getCorrectType

public static Class<?> **getCorrectType** (Member *object*)

Gets member type. It will try to resolve the type by checking it directly, or via getter/setter methods. If member is neither a field or getter/setter method, it returns null.

Parameters

- **object** – annotated element to retrieve type from

Returns type of the element, or null if not applicable

getDeclaringClass

public static Class<?> **getDeclaringClass** (AccessibleObject *ac*)

Retrieves a declaring class for the given object. Returns null if the given object is not a member of a class.

Parameters

- **ac** – object to verify

Returns A class, to which this object belongs

getDefaultEnumName

public static String **getDefaultEnumName** (String *entityClassName*, String *fieldName*)

Builds an enum class name for given entity class name and field name.

Parameters

- **entityClassName** – the class name of the entity
- **fieldName** – the field name

Returns enum class name

getFieldAndAccessorsForElement

public static List<AccessibleObject> **getFieldAndAccessorsForElement** (AccessibleObject *ao*)

Returns a list of objects, that are either field or getter/setter methods of this field, based on single accessible object of a class. If it fails to find anything, it returns the passed object.

Parameters

- **ao** – an object to find field, getter/setter method for

Returns a list of field and getter/setter methods or `ao` if nothing has been found

getFieldName

public static String **getFieldName** (AnnotatedElement *object*)

Gets field name, from the specified annotated element. It will return null, if annotated element is not a class member. Otherwise, it will try to resolve the field name, by either reading it directly from the member, or by determining the name, based on the getter/setter method. It will return null if member is neither a field or getter/setter.

Parameters

- **object** – annotated element to retrieve field name from

Returns field name, if possible; null otherwise

getFieldName

public static String **getFieldName** (Member *object*)

Gets field name, from the specified member. It will try to resolve the field name, by either reading it directly from the member, or by determining the name, based on the getter/setter method. It will return null if member is neither a field or getter/setter.

Parameters

- **object** – class member to retrieve field name from

Returns field name, if possible; null otherwise

getFieldNameFromGetterSetterName

public static String **getFieldNameFromGetterSetterName** (String *getterSetterName*)

Attempts to retrieve field name from the getter/setter method name. It will throw `java.lang.IllegalArgumentException` if provided value is empty or does not match the setter/getter naming convention.

Parameters

- **getterSetterName** – getter/setter method name

Returns field name

getGenericType

public static Class<?> **getGenericType** (AnnotatedElement *object*)

Retrieves an actual type from the parameterized class member. If annotated element is not a class member, it returns null. It always checks for the first parameter. If you want to specify which parameter to retrieve, use `getGenericType(java.lang.reflect.AnnotatedElement,int)`. This will work on fields and getter/setter methods. It will return null for other class members or if there is no parameterized type on them.

Parameters

- **object** – annotated element to retrieve actual type from

Returns Actual type of the parameterized class member

getGenericType

public static Class<?> **getGenericType** (AnnotatedElement *object*, int *typeNumber*)

Retrieves an actual type from the parameterized class member. If annotated element is not a class member, it returns null. It will check the parameter on position `typeNumber`. This will work on fields and getter/setter methods. It will return null for other class members or if there is no parameterized type on them.

Parameters

- **object** – annotated element to retrieve actual type from
- **typeNumber** – position of the parameterized type

Returns Actual type of the parameterized class member

getGenericType

public static Class<?> **getGenericType** (Member *object*, int *typeNumber*)

Retrieves an actual type from the parameterized class member. It will check the parameter on position `typeNumber` This will work on fields and getter/setter methods. It will return null for other class members or if there is no parameterized type on them.

Parameters

- **object** – class member to retrieve actual type from
- **typeNumber** – position of the parameterized type

Returns Actual type of the parameterized class member

getGetterName

public static String **getGetterName** (String *fieldName*, CtClass *declaring*)

Returns getter method name for the given field name and the class declaration.

Parameters

- **fieldName** – field name
- **declaring** – the class declaration that contains the field

Returns getter name

getMembers

public static List<Member> **getMembers** (Class<?> *clazz*, Predicate *memberPredicate*)

Gets all members of a class, that match the specified predicate.

Parameters

- **clazz** – class to retrieve members from
- **memberPredicate** – predicate that must be fulfilled by class members

Returns list of class members

getSetterName

public static String **getSetterName** (String *fieldName*)

Returns setter method name for the given field name.

Parameters

- **fieldName** – field name

Returns setter name

isGetter

public static boolean **isGetter** (Member *member*)

Checks if given class member is a getter method. This includes boolean-specific getters, starting with "is" prefix.

Parameters

- **member** – class member to verify

Returns true if given class member is a getter method; false otherwise

isSetter

public static boolean **isSetter** (Member *member*)
Checks if given class member is a setter method.

Parameters

- **member** – class member to verify

Returns true if given class member is a setter method; false otherwise

12.65.26 NumberPredicate

public class **NumberPredicate** implements Predicate
Implementation of a `Predicate` that allows a type-independent comparision of two numbers.

Constructors

NumberPredicate

public **NumberPredicate** (Number *element*)

Methods

evaluate

public boolean **evaluate** (Object *candidate*)

12.65.27 ObjectReferenceRepository

public class **ObjectReferenceRepository**
Represents an object reference repository. It holds historical objects for the real objects with the given id and class name.

Methods

getHistoricalObject

public Object **getHistoricalObject** (Object *nonHistoricalObject*)

saveHistoricalObject

public void **saveHistoricalObject** (Object *nonHistoricalObject*, Object *historicalObject*)

12.65.28 Order

public class **Order** implements Serializable
Represents an order in a query

Constructors

Order

public **Order** (String *field*)
Creates order, with ascending direction.

Parameters

- **field** – field to order results by

Order

public **Order** (String *field*, String *direction*)
Creates order.

Parameters

- **field** – field to order results by
- **direction** – `java.lang.String` representation of a direction

Order

public **Order** (String *field*, Direction *direction*)
Creates order.

Parameters

- **field** – field to order results by
- **direction** – direction of the order

Methods

getDirection

public Direction **getDirection** ()

getField

public String **getField** ()

toString

public String **toString** ()

12.65.29 Order.Direction

public static enum **Direction**

Represents a direction of the order.

Enum Constants

ASC

public static final Order.Direction **ASC**

Ascending direction order (eg. 1, 2, 3...)

DESC

public static final Order.Direction **DESC**

Descending direction order (eg. 100, 99, 98...)

12.65.30 Pair

public interface **Pair**<N, V>

The `Pair` util interface should use everywhere where developer needs a pair of key-value

Parameters

- **<N>** – type of key
- **<V>** – type of value

See also: `org.motechproject.mds.domain.FieldMetadata`, `org.motechproject.mds.domain.FieldSet` `org.motechproject.mds.dto.MetadataDto`, `org.motechproject.mds.dto.SettingDto`

Methods

getKey

N **getKey**()

getValue

V **getValue**()

12.65.31 PropertyUtil

public final class **PropertyUtil** extends PropertyUtils

The `PropertyUtil` util class provides the same method like `org.apache.commons.beanutils.PropertyUtils` and two additional methods for safe writing and reading property in the given bean.

Methods

copyProperties

public static void **copyProperties** (Object *target*, Object *object*)

copyProperties

public static void **copyProperties** (Object *target*, Object *object*, Set<String> *fieldsToUpdate*)

getPropertyDescriptors

public static PropertyDescriptor[] **getPropertyDescriptors** (Object *bean*)

safeGetProperty

public static Object **safeGetProperty** (Object *bean*, String *name*)

safeGetPropertyType

public static Class<?> **safeGetPropertyType** (Object *bean*, String *name*)

safeSetCollectionProperty

public static void **safeSetCollectionProperty** (Object *bean*, String *name*, Collection *values*)

safeSetProperty

public static void **safeSetProperty** (Object *bean*, String *name*, Object *value*)

12.65.32 SecurityMode

public enum **SecurityMode**

This enum describes security mode for an entity

Enum Constants

CREATOR

public static final SecurityMode **CREATOR**

Only user that created an instance can access it.

EVERYONE

public static final SecurityMode **EVERYONE**

Everyone has got an access to the instances of an entity

NO_ACCESS

public static final SecurityMode **NO_ACCESS**
Nobody can access the instances of an entity

OWNER

public static final SecurityMode **OWNER**
Only user marked as an owner can access the instance of an entity

PERMISSIONS

public static final SecurityMode **PERMISSIONS**
Only users with specified permissions can access the instances of an entity.

USERS

public static final SecurityMode **USERS**
Only specified users can access the instances of an entity.

12.65.33 SecurityUtil

public final class **SecurityUtil**
The `SecurityUtil` class provides helper methods to retrieve logged user details, such as username, roles or permissions

Methods

getUserPermissions

public static Set<String> **getUserPermissions**()
Returns authenticated user's permissions. It will return empty set, in case there's no active authentication or if the current user does not have any permissions assigned.

Returns set of permissions assigned to the current user

getUserRoles

public static List<String> **getUserRoles**()
Returns authenticated user's roles. It will return empty list, in case there's no active authentication or if the current user does not have any roles assigned.

Returns list of roles assigned to the current user

getUsername

public static String **getUsername** ()

Retrieves current username from the Spring security context. Returns null, if there's no active authentication object.

Returns username or null, if no user is authenticated

12.65.34 TypeHelper

public final class **TypeHelper**

A helper class for parsing and formatting MDS supported types.

Methods

breakString

public static String[] **breakString** (String *str*)

breakString

public static String[] **breakString** (String *str*, String[] *removes*, String[] *search*, String[] *replacement*, String *separator*)

breakStringForCollection

public static String[] **breakStringForCollection** (String *str*)

buildStringFromList

public static String **buildStringFromList** (List<String> *items*)

format

public static String **format** (Object *obj*)

Creates a `java.lang.String` representation of the given value. If given object is a `java.util.List`, each new element is placed in a new line.

Parameters

- **obj** – value to retrieve `java.lang.String` representation for

Returns `java.lang.String` representation of an object

format

public static String **format** (Object *obj*, char *listJoinChar*)

Creates a `java.lang.String` representation of the given value. If given object is a `java.util.List` a character put between next values can be specified.

Parameters

- **obj** – value to retrieve `java.lang.String` representation for
- **listJoinChar** – character to put between next elements of a list; applicable if given object is a list

Returns `java.lang.String` representation of an object

getPrimitive

public static Class<?> **`getPrimitive`** (Class<?> *clazz*)

Retrieves equivalent primitive class for the given wrapper class.

Parameters

- **clazz** – wrapper class

Returns equivalent primitive class

getWrapperForPrimitive

public static Class<?> **`getWrapperForPrimitive`** (Class<?> *clazz*)

Retrieves equivalent wrapper class for the given primitive type.

Parameters

- **clazz** – primitive type

Returns equivalent wrapper class

hasPrimitive

public static boolean **`hasPrimitive`** (Class<?> *clazz*)

Verifies whether given class has got a primitive equivalent.

Parameters

- **clazz** – class to verify

Returns true if given wrapper class has got a primitive equivalent; false otherwise

isPrimitive

public static boolean **`isPrimitive`** (Class<?> *clazz*)

Verifies whether given class is primitive.

Parameters

- **clazz** – class to verify

Returns true if given class is primitive; false otherwise

isPrimitive

public static boolean **isPrimitive** (String *className*)

Verifies whether given class is primitive.

Parameters

- **className** – fully qualified class name to verify

Returns true if given class is primitive; false otherwise

isTypeSupportedInMap

public static boolean **isTypeSupportedInMap** (String *type*, boolean *isKey*)

Returns true if the given type is supported in map, otherwise false.

Parameters

- **type** – the type to be checked
- **isKey** – true if it is key of the map, otherwise false

Returns true if the given type is supported in map, otherwise false

parse

public static Object **parse** (Object *val*, Class<?> *toClass*)

Attempts to parse given value to an instance of a given class. Throws `java.lang.IllegalArgumentException` if this method is unable to parse the value.

Parameters

- **val** – value to parse
- **toClass** – a class to turn value into

Returns parsed value, and instance of the given class

parse

public static Object **parse** (Object *val*, String *toClass*)

Attempts to parse given value to an instance of a given class. Throws `java.lang.IllegalArgumentException` if this method is unable to parse the value.

Parameters

- **val** – value to parse
- **toClass** – fully qualified class name

Returns parsed value, and instance of the given class

parse

public static Object **parse** (Object *val*, String *toClass*, ClassLoader *classLoader*)

Attempts to parse given value to an instance of a given class. The class may be loaded using custom class loader, in case of a failure to load it via default class loader. Throws `java.lang.IllegalArgumentException` if this method is unable to parse the value.

Parameters

- **val** – value to parse
- **toClass** – fully qualified class name
- **classLoader** – class loader to use, in case of a failure to find class of name `toClass`

Returns parsed value, and instance of the given class

parse

public static Object **`parse`** (Object *val*, String *toClass*, String *genericType*)

Attempts to parse given value to an instance of a given class. The class may have a generic type. Throws `java.lang.IllegalArgumentException` if this method is unable to parse the value.

Parameters

- **val** – value to parse
- **toClass** – fully qualified class name
- **genericType** – fully qualified class name of a generic type

Returns parsed value, and instance of the given class

parse

public static Object **`parse`** (Object *val*, String *toClass*, String *genericType*, ClassLoader *classLoader*)

Attempts to parse given value to an instance of a given class. The class may have a generic type and can be loaded using custom class loader, in case of a failure to load it via default class loader. Throws `java.lang.IllegalArgumentException` if this method is unable to parse the value.

Parameters

- **val** – value to parse
- **toClass** – fully qualified class name
- **genericType** – fully qualified class name of a generic type
- **classLoader** – class loader to use, in case of a failure to find class of name `toClass`

Returns parsed value, and instance of the given class

parseCollection

public static Collection **`parseCollection`** (Collection *val*, Class<?> *toClassDefinition*, Class<?> *generic*)

parseDateToDate

public static Object **`parseDateToDate`** (Object *val*, String *toClass*)

Allows parsing of the various date types. Parsing from the `org.joda.time.LocalDate` and `org.motechproject.commons.date.model.Time` is not supported at the moment.

Parameters

- **val** – value to parse

- **toClass** – destination class

Returns date, parsed to the specified type

parseIntToBool

public static boolean **parseIntToBool** (Integer *val*)

Turns given `java.lang.Integer` into `boolean`.

Parameters

- **val** – value to parse

Returns true if value is not null and greater than 0; false otherwise

parseMapValue

public static Object **parseMapValue** (Object *valueToParse*, String *type*, boolean *isKey*)

parseNumber

public static Number **parseNumber** (Object *val*, String *toClass*)

Parses given value into `java.lang.Number` or one of the standard Java types, extending Number.

Parameters

- **val** – value to parse
- **toClass** – fully qualified class name

Returns parsed value

parseString

public static Object **parseString** (String *str*, Class<?> *toClass*)

Attempts to parse given String to an instance of a given class. It will throw `java.lang.IllegalStateException` in case this method was not able to parse the value.

Parameters

- **str** – String to parse
- **toClass** – a class to turn value into

Returns parsed value, an instance of the given class

parseString

public static Object **parseString** (String *str*, String *toClass*)

Attempts to parse given String to an instance of a given class. It will throw `java.lang.IllegalStateException` in case this method was not able to parse the value.

Parameters

- **str** – String to parse
- **toClass** – fully qualified class name

Returns parsed value, an instance of the given class

parseString

public static Object **parseString** (String *str*, Class<?> *toClass*, Class<?> *generic*)

Attempts to parse given String to an instance of a given class. The class may also have a generic type. It will throw `java.lang.IllegalStateException` in case this method was not able to parse the value.

Parameters

- **str** – String to parse
- **toClass** – a class to turn value into
- **generic** – generic class

Returns parsed value, an instance of the given class

parseStringToMap

public static Map **parseStringToMap** (String *str*)

Parses given `java.lang.String` to `java.util.Map`. Each new entry should be preceeded by a comma mark (,). The key and value should be split with a colon mark (:). By default parsed values will be String type.

Parameters

- **str** – String to parse

Returns Map, parsed from the given String

parseStringToMap

public static Map **parseStringToMap** (String *keyClass*, String *valueClass*, String *str*)

Parses given `java.lang.String` to `java.util.Map`. Each new entry should be preceeded by a comma mark (,). The key and value should be split with a colon mark (:). Types of the parsed values depend on the given keyClass and valueClass.

Parameters

- **keyClass** – the type of key
- **valueClass** – the type of value
- **str** – String String to parse

Returns Map, parsed from the given String

suggestCollectionImplementation

public static Class **suggestCollectionImplementation** (String *collectionClass*)

Returns concrete class, for the given collection interface or abstract class. If given class is already concrete, it will return that class. Throws `java.lang.IllegalArgumentException` if class of given name cannot be loaded.

Parameters

- **collectionClass** – fully qualified class name to find implementation for

Returns concrete class

suggestCollectionImplementation

public static Class **suggestCollectionImplementation** (Class *collectionClass*)

Returns concrete class, for the given collection interface or abstract class. If given class is already concrete, it will return that class.

Parameters

- **collectionClass** – collection class to find implementation for

Returns concrete class

toRange

public static Range **toRange** (Object *object*, String *typeClass*)

Parses given value to `org.motechproject.commons.api.Range`. If passed value is assignable neither to range nor to map, it throws `java.lang.IllegalArgumentException`. If value is a map, it should contain keys "min" and "max".

Parameters

- **object** – value to parse
- **typeClass** – fully qualified class name of the range values

Returns `org.motechproject.commons.api.Range` value

toSet

public static Set **toSet** (Object *object*, String *typeClass*, ClassLoader *classLoader*)

Parses given `java.util.Collection` class into `java.util.Set`. If given value is not a subtype of `java.util.Collection` it throws `java.lang.IllegalArgumentException`.

Parameters

- **object** – value to parse
- **typeClass** – type of the values that should be placed in `java.util.Set`
- **classLoader** – optional class loader to use, loading type class

Returns Set, parsed from the given value

12.65.35 ValidationUtil

public final class **ValidationUtil**

Common validation utils for mds.

Methods

validateNoJavaKeyword

public static void **validateNoJavaKeyword** (String *str*)

Verifies that given string is not a reserved Java keyword. Throws

`org.motechproject.mds.ex.entity.ReservedKeywordException` if given String is reserved.

Parameters

- **str** – String to verify

12.66 org.motechproject.mdsmigration.java

12.66.1 AbstractMDSMigration

public abstract class **`AbstractMDSMigration`** implements SpringJdbcMigration

This is an abstract class for java migrations. Flyway-core does not see java migration files inside motech-platform-dataservices, because of that we need this class here to invoke by reflections the real implementation of migrations.

Methods

getMigrationImplClassName

public abstract String **`getMigrationImplClassName`**()

Returns the class name of migration.

Returns the class name of migration

migrate

public void **`migrate`**(JdbcTemplate *jdbcTemplate*)

12.66.2 V33__MOTECH1359

public class **`V33__MOTECH1359`** extends AbstractMDSMigration

Methods

getMigrationImplClassName

public String **`getMigrationImplClassName`**()

12.67 org.motechproject.osgi.web

12.67.1 ApplicationContextTracker

public abstract class **`ApplicationContextTracker`** extends ServiceTracker

Base class for every class that wishes to track Spring application context. Contains a methods that help with synchronous processing.

Constructors

ApplicationContextTracker

public **ApplicationContextTracker** (BundleContext *context*)

Methods

contextInvalidOrProcessed

protected boolean **contextInvalidOrProcessed** (ServiceReference *serviceReference*, ApplicationContext *applicationContext*)

Checks whether the given context is still valid (by checking its service reference) and not yet processed.

Parameters

- **serviceReference** – the service reference for the context
- **applicationContext** – the context to check

Returns true if the context is invalid or already processed, false otherwise

getLock

protected Object **getLock** ()

Returns an object that should be used as a lock for synchronization of context processing

Returns a lock for synchronization

markAsProcessed

protected void **markAsProcessed** (ApplicationContext *applicationContext*)

Marks the given application context as already processed by this tracker, by saving its id.

Parameters

- **applicationContext** – the application context to be marked as processed

removeFromProcessed

protected void **removeFromProcessed** (ApplicationContext *applicationContext*)

Undoes marking an application context as processed by this tracked. Its id is removed from the list of processed ids.

Parameters

- **applicationContext** – the application context to remove from the list of processed contexts

12.67.2 BlueprintActivator

public class **BlueprintActivator** implements BundleActivator

The bundle activator used by this (osgi-web-util) module. It launches a `org.motechproject.osgi.web.BlueprintApplicationContextTracker` that tracks

blueprint contexts and does all the processing required for making those contexts into fully functional modules.

See also: `org.motechproject.osgi.web.BlueprintApplicationContextTracker`

Methods

start

public void **`start`** (BundleContext *context*)

stop

public void **`stop`** (BundleContext *context*)

12.67.3 BlueprintApplicationContextTracker

public class **`BlueprintApplicationContextTracker`** extends ApplicationContextTracker

The `BlueprintApplicationContextTracker` class tracks application contexts, which are registered as services by the Gemini extender. This is the main processor for MOTECH modules. For each module it will create an `org.motechproject.osgi.web.HttpServiceTracker` and a `org.motechproject.osgi.web.UIServiceTracker`. These trackers will be responsible for registering the module with `org.osgi.service.http.HttpService` (so that they can expose an HTTP endpoint) and the `org.motechproject.osgi.web.UIFrameworkService` (so that they can register their UI) respectively. This module also uses the `org.motechproject.osgi.web.Log4JBundleLoader` for loading log4j configuration files from the registered modules. The processing is only performed for bundles that have the `Blueprint-Enabled` header in their manifest.

Constructors

BlueprintApplicationContextTracker

public **`BlueprintApplicationContextTracker`** (BundleContext *context*)

Methods

addingService

public Object **`addingService`** (ServiceReference *serviceReference*)

removedService

public void **`removedService`** (ServiceReference *reference*, Object *service*)

12.67.4 BundleContextWrapper

public class **BundleContextWrapper** implements BundleContextAware

This is a wrapper class for the OSGi `org.osgi.framework.BundleContext` class. It provides convenience methods for processing Blueprint contexts of modules. This class implements `org.eclipse.gemini.blueprint.context.BundleContextAware`, so if it's published as a Spring bean, the bundle context object should get injected by Spring.

Fields

CONTEXT_SERVICE_NAME

public static final String **CONTEXT_SERVICE_NAME**

The service property set by Blueprint for application contexts published at services. The value for this property will be equal to the symbolic name of the bundle from which the context comes from. It allows to retrieve a published context for a given bundle from the bundle context.

Constructors

BundleContextWrapper

public **BundleContextWrapper**()

The default constructor, expects the bundle context to be injected by Spring.

BundleContextWrapper

public **BundleContextWrapper**(BundleContext *context*)

Constructs this wrapper for a given bundle context.

Parameters

- **context** – the bundle context to wrap around

Methods

getBundleApplicationContext

public ApplicationContext **getBundleApplicationContext**()

Returns the Spring `org.springframework.context.ApplicationContext` created by Blueprint for the bundle the underlying bundle context comes from. The context is retrieved from the bundle context, since Blueprint publishes application contexts as OSGi services.

Returns the context created by Blueprint for the bundle the underlying context comes from, or null if there is no context

getBundleContext

public BundleContext **getBundleContext**()

Returns the underlying `org.osgi.framework.BundleContext` object

getCurrentBundleSymbolicName

public String **getCurrentBundleSymbolicName**()

Returns the symbolic name of the bundle from which the underlying context comes from

getService

public <T> T **getService**(Class<T> *clazz*)

Retrieves an OSGi service using the underlying bundle context. Note that this will return the service from the first reference, so multiple services for one class are not supported by this method (you will get a random instance).

Parameters

- **clazz** – the class of the service to retrieve
- **<T>** – the class of the service to retrieve

Returns an OSGi service for the class, or null if there is no such service available

setBundleContext

public void **setBundleContext**(BundleContext *bundleContext*)

12.67.5 BundleRegister

public final class **BundleRegister**

The `BundleRegister` Singleton class is used for recording bundles. This class will help to reconfigure logger's levels.

Methods

addBundle

public void **addBundle**(Bundle *bundle*)

getBundleList

public List<Bundle> **getBundleList**()

getInstance

public static BundleRegister **getInstance**()

12.67.6 BundledJspView

public class **BundledJspView** extends JstlView implements BundleContextAware

This is a class that should be used as the **viewClass** with Spring view resolvers in order to support loading of JSP pages coming from OSGi bundles in Tomcat. This class will set the **org.apache.catalina.jsp_file** attribute in the request. That attribute is recognized by Tomcat and will make it load the JSP from the bundle. When registered as bean, this will obtain the bundle context (since it implements `org.eclipse.gemini.blueprint.context.BundleContextAware` and will point Tomcat to resources of the bundle the context comes from.

Methods

render

public void **render** (Map<String, ?> *model*, HttpServletRequest *request*, HttpServletResponse *response*)

setBundleContext

public void **setBundleContext** (BundleContext *bundleContext*)

12.67.7 HttpServiceTracker

public class **HttpServiceTracker** extends ServiceTracker

This is the HttpServiceTracker that will be created by `org.motechproject.osgi.web.BlueprintApplicationCont` for bundles that have a Gemini Blueprint context and the `Blueprint-Enabled` header in their manifest. This class is responsible for tracking the `org.osgi.service.http.HttpService`. Once it becomes available, an `OSGiDispatcherServlet` is created and registered with service, which means exposing and HTTP endpoint for the bundle. We also create and register an `org.motechproject.osgi.web.OSGiDispatcherServlet` with a context built upon the context created by the Gemini Extender. The dispatcher servlet created here allows HTTP access to the bundle, by making its Spring context the parent of the dispatchers context.

Constructors

HttpServiceTracker

public **HttpServiceTracker** (BundleContext *context*, Map<String, String> *resourceMapping*)

Methods

addingService

public Object **addingService** (ServiceReference *ref*)

removedService

public void **removedService** (ServiceReference *ref*, Object *service*)

start

public void **start** ( )

unregister

public void **unregister** ( )

12.67.8 HttpServiceTrackers

public class **HttpServiceTrackers**

This is responsible for creating and keeping track of `org.motechproject.osgi.web.HttpServiceTracker` instances.

Constructors

HttpServiceTrackers

public **HttpServiceTrackers** (BundleContext *bundleContext*)

Creates this http tracker registry and registers a bundle listener, that will close the trackers for a bundle once it is stopped.

Parameters

- **bundleContext** – the context used for registering the bundle listener

Methods

addTrackerFor

public HttpServiceTracker **addTrackerFor** (Bundle *bundle*)

Creates an `org.motechproject.osgi.web.HttpServiceTracker` instance for the given bundle.

Parameters

- **bundle** – the bundle for which the tracker should be created

Returns the newly created tracker

isBeingTracked

public boolean **isBeingTracked** (Bundle *bundle*)

Checks whether an `org.motechproject.osgi.web.HttpServiceTracker` exists for the given bundle.

Parameters

- **bundle** – the bundle to check

Returns true if a tracker exists, false otherwise

removeTrackerFor

public HttpServiceTracker **removeTrackerFor** (Bundle *bundle*)

Removes a tracker for a given bundle. The tracker is also unregistered and closed cleanly.

Parameters

- **bundle** – the bundle to remove the tracker for

Returns the closed tracker instance, dropped from this registry

12.67.9 LocaleService

public interface **LocaleService**

A service responsible for localization. Allows retrieval/settings of user language as well as retrieving localized messages for a user request. Can also be used to retrieve a list of usable languages.

Methods

getMessages

Map<String, String> **getMessages** (HttpServletRequest *request*)

getSupportedLanguages

NavigableMap<String, String> **getSupportedLanguages** ()

getUserLocale

Locale **getUserLocale** (HttpServletRequest *request*)

setSessionLocale

void **setSessionLocale** (HttpServletRequest *request*, HttpServletResponse *response*, Locale *locale*)

setUserLocale

void **setUserLocale** (HttpServletRequest *request*, HttpServletResponse *response*, Locale *locale*)

12.67.10 Log4JBundleLoader

public class **Log4JBundleLoader**

This `Log4JBundleLoader` class is responsible for loading configuration of loggers from properties located in bundle classpaths (log4j.xml).

Methods

checkListContainLogger

public boolean **checkListContainLogger** (List<LogMapping> *loggers*, String *log*)

checkLogXmlConfiguration

public boolean **checkLogXmlConfiguration** (Document *log4jDoc*)

createLoggerProperties

public Properties **createLoggerProperties** (List<LogMapping> *log*)

loadBundle

public void **loadBundle** (Bundle *bundle*)

loadLoggerDbConfiguration

public void **loadLoggerDbConfiguration** ()

setLog4jConf

public void **setLog4jConf** (String *log4jConf*)

12.67.11 ModuleRegistrationData

public class **ModuleRegistrationData**

Object used to registered a module withing the Motech UI system. Represents a module and is used for building the common user interface. All modules that wish to register within the UI system must either expose this class as a spring bean in their application context or manually register it through the `UIFrameworkService` OSGi service.

See also: `UIFrameworkService`

Constructors

ModuleRegistrationData

public **ModuleRegistrationData** ()

ModuleRegistrationData

public **ModuleRegistrationData** (String *moduleName*, String *url*)

Parameters

- **moduleName** – the name of the module
- **url** – the url under which this module can be accessed

ModuleRegistrationData

public **ModuleRegistrationData** (String *moduleName*, Map<String, String> *i18n*)

Constructor for modules that just want to register i18n files (i.e. for tasks).

Parameters

- **moduleName** – the name of the module
- **i18n** – a map, where the keys are the names of the i18n files and values are their locations (HTTP locations)

ModuleRegistrationData

public **ModuleRegistrationData** (String *moduleName*, String *url*, List<String> *angularModules*, Map<String, String> *i18n*)

Constructor for modules that want to register their own panels in the UI.

Parameters

- **moduleName** – the name of the module
- **url** – the url under which this module can be accessed
- **angularModules** – the list of angular modules that should be loaded on the UI for this module
- **i18n** – a map, where the keys are the names of the i18n files and values are their locations (HTTP locations)

Methods

addAngularModule

public void **addAngularModule** (String *moduleName*)

Adds an AngularJS module that should be loaded with this module.

Parameters

- **moduleName** – the name of the angular module that should be loaded for this module on the UI

addI18N

public void **addI18N** (String *fileName*, String *fileLocation*)

Adds i18n messages file entry for this module. Messages from this file will be loaded on the UI.

Parameters

- **fileName** – the name of the file
- **fileLocation** – the location of the file (HTTP location, ../mymodule/messages for example)

addSubMenu

public void **addSubMenu** (String *url*, String *label*)

Adds a submenu to this module. Submenu is a link on the left side of the UI.

Parameters

- **url** – the url to which the link will redirect to
- **label** – the label that will be displayed on the UI

addSubMenu

public void **addSubMenu** (String *url*, String *label*, String *roleForAccess*)

Adds a submenu to this module. Submenu is a link on the left side of the UI.

Parameters

- **url** – the url to which the link will redirect to
- **label** – the label that will be displayed on the UI
- **roleForAccess** – the permission required to view this sub menu (will be hidden if the user doesn't have the permission)

equals

public boolean **equals** (Object *o*)

getAngularModules

public List<String> **getAngularModules** ()

Returns the list of angular modules that should be loaded on the UI for this module

getAngularModulesStr

public String **getAngularModulesStr** ()

Returns the list of angular modules that should be loaded for this module in a format that can be used in Javascript.

Returns the list of Angular modules in javascript format

getBundle

public Bundle **getBundle** ()

Returns the underlying OSGi bundle for this module

getCriticalMessage

public String **getCriticalMessage**()

Returns a critical message for this module. That message should be displayed on the UI if the module needs attention.

Returns the critical message that should be communicated to the user

getDefaultURL

public String **getDefaultURL**()

Returns the default url for this module. Usually points to one of its sub-views.

getDocumentationUrl

public String **getDocumentationUrl**()

Returns the documentation url for this module. A small link at the bottom of the screen will point to this specific documentation, the admin module will also link to it. The link can be external.

Returns the url of the documentation

getI18n

public Map<String, String> **getI18n**()

Returns the list of i18n message files for this module.

Returns a map, where the keys are the names of the i18n files and values are their locations (HTTP locations)

getModuleName

public String **getModuleName**()

Returns the name of the module represented by this object

getResourcePath

public String **getResourcePath**()

Returns the path of the module's static resources

getRestDocsPath

public String **getRestDocsPath**()

Returns the path to the REST API specification for this module. This file will be used for generating a Swagger UI for the API.

Returns the path to REST API specification for this module

getRoleForAccess

public List<String> **getRoleForAccess**()

Returns the permission names required to access this module - used to hide its menus on the UI. This module should be exposed only if the user has at least one of these permissions.

Returns the permissions required for accessing the module

getSettingsURL

public String **getSettingsURL**()

Returns the settings url for this module. This is used for linking to the custom settings page for a module in the admin UI instead of generating a UI for the settings.

Returns the path to the custom setting page for this module

getSubMenu

public Map<String, SubmenuInfo> **getSubMenu**()

Returns the list of sub-menus for this module.

Returns a map where the keys are the names of the sub-menus and values are their representations

getUrl

public String **getUrl**()

Returns the url for accessing this module

hashCode

public int **hashCode**()

isNeedsAttention

public boolean **isNeedsAttention**()

Checks whether this module needs attention - meaning it requires a UI notification pointing to it.

Returns true if the module needs attention, false otherwise

removeAngularModule

public void **removeAngularModule**(String *moduleName*)

Removes an AngularJS module from the list of modules should be loaded with this module.

Parameters

- **moduleName** – the name of the angular module that should no longer be loaded for this module on the UI

setBundle

public void **setBundle** (Bundle *bundle*)

Parameters

- **bundle** – the underlying OSGi bundle for this module

setCriticalMessage

public void **setCriticalMessage** (String *criticalMessage*)

Sets a critical message for this module. That message should be displayed on the UI if the module needs attention.

Parameters

- **criticalMessage** – the critical message that should be communicated to the user

setDefaultURL

public void **setDefaultURL** (String *defaultURL*)

Parameters

- **defaultURL** – the default url for this module. Usually points to one of its sub-views.

setModuleName

public void **setModuleName** (String *moduleName*)

Parameters

- **moduleName** – the name of the module represented by this object

setNeedsAttention

public void **setNeedsAttention** (boolean *needsAttention*)

Sets whether this module needs attention - meaning it requires a UI notification pointing to it.

Parameters

- **needsAttention** – true if the module needs attention, false otherwise

setResourcePath

public void **setResourcePath** (String *resourcePath*)

Parameters

- **resourcePath** – the path of the module's static resources

setRestDocsPath

public void **setRestDocsPath** (String *restDocsPath*)

Sets the path to the REST API specification for this module. This file will be used for generating a Swagger UI for the API.

Parameters

- **restDocsPath** – the path to REST API specification for this module

setRoleForAccess

public void **setRoleForAccess** (String *role*)

Sets a permission name required to access this module - used to hide its menus on the UI. This module should be exposed only if the user has this permissions.

Parameters

- **role** – the permission required for accessing the module

setRoleForAccess

public void **setRoleForAccess** (List<String> *roles*)

Sets the permission names required to access this module - used to hide its menus on the UI. This module should be exposed only if the user has at least one of these permissions.

Parameters

- **roles** – the permissions required for accessing the module

setSettingsURL

public void **setSettingsURL** (String *settingsURL*)

Sets the settings url for this module. This is used for linking to the custom settings page for a module in the admin UI instead of generating a UI for the settings.

Parameters

- **settingsURL** – the path to the custom setting page for this module

setSubMenu

public void **setSubMenu** (Map<String, SubmenuInfo> *subMenu*)

Sets the list of sub-menus for this module.

Parameters

- **subMenu** – a map where the keys are the names of the sub-menus and values are their representations

setUrl

public void **setUrl** (String *url*)

Parameters

- **url** – the url for accessing this module

subMenuNeedsAttention

public void **subMenuNeedsAttention** (String *submenu*)

Marks a sub-menu of this module as needing attention - meaning it requires a UI notification pointing to it.

Parameters

- **submenu** – the sub-menu to mark as requiring attention

submenuBackToNormal

public void **submenuBackToNormal** (String *submenu*)

Marks a sub-menu of this module as not needing attention anymore - meaning no special UI notification should point to it.

Parameters

- **submenu** – the sub-menu to mark as requiring attention

12.67.12 MotechOSGiWebApplicationContext

public class **MotechOSGiWebApplicationContext** extends OsgiBundleXmlApplicationContext implements ConfigurableWebA

The context that is created for all Blueprint-Enabled bundles. This context will be used for the `org.motechproject.osgi.web.OSGiDispatcherServlet` that we create in the `org.motechproject.osgi.web.HttpServiceTracker`.

Constructors

MotechOSGiWebApplicationContext

public **MotechOSGiWebApplicationContext** ()

Methods

getNamespace

public String **getNamespace** ()

getServletConfig

public ServletConfig **getServletConfig** ()

getServletContext

public ServletContext **getServletContext**()

isInitialized

public boolean **isInitialized**()

setConfigLocation

public void **setConfigLocation**(String *configLocation*)

setNamespace

public void **setNamespace**(String *namespace*)

setServletConfig

public void **setServletConfig**(ServletConfig *servletConfig*)

setServletContext

public void **setServletContext**(ServletContext *servletContext*)

waitForContext

public void **waitForContext**(int *waitTimeInMillis*)

12.67.13 OSGiDispatcherServlet

public class **OSGiDispatcherServlet** extends DispatcherServlet

This class extends Spring's `org.springframework.web.servlet.DispatcherServlet` and is used by MOTECH for registering HTTP endpoints. This extension adds support for OSGi by making sure that `MotechOSGiWebApplicationContext` instances are connected with their parent context instances, created by the Gemini Extender. It also injects the bundle context into those contexts.

Constructors

OSGiDispatcherServlet

public **OSGiDispatcherServlet**(BundleContext *bundleContext*)

OSGiDispatcherServlet

public **OSGiDispatcherServlet**(BundleContext *bundleContext*, ConfigurableWebApplicationContext *configurableWebApplicationContext*)

Methods

initFrameworkServlet

protected void **initFrameworkServlet** ( )

postProcessWebApplicationContext

protected void **postProcessWebApplicationContext** (ConfigurableWebApplicationContext *wac*)

12.67.14 SubmenuInfo

public class **SubmenuInfo**

Class to encapsulate information about sub-menu links to be shown on UI. This represents a link shown on the left side of the UI.

Constructors

SubmenuInfo

public **SubmenuInfo** ( )

SubmenuInfo

public **SubmenuInfo** (String *url*)

Constructs an instance for a given url.

Parameters

- **url** – the url this links to

Methods

getCriticalMessage

public String **getCriticalMessage** ( )

Returns the critical message for this sub-menu, if it has been set.

Returns the critical message for this link

getRoleForAccess

public List<String> **getRoleForAccess** ( )

Returns a list of permissions required to access this sub-menu item. It is required for the user to have at least permission from this, not all. This link will be hidden for users without the permissions.

Returns the list of permissions for access

getUrl

public String **getUrl** ()

Returns the url this links to

isNeedsAttention

public boolean **isNeedsAttention** ()

Returns true if this sub-menu should be marked as requiring attention on the UI, false otherwise

setCriticalMessage

public void **setCriticalMessage** (String *criticalMessage*)

Sets the critical message for this sub-menu, it will displayed on the UI as a tooltip.

Parameters

- **criticalMessage** – the critical message for this link

setNeedsAttention

public void **setNeedsAttention** (boolean *needsAttention*)

Parameters

- **needsAttention** – true if this sub-menu should be marked as requiring attention on the UI, false otherwise

setRoleForAccess

public void **setRoleForAccess** (String *roleForAccess*)

setRoleForAccess

public void **setRoleForAccess** (List<String> *roleForAccess*)

Sets the list of permissions required to access this sub-menu item. It is required for the user to have at least permission from this, not all. This link will be hidden for users without the permissions.

Parameters

- **roleForAccess** – the list of permissions for access

setUrl

public void **setUrl** (String *url*)

Parameters

- **url** – the url this links to

12.67.15 UIFrameworkService

public interface **UIFrameworkService**

Service responsible for managing the user interface. Provides methods for registering/un-registering modules. All modules are represented by `ModuleRegistrationData` objects, either registered directly through this service or automatically by exposing it in their spring context. This service also allows manipulation of module state, by marking given modules as requiring attention on the UI.

Methods

getModuleData

ModuleRegistrationData **getModuleData** (String *moduleName*)

Gets registration data for the module with the given name.

Parameters

- **moduleName** – the name of the module for which the registration data should be retrieved

Returns the registration data for the given module, or null if such a module is not registered

getModuleDataByAngular

ModuleRegistrationData **getModuleDataByAngular** (String *angularModule*)

Retrieves module registration data which registers the given AngularJS module.

Parameters

- **angularModule** – the name of the Angular module

Returns the registration for the module that registers the Angular module

getModuleDataByBundle

ModuleRegistrationData **getModuleDataByBundle** (Bundle *bundle*)

Retrieves the module registration data for a given bundle. Since `org.motechproject.osgi.web.ModuleRegistrationData` held by the service contain references to the bundles registering them, this will match relying on Bundle.equals().

Parameters

- **bundle** – the bundle for which the registration data should be retrieved

Returns the registration data for the given bundle (module)

getRegisteredModules

ModuleRegistrations **getRegisteredModules** ()

Returns information about all modules registered with the UI system. The modules are grouped into module with their own submenus (links to menus in the top menu), those without their submenus (they are all placed in the modules section) and those without UI altogether (messages from them will be loaded though, for usage in tasks and possibly other places).

Returns all modules registered with the system

getRestDocLinks

Map<String, String> **getRestDocLinks** ()

Returns a map of links to Swagger rest specifications registered with the system. Keys are module names, values are urls to specifications.

Returns the map containing all registered rest doc urls as values, and module names as keys

isModuleRegistered

boolean **isModuleRegistered** (String *moduleName*)

Checks whether the module with a given name is registered in the UI system.

Parameters

- **moduleName** – the name of the module

Returns true if the module is registered, false otherwise

moduleBackToNormal

void **moduleBackToNormal** (String *moduleName*)

Marks the module as no longer requiring attention, undoing the `moduleNeedsAttention(String,String)` call. The red marker will disappear from the link of that module.

Parameters

- **moduleName** – the name of the module that is back to normal

moduleBackToNormal

void **moduleBackToNormal** (String *moduleName*, String *submenu*)

Marks the module and submenu as no longer requiring attention, undoing the `moduleNeedsAttention(String,String,String)` call. The red marker will disappear from the link of that module and submenu.

Parameters

- **moduleName** – the name of the module that is back to normal
- **submenu** – the name of the submenu that is back to normal

moduleNeedsAttention

void **moduleNeedsAttention** (String *moduleName*, String *message*)

Marks a module as requiring attention, by setting the `org.motechproject.osgi.web.ModuleRegistrationData.se` flag and the critical message for the module using `org.motechproject.osgi.web.ModuleRegistrationData.setC` This will mark the module as needing attention on the UI by display a red warning sign next to its link and displaying the message.

Parameters

- **moduleName** – the name of the module to mark as requiring attention
- **message** – the message explaining the reason for this alert

moduleNeedsAttention

void **moduleNeedsAttention** (String *moduleName*, String *submenu*, String *message*)

Does the same as `moduleNeedsAttention(String,String)`, with the difference that an entire submenu (the top level menu) will be marked as requiring attention.

Parameters

- **moduleName** – the name of the module to mark as requiring attention
- **submenu** – the name of the submenu to be marked as requiring attention
- **message** – the message explaining the reason for this alert

registerModule

void **registerModule** (ModuleRegistrationData *module*)

Registers a module in the UI system.

Parameters

- **module** – a `org.motechproject.osgi.web.ModuleRegistrationData` representing the module to register

unregisterModule

void **unregisterModule** (String *moduleName*)

Unregisters module from the UI system.

Parameters

- **moduleName** – the name of the module to unregister

12.67.16 UIServiceTracker

public class **UIServiceTracker** extends ServiceTracker

A tracker created for each bundle with the `Blueprint-Enabled` header in its manifest. This tracker will track the `org.motechproject.osgi.web.UIFrameworkService`, once it becomes active it registers the bundle with it - thanks to this the `org.motechproject.osgi.web.ModuleRegistrationData` beans defined in the modules will be respected and will make the module incorporated into the UI.

Constructors

UIServiceTracker

public **UIServiceTracker** (BundleContext *context*, ModuleRegistrationData *moduleRegistrationData*)

Constructs the tracker instance for a bundle.

Parameters

- **context** – the context of the bundle for which this tracker should work
- **moduleRegistrationData** – the module registration data for the bundle that will be used when registering with the UI service

UIServiceTracker

public **UIServiceTracker** (BundleContextWrapper *wrapper*, ModuleRegistrationData *moduleRegistrationData*)

Constructs the tracker instance for a bundle.

Parameters

- **wrapper** – a wrapper of the context of the bundle for which this tracker should work
- **moduleRegistrationData** – the module registration data for the bundle that will be used when registering with the UI service

Methods

addingService

public Object **addingService** (ServiceReference *ref*)

removedService

public void **removedService** (ServiceReference *ref*, Object *service*)

start

public void **start** ()

12.67.17 UIServiceTrackers

public class **UIServiceTrackers**

The registry that handles `org.motechproject.osgi.web.UIServiceTracker` instances created for bundles with `Blueprint-Enabled` header in their manifest.

See also: `org.motechproject.osgi.web.UIServiceTracker`

Constructors

UIServiceTrackers

public **UIServiceTrackers** (BundleContext *bundleContext*)

Constructs the registry and registers a bundle listener that will remove trackers for bundles that are stopping.

Parameters

- **bundleContext** – the bundle context used to register the bundle listener

Methods

addTrackerFor

public UIServiceTracker **addTrackerFor** (Bundle *bundle*, ApplicationContext *applicationContext*)

Creates a `org.motechproject.osgi.web.UIServiceTracker` for the given bundle. The `org.motechproject.osgi.web.ModuleRegistrationData` from the provided Spring context of the bundle will be used for registering the UI.

Parameters

- **bundle** – the bundle to create the tracker for
- **applicationContext** – the Spring context of the bundle

Returns the newly created `org.motechproject.osgi.web.UIServiceTracker` instance

isBeingTracked

public boolean **isBeingTracked** (Bundle *bundle*)

Checks whether this registry already has a tracker for the given bundle.

Parameters

- **bundle** – the bundle to check

Returns true if the registry has a tracker for the bundle, false otherwise

removeTrackerFor

public UIServiceTracker **removeTrackerFor** (Bundle *bundle*)

Closes and removes the `org.motechproject.osgi.web.UIServiceTracker` for the bundle.

Parameters

- **bundle** – the bundle to remove the tracker for

Returns the closed and removed tracker

12.68 org.motechproject.osgi.web.domain

12.68.1 LogMapping

public class **LogMapping**

The `LogMapping` class is used for mapping logger objects from the saved log4j properties.

Constructors

LogMapping

public **LogMapping** ()

LogMapping

public **LogMapping** (String *logName*, String *logLevel*)

Methods

equals

public boolean **equals** (Object *obj*)

getLogLevel

public String **getLogLevel** ()

Returns the log level for the logger represented by this mapping

getLogName

public String **getLogName** ()

Returns the name of the logger represented by this mapping

hashCode

public int **hashCode** ()

setLogLevel

public void **setLogLevel** (String *logLevel*)

Parameters

- **logLevel** – the log level for the logger represented by this mapping

setLogName

public void **setLogName** (String *logName*)

Parameters

- **logName** – the name of the logger represented by this mapping

toString

public String **toString** ()

12.69 org.motechproject.osgi.web.exception

12.69.1 RenderException

public class **RenderException** extends Exception
Signals an exception with rendering a JSP view.

Constructors

RenderException

public **RenderException** (String *message*)

RenderException

public **RenderException** (String *message*, Throwable *cause*)

RenderException

public **RenderException** (Throwable *cause*)

12.69.2 ServiceWaitInterruptedException

public class **ServiceWaitInterruptedException** extends RuntimeException
Exception that signals that waiting for an OSGi service was interrupted.

Constructors

ServiceWaitInterruptedException

public **ServiceWaitInterruptedException** (String *serviceCLassName*, InterruptedException *cause*)

12.69.3 ServletRegistrationException

public class **ServletRegistrationException** extends RuntimeException
Thrown when an error occurs during registration of a module servlet.

Constructors

ServletRegistrationException

public **ServletRegistrationException** (String *message*)

ServletRegistrationException

public **ServletRegistrationException** (String *message*, Throwable *cause*)

ServletRegistrationException

public **ServletRegistrationException** (Throwable *cause*)

12.70 org.motechproject.osgi.web.ext

12.70.1 ApplicationEnvironment

public final class **ApplicationEnvironment**

Utility class for handling the `ENVIRONMENT` system variable. The only meaningful value for the variable at the moment `DEVELOPMENT`, which will cause MOTECH to load static resources from disk paths instead of jar classpaths. It also allows resolving of these disk paths for given bundle name.

Fields

DEVELOPMENT

public static final String **DEVELOPMENT**

The value representing development mode.

ENVIRONMENT

public static final String **ENVIRONMENT**

The name of the variable controlling whether we are in development mode.

Methods

getEnvironment

public static String **getEnvironment** ()

Returns the value of the environment variable

getModulePath

public static String **getModulePath** (BundleName *bundleName*)

Returns the root disk path from which resources for the bundle with a given name should be loaded. This is controlled by system variables with names equal to bundle symbolic names with dots and dashes replaced with underscores. For example, the path for a bundle with the symbolic name `org.motechproject.cms-lite` is controlled by the system variable `org_motechproject_cms_lite`.

Parameters

- **bundleName** – the name of bundle

Returns the root path from which to load bundle resources or null if it is not set

isInDevelopmentMode

public static boolean **isInDevelopmentMode**()

Checks whether we are in development mode, meaning that the `ENVIRONMENT` system variable was set to `DEVELOPMENT`.

Returns true if we are in development mode, false otherwise

12.70.2 BundleName

public class **BundleName**

A wrapper for a bundle symbolic name. Provides a convenience method for converting the name to a form that can be used as a system variable name by replacing dashes and dots with underscores.

Constructors

BundleName

public **BundleName**(String *bundleSymbolicName*)

Methods

equals

public boolean **equals**(Object *bundle*)

hashCode

public int **hashCode**()

underscore

public String **underscore**()

Converts the symbolic name represented by this object to a form that can be used as a system variable name. It will replace dots and dashes with underscores. For example `org.motechproject.cms-lite` will will be converted to org_motechproject_cms_lite.

Returns the symbolic name in a form that can be used as a system variable name

12.70.3 FileSystemAwareUIHttpContext

public class **FileSystemAwareUIHttpContext** extends UiHttpContext

An extension of the `org.motechproject.osgi.web.ext.UiHttpContext`. This class will be used in development mode for bundles that are configured to load their resources from the hard drive directly, not the jar classpath. The idea is to allow rapid UI development, changes to static html/css/js files will be reflected directly on the UI right after changes are made. If this context fails to load a resource from disk, it will fall back to loading from classpath. This context is a decorator, that decorates the HTTP context coming from Felix.

See also: `org.motechproject.osgi.web.ext.ApplicationEnvironment`

Constructors

FileSystemAwareUIHttpContext

public **FileSystemAwareUIHttpContext** (HttpContext *context*, String *resourceRootDirectoryPath*)

Creates a new instance by decorating the given HTTP context.

Parameters

- **context** – the context to decorate
- **resourceRootDirectoryPath** – the root path from which this context should attempt to read resources

Methods

getResource

public URL **getResource** (String *name*)

getResourceRootDirectoryPath

public String **getResourceRootDirectoryPath** ()

Returns the root path from which this context should attempt to read resources

12.70.4 HttpContextFactory

public final class **HttpContextFactory**

This factory is responsible for creating `org.osgi.service.http.HttpContext` decorator objects for bundles. If dynamic resource loading is set for a given bundle, meaning the `ENVIRONMENT` variable is set to `DEVELOPMENT` and variable with an underscored version of the bundle symbolic name is defined, an instance of `org.motechproject.osgi.web.ext.FileSystemAwareUIHttpContext` will be created for the bundle. In other cases the provided contex is unchaged.

Methods

getHttpContext

public static HttpContext **getHttpContext** (HttpContext *httpContext*, Bundle *bundle*)

Decorates the given HttpContext with a @{link FileSystemAwareUIHttpContext} if dynamic resource loading is set up.

Parameters

- **httpContext** – the default http context for a given bundle
- **bundle** – the bundle for which this http context will be registered

Returns the decorated instance of the provided context if dynamic loading was set up, the original instance otherwise

12.70.5 UiHttpContext

public class **UiHttpContext** implements HttpContext

This is the extension of the Felix `org.osgi.service.http.HttpContext` used by MOTECH. It acts as a decorator for the default context provided by Felix, its only function is to resolve resource names, so that calls to root (`/webapp`) map to the `index.html` file from the root directory.

Constructors

UiHttpContext

public **UiHttpContext** (HttpContext *context*)

Constructs the instance by decorating the provided HTTP context.

Parameters

- **context** – the context to decorate

Methods

getContext

protected HttpContext **getContext** ()

getMimeType

public String **getMimeType** (String *name*)

getResource

public URL **getResource** (String *name*)

handleSecurity

public boolean **handleSecurity** (HttpServletRequest *request*, HttpServletResponse *response*)

12.71 org.motechproject.osgi.web.service

12.71.1 ServerLogService

public interface **ServerLogService**

Interface for accessing log4j Logger configuration.

Fields

ROOT_LOGGER_NAME

String **ROOT_LOGGER_NAME**

Methods

changeLogLevel

void **changeLogLevel** (String *name*, String *level*)

Changes the logging level for one, specified logger.

Parameters

- **name** – name of the logger you wish to change the level for
- **level** – a new level for the logger

changeRootLogLevel

void **changeRootLogLevel** (String *level*)

Changes logging level for the root logger

Parameters

- **level** – a new logging level for the root logger

getAllLogMappings

List<LogMapping> **getAllLogMappings** ()

Returns details for all loggers. Root logger included.

Returns all loggers details, including the root logger

getLogLevels

List<LogMapping> **getLogLevels** ()

Returns details for all loggers. Root logger is NOT included.

Returns details for all loggers, except the root logger

getRootLogLevel

LogMapping **getRootLogLevel** ()

Returns the logger details for the root logger

Returns root logger details

reconfigure

void **reconfigure**()
Sets the logger properties from scratch.

removeLogger

void **removeLogger**(String *name*)
Removes the specified logger.

Parameters

- **name** – name of the logger you wish to remove

12.72 org.motechproject.osgi.web.settings

12.72.1 Loggers

public class **Loggers**
Holds information about all log4j loggers in our system.

Constructors

Loggers

public **Loggers**()

Loggers

public **Loggers**(List<LogMapping> *loggers*, LogMapping *root*)

Methods

equals

public boolean **equals**(Object *obj*)

getLoggers

public List<LogMapping> **getLoggers**()

getRoot

public LogMapping **getRoot**()

getTrash

public List<LogMapping> **getTrash** ( )

hashCode

public int **hashCode** ( )

setLoggers

public void **setLoggers** (List<LogMapping> *loggers*)

setRoot

public void **setRoot** (LogMapping *root*)

setTrash

public void **setTrash** (List<LogMapping> *trash*)

12.73 org.motechproject.osgi.web.util

12.73.1 BundleHeaders

public class **BundleHeaders**

A convenience class for reading bundle headers.

Fields

BLUEPRINT_ENABLED

public static final String **BLUEPRINT_ENABLED**

CONTEXT_PATH

public static final String **CONTEXT_PATH**

RESOURCE_PATH

public static final String **RESOURCE_PATH**

Constructors

BundleHeaders

public **BundleHeaders** (BundleContext *bundleContext*)

Constructs this class using a bundle from which the given bundle context from.

Parameters

- **bundleContext** – the context of the bundle for which we want to read headers

BundleHeaders

public **BundleHeaders** (Bundle *bundle*)

Constructs this class using a given bundle.

Parameters

- **bundle** – the bundle for which we want to read headers

Methods

get

public Object **get** (Object *key*)

Retrieves the header value for a given key.

Parameters

- **key** – the key for the header

Returns the value of the header

getContextPath

public String **getContextPath** ()

Returns the header representing the http path under which the servlet for this bundle should be published.

Returns the **Context-Path** header

getName

public String **getName** ()

Reads the name of the bundle.

Returns the **Bundle-Name** header

getResourcePath

public String **getResourcePath** ()

Returns the header representing the http path under which the static resources (from the /webapp directory) for this bundle should be published.

Returns the **Resource-Path** header

getStringValue

public String **getStringValue** (String *key*)

Gets the header value after casting it to a String.

Parameters

- **key** – the key for which the header should be retrieved

Returns the header value as a String

getSymbolicName

public String **getSymbolicName** ()

Reads the symbolic name of the bundle.

Returns the **Bundle-SymbolicName** header

getVersion

public String **getVersion** ()

Reads the version of the bundle.

Returns the **Bundle-Version** header

isBluePrintEnabled

public boolean **isBluePrintEnabled** ()

Checks if the bundle is Blueprint enabled, meaning its "Blueprint-Enabled" header is set to `true`

Returns true if the bundle is Blueprint enabled, false otherwise

12.73.2 ModuleRegistrations

public class **ModuleRegistrations**

Represents all modules registered with the system. Modules are grouped into 3 categories: modules with sub-menu - modules that have multiple menu items, they get a link in the top level, the admin module is an example modules without sub-menu - modules that are placed in the "Modules" section on the UI, the email module is an example modules without UI - modules that don't have any UI, but register i18n files for example, pill-reminder is an example

Constructors

ModuleRegistrations

public **ModuleRegistrations** ()

Constructs a new instance, initializing all 3 groups as empty lists.

ModuleRegistrations

public **ModuleRegistrations** (Collection<ModuleRegistrationData> *modulesWithSubMenu*, Collection<ModuleRegistrationData> *modulesWithoutSubmenu*, Collection<ModuleRegistrationData> *modulesWithoutUI*)

Constructs a new instances with the 3 groups provided as collections.

Parameters

- **modulesWithSubMenu** – modules with sub menus
- **modulesWithoutSubmenu** – modules without sub-menus
- **modulesWithoutUI** – modules without UI

Methods

allRegistrations

public List<ModuleRegistrationData> **allRegistrations** ()

Returns all registered modules from all the 3 groups as one list.

Returns all registered modules

getModulesWithSubMenu

public Collection<ModuleRegistrationData> **getModulesWithSubMenu** ()

Returns registered modules with sub-menus

getModulesWithoutSubmenu

public Collection<ModuleRegistrationData> **getModulesWithoutSubmenu** ()

Returns registered modules without sub-menus

getModulesWithoutUI

public Collection<ModuleRegistrationData> **getModulesWithoutUI** ()

Returns registered modules without UI

setModulesWithSubMenu

public void **setModulesWithSubMenu** (Collection<ModuleRegistrationData> *modulesWithSubMenu*)

Parameters

- **modulesWithSubMenu** – registered modules with sub-menus

setModulesWithoutSubmenu

public void **setModulesWithoutSubmenu** (Collection<ModuleRegistrationData> *modulesWithoutSubmenu*)

Parameters

- **modulesWithoutSubmenu** – registered modules without sub-menus

setModulesWithoutUI

public void **setModulesWithoutUI** (Collection<ModuleRegistrationData> *modulesWithoutUI*)

Parameters

- **modulesWithoutUI** – registered modules without UI

12.73.3 OSGiServiceUtils

public final class **OSGiServiceUtils**

Utility class for retrieving OSGi services.

Methods

findService

public static <T> T **findService** (BundleContext *bundleContext*, Class<T> *clazz*)

Retrieves the service of the given class from the bundle context. This will retrieve the service with highest priority if there are multiple instances of the service.

Parameters

- **bundleContext** – the bundleContext which will be used for service retrieval
- **clazz** – the class of the service to be retrieved
- **<T>** – the type of the service to be returned

Returns the service found or `null` if there is no such service in the bundle context

findService

public static <T> T **findService** (BundleContext *bundleContext*, String *className*)

Retrieves the service of the given class from the bundle context. This will retrieve the service with highest priority if there are multiple instances of the service.

Parameters

- **bundleContext** – the bundleContext which will be used for service retrieval
- **className** – the class name of the service to be retrieved
- **<T>** – the type of the service to be returned

Returns the service found or `null` if there is no such service in the bundle context

findService

public static <T> T **findService** (BundleContext *bundleContext*, Class<T> *clazz*, long *timeout*)

Retrieves the service of the given class from the bundle context. This will retrieve the service with highest priority if there are multiple instances of the service. Based on the timeout parameter representing the max wait time for the service, the lookup for the service will be performed multiple times in one second intervals. The lookup will be performed at least once.

Parameters

- **bundleContext** – the bundleContext which will be used for service retrieval
- **clazz** – the class of the service to be retrieved
- **timeout** – the max time that will be spent waiting for the service, in miliseconds
- **<T>** – the type of the service to be returned

Returns the service found or `null` if there is no such service in the bundle context

findService

public static <T> T **findService** (BundleContext *bundleContext*, String *className*, long *timeout*)

Retrieves the service of the given class from the bundle context. This will retrieve the service with highest priority if there are multiple instances of the service. Based on the timeout parameter representing the max wait time for the service, the lookup for the service will be performed multiple times in one second intervals. The lookup will be performed at least once.

Parameters

- **bundleContext** – the bundleContext which will be used for service retrieval
- **className** – the class name of the service to be retrieved
- **timeout** – the max time that will be spent waiting for the service, in miliseconds
- **<T>** – the type of the service to be returned

Returns the service found or `null` if there is no such service in the bundle context

12.73.4 WebBundleUtil

public final class **WebBundleUtil**

Utility class for easing bundle related operations/searches.

Methods

findBundleByName

public static Bundle **findBundleByName** (BundleContext *bundleContext*, String *name*)

Does a search for a bundle with a matching Bundle-Name header in its manifest. Note that if there two bundles installed with the same name, the first one found will be returned.

Parameters

- **bundleContext** – the bundle context used for the search
- **name** – the Bundle-Name header value of the bundle we are searching for

Returns the matching bundle, or null if there is no such bundle

findBundleBySymbolicName

public static Bundle **findBundleBySymbolicName**(BundleContext *bundleContext*, String *symbolicName*)

Does a search for a bundle with a matching symbolic name. Note that if there two bundles installed with the same symbolic name, the first one found will be returned.

Parameters

- **bundleContext** – the bundle context used for the search
- **symbolicName** – symbolic name of the bundle we are looking for

Returns the matching bundle, or null if no such bundle exists

getContextLocation

public static String **getContextLocation**(Bundle *bundle*)

Returns the context file location for the bundle, by using reading its **Context-File** header.

Parameters

- **bundle** – the bundle for which we want to retrieve the context file location for

Returns the location of the context file, or null if it is not defined

getContextPath

public static String **getContextPath**(Bundle *bundle*)

Returns the HTTP context path for the bundle, by using reading its **Context-Path** header.

Parameters

- **bundle** – the bundle for which we want to retrieve the context path for

Returns the context path file, or null if it is not defined

getModuleId

public static String **getModuleId**(Bundle *bundle*)

Returns an id for a given bundle for internal use. This is the context-path header value, or the symbolic name if the former is not defined.

Parameters

- **bundle** – the bundle for which we need an id

Returns the id for the bundle

getSymbolicNames

public static List<String> **getSymbolicNames**(BundleContext *context*)

Returns the list of bundles symbolic names.

Parameters

- **context** – the context of the bundle, not null

Returns the list of the bundles symbolic names

12.74 org.motechproject.scheduler.builder

12.74.1 CronJobExpressionBuilder

public class **`CronJobExpressionBuilder`**

Builder for creating cron expressions for jobs, which should be triggered every `repeatIntervalInMinutes` for `repeatWindowInHours` hours or stop at 23:`startTime.getMinute()`.

Constructors

CronJobExpressionBuilder

public **`CronJobExpressionBuilder`**(Time *startTime*, Integer *repeatWindowInHours*, Integer *repeatIntervalInMinutes*)

Constructor.

Parameters

- **startTime** – the time at which job should become active, not null
- **repeatWindowInHours** – the period(in hours) in which job should be active
- **repeatIntervalInMinutes** – the interval between job fires

Methods

build

public String **`build`**()

Builds cron expression with "0 M/M H-H + + ?" pattern.

Returns the cron expression ready to be used with CronSchedulableJob

12.74.2 CronJobSimpleExpressionBuilder

public class **`CronJobSimpleExpressionBuilder`**

Builder for simple cron expressions for jobs, which should be fired every `repeatIntervalInDays` days.

Constructors

CronJobSimpleExpressionBuilder

public **`CronJobSimpleExpressionBuilder`**(Time *startTime*)

Constructor.

Parameters

- **startTime** – the time at which job should become active, not null

Methods

build

public String **build**()

Builds cron expression with "0 M H D/D * ?" pattern for non-zero interval, or with "0 M H D * ?" pattern for zero interval.

Returns the cron expression ready to use with CronSchedulableJob

withRepeatIntervalInDays

public CronJobSimpleExpressionBuilder **withRepeatIntervalInDays** (int *repeatIntervalInDays*)

Sets interval on which job should be fired.

Parameters

- **repeatIntervalInDays** – the interval(in days) between job fires, 0 means everyday

Returns the `CronJobSimpleExpressionBuilder` ready to build cron expressions

12.74.3 WeeklyCronJobExpressionBuilder

public class **WeeklyCronJobExpressionBuilder**

Cron expression builder for jobs, which should be triggered on given day of week at given time.

Constructors

WeeklyCronJobExpressionBuilder

public **WeeklyCronJobExpressionBuilder** (DayOfWeek *dayOfWeek*)

Constructor.

Parameters

- **dayOfWeek** – the day of week at which job should be fired, not null

WeeklyCronJobExpressionBuilder

public **WeeklyCronJobExpressionBuilder** (int *dayOfWeekNumber*)

Constructor.

Parameters

- **dayOfWeekNumber** – the day of week at which job should be fired, must be in range from 1 to 7

Throws

- **java.lang.IllegalArgumentException** – when dayOfWeekNumber isn't in range from 1 to 7

Methods

build

public String **build**()

Builds cron expression with "0 M H ? * D" pattern.

Returns the cron expression as String ready to be used with SchedulableJob classes

withTime

public WeeklyCronJobExpressionBuilder **withTime**(Time *time*)

Sets time, at which built job should be fired.

Parameters

- **time** – the time at which job should be fired, not null

Returns the `WeeklyCronJobExpressionBuilder` ready to build cron expressions

12.75 org.motechproject.scheduler.contract

12.75.1 CronJobId

public class **CronJobId** extends JobId

Represents ID for CronSchedulableJobs.

Constructors

CronJobId

public **CronJobId**(String *subject*, String *id*)

Constructor.

Parameters

- **subject** – the subject of `MotechEvent` fired, when job is triggered, not null
- **id** – the "JobID" parameter for `MotechEvent` fired, when job is triggered, not null

CronJobId

public **CronJobId**(MotechEvent *event*)

Constructor.

Parameters

- **event** – the `MotechEvent` fired, when job is triggered, not null

12.75.2 CronSchedulableJob

public class **CronSchedulableJob** implements SchedulableJob, Serializable

Schedulable Job - a data carrier class for a scheduled job that can be fired unlimited number of times as specified with the cron expression

Author Igor (iopushnyev@2paths.com) Date: 16/02/11 Time: 1:43 PM

Constructors

CronSchedulableJob

public **CronSchedulableJob**(MotechEvent *motechEvent*, String *cronExpression*, Date *startTime*, Date *endTime*)

Constructor.

Parameters

- **motechEvent** – the `MotechEvent` fired, when job triggers, not null
- **cronExpression** – the cron expression, which defines when job should be fired, not null
- **startTime** – the `Date` at which job should become ACTIVE, not null
- **endTime** – the `Date` at which job should be stopped, null treated as never end

CronSchedulableJob

public **CronSchedulableJob**(MotechEvent *motechEvent*, String *cronExpression*)

Constructor.

Parameters

- **motechEvent** – the `MotechEvent` fired, when job triggers, not null
- **cronExpression** – the cron expression, which defines when job should be fired, not null

CronSchedulableJob

public **CronSchedulableJob**(MotechEvent *motechEvent*, String *cronExpression*, Date *startTime*, Date *endTime*, boolean *ignorePastFiresAtStart*)

Constructor.

Parameters

- **motechEvent** – the `MotechEvent` fired, when job triggers, not null
- **cronExpression** – the cron expression, which defines when job should be fired, not null
- **startTime** – the `Date` at which job should become ACTIVE, not null
- **endTime** – the `Date` at which job should be stopped, null treated as never end
- **ignorePastFiresAtStart** – the flag defining, whether job should ignore past fires at start or not

Methods

equals

public boolean **equals** (Object *obj*)

getCronExpression

public String **getCronExpression** ()

getEndTime

public Date **getEndTime** ()

getMotechEvent

public MotechEvent **getMotechEvent** ()

getStartTime

public Date **getStartTime** ()

hashCode

public int **hashCode** ()

isIgnorePastFiresAtStart

public boolean **isIgnorePastFiresAtStart** ()

toString

public String **toString** ()

12.75.3 DayOfWeekSchedulableJob

public final class **DayOfWeekSchedulableJob** implements SchedulableJob, Serializable

Job that is scheduled on particular days of week

Constructors

DayOfWeekSchedulableJob

public **DayOfWeekSchedulableJob**(MotechEvent *motechEvent*, LocalDate *start*, LocalDate *end*, List<DayOfWeek> *days*, Time *time*, boolean *ignorePastFiresAtStart*)

Constructor.

Parameters

- **motechEvent** – the `MotechEvent` fired, when job triggers, not null
- **start** – the `Date` at which job should become ACTIVE, not null
- **end** – the `Date` at which job should be stopped, null treated as never end
- **days** – the list of days at which job should be fired, not null
- **time** – the time at which job should be fired, not null
- **ignorePastFiresAtStart** – the flag defining whether job should ignore past fires at start or not

DayOfWeekSchedulableJob

public **DayOfWeekSchedulableJob**(MotechEvent *motechEvent*, LocalDate *start*, LocalDate *end*, List<DayOfWeek> *days*, Time *time*)

Constructor.

Parameters

- **motechEvent** – the `MotechEvent` fired, when job triggers, not null
- **start** – the `Date` at which job should become ACTIVE, not null
- **end** – the `Date` at which job should be stopped, null treated as never end
- **days** – the list of days at which job should be fired, not null
- **time** – the time at which job should be fired, not null

Methods

getCronDays

public List<Integer> **getCronDays**()

Returns list of days(as `Integer`) at which Job should be fired.

Returns list of days(as `Integer`) at which Job should be fired

getEndDate

public LocalDate **getEndDate**()

getMotechEvent

public MotechEvent **getMotechEvent**()

getStartDate

public LocalDate **getStartDate**()

getTime

public Time **getTime**()

isIgnorePastFiresAtStart

public boolean **isIgnorePastFiresAtStart**()

12.75.4 EventInfo

public class **EventInfo**

EventInfo is the class which contains information about event associated with scheduled job.

Constructors

EventInfo

public **EventInfo**()

Methods

getParameters

public Map<String, Object> **getParameters**()

getSubject

public String **getSubject**()

setParameters

public void **setParameters**(Map<String, Object> *parameters*)

setSubject

public void **setSubject**(String *subject*)

12.75.5 JobBasicInfo

public class **JobBasicInfo**

JobBasicInfo is the class which contains information about scheduled job and its current state.

Fields

ACTIVITY_ACTIVE

public static final String **ACTIVITY_ACTIVE**

ACTIVITY_FINISHED

public static final String **ACTIVITY_FINISHED**

ACTIVITY_NOTSTARTED

public static final String **ACTIVITY_NOTSTARTED**

JOBTYPE_CRON

public static final String **JOBTYPE_CRON**

JOBTYPE_PERIOD

public static final String **JOBTYPE_PERIOD**

JOBTYPE_REPEATING

public static final String **JOBTYPE_REPEATING**

JOBTYPE_RUNONCE

public static final String **JOBTYPE_RUNONCE**

STATUS_BLOCKED

public static final String **STATUS_BLOCKED**

STATUS_ERROR

public static final String **STATUS_ERROR**

STATUS_OK

public static final String **STATUS_OK**

STATUS_PAUSED

public static final String **STATUS_PAUSED**

Constructors

JobBasicInfo

public **JobBasicInfo**()

JobBasicInfo

public **JobBasicInfo**(String *activity*, String *status*, String *name*, String *startDate*, String *nextFireDate*, String *endDate*, String *jobType*, String *info*)

Constructor.

Parameters

- **activity** – the jobs activity (NONSTARTED, ACTIVE, FINISHED)
- **status** – the jobs status (ERROR, OK, PAUSED, BLOCKED)
- **name** – the jobs name
- **startDate** – the jobs start date
- **nextFireDate** – the jobs next fire date
- **endDate** – the jobs end date
- **jobType** – the type of job
- **info** – the information about job, different for each type of Job

Methods

getActivity

public String **getActivity**()

getEndDate

public String **getEndDate**()

getInfo

public String **getInfo**()

getJobType

public String **getJobType**()

getName

public String **getName**()

getNextFireDate

public String **getNextFireDate**()

getStartDate

public String **getStartDate**()

getStatus

public String **getStatus**()

setActivity

public void **setActivity**(String *activity*)

setEndDate

public void **setEndDate**(String *endDate*)

setInfo

public void **setInfo**(String *info*)

setJobType

public void **setJobType**(String *jobType*)

setName

public void **setName**(String *name*)

setNextFireDate

public void **setNextFireDate**(String *nextFireDate*)

setStartDate

public void **setStartDate** (String *startDate*)

setStatus

public void **setStatus** (String *status*)

12.75.6 JobDetailedInfo

public class **JobDetailedInfo**

JobDetailedInfo is the class which wraps the EventInfo list.

See also: `EventInfo`

Constructors

JobDetailedInfo

public **JobDetailedInfo** ()

JobDetailedInfo

public **JobDetailedInfo** (List<EventInfo> *eventInfoList*)

Constructor.

Parameters

- **eventInfoList** – the list of information about event, not null

Methods

getEventInfoList

public List<EventInfo> **getEventInfoList** ()

setEventInfoList

public void **setEventInfoList** (List<EventInfo> *eventInfoList*)

12.75.7 JobId

public abstract class **JobId** implements Serializable

ID used to distinguish one job from others.

Fields

JOB_ID_KEY

public static final String **JOB_ID_KEY**

Constructors

JobId

public **JobId** (String *subject*, String *id*, String *suffix*)

Constructor.

Parameters

- **subject** – the subject of `MotechEvent` fired, when job is triggered, not null
- **id** – the "JobID" parameter for `MotechEvent` fired, when job is triggered, not null
- **suffix** – the type of job

JobId

public **JobId** (MotechEvent *motechEvent*, String *suffix*)

Constructor.

Parameters

- **motechEvent** – the `MotechEvent` fired, when job is triggered, not null
- **suffix** – the type of job, not null

Methods

toString

public String **toString** ()

value

public String **value** ()

Returns jobs ID representation as String.

Returns String representation of jobs ID

12.75.8 JobsSearchSettings

public class **JobsSearchSettings**

`JobsSearchSettings` is the class used for passing search criteria to the Service Layer, it tells how the `MotechSchedulerDatabaseService` should filter jobs information.

See also: `org.motechproject.scheduler.service.MotechSchedulerDatabaseService`, `org.motechproject.scheduler.web.controller.JobsController`

Methods

getActivity

public String **getActivity**()

getName

public String **getName**()

getPage

public Integer **getPage**()

getRows

public Integer **getRows**()

getSortColumn

public String **getSortColumn**()

getSortDirection

public String **getSortDirection**()

getStatus

public String **getStatus**()

getTimeFrom

public String **getTimeFrom**()

getTimeTo

public String **getTimeTo**()

setActivity

public void **setActivity**(String *activity*)

setName

public void **setName**(String *name*)

setPage

public void **setPage** (Integer *page*)

setRows

public void **setRows** (Integer *rows*)

setSortColumn

public void **setSortColumn** (String *sortColumn*)

setSortDirection

public void **setSortDirection** (String *sortDirection*)

setStatus

public void **setStatus** (String *status*)

setTimeFrom

public void **setTimeFrom** (String *timeFrom*)

setTimeTo

public void **setTimeTo** (String *timeTo*)

12.75.9 RepeatingJobId

public class **RepeatingJobId** extends JobId
Jobs ID for RepeatingSchedulableJobs.

Fields

SUFFIX_REPEATJOBID

public static final String **SUFFIX_REPEATJOBID**

Constructors

RepeatingJobId

public **RepeatingJobId** (String *subject*, String *id*)
Constructor.

Parameters

- **subject** – the subject of `MotechEvent` fired, when job is triggered
- **id** – the "JobID" parameter for `MotechEvent`, when job is triggered

RepeatingJobId

public **`RepeatingJobId`** (MotechEvent *repeatingEvent*)
Constructor.

Parameters

- **repeatingEvent** – the `MotechEvent` fired, when job is triggered

12.75.10 RepeatingPeriodJobId

public class **`RepeatingPeriodJobId`** extends JobId
Jobs ID for RepeatingPeriodSchedulableJobs.

Fields

SUFFIX_REPEATPERIODJOBID

public static final String **`SUFFIX_REPEATPERIODJOBID`**

Constructors

RepeatingPeriodJobId

public **`RepeatingPeriodJobId`** (String *subject*, String *id*)
Constructor.

Parameters

- **subject** – the subject of `MotechEvent` fired, when job is triggered
- **id** – the "JobID" parameter for `MotechEvent` fired, when job is triggered

RepeatingPeriodJobId

public **`RepeatingPeriodJobId`** (MotechEvent *repeatingEvent*)
Constructor.

Parameters

- **repeatingEvent** – the `MotechEvent` fired, when job is triggered

12.75.11 RepeatingPeriodSchedulableJob

public class **`RepeatingPeriodSchedulableJob`** implements SchedulableJob, Serializable
Job that will be fired every `org.joda.time.Period` of time

Constructors

RepeatingPeriodSchedulableJob

public **RepeatingPeriodSchedulableJob**()

Constructor. It will create a job, which will never end, won't ignore past fires at start and will use original fire time after misfire. Start time, `MotechEvent` and repeat period are not assigned, which means that further usage, without setting them, can cause exceptions.

RepeatingPeriodSchedulableJob

public **RepeatingPeriodSchedulableJob**(MotechEvent *motechEvent*, Date *startTime*, Date *endTime*, Period *repeatPeriod*, boolean *ignorePastFiresAtStart*)

Constructor.

Parameters

- **motechEvent** – the `MotechEvent` which will be fired when the job triggers, not null
- **startTime** – the `Date` at which job should become ACTIVE, not null
- **endTime** – the `Date` at which job should be stopped, null treated as never end
- **repeatPeriod** – the `Period` between job fires, not null
- **ignorePastFiresAtStart** – the flag defining whether job should ignore past fires at start or not

Methods

equals

public boolean **equals**(Object *obj*)

getEndTime

public Date **getEndTime**()

getMotechEvent

public MotechEvent **getMotechEvent**()

getRepeatPeriod

public Period **getRepeatPeriod**()

getStartTime

public Date **getStartTime**()

hashCode

public int **hashCode** ( )

isIgnorePastFiresAtStart

public boolean **isIgnorePastFiresAtStart** ( )

isUseOriginalFireTimeAfterMisfire

public boolean **isUseOriginalFireTimeAfterMisfire** ( )

setEndTime

public RepeatingPeriodSchedulableJob **setEndTime** (Date *endTime*)

setIgnorePastFiresAtStart

public RepeatingPeriodSchedulableJob **setIgnorePastFiresAtStart** (boolean *ignorePastFiresAtStart*)

setMotechEvent

public RepeatingPeriodSchedulableJob **setMotechEvent** (MotechEvent *motechEvent*)

setRepeatPeriod

public void **setRepeatPeriod** (Period *repeatPeriod*)

setStartTime

public RepeatingPeriodSchedulableJob **setStartTime** (Date *startTime*)

setUseOriginalFireTimeAfterMisfire

public RepeatingPeriodSchedulableJob **setUseOriginalFireTimeAfterMisfire** (boolean *useOriginalFireTimeAfterMisfire*)

toString

public String **toString** ( )

12.75.12 RepeatingSchedulableJob

public class **RepeatingSchedulableJob** implements SchedulableJob, Serializable

Schedulable Job - a data carrier class for a scheduled job that can be fired set number of times

Constructors

RepeatingSchedulableJob

public **RepeatingSchedulableJob**()

Constructor. It will create a job, which will never end, won't ignore past fires at start and will use original fire time after misfire. Start time, `MotechEvent`, repeat count and repeat interval are not assigned, which means that further usage, without setting them, can cause exceptions.

RepeatingSchedulableJob

public **RepeatingSchedulableJob**(MotechEvent *motechEvent*, Integer *repeatCount*, Integer *repeatIntervalInSeconds*, Date *startTime*, Date *endTime*, boolean *ignorePastFiresAtStart*)

Constructor.

Parameters

- **motechEvent** – the `MotechEvent` which will be fired when the job triggers, not null
- **repeatCount** – the number of times job should be repeated, null treated as infinite
- **repeatIntervalInSeconds** – the interval(in seconds) between job fires
- **startTime** – the `Date` at which job should become ACTIVE, not null
- **endTime** – the `Date` at which job should be stopped, null treated as never end
- **ignorePastFiresAtStart** – the flag defining whether job should ignore past fires at start or not

RepeatingSchedulableJob

public **RepeatingSchedulableJob**(MotechEvent *motechEvent*, Integer *repeatIntervalInSeconds*, Date *startTime*, Date *endTime*, boolean *ignorePastFiresAtStart*)

Constructor.

Parameters

- **motechEvent** – the `MotechEvent` which will be fired when the job triggers, not null
- **repeatIntervalInSeconds** – the interval(in seconds) between job fires
- **startTime** – the `Date` at which job should become ACTIVE, not null
- **endTime** – the `Date` at which job should be stopped, null treated as never end
- **ignorePastFiresAtStart** – the flag defining whether job should ignore past fires at start or not

Methods

equals

public boolean **equals** (Object *obj*)

getEndTime

public Date **getEndTime** ()

getMotechEvent

public MotechEvent **getMotechEvent** ()

getRepeatCount

public Integer **getRepeatCount** ()

getRepeatIntervalInSeconds

public Integer **getRepeatIntervalInSeconds** ()

getStartTime

public Date **getStartTime** ()

hashCode

public int **hashCode** ()

isIgnorePastFiresAtStart

public boolean **isIgnorePastFiresAtStart** ()

isUseOriginalFireTimeAfterMisfire

public boolean **isUseOriginalFireTimeAfterMisfire** ()

setEndTime

public RepeatingSchedulableJob **setEndTime** (Date *endTime*)

setIgnorePastFiresAtStart

public RepeatingSchedulableJob **setIgnorePastFiresAtStart** (boolean *ignorePastFiresAtStart*)

Ignore past fires when start time of job is in past.

```
ex : repeating job with interval of 5 unit, and current time in between fire 2 and 3 will start
 1     2     3     4
 +-----+-----+-----+
 start    ^current time
```

Parameters

- **ignorePastFiresAtStart** –

setMotechEvent

public RepeatingSchedulableJob **setMotechEvent** (MotechEvent *motechEvent*)

setRepeatCount

public RepeatingSchedulableJob **setRepeatCount** (Integer *repeatCount*)

setRepeatIntervalInSeconds

public RepeatingSchedulableJob **setRepeatIntervalInSeconds** (Integer *repeatIntervalInSeconds*)

setStartTime

public RepeatingSchedulableJob **setStartTime** (Date *startTime*)

setUseOriginalFireTimeAfterMisfire

public RepeatingSchedulableJob **setUseOriginalFireTimeAfterMisfire** (boolean *useOriginalFireTimeAfterMisfire*)

toString

public String **toString** ()

12.75.13 RunOnceJobId

public class **RunOnceJobId** extends JobId

Represents ID for RunOnceSchedulableJob.

Fields

SUFFIX_RUNONCEJOBID

public static final String **SUFFIX_RUNONCEJOBID**

Constructors

RunOnceJobId

public **RunOnceJobId** (String *subject*, String *id*)
Constructor.

Parameters

- **subject** – the subject of `MotechEvent` fired, when job is triggered
- **id** – the "JobID" parameter for `MotechEvent` fired, when job is triggered

RunOnceJobId

public **RunOnceJobId** (MotechEvent *runOnceEvent*)
Constructor.

Parameters

- **runOnceEvent** – the `MotechEvent` fired, when job is triggered

12.75.14 RunOnceSchedulableJob

public final class **RunOnceSchedulableJob** implements SchedulableJob, Serializable
Run Once Schedulable Job - a data carrier class for a job scheduled in the future that can be fired only once

This class is immutable

User: Igor (iopushnyev@2paths.com) Date: 16/02/11 Time: 1:43 PM

Constructors

RunOnceSchedulableJob

public **RunOnceSchedulableJob** (MotechEvent *motechEvent*, Date *startDate*)
Constructor

Parameters

- **motechEvent** –
  - event data message that will be send by Motech Scheduler when this job is fired
- **startDate** –
  - date and time when the job fill be fired

Throws

- **IllegalArgumentException** – if motechEvent or startDate is null or startDate is in past

Methods

equals

public boolean **equals** (Object *o*)

getMotechEvent

public MotechEvent **getMotechEvent** ()

getStartDate

public Date **getStartDate** ()

hashCode

public int **hashCode** ()

isIgnorePastFiresAtStart

public boolean **isIgnorePastFiresAtStart** ()

toString

public String **toString** ()

12.75.15 SchedulableJob

public interface **SchedulableJob**

Represents Job that can be scheduled

Methods

getMotechEvent

MotechEvent **getMotechEvent** ()

isIgnorePastFiresAtStart

boolean **isIgnorePastFiresAtStart** ()

12.76 org.motechproject.scheduler.exception

12.76.1 MotechSchedulerException

public class **MotechSchedulerException** extends RuntimeException

Thrown when error within MOTECH Scheduler occurs. Can be caused by updating non-existent job, job having invalid cron expression etc. User: Igor (iopushnyev@2paths.com) Date: 17/02/11 Time: 4:20 PM

Constructors

MotechSchedulerException

public **MotechSchedulerException** ( )

MotechSchedulerException

public **MotechSchedulerException** (String *message*)

MotechSchedulerException

public **MotechSchedulerException** (String *message*, Throwable *cause*)

MotechSchedulerException

public **MotechSchedulerException** (Throwable *cause*)

12.76.2 MotechSchedulerJobRetrievalException

public class **MotechSchedulerJobRetrievalException** extends RuntimeException

Indicates an error when retrieving scheduled jobs from the database.

Constructors

MotechSchedulerJobRetrievalException

public **MotechSchedulerJobRetrievalException** ( )

MotechSchedulerJobRetrievalException

public **MotechSchedulerJobRetrievalException** (String *message*, Throwable *cause*)

12.76.3 SchedulerInstantiationException

public class **SchedulerInstantiationException** extends RuntimeException

Thrown when scheduler can't be instantiated.

Constructors

SchedulerInstantiationException

public **SchedulerInstantiationException** (String *message*, Throwable *e*)

12.76.4 SchedulerShutdownException

public class **SchedulerShutdownException** extends RuntimeException

The `SchedulerShutdownException` exception informs about that there were problems with shutdown scheduler.

Constructors

SchedulerShutdownException

public **SchedulerShutdownException** (String *message*, Throwable *e*)

12.77 org.motechproject.scheduler.factory

12.77.1 MotechSchedulerFactoryBean

public class **MotechSchedulerFactoryBean**

The `MotechSchedulerFactoryBean` is used to create scheduler and start it.

Constructors

MotechSchedulerFactoryBean

public **MotechSchedulerFactoryBean** (ApplicationContext *applicationContext*, Properties *schedulerProperties*)

Constructor.

Parameters

- **applicationContext** – the Spring context of the Scheduler module, not null
- **schedulerProperties** – the properties of scheduler, not null

Methods

getQuartzScheduler

public Scheduler **getQuartzScheduler** ()

Returns created scheduler.

Returns the created scheduler

getQuartzSchedulerFactoryBean

public SchedulerFactoryBean **getQuartzSchedulerFactoryBean**()

init

public void **init**()
Creates the Spring `SchedulerFactoryBean`.

shutdown

public void **shutdown**()
Shuts down `MotechSchedulerFactoryBean`.

12.78 org.motechproject.scheduler.service

12.78.1 MotechSchedulerActionProxyService

public interface **MotechSchedulerActionProxyService**
Proxy registered with the task channel, exposing the scheduler service as task actions.

Methods

scheduleCronJob

void **scheduleCronJob**(String *subject*, Map<Object, Object> *parameters*, String *cronExpression*, DateTime *startTime*, DateTime *endTime*, Boolean *ignorePastFiresAtStart*)
Schedules job, which will be fired on every match with given cron expression.

Parameters

- **subject** – the subject for `MotechEvent` fired, when job is triggered, not null
- **parameters** – the parameters for `MotechEvent`, not null
- **cronExpression** – the cron expression defining when job should be triggered, not null
- **startTime** – the `DateTime` at which should become ACTIVE, not null
- **endTime** – the `DateTime` at which job should be stopped, null treated as never end.
- **ignorePastFiresAtStart** – the flag defining whether job should ignore past fires at start or not, not null

scheduleDayOfWeekJob

void **scheduleDayOfWeekJob**(String *subject*, Map<Object, Object> *parameters*, DateTime *start*, DateTime *end*, List<Object> *days*, DateTime *time*, Boolean *ignorePastFiresAtStart*)
Schedules job, which will be fired at given time on days of week provided by user.

Parameters

- **subject** – the subject for `MotechEvent` fired, when job is triggered, not null
- **parameters** – the parameters for `MotechEvent`, not null
- **start** – the `DateTime` at which should become ACTIVE, not null
- **end** – the `DateTime` at which job should be stopped, null treated as never end
- **days** – the list of days at which job should be fired, not null
- **time** – the `Time` at which job should be fired
- **ignorePastFiresAtStart** – the flag defining whether job should ignore past fires at start or not, not null

scheduleRepeatingJob

void **scheduleRepeatingJob**(String *subject*, Map<Object, Object> *parameters*, DateTime *startTime*, DateTime *endTime*, Integer *repeatCount*, Integer *repeatIntervalInSeconds*, Boolean *ignorePastFiresAtStart*, Boolean *useOriginalFireTimeAfterMisfire*)

Schedules job, which will be fired every user-specified time interval(in milliseconds), but won't be fired more than given number of times.

Parameters

- **subject** – the subject for `MotechEvent` fired, when job is triggered, not null
- **parameters** – the parameters for `MotechEvent`, not null
- **startTime** – the `DateTime` at which should become ACTIVE, not null
- **endTime** – the `DateTime` at which job should be stopped, null treated as never end
- **repeatCount** – the number of time job should be triggered, -1 treated as infinite, not null
- **repeatIntervalInSeconds** – the interval(in seconds) between job fires, not null
- **ignorePastFiresAtStart** – the flag defining whether job should ignore past fires at start or not, not null
- **useOriginalFireTimeAfterMisfire** – the flag defining whether job should use original fire time after misfire, not null

scheduleRepeatingPeriodJob

void **scheduleRepeatingPeriodJob**(String *subject*, Map<Object, Object> *parameters*, DateTime *startTime*, DateTime *endTime*, Period *repeatPeriod*, Boolean *ignorePastFiresAtStart*, Boolean *useOriginalFireTimeAfterMisfire*)

Schedules job, which will be fired every, user-specified period.

Parameters

- **subject** – the subject for `MotechEvent` fired, when job is triggered, not null
- **parameters** – the parameters for `MotechEvent`, not null
- **startTime** – the `DateTime` at which should become ACTIVE, not null
- **endTime** – the `DateTime` at which job should be stopped, null treated as never end
- **repeatPeriod** – the `Period` between job fires, not null

- **ignorePastFiresAtStart** – the flag defining whether job should ignore past fires at start or not, not null
- **useOriginalFireTimeAfterMisfire** – the flag defining whether job should use original fire time after misfire

scheduleRunOnceJob

void **scheduleRunOnceJob** (String *subject*, Map<Object, Object> *parameters*, DateTime *startDate*)

Schedules job, which will be fired only once at date given by user.

Parameters

- **subject** – the subject for `MotechEvent` fired, when job is triggered, not null
- **parameters** – the parameters for `MotechEvent`, not null
- **startDate** – the `DateTime` at which should become ACTIVE, not null

unscheduleJobs

void **unscheduleJobs** (String *subject*)

Unschedules all jobs with given subject in their name.

Parameters

- **subject** – the subject for deleting jobs

12.78.2 MotechSchedulerDatabaseService

public interface **MotechSchedulerDatabaseService**

Service provides methods used to get data from Scheduler. Also provides pagination to use with jqGrid.

Methods

countJobs

int **countJobs** (JobsSearchSettings *jobsSearchSettings*)

Counts all triggers in TRIGGER table which matches the filters built from grid settings.

Parameters

- **jobsSearchSettings** – contains filter jobs information.

Throws

- **MotechSchedulerJobRetrievalException** – when the query fails.

Returns number of all triggers which matches the filters built from grid settings.

getScheduledJobDetailedInfo

JobDetailedInfo **getScheduledJobDetailedInfo** (JobBasicInfo *jobBasicInfo*)

Returns detailed information about job matching given `JobBasicInfo`.

Parameters

- **jobBasicInfo** – the `JobBasicInfo` about the job

Returns the detailed information about job

getScheduledJobsBasicInfo

List<JobBasicInfo> **`getScheduledJobsBasicInfo`** (JobsSearchSettings *jobsSearchSettings*)

Returns info about scheduled jobs for given filter information Sorts all jobs with ascending or descending order for given column.

Parameters

- **jobsSearchSettings** – contains filter, sorting and pagination jobs options.

Throws

- **MotechSchedulerJobRetrievalException** – when the query fails.

Returns list with `org.motechproject.scheduler.contract.JobBasicInfo` for given sorting and pagination option

12.78.3 MotechSchedulerService

public interface **`MotechSchedulerService`**

ingroup scheduler Motech Scheduler Service Interface provides methods to schedule reschedule and unschedule a job Set a global policy that determines trigger fire behaviour for misfired triggers. For details see quartz documentations for misfire policy do_nothing -> @see CronTrigger.MISFIRE_INSTRUCTION_DO_NOTHING fire_once_now -> @see CronTrigger.MISFIRE_INSTRUCTION_FIRE_ONCE_NOW ignore -> @see CronTrigger.MISFIRE_INSTRUCTION_IGNORE_MISFIRE_POLICY fire_now -> @see SimpleTrigger.MISFIRE_INSTRUCTION_FIRE_NOW ignore -> @see SimpleTrigger.MISFIRE_INSTRUCTION_IGNORE_MISFIRE_POLICY reschedule_next_with_existing_count -> @see SimpleTrigger.MISFIRE_INSTRUCTION_RESCHEDULE_NEXT_WITH_EXISTING_COUNT reschedule_next_with_remaining_count -> @see SimpleTrigger.MISFIRE_INSTRUCTION_RESCHEDULE_NEXT_WITH_REMAINI reschedule_now_with_existing_count -> @see SimpleTrigger.MISFIRE_INSTRUCTION_RESCHEDULE_NOW_WITH_EXIST reschedule_now_with_remaining_count -> @see SimpleTrigger.MISFIRE_INSTRUCTION_RESCHEDULE_NOW_WITH_REM

Author Igor (iopushnyev@2paths.com) Date: 16/02/11

Fields

JOB_ID_KEY

String **`JOB_ID_KEY`**

Methods

getNextFireDate

DateTime **`getNextFireDate`** (JobId *jobId*)

Returns next fire date of job with given ID.

Parameters

- **jobId** – the `JobId` of job, not null

Returns next fire date of job

getPreviousFireDate

DateTime **getPreviousFireDate** (JobId *jobId*)

Returns last fire date of job with given ID.

Parameters

- **jobId** – the `JobId` of job, not null

Returns last fire date of job

getScheduledJobTimings

List<Date> **getScheduledJobTimings** (String *subject*, String *externalJobId*, Date *startDate*, Date *endDate*)

Returns list of dates at which job will be triggered.

Parameters

- **subject** – the subject of job, not null
- **externalJobId** – the external ID of job, not null
- **startDate** – the `Date` after which dates should be added, not null
- **endDate** – the `Date` before which dates should be added, not null

Returns the list of dates, null if exception was thrown

getScheduledJobTimingsWithPrefix

List<Date> **getScheduledJobTimingsWithPrefix** (String *subject*, String *externalJobIdPrefix*, Date *startDate*, Date *endDate*)

Returns list of dates at which jobs will be triggered.

Parameters

- **subject** – the subject of job, not null
- **externalJobIdPrefix** – the prefix of jobs
- **startDate** – the `Date` after which dates should be added, not null
- **endDate** – the `Date` before which dates should be added, not null

Returns the list of dates

rescheduleJob

void **rescheduleJob** (String *subject*, String *externalId*, String *cronExpression*)

Reschedules a job with the given job ID to be fired according to the given Cron Expression Previous version of the configured Motech Scheduled Event that will be created when the job is fired remains as it was

Parameters

- **subject** –
- **externalId** –

- **cronExpression** –

safeScheduleJob

void **safeScheduleJob** (CronSchedulableJob *cronSchedulableJob*)

Same as scheduleJob, except that it would update existing job if one exists instead of creating a new one

Parameters

- **cronSchedulableJob** –

safeScheduleRepeatingJob

void **safeScheduleRepeatingJob** (RepeatingSchedulableJob *repeatingSchedulableJob*)

Same as safeScheduleRepeatingJob with intervening = true

Parameters

- **repeatingSchedulableJob** –

safeScheduleRepeatingPeriodJob

void **safeScheduleRepeatingPeriodJob** (RepeatingPeriodSchedulableJob *repeatingPeriodSchedulableJob*)

Same as scheduleRepeatingPeriodJob, except that it would update existing job if one exists instead of creating a new one

Parameters

- **repeatingPeriodSchedulableJob** –

safeScheduleRunOnceJob

void **safeScheduleRunOnceJob** (RunOnceSchedulableJob *schedulableJob*)

Same as scheduleRunOnceJob, except that it would update existing job if one exists instead of creating a new one

Parameters

- **schedulableJob** –

safeUnscheduleAllJobs

void **safeUnscheduleAllJobs** (String *jobIdPrefix*)

Unschedules all jobs with given prefix. Logs all exceptions instead of throwing them.

Parameters

- **jobIdPrefix** – the jobs prefix

safeUnscheduleJob

void **safeUnscheduleJob** (String *subject*, String *externalId*)

Same as unscheduleJob except that it would not throw an exception if the job doesn't exist

Parameters

- **subject** –
- **externalId** –

safeUnscheduleRepeatingJob

void **safeUnscheduleRepeatingJob** (String *subject*, String *externalId*)

Same as unscheduleRepeatingJob except that it would not throw an exception if the job doesn't exist

Parameters

- **subject** –
- **externalId** –

safeUnscheduleRunOnceJob

void **safeUnscheduleRunOnceJob** (String *subject*, String *externalId*)

Same as unscheduleRunOnceJob except that it would not throw an exception if the job doesn't exist

Parameters

- **subject** –
- **externalId** –

scheduleDayOfWeekJob

void **scheduleDayOfWeekJob** (DayOfWeekSchedulableJob *dayOfWeekSchedulableJob*)

Same as safeScheduleDayOfWeekJob with intervening = true

Parameters

- **dayOfWeekSchedulableJob** –

scheduleJob

void **scheduleJob** (CronSchedulableJob *cronSchedulableJob*)

Schedules the given schedulable job. The Job ID by which the job will be referencing in the future should be provided in an Instance of MotechEvent in SchedulableJob (see MotechEvent.jobId) If a job with the same job ID as the given exists, this job will be unscheduled and the given schedulable job will be scheduled

Parameters

- **cronSchedulableJob** –

scheduleRepeatingJob

void **scheduleRepeatingJob** (RepeatingSchedulableJob *repeatingSchedulableJob*)

Schedules the given schedulable job. The Job ID by which the job will be referencing in the future should be provided in an Instance of MotechEvent in SchedulableJob (see MotechEvent.jobId) If a job with the same job ID as the given exists, this job will be unscheduled and the given schedulable job will be scheduled

Parameters

- **repeatingSchedulableJob** –

scheduleRepeatingPeriodJob

void **scheduleRepeatingPeriodJob** (RepeatingPeriodSchedulableJob *repeatingPeriodSchedulableJob*)

Same as scheduleRepeatingJob but schedules RepeatingPeriodSchedulableJob

Parameters

- **repeatingPeriodSchedulableJob** –

scheduleRunOnceJob

void **scheduleRunOnceJob** (RunOnceSchedulableJob *schedulableJob*)

Schedules `RunOnceSchedulableJob`.

Parameters

- **schedulableJob** – the `RunOnceSchedulableJob` to be scheduled, not null

unscheduleAllJobs

void **unscheduleAllJobs** (String *jobIdPrefix*)

Unschedules all jobs with given prefix.

Parameters

- **jobIdPrefix** – the jobs prefix

unscheduleJob

void **unscheduleJob** (String *subject*, String *externalId*)

Unschedules a job with the given job ID

Parameters

- **subject** – : String representing domain operation eg. "pill-reminder", "outbox-call" or motechEvent.getSubject()
- **externalId** – : domain specific id as String.

unscheduleJob

void **unscheduleJob** (JobId *job*)

Unschedules job with given job ID.

Parameters

- **job** – the `JobId` of job which should be unscheduled, not null

unscheduleRepeatingJob

void **unscheduleRepeatingJob** (String *subject*, String *externalId*)

Unschedules `RepeatingSchedulableJob` with given subject and external ID.

Parameters

- **subject** – the subject of job, not null
- **externalId** – the external ID of job, not null

unscheduleRunOnceJob

void **unscheduleRunOnceJob** (String *subject*, String *externalId*)

Unschedules a run once job with the given job ID

Parameters

- **subject** – : String representing domain operation eg. "pill-reminder", "outbox-call" or motechEvent.getSubject()
- **externalId** – : domain specific id as String.

updateScheduledJob

void **updateScheduledJob** (MotechEvent *motechEvent*)

Updates MotechEvent data of the job defined by jobIb in the given instance of that class

Parameters

- **motechEvent** –

12.79 org.motechproject.scheduler.service.impl

12.79.1 MotechScheduledJob

public class **MotechScheduledJob** implements Job

Represents a MOTECH job scheduled with quartz. This class implements the `org.quartz.Job` interface - its execute method will be called when a MOTECH job in quartz triggers. Since jobs in MOTECH are basically `org.motechproject.event.MotechEvent`s getting published on a quartz schedule, upon execution this class retrieves the `org.motechproject.event.listener.EventRelay` from the application context and uses it to immediately publish the event scheduled with this job. For every execution a new copy of the event is constructed.

Methods

execute

public void **execute** (JobExecutionContext *jobExecutionContext*)

Executes the job called by Quartz.

Parameters

- **jobExecutionContext** – the executionContext of the job provided by Quartz

12.79.2 MotechScheduler

public final class **MotechScheduler**

ingroup scheduler

Main class that can bootstrap a Motech Scheduler

Author Igor (iopushnyev@2paths.com)

Methods

main

public static void **main** (String[] *args*)

12.79.3 MotechSchedulerActionProxyServiceImpl

public class **MotechSchedulerActionProxyServiceImpl** implements MotechSchedulerActionProxyService

Constructors

MotechSchedulerActionProxyServiceImpl

public **MotechSchedulerActionProxyServiceImpl** (MotechSchedulerService *schedulerService*)

Methods

scheduleCronJob

public void **scheduleCronJob** (String *subject*, Map<Object, Object> *parameters*, String *cronExpression*, DateTime *startTime*, DateTime *endTime*, Boolean *ignorePastFiresAtStart*)

scheduleDayOfWeekJob

public void **scheduleDayOfWeekJob** (String *subject*, Map<Object, Object> *parameters*, DateTime *start*, DateTime *end*, List<Object> *days*, DateTime *time*, Boolean *ignorePastFiresAtStart*)

scheduleRepeatingJob

public void **scheduleRepeatingJob** (String *subject*, Map<Object, Object> *parameters*, DateTime *startTime*, DateTime *endTime*, Integer *repeatCount*, Integer *repeatIntervalInSeconds*, Boolean *ignorePastFiresAtStart*, Boolean *useOriginalFireTimeAfterMisfire*)

scheduleRepeatingPeriodJob

public void **scheduleRepeatingPeriodJob** (String *subject*, Map<Object, Object> *parameters*, DateTime *startTime*, DateTime *endTime*, Period *repeatPeriod*, Boolean *ignorePastFiresAtStart*, Boolean *useOriginalFireTimeAfterMisfire*)

scheduleRunOnceJob

public void **scheduleRunOnceJob** (String *subject*, Map<Object, Object> *parameters*, DateTime *startDate*)

unscheduleJobs

public void **unscheduleJobs** (String *subject*)

12.79.4 MotechSchedulerDatabaseServiceImpl

public class **MotechSchedulerDatabaseServiceImpl** implements MotechSchedulerDatabaseService

Motech Scheduler Database Service implementation

See also: `MotechSchedulerDatabaseService`

Methods

countJobs

public int **countJobs** (JobsSearchSettings *jobsSearchSettings*)

getScheduledJobDetailedInfo

public JobDetailedInfo **getScheduledJobDetailedInfo** (JobBasicInfo *jobBasicInfo*)

getScheduledJobsBasicInfo

public List<JobBasicInfo> **getScheduledJobsBasicInfo** (JobsSearchSettings *jobsSearchSettings*)

init

public void **init** ()

12.79.5 MotechSchedulerServiceImpl

public class **MotechSchedulerServiceImpl** implements MotechSchedulerService

Motech Scheduler Service implementation

See also: `MotechSchedulerService`

Fields

JOB_GROUP_NAME

public static final String **JOB_GROUP_NAME**

Constructors

MotechSchedulerServiceImpl

public **MotechSchedulerServiceImpl** (MotechSchedulerFactoryBean *motechSchedulerFactoryBean*, SettingsFacade *schedulerSettings*)

Methods

assertArgumentNotNull

protected void **assertArgumentNotNull** (String *objectName*, Object *object*)

Asserts that given object is not null.

Parameters

- **objectName** – the objects name
- **object** – the object to be checked for being null

Throws

- **IllegalArgumentException** – if object is null

getNextFireDate

public DateTime **getNextFireDate** (JobId *jobId*)

getPreviousFireDate

public DateTime **getPreviousFireDate** (JobId *jobId*)

getScheduledJobTimings

public List<Date> **getScheduledJobTimings** (String *subject*, String *externalJobId*, Date *startDate*, Date *endDate*)

getScheduledJobTimingsWithPrefix

public List<Date> **getScheduledJobTimingsWithPrefix** (String *subject*, String *externalJobIdPrefix*, Date *startDate*, Date *endDate*)

logObjectIfNotNull

protected void **logObjectIfNotNull** (Object *obj*)

Logs object if it is not null.

Parameters

- **obj** – the object to be checked for being null

rescheduleJob

public void **rescheduleJob** (String *subject*, String *externalId*, String *cronExpression*)

safeScheduleJob

public void **safeScheduleJob** (CronSchedulableJob *cronSchedulableJob*)

safeScheduleRepeatingJob

public void **safeScheduleRepeatingJob** (RepeatingSchedulableJob *repeatingSchedulableJob*)

safeScheduleRepeatingPeriodJob

public void **safeScheduleRepeatingPeriodJob** (RepeatingPeriodSchedulableJob *repeatingPeriodSchedulableJob*)

safeScheduleRunOnceJob

public void **safeScheduleRunOnceJob** (RunOnceSchedulableJob *schedulableJob*)

safeUnscheduleAllJobs

public void **safeUnscheduleAllJobs** (String *jobIdPrefix*)

safeUnscheduleJob

public void **safeUnscheduleJob** (String *subject*, String *externalId*)

safeUnscheduleRepeatingJob

public void **safeUnscheduleRepeatingJob** (String *subject*, String *externalId*)

safeUnscheduleRunOnceJob

public void **safeUnscheduleRunOnceJob** (String *subject*, String *externalId*)

scheduleDayOfWeekJob

public void **scheduleDayOfWeekJob** (DayOfWeekSchedulableJob *dayOfWeekSchedulableJob*)

scheduleJob

public void **scheduleJob** (CronSchedulableJob *cronSchedulableJob*)

scheduleRepeatingJob

public void **scheduleRepeatingJob** (RepeatingSchedulableJob *repeatingSchedulableJob*)

scheduleRepeatingPeriodJob

public void **scheduleRepeatingPeriodJob** (RepeatingPeriodSchedulableJob *repeatingPeriodSchedulableJob*)

scheduleRunOnceJob

public void **scheduleRunOnceJob** (RunOnceSchedulableJob *schedulableJob*)

unscheduleAllJobs

public void **unscheduleAllJobs** (String *jobIdPrefix*)

unscheduleJob

public void **unscheduleJob** (String *subject*, String *externalId*)

unscheduleJob

public void **unscheduleJob** (JobId *job*)

unscheduleRepeatingJob

public void **unscheduleRepeatingJob** (String *subject*, String *externalId*)

unscheduleRunOnceJob

public void **unscheduleRunOnceJob** (String *subject*, String *externalId*)

updateScheduledJob

public void **updateScheduledJob** (MotechEvent *motechEvent*)

12.80 org.motechproject.security.annotations

12.80.1 SecurityAnnotationBeanPostProcessor

public class **SecurityAnnotationBeanPostProcessor** implements BeanPostProcessor

A `BeanPostProcessor` used by Motech to load permissions from modules. Given a module context, it looks for `PreAuthorize` and `PostAuthorize` annotations. These annotations are then parsed using an `ExpressionParser`. The permission names are deduced from `hasRole` and `hasAnyRole` in the annotation value. The names of permissions are then saved using the `MotechPermissionService`. The bundle name used to construct the permission is retrieved from the application context.

Constructors

SecurityAnnotationBeanPostProcessor

public **SecurityAnnotationBeanPostProcessor** (MotechPermissionService *permissionService*)

Methods

postProcessAfterInitialization

public Object **postProcessAfterInitialization** (Object *bean*, String *beanName*)

Searches for `org.springframework.security.access.prepost.PreAuthorize` and `org.springframework.security.access.prepost.PostAuthorize` annotations representing permissions and parses them. Parsed annotations are used to find permissions. After that those permissions are added to `org.motechproject.security.service.MotechPermissionService`

Parameters

- **bean** – to be processed
- **beanName** – name of the bean

Returns processed bean

postProcessBeforeInitialization

public Object **postProcessBeforeInitialization** (Object *bean*, String *beanName*)

processAnnotations

public void **processAnnotations** (ApplicationContext *applicationContext*)

Processes security annotations from bundles using `postProcessAfterInitialization(Object,String)` method

Parameters

- **applicationContext** – bundle context used to look for annotations

12.81 org.motechproject.security.authentication

12.81.1 MotechAccessVoter

public class **MotechAccessVoter** implements AccessDecisionVoter<Object>

A custom AccessDecisionVoter for voting on whether a specific user has access to a particular URL. For example, a security rule can specify that the users motech and admin have access to a particular URL. This loads the metadata source with attributes for ACCESS_motech and ACCESS_admin. When a user invokes that URL, an affirmative based voting system will check whether or not the user is motech or admin. If not, they are denied permission, otherwise they are granted access.

Methods

supports

public boolean **supports** (ConfigAttribute *attribute*)

supports

public boolean **supports** (Class<?> *clazz*)

vote

public int **vote** (Authentication *authentication*, Object *object*, Collection<ConfigAttribute> *attributes*)

Checks if given user has access to given URL. If authentication details are not instance of MotechUserProfile or ConfigAttributes are empty then return ACCESS_ABSTAIN. If attribute is supported but User is not allowed then return ACCESS_DENIED, otherwise return ACCESS_GRANTED

Parameters

- **authentication** – to be used for check
- **attributes** – that contains information about access for users

Returns ACCESS_ABSTAIN, ACCESS_DENIED or ACCESS_GRANTED

12.81.2 MotechBasicAuthenticationEntryPoint

public class **MotechBasicAuthenticationEntryPoint** extends BasicAuthenticationEntryPoint

An entry point for BASIC authentications, sets the correct server realm key.

Fields

SECURITY_REALM_KEY

public static final String **SECURITY_REALM_KEY**

Constructors

MotechBasicAuthenticationEntryPoint

public **MotechBasicAuthenticationEntryPoint** (SettingsFacade *settingsFacade*)

12.81.3 MotechLoginErrorHandler

public class **MotechLoginErrorHandler** extends SimpleUrlAuthenticationFailureHandler

Class responsible for increasing user failure login counter. Extends `SimpleUrlAuthenticationFailureHandler`. It also redirect user to error login page.

See also: `org.springframework.security.web.authentication.SimpleUrlAuthenticationFailureHa`

Methods

onAuthenticationFailure

public void **onAuthenticationFailure** (HttpServletRequest *request*, HttpServletResponse *response*, AuthenticationException *exception*)

setUserBlockedUrl

public void **setUserBlockedUrl** (String *userBlockedUrl*)

Sets the URL which will be used as the user blocked destination.

Parameters

- **userBlockedUrl** – the user blocked URL.

12.81.4 MotechLoginSuccessHandler

public class **MotechLoginSuccessHandler** extends SavedRequestAwareAuthenticationSuccessHandler

Class responsible for logging info about users that log in and for resetting their failure login counter. Extends `SavedRequestAwareAuthenticationSuccessHandler`. It also serves as a fallback for storing sessions that started with the server before web-security was started.

See also: `org.springframework.security.web.authentication.SavedRequestAwareAuthenticationS`

Methods

onAuthenticationSuccess

public void **onAuthenticationSuccess** (HttpServletRequest *request*, HttpServletResponse *response*, Authentication *authentication*)

12.81.5 MotechLoginUrlAuthenticationEntryPoint

public class **MotechLoginUrlAuthenticationEntryPoint** extends LoginUrlAuthenticationEntryPoint

Used to commence a form login authentication.

Methods

commence

public void **commence** (HttpServletRequest *request*, HttpServletResponse *response*, AuthenticationException *authException*)

Performs the redirect (or forward) to the login form URL.

12.81.6 MotechLogoutSuccessHandler

public class **MotechLogoutSuccessHandler** implements LogoutHandler

A logout handler for logging users that log out from MOTECH.

Methods

logout

public void **logout** (HttpServletRequest *request*, HttpServletResponse *response*, Authentication *authentication*)

12.81.7 MotechPasswordEncoder

public class **MotechPasswordEncoder** extends BCryptPasswordEncoder

Class responsible for password encoding

Methods

encodePassword

public String **encodePassword** (String *rawPassword*)

Encodes given password using BCrypt hashing

Parameters

- **rawPassword** – password to be encoded

Returns encoded password

isPasswordValid

public boolean **isPasswordValid** (String *encPassword*, String *rawPassword*)

Encodes rawPassword and checks if it's the same as encoded one

Parameters

- **encPassword** – encoded password
- **rawPassword** – not encoded password

Returns true if passwords are the same, false otherwise

12.81.8 MotechRestBasicAuthenticationEntryPoint

public class **MotechRestBasicAuthenticationEntryPoint** extends BasicAuthenticationEntryPoint

A custom entry point that is invoked when there is an authentication exception within the filter. This ensures that when a user does not have login privileges and are unable to authenticate, a 401 unauthorized response is returned.

Fields

SECURITY_REALM_KEY

public static final String **SECURITY_REALM_KEY**

Constructors

MotechRestBasicAuthenticationEntryPoint

public **MotechRestBasicAuthenticationEntryPoint** (SettingsFacade *settingsFacade*)

Methods

commence

public void **commence** (HttpServletRequest *request*, HttpServletResponse *response*, AuthenticationException *authException*)

12.82 org.motechproject.security.builder

12.82.1 SecurityRuleBuilder

public class **SecurityRuleBuilder**

The security rule builder is responsible for building a SecurityFilterChain, which consists of a matcher pattern and a list of Spring security filters. The filters are created and configured base upon the security rule's settings.

Fields

NO_METHODS_REQUIRED_EXCEPTION_MESSAGE

public static final String **NO_METHODS_REQUIRED_EXCEPTION_MESSAGE**

NO_PATTERN_EXCEPTION_MESSAGE

public static final String **NO_PATTERN_EXCEPTION_MESSAGE**

NO_PROTOCOL_EXCEPTION_MESSAGE

public static final String **NO_PROTOCOL_EXCEPTION_MESSAGE**

NO_SUPPORTED_SCHEMES_EXCEPTION_MESSAGE

public static final String **NO_SUPPORTED_SCHEMES_EXCEPTION_MESSAGE**

Methods

buildSecurityChain

public synchronized SecurityFilterChain **buildSecurityChain** (MotechURLSecurityRule *securityRule*, HTTPMethod *method*)

Builds SecurityFilterChain which is capable of being matched against HttpServletRequest in order to decide whether it applies to that request

Parameters

- **securityRule** – that will be used as pattern
- **method** – to be used in filter

Returns new filter chain with security rule, matcher and filters

setAuthenticationManager

public void **setAuthenticationManager** (AuthenticationManager *authenticationManager*)

setBasicAuthenticationEntryPoint

public void **setBasicAuthenticationEntryPoint** (AuthenticationEntryPoint *basicAuthenticationEntryPoint*)

setChannelDecisionManager

public void **setChannelDecisionManager** (ChannelDecisionManager *channelDecisionManager*)

setLoginAuthenticationEntryPoint

public void **setLoginAuthenticationEntryPoint** (AuthenticationEntryPoint *loginAuthenticationEntryPoint*)

setMotechLogoutHandler

public void **setMotechLogoutHandler** (MotechLogoutSuccessHandler *motechLogoutHandler*)

setOpenIDAuthenticationFilter

public void **setOpenIDAuthenticationFilter**(OpenIDAuthenticationFilter *openIDAuthenticationFilter*)

setSettingsFacade

public void **setSettingsFacade**(SettingsFacade *settingsFacade*)

setUsernamePasswordAuthenticationFilter

public void **setUsernamePasswordAuthenticationFilter**(UsernamePasswordAuthenticationFilter *usernamePasswordAuthenticationFilter*)

12.83 org.motechproject.security.constants

12.83.1 HTTPMethod

public enum **HTTPMethod**

Enumeration of HTTP request method.

Enum Constants

ANY

public static final HTTPMethod **ANY**

DELETE

public static final HTTPMethod **DELETE**

GET

public static final HTTPMethod **GET**

HEAD

public static final HTTPMethod **HEAD**

OPTIONS

public static final HTTPMethod **OPTIONS**

POST

public static final HTTPMethod **POST**

PUT

public static final HTTPMethod **PUT**

TRACE

public static final HTTPMethod **TRACE**

12.83.2 PermissionNames

public final class **PermissionNames**
Contains permission names constants

Fields

MANAGE_ROLE_AND_PERMISSION_PERMISSION

public static final String **MANAGE_ROLE_AND_PERMISSION_PERMISSION**

MANAGE_URL_PERMISSION

public static final String **MANAGE_URL_PERMISSION**

MANAGE_USER_PERMISSION

public static final String **MANAGE_USER_PERMISSION**

MDS_DATA_ACCESS_PERMISSION

public static final String **MDS_DATA_ACCESS_PERMISSION**

MDS_SCHEMA_ACCESS_PERMISSION

public static final String **MDS_SCHEMA_ACCESS_PERMISSION**

MDS_SETTINGS_ACCESS_PERMISSION

public static final String **MDS_SETTINGS_ACCESS_PERMISSION**

VIEW_BASIC_EMAIL_LOGS_PERMISSION

public static final String **VIEW_BASIC_EMAIL_LOGS_PERMISSION**

VIEW_DETAILED_EMAIL_LOGS_PERMISSION

public static final String **VIEW_DETAILED_EMAIL_LOGS_PERMISSION**

12.83.3 Protocol

public enum **Protocol**
Enumeration of Protocols supported by the module

Enum Constants

HTTP

public static final Protocol **HTTP**

HTTPS

public static final Protocol **HTTPS**

12.83.4 Scheme

public enum **Scheme**
Enumeration of scheme supported by the module

Enum Constants

BASIC

public static final Scheme **BASIC**

NO_SECURITY

public static final Scheme **NO_SECURITY**

OATH

public static final Scheme **OATH**
not supported yet

OPEN_ID

public static final Scheme **OPEN_ID**

USERNAME_PASSWORD

public static final Scheme **USERNAME_PASSWORD**

12.83.5 SecurityConfigConstants

public final class **SecurityConfigConstants**

A class for holding constants related to the options available for dynamic security rules. MotechURLSecurityRule is where these options are used. Prefixes related to security voting are also stored in this class.

Fields

ANY_PATTERN

public static final String **ANY_PATTERN**

ROLE_ACCESS_PREFIX

public static final String **ROLE_ACCESS_PREFIX**

SYSTEM_ORIGIN

public static final String **SYSTEM_ORIGIN**

USER_ACCESS_PREFIX

public static final String **USER_ACCESS_PREFIX**

12.83.6 UserRoleNames

public final class **UserRoleNames**

Contains role names constants

Fields

ADMIN_USER

public static final String **ADMIN_USER**

BUNDLE_ADMIN_ROLE

public static final String **BUNDLE_ADMIN_ROLE**

EMAIL_ADMIN_ROLE

public static final String **EMAIL_ADMIN_ROLE**

MDS_ADMIN

public static final String **MDS_ADMIN**

MOTECH_ADMIN

public static final String **MOTECH_ADMIN**

ROLES_ADMIN

public static final String **ROLES_ADMIN**

SECURITY_ADMIN_ROLE

public static final String **SECURITY_ADMIN_ROLE**

USER_ADMIN_ROLE

public static final String **USER_ADMIN_ROLE**

12.83.7 WebSecurityRoles

public final class **WebSecurityRoles**

Contains constants used for securing parts of the web-security module.

Fields

HAS_MANAGE_ROLE_AND_PERMISSION

public static final String **HAS_MANAGE_ROLE_AND_PERMISSION**

HAS_MANAGE_URL

public static final String **HAS_MANAGE_URL**

HAS_MANAGE_USER

public static final String **HAS_MANAGE_USER**

12.84 org.motechproject.security.domain

12.84.1 MotechPermission

public class **MotechPermission**
Entity representing permission

Constructors

MotechPermission

public **MotechPermission** ( )

MotechPermission

public **MotechPermission** (String *permissionName*, String *bundleName*)

Methods

getBundleName

public String **getBundleName** ( )

getPermissionName

public String **getPermissionName** ( )

setBundleName

public void **setBundleName** (String *bundleName*)

setPermissionName

public void **setPermissionName** (String *permissionName*)

toString

public String **toString** ( )

12.84.2 MotechRole

public class **MotechRole**
Entity that represents role in Motech

Constructors

MotechRole

public **MotechRole** ( )

MotechRole

public **MotechRole** (String *roleName*, List<String> *permissionNames*, boolean *deletable*)

Methods

getPermissionNames

public List<String> **getPermissionNames** ( )

getRoleName

public String **getRoleName** ( )

hasPermission

public boolean **hasPermission** (String *permissionName*)

isDeletable

public boolean **isDeletable** ( )

removePermission

public void **removePermission** (String *permissionName*)

setDeletable

public void **setDeletable** (boolean *deletable*)

setPermissionNames

public void **setPermissionNames** (List<String> *permissionNames*)

setRoleName

public void **setRoleName** (String *roleName*)

12.84.3 MotechSecurityConfiguration

public class **MotechSecurityConfiguration**

The MotechSecurityConfiguration is a single document that contains all of the URL security rule configuration. The configuration was designed as one document because the entire filter chain must be reconstructed each time it is updated, therefore managing many references is unnecessary.

Constructors

MotechSecurityConfiguration

public **MotechSecurityConfiguration** ()

MotechSecurityConfiguration

public **MotechSecurityConfiguration** (List<MotechURLSecurityRule> *securityRules*)

Methods

getSecurityRules

public List<MotechURLSecurityRule> **getSecurityRules** ()

setSecurityRules

public void **setSecurityRules** (List<MotechURLSecurityRule> *securityRules*)

12.84.4 MotechURLSecurityRule

public class **MotechURLSecurityRule**

The MotechURLSecurityRule specifies the configuration for setting up a Spring SecurityFilterChain.

Details regarding configuration:

- **pattern** - URL pattern the security rule applies to
- **supportedSchemes** - Security rules that should apply to the URL, such as BASIC or OPEN_ID
- **protocol** - Protocol the security rule applies to, such as HTTP or HTTPS
- **permissionAccess** - Requires user has at least one of the listed permission to access the URL
- **userAccess** - User specific access for a URL, such as motech or admin, when combined with permission access they act as an either or (one must be true)
- **priority** - For future use in determining the ordering of filter chains, may be deprecated depending on UI implementation

- **rest** - Whether the endpoint is meant for a form login process or as an REST end-point that does not create a session for the
- **origin** - The module or user the rule originated from
- **version** - The version of the module or platform the rule was created
- **methodsRequired** - HTTP methods the rule applies to, if ANY is used then any method is matched, if a set is used, such as GET, POST, etc, then each will have its own corresponding filter chain with the same security found in that rule

Methods

getId

public Long **getId**()

getMethodsRequired

public List<HTTPMethod> **getMethodsRequired**()

getOrigin

public String **getOrigin**()

getPattern

public String **getPattern**()

getPermissionAccess

public List<String> **getPermissionAccess**()

getPriority

public int **getPriority**()

getProtocol

public Protocol **getProtocol**()

getSupportedSchemes

public List<Scheme> **getSupportedSchemes**()

getUserAccess

public List<String> **getUserAccess** ()

getVersion

public String **getVersion** ()

isActive

public boolean **isActive** ()

isDeleted

public boolean **isDeleted** ()

isRest

public boolean **isRest** ()

setActive

public void **setActive** (boolean *active*)

setDeleted

public void **setDeleted** (boolean *deleted*)

setId

public void **setId** (Long *id*)

setMethodsRequired

public void **setMethodsRequired** (List<HTTPMethod> *methodsRequired*)

setOrigin

public void **setOrigin** (String *origin*)

setPattern

public void **setPattern** (String *pattern*)

setPermissionAccess

public void **setPermissionAccess** (List<String> *permissionAccess*)

setPriority

public void **setPriority** (int *priority*)

setProtocol

public void **setProtocol** (Protocol *protocol*)

setRest

public void **setRest** (boolean *rest*)

setSupportedSchemes

public void **setSupportedSchemes** (List<Scheme> *supportedSchemes*)

setUserAccess

public void **setUserAccess** (List<String> *userAccess*)

setVersion

public void **setVersion** (String *version*)

toString

public String **toString** ()

12.84.5 MotechUser

public class **MotechUser**

Entity that represents Motech user

Constructors

MotechUser

public **MotechUser** ()

MotechUser

public **MotechUser** (String *userName*, String *password*, String *email*, String *externalId*, List<String> *roles*, String *openId*, Locale *locale*)

Methods

equals

public boolean **equals** (Object *o*)

getEmail

public String **getEmail** ()

getExternalId

public String **getExternalId** ()

getFailureLoginCounter

public Integer **getFailureLoginCounter** ()

getLastPasswordChange

public DateTime **getLastPasswordChange** ()

getLocale

public Locale **getLocale** ()

getOpenId

public String **getOpenId** ()

getPassword

public String **getPassword** ()

getRoles

public List<String> **getRoles** ()

getUserName

public String **getUserName**()

getUserStatus

public UserStatus **getUserStatus**()

hasRole

public boolean **hasRole**(String *role*)

hashCode

public int **hashCode**()

isActive

public boolean **isActive**()

setEmail

public void **setEmail**(String *email*)

setExternalId

public void **setExternalId**(String *externalId*)

setFailureLoginCounter

public void **setFailureLoginCounter**(Integer *failureLoginCounter*)

setLastPasswordChange

public void **setLastPasswordChange**(DateTime *lastPasswordChange*)

setLocale

public void **setLocale**(Locale *locale*)

setOpenId

public void **setOpenId**(String *openId*)

setPassword

public void **setPassword** (String *password*)

setRoles

public void **setRoles** (List<String> *roles*)

setUserName

public void **setUserName** (String *userName*)

setUserStatus

public void **setUserStatus** (UserStatus *userStatus*)

toString

public String **toString** ()

12.84.6 MotechUserProfile

public class **MotechUserProfile** implements Serializable
Represents Motech user

Constructors

MotechUserProfile

public **MotechUserProfile** (MotechUser *user*)

Methods

getExternalId

public String **getExternalId** ()

getLocale

public Locale **getLocale** ()

getRoles

public List<String> **getRoles** ()

getUserName

public String **getUserName**()

getUserStatus

public UserStatus **getUserStatus**()

hasRole

public boolean **hasRole**(String *role*)

isActive

public boolean **isActive**()

12.84.7 PasswordRecovery

public class **PasswordRecovery**

Entity that holds data used for password recovery

Methods

getEmail

public String **getEmail**()

getExpirationDate

public DateTime **getExpirationDate**()

getLocale

public Locale **getLocale**()

getToken

public String **getToken**()

getUsername

public String **getUsername**()

setEmail

public void **setEmail** (String *email*)

setExpirationDate

public void **setExpirationDate** (DateTime *expirationDate*)

setLocale

public void **setLocale** (Locale *locale*)

setToken

public void **setToken** (String *token*)

setUsername

public void **setUsername** (String *username*)

12.84.8 SecurityRuleComparator

public class **SecurityRuleComparator** implements Comparator<MotechURLSecurityRule>

Class that helps to compare `org.motechproject.security.domain.MotechURLSecurityRule`

Methods

compare

public int **compare** (MotechURLSecurityRule *o1*, MotechURLSecurityRule *o2*)

Compares two MotechURLSecurityRules to select more important one (one with higher priority or one that comes from the system or one with longer pattern). First checks if both priorities are the same, if yes then checks for origin of both rules. If both of origins equals `org.motechproject.security.constants.SecurityConfigConstants.SYSTEM_ORIGIN` or if none of them then compares length of patterns from both rules. If origin of only one of the rules is equal then returns number that represents given rule. If priorities of both rules are not the same then just compares them.

Parameters

- **o1** – first rule
- **o2** – second rule

Returns number that represents one of the rules - will return 1 if first rule is more important or -1 if the second one

12.84.9 UserStatus

public enum **UserStatus**
Represents the user status.

Enum Constants

ACTIVE

public static final UserStatus **ACTIVE**
User is active.

BLOCKED

public static final UserStatus **BLOCKED**
User is blocked.

12.85 org.motechproject.security.ex

12.85.1 EmailExistsException

public class **EmailExistsException** extends RuntimeException
Exception that signalizes that given email is already used by other user

Constructors

EmailExistsException

public **EmailExistsException** (String *message*)

12.85.2 InvalidTokenException

public class **InvalidTokenException** extends Exception
Exception that signalizes that given token is invalid

Constructors

InvalidTokenException

public **InvalidTokenException** ()

InvalidTokenException

public **InvalidTokenException** (String *message*)

12.85.3 NonAdminUserException

public class **NonAdminUserException** extends Exception
Exception that signalizes that given user is not an admin

Constructors

NonAdminUserException

public **NonAdminUserException** (String *message*)

12.85.4 PasswordTooShortException

public class **PasswordTooShortException** extends RuntimeException
Signals that password is shorter than the configured minimal length.

Constructors

PasswordTooShortException

public **PasswordTooShortException** (int *minLength*)

Parameters

- **minLength** – the configured minimal length of the password

Methods

getMinLength

public int **getMinLength** ()

Returns the configured minimal length of the password

12.85.5 PasswordValidatorException

public class **PasswordValidatorException** extends RuntimeException
Signals that the password didn't pass validation.

Constructors

PasswordValidatorException

public **PasswordValidatorException** (String *msg*)

12.85.6 RoleHasUserException

public class **RoleHasUserException** extends RuntimeException
Represents a failed attempt to delete a role currently assigned to a user.

Constructors

RoleHasUserException

public **RoleHasUserException** (String *message*)

12.85.7 SecurityConfigException

public class **SecurityConfigException** extends RuntimeException

A runtime exception thrown when the security config does not pass validation constraints required in order to construct a new security chain. Ideally should not be thrown as the UI should not allow invalid data to be submitted.

Constructors

SecurityConfigException

public **SecurityConfigException** (String *message*)

12.85.8 ServerUrlIsEmptyException

public class **ServerUrlIsEmptyException** extends RuntimeException

Exception which signalizes that server url property in platform settings is empty

Constructors

ServerUrlIsEmptyException

public **ServerUrlIsEmptyException** ()

ServerUrlIsEmptyException

public **ServerUrlIsEmptyException** (String *message*)

12.85.9 UserNotFoundException

public class **UserNotFoundException** extends Exception

Exception that signalizes that given user was not found

Constructors

UserNotFoundException

public **UserNotFoundException** ()

UserNotFoundException

public **UserNotFoundException** (String *message*)

12.86 org.motechproject.security.model

12.86.1 PermissionDto

public class **PermissionDto** implements Serializable

The `PermissionDto` contains information about permission.

Constructors

PermissionDto

public **PermissionDto** ()

PermissionDto

public **PermissionDto** (MotechPermission *motechPermission*)

PermissionDto

public **PermissionDto** (String *permissionName*, String *bundleName*)

Methods

equals

public boolean **equals** (Object *obj*)

getBundleName

public String **getBundleName** ()

getPermissionName

public String **getPermissionName** ()

hashCode

public int **hashCode** ()

setBundleName

public void **setBundleName** (String *bundleName*)

setPermissionName

public void **setPermissionName** (String *permissionName*)

toString

public String **toString** ()

12.86.2 RoleDto

public class **RoleDto**

Transfer Motech role data between representations.

Role data transfer object facilitates exchange of role data among services, repository, and client user interface.

Constructors

RoleDto

public **RoleDto** ()

RoleDto

public **RoleDto** (MotechRole *motechRole*)

RoleDto

public **RoleDto** (String *roleName*, List<String> *permissionNames*)

RoleDto

public **RoleDto** (String *roleName*, List<String> *permissionNames*, boolean *deletable*)

Methods

equals

public boolean **equals** (Object *obj*)

getOriginalRoleName

public String **getOriginalRoleName** ()

getPermissionNames

public List<String> **getPermissionNames** ( )

getRoleName

public String **getRoleName** ( )

hashCode

public int **hashCode** ( )

isDeletable

public boolean **isDeletable** ( )

setDeletable

public void **setDeletable** (boolean *deletable*)

setOriginalRoleName

public void **setOriginalRoleName** (String *originalRoleName*)

setPermissionNames

public void **setPermissionNames** (List<String> *permissionNames*)

setRoleName

public void **setRoleName** (String *roleName*)

toString

public String **toString** ( )

12.86.3 SecurityConfigDto

public class **SecurityConfigDto**

Used to transfer security configuration to and from a web request and UI

Methods

getSecurityRules

public List<SecurityRuleDto> **getSecurityRules**()

setSecurityRules

public void **setSecurityRules**(List<SecurityRuleDto> *securityRules*)

12.86.4 SecurityRuleDto

public class **SecurityRuleDto**

Transfer Motech security rule data between representations.

Methods

getId

public Long **getId**()

getMethodsRequired

public List<String> **getMethodsRequired**()

getOrigin

public String **getOrigin**()

getPattern

public String **getPattern**()

getPermissionAccess

public List<String> **getPermissionAccess**()

getPriority

public int **getPriority**()

getProtocol

public String **getProtocol**()

getSupportedSchemes

public List<String> **getSupportedSchemes**()

getUserAccess

public List<String> **getUserAccess**()

getVersion

public String **getVersion**()

isActive

public boolean **isActive**()

isDeleted

public boolean **isDeleted**()

isRest

public boolean **isRest**()

setActive

public void **setActive**(boolean *active*)

setDeleted

public void **setDeleted**(boolean *deleted*)

setId

public void **setId**(Long *id*)

setMethodsRequired

public void **setMethodsRequired**(List<String> *methodsRequired*)

setOrigin

public void **setOrigin**(String *origin*)

setPattern

public void **setPattern** (String *pattern*)

setPermissionAccess

public void **setPermissionAccess** (List<String> *permissionAccess*)

setPriority

public void **setPriority** (int *priority*)

setProtocol

public void **setProtocol** (String *protocol*)

setRest

public void **setRest** (boolean *rest*)

setSupportedSchemes

public void **setSupportedSchemes** (List<String> *supportedSchemes*)

setUserAccess

public void **setUserAccess** (List<String> *userAccess*)

setVersion

public void **setVersion** (String *version*)

12.86.5 UserDto

public class **UserDto**

Transfers Motech user data between representations

Constructors

UserDto

public **UserDto** ()

UserDto

public **UserDto** (MotechUser *motechUser*)

Methods

getEmail

public String **getEmail** ()

getExternalId

public String **getExternalId** ()

getLocale

public Locale **getLocale** ()

getOpenId

public String **getOpenId** ()

getPassword

public String **getPassword** ()

getRoles

public List<String> **getRoles** ()

getUserName

public String **getUserName** ()

getUserStatus

public UserStatus **getUserStatus** ()

isGeneratePassword

public boolean **isGeneratePassword** ()

setEmail

public void **setEmail** (String *email*)

setExternalId

public void **setExternalId** (String *externalId*)

setGeneratePassword

public void **setGeneratePassword** (boolean *generatePassword*)

setLocale

public void **setLocale** (Locale *locale*)

setOpenId

public void **setOpenId** (String *openId*)

setPassword

public void **setPassword** (String *password*)

setRoles

public void **setRoles** (List<String> *roles*)

setUserName

public void **setUserName** (String *userName*)

setUserStatus

public void **setUserStatus** (UserStatus *userStatus*)

12.87 org.motechproject.security.repository

12.87.1 AllMotechPermissions

public class **AllMotechPermissions**

Implementation of DAO interface that utilizes a MDS back-end for storage. Class responsible for handling MotechPermission.

Methods

add

public void **add** (MotechPermission *permission*)

Adds new MotechPermission and update Motech Admin role to contain it

Parameters

- **permission** – to be added

delete

public void **delete** (MotechPermission *permission*)

Deletes given MotechPermission

Parameters

- **permission** – to be removed

findByPermissionName

public MotechPermission **findByPermissionName** (String *permissionName*)

Returns MotechPermission with given name

Parameters

- **permissionName** – name of permission

Returns MotechPermission

getPermissions

public List<MotechPermission> **getPermissions** ()

Returns all MotechPermissions

Returns list that contains permissions

setAllMotechRoles

public void **setAllMotechRoles** (AllMotechRoles *allMotechRoles*)

setDataService

public void **setDataService** (MotechPermissionsDataService *dataService*)

12.87.2 AllMotechRoles

public class **AllMotechRoles**

Implementation of DAO interface that utilizes a MDS back-end for storage. Class responsible for handling MotechRoles.

Methods

add

public void **add** (MotechRole *role*)
Creates MotechRole if it doesn't exists

Parameters

- **role** – to be created

findByRoleName

public MotechRole **findByRoleName** (String *roleName*)
Looks for and returns MotechRole with given name

Parameters

- **roleName** – name of MotechRole

Returns MotechRole or null if name is a null

getRoles

public List<MotechRole> **getRoles** ()
Returns all MotechRoles

Returns list that contains roles

remove

public void **remove** (MotechRole *motechRole*)
Removes given MotechRole

Parameters

- **motechRole** – to be removed

setDataService

public void **setDataService** (MotechRolesDataService *dataService*)

update

public void **update** (MotechRole *motechRole*)
Updates given MotechRole

Parameters

- **motechRole** – to be updated

12.87.3 AllMotechSecurityRules

public class **AllMotechSecurityRules**

Implementation of DAO interface that utilizes a MDS back-end for storage. Only one MotechSecurityConfiguration file should be saved at a time, so adding the document looks for the old document in order to update it if it already exists. Rather than updating the object reference, the old configuration's ID and revision are used for the new document.

Methods

addOrUpdate

public void **addOrUpdate** (MotechSecurityConfiguration *config*)

Reads rules from `org.motechproject.security.domain.MotechSecurityConfiguration` and split them into those to be created, updated or removed. Before updating `org.motechproject.security.repository.MotechURLSecurityRuleDataService` is checked for old rule with the same id - update will be done only if it exists. Same thing happens for rules to be removed.

Parameters

- **config** –

getMotechSecurityConfiguration

public MotechSecurityConfiguration **getMotechSecurityConfiguration** ()

Gets MotechSecurityConfiguration

Returns configuration

getRuleById

public MotechURLSecurityRule **getRuleById** (Long *id*)

Returns MotechURLSecurityRule for given id

Parameters

- **id** – of security rule

Returns rule with given id or null in case when id == null

getRules

public List<MotechURLSecurityRule> **getRules** ()

Returns all MotechURLSecurityRules

Returns list that contains rules

getRulesByOrigin

public List<MotechURLSecurityRule> **getRulesByOrigin** (String *origin*)

Returns all MotechURLSecurityRules for given origin

Parameters

- **origin** – of security rules

Returns list that contains rules or null in case origin is null

getRulesByOriginAndVersion

public List<MotechURLSecurityRule> **getRulesByOriginAndVersion** (String *origin*, String *version*)

Returns all MotechURLSecurityRules for given origin and version

Parameters

- **origin** – of security rules
- **version** – of security rules

Returns list that contains rules or empty list in case origin or version is null

remove

public void **remove** (MotechSecurityConfiguration *config*)

Removes all rules from given MotechSecurityConfiguration

Parameters

- **config** – with rules to be removed

setDataService

public void **setDataService** (MotechURLSecurityRuleDataService *dataService*)

12.87.4 AllMotechUsers

public class **AllMotechUsers**

Implementation of DAO interface that utilizes a MDS back-end for storage. Class responsible for handling MotechUsers.

Methods

add

public void **add** (MotechUser *user*)

Adds new MotechUser if its name and email are not null

Parameters

- **user** – to be added

addOpenIdUser

public void **addOpenIdUser** (MotechUser *user*)

Adds new MotechUser with OpenId as long as its not a null

Parameters

- **user** – to be added

findByRole

public List<MotechUser> **findByRole** (String *role*)

Returns MotechUsers with given role

Parameters

- **role** – of users

Returns list that contains users with given role or null in case when role == null

findByUserName

public MotechUser **findByUserName** (String *userName*)

Gets MotechUser with given name

Parameters

- **userName** – name of user

Returns user with given name or null in case when userName == null

findUserByEmail

public MotechUser **findUserByEmail** (String *email*)

Gets MotechUser with given email

Parameters

- **email** – of user

Returns user with given email or null in case when email == null

findUserByOpenId

public MotechUser **findUserByOpenId** (String *openId*)

Gets MotechUser with given OpenId

Parameters

- **openId** – of user

Returns user with given OpenId or null in case when openId == null

getOpenIdUsers

public List<MotechUser> **getOpenIdUsers** ()

Returns all MotechUsers that comes from `org.motechproject.server.config.domain.LoginMode.OPEN_ID`

Returns list that contains users

getUsers

public List<MotechUser> **getUsers** ()

Returns all MotechUsers that comes from `org.motechproject.server.config.domain.LoginMode.REPOSITORY`

Returns list that contains users

remove

public void **remove** (MotechUser *motechUser*)

Deletes given MotechUser

Parameters

- **motechUser** – to be removed

setDataService

public void **setDataService** (MotechUsersDataService *dataService*)

update

public void **update** (MotechUser *motechUser*)

Updates given MotechUser as long as his email is not used by another user

Parameters

- **motechUser** – to be updated

12.87.5 AllPasswordRecoveries

public class **AllPasswordRecoveries**

Implementation of DAO interface that utilizes a MDS back-end for storage. Class responsible for handling PasswordRecoveries.

Methods

add

public void **add** (PasswordRecovery *passwordRecovery*)

Adds given PasswordRecovery provided tha one doesn't exist yet for the user

Parameters

- **passwordRecovery** – to be added

allRecoveries

public List<PasswordRecovery> **allRecoveries**()

Returns all PasswordRecoveries

Returns list that contains recoveries

createRecovery

public PasswordRecovery **createRecovery**(String *username*, String *email*, String *token*, DateTime *expirationDate*, Locale *locale*)

Creates PasswordRecovery for given informations and return it

Parameters

- **username** – for recovery
- **email** – for recovery
- **token** – for recovery
- **expirationDate** – for recovery
- **locale** – for recovery

Returns recovery with given informations

findForToken

public PasswordRecovery **findForToken**(String *token*)

Gets PasswordRecovery for given token

Parameters

- **token** – for recovery

Returns recovery for given token or null in case when token is a null

findForUser

public PasswordRecovery **findForUser**(String *username*)

Gets PasswordRecovery for user with given name

Parameters

- **username** – name of user

Returns recovery for given name or null in case when username is a null

getExpired

public List<PasswordRecovery> **getExpired**()

Returns all expired PasswordRecoveries

Returns list that contains recoveries

remove

public void **remove** (PasswordRecovery *passwordRecovery*)
Deletes given PasswordRecovery

Parameters

- **passwordRecovery** – to be removed

setDataService

public void **setDataService** (PasswordRecoveriesDataService *dataService*)

update

public void **update** (PasswordRecovery *passwordRecovery*)
Updates given PasswordRecovery

Parameters

- **passwordRecovery** – to be updated

12.87.6 MotechPermissionsDataService

public interface **MotechPermissionsDataService** extends MotechDataService<MotechPermission>
Interface for data service injected by MDS

Methods

findByPermissionName

MotechPermission **findByPermissionName** (String *permissionName*)

12.87.7 MotechRolesDataService

public interface **MotechRolesDataService** extends MotechDataService<MotechRole>
Interface for data service injected by MDS

Methods

findByRoleName

MotechRole **findByRoleName** (String *roleName*)

12.87.8 MotechURLSecurityRuleDataService

public interface **MotechURLSecurityRuleDataService** extends MotechDataService<MotechURLSecurityRule>
Interface for data service injected by MDS

Methods

findByOrigin

List<MotechURLSecurityRule> **findByOrigin** (String *origin*)

findByOriginAndVersion

List<MotechURLSecurityRule> **findByOriginAndVersion** (String *origin*, String *version*)

12.87.9 MotechUsersDataService

public interface **MotechUsersDataService** extends MotechDataService<MotechUser>

Interface for data service injected by MDS

Methods

findByEmail

MotechUser **findByEmail** (String *email*)

findByOpenId

MotechUser **findByOpenId** (String *openId*)

findByRole

List<MotechUser> **findByRole** (String *role*)

findByUserName

MotechUser **findByUserName** (String *userName*)

12.87.10 PasswordRecoveriesDataService

public interface **PasswordRecoveriesDataService** extends MotechDataService<PasswordRecovery>

Interface for data service injected by MDS

Methods

findByExpirationDate

List<PasswordRecovery> **findByExpirationDate** (Range<DateTime> *range*)

findForToken

PasswordRecovery **findForToken** (String *token*)

findForUser

PasswordRecovery **findForUser** (String *username*)

12.88 org.motechproject.security.service

12.88.1 AuthoritiesService

public interface **AuthoritiesService**

Service interface to retrieve authorities(permissions) for a given MotechUser

Methods

authoritiesFor

List<GrantedAuthority> **authoritiesFor** (MotechUser *user*)

Gets list of `org.springframework.security.core.GrantedAuthority` for given user

Parameters

- **user** – for whom we want to get list

Returns list that contains `org.springframework.security.core.GrantedAuthority`

12.88.2 MotechPermissionService

public interface **MotechPermissionService**

Service for managing Motech permissions.

Methods

addPermission

void **addPermission** (PermissionDto *permission*)

Adds a new permission

Parameters

- **permission** – to be added

deletePermission

void **deletePermission** (String *permissionName*)

Deletes permission with given name

Parameters

- **permissionName** – name of the permission to be removed

getPermissions

List<PermissionDto> **getPermissions**()

Gets list of all permissions

Returns list that contains permissions

12.88.3 MotechProxyManager

public class **MotechProxyManager**

The MotechProxyManager acts as a wrapper around Spring's FilterChainProxy. The FilterChainProxy contains a list of immutable SecurityFilterChain objects which Spring's security consults for filters when handling requests. In order to dynamically define new secure, a new FilterChainProxy is constructed and the reference is updated. The MotechProxyManager acts as a customized delegate in MotechDelegatingFilterProxy.

Methods

getDefaultSecurityConfiguration

public MotechSecurityConfiguration **getDefaultSecurityConfiguration**()

This method reads default security configuration from the file containing security rules and returns it.

Returns MotechSecurityConfiguration default security rules

getFilterChainProxy

public FilterChainProxy **getFilterChainProxy**()

initializeProxyChain

public void **initializeProxyChain**()

This method serves the same purpose of rebuildProxyChain, but does not require any kind of security authentication so it should only ever be used by the activator, which does not have an authentication object.

rebuildProxyChain

public synchronized void **rebuildProxyChain**()

Method to invoke to dynamically re-define the Spring security. All rules converted into security filter chains in order to create a new FilterChainProxy. The order of the rules in the list matters for filtering purposes.

setProxy

public void **setProxy**(FilterChainProxy *proxy*)

setSecurityRuleBuilder

public void **setSecurityRuleBuilder** (SecurityRuleBuilder *securityRuleBuilder*)

setSecurityRulesDAO

public void **setSecurityRulesDAO** (AllMotechSecurityRules *securityRulesDAO*)

12.88.4 MotechRoleService

public interface **MotechRoleService**
Service for managing Motech roles

Methods

createRole

void **createRole** (RoleDto *role*)
Creates new role

Parameters

- **role** – to be created

deleteRole

void **deleteRole** (RoleDto *role*)
Deletes given role

Parameters

- **role** – to be deleted

getRole

RoleDto **getRole** (String *roleName*)
Returns role with given name

Parameters

- **roleName** – name of the role that should be returned

Returns role with given name

getRoles

List<RoleDto> **getRoles** ()
Returns all roles

Returns list that contains roles

updateRole

void **updateRole** (RoleDto *role*)
Updates given role

Parameters

- **role** – to be updated

12.88.5 MotechURLSecurityService

public interface **MotechURLSecurityService**
Service to access and update security configuration details from the platform. Permission based, method level security is defined to prevent unauthorized users from updating security.

Methods

findAllSecurityRules

List<SecurityRuleDto> **findAllSecurityRules** ()
A protected method for viewing security rule information for the platform.

Returns All URL security rules found in the database

updateSecurityConfiguration

void **updateSecurityConfiguration** (SecurityConfigDto *configuration*)
A protected method for updating security configuration for the platform.

Parameters

- **configuration** – The updated security information, which will cause an updating of the motech proxy manager

12.88.6 MotechUserService

public interface **MotechUserService**
Service interface that defines APIs to retrieve and manage user details

Methods

activateUser

void **activateUser** (String *username*)
Activates user

Parameters

- **username** – of user to be activated

changeEmail

void **changeEmail** (String *email*)

Changes the e-mail address of a currenty logged user

Parameters

- **email** – a new e-mail address

changePassword

MotechUserProfile **changePassword** (String *oldPassword*, String *newPassword*)

Allows to change a password of a currently logged-in user.

Parameters

- **oldPassword** – An old password of currently logged user
- **newPassword** – A new password for the currently logged user

Returns MotechUserProfile with updated user information

changePassword

MotechUserProfile **changePassword** (String *userName*, String *oldPassword*, String *newPassword*)

Changes password of user with given username and return his `org.motechproject.security.domain.MotechUserPr`

Parameters

- **userName** – of user
- **oldPassword** – password that was used before
- **newPassword** – new password for user

Returns user profile after password change

deleteUser

void **deleteUser** (UserDto *user*)

Deletes given user

Parameters

- **user** – to be removed

getCurrentUser

UserDto **getCurrentUser** ()

Returns user that is logged in current session

Returns current user

getLocale

Locale **getLocale** (String *userName*)

Returns `java.util.Locale` of user with given name

Parameters

- **userName** – of user

Returns locale of user

getOpenIdUsers

List<MotechUserProfile> **getOpenIdUsers** ()

Returns `org.motechproject.security.domain.MotechUserProfile` of users with set OpenId

Returns list that contains users with OpenId

getRoles

List<String> **getRoles** (String *userName*)

Returns all roles of user with given name

Parameters

- **userName** – name of user

Returns list that contains user roles

getUser

UserDto **getUser** (String *userName*)

Returns user with given name

Parameters

- **userName** – of user

Returns user with given name

getUserByEmail

UserDto **getUserByEmail** (String *email*)

Returns user with given email

Parameters

- **email** – of user

Returns user with given email

getUsers

List<MotechUserProfile> **getUsers**()

Returns all `org.motechproject.security.domain.MotechUserProfile`

Returns list that contains profiles

hasActiveMotechAdmin

boolean **hasActiveMotechAdmin**()

Checks if there active user with Admin role

Returns true if user exists, otherwise false

hasEmail

boolean **hasEmail**(String *email*)

Checks if user with given email exists

Parameters

- **email** – of user

Returns true if user exists, otherwise false

hasUser

boolean **hasUser**(String *username*)

Checks if user with given name exists

Parameters

- **username** – of user

Returns true if user exists, otherwise return false

register

void **register**(String *username*, String *password*, String *email*, String *externalId*, List<String> *roles*, Locale *locale*)

Registers new user

Parameters

- **username** – of new user
- **password** – of new user
- **email** – of new user
- **externalId** – of new user
- **roles** – list that contains roles for new user
- **locale** – to be set as default for new user

register

void **register** (String *username*, String *password*, String *email*, String *externalId*, List<String> *roles*, Locale *locale*, UserStatus *userStatus*, String *openId*)

Registers new user

Parameters

- **username** – of new user
- **password** – of new user
- **email** – of new user
- **externalId** – of new user
- **roles** – list that contains roles for new user
- **locale** – to be set as default for new user
- **userStatus** – user status, `org.motechproject.security.domain.UserStatus`
- **openId** – of new user

registerMotechAdmin

void **registerMotechAdmin** (String *username*, String *password*, String *email*, Locale *locale*)

A method that allows to register the first MOTECH Admin in the application. Throws `java.lang.IllegalStateException` when an active Admin User is already registered.

Parameters

- **username** – Username of a new user
- **password** – Password of a new user
- **email** – Email address of a new user
- **locale** – Selected locale for the new user

retrieveUserByCredentials

MotechUserProfile **retrieveUserByCredentials** (String *username*, String *password*)

Returns `org.motechproject.security.domain.MotechUserProfile` for user with given username and password

Parameters

- **username** – of user to be returned
- **password** – of user to be returned

Returns profile of user with given credentials

sendLoginInformation

void **sendLoginInformation** (String *userName*)

Sends login information by email using address set for user with given name

Parameters

- **userName** – name of user

Throws

- **UserNotFoundException** – when user has not been found
- **NonAdminUserException** – when user is not an admin

setLocale

void **`setLocale`** (Locale *locale*)

Sets `org.motechproject.security.domain.MotechUserProfile` for user in current session

Parameters

- **locale** – to be set for user

updateUserDetailsWithPassword

void **`updateUserDetailsWithPassword`** (UserDto *user*)

Updates user and set new password

Parameters

- **user** – to be updated

updateUserDetailsWithoutPassword

void **`updateUserDetailsWithoutPassword`** (UserDto *user*)

Updates user without setting new password

Parameters

- **user** – to be updated

validatePassword

void **`validatePassword`** (String *password*)

Checks whether the password meets requirements

Parameters

- **password** – the password to validate

Throws

- **PasswordValidatorException** – when password is not valid

12.88.7 PasswordRecoveryService

public interface **`PasswordRecoveryService`**

Service that defines APIs to manage password recovery

Methods

cleanUpExpiredRecoveries

void **cleanUpExpiredRecoveries**()

Removes all expired recoveries

oneTimeTokenOpenId

String **oneTimeTokenOpenId**(String *email*)

Creates an one time token for OpenId for the user with the given email address and sends a recovery email

Parameters

- **email** – address of the user

Throws

- **UserNotFoundException** – when no user for the given email exists
- **NonAdminUserException** – when the user for the given email is not an admin (don't have Admin role)

Returns the recovery token that can be used for resetting the password

oneTimeTokenOpenId

String **oneTimeTokenOpenId**(String *email*, boolean *notify*)

Creates an one time token for OpenId for the user with the given email address, with an optional email notification.

Parameters

- **email** – address of the user
- **notify** – about the recovery

Throws

- **UserNotFoundException** – when no user with the given email exists
- **NonAdminUserException** – when the user for the given email is not an admin (don't have Admin role)

Returns the recovery token that can be used for resetting the password

oneTimeTokenOpenId

String **oneTimeTokenOpenId**(String *email*, DateTime *expiration*, boolean *notify*)

Creates an one time token for OpenId for the user with the given email address, with an optional email notification. The recovery will expire on the given date.

Parameters

- **email** – address of the user
- **expiration** – date of recovery, it shouldn't be a past date
- **notify** – about the recovery

Throws

- **UserNotFoundException** – when no user with the given email exists
- **NonAdminUserException** – when the user for the given email is not an admin (don't have Admin role)

Returns the recovery token that can be used for resetting the password

passwordRecoveryRequest

String **passwordRecoveryRequest** (String *email*)

Creates password recovery for the user with the given email address and sends a recovery email

Parameters

- **email** – address of the user

Throws

- **UserNotFoundException** – when no user for the given email exists

Returns the recovery token that can be used for resetting the password

passwordRecoveryRequest

String **passwordRecoveryRequest** (String *email*, boolean *notify*)

Creates password recovery for the user with the given email address, with an optional email notification.

Parameters

- **email** – address of the user
- **notify** – about the recovery

Throws

- **UserNotFoundException** – when no user for the given email exists

Returns the recovery token that can be used for resetting the password

passwordRecoveryRequest

String **passwordRecoveryRequest** (String *email*, DateTime *expiration*)

Creates password recovery for the user with the given email address and sends a recovery email. The recovery will expire on the given date.

Parameters

- **email** – address of the user
- **expiration** – date of recovery, it shouldn't be a past date

Throws

- **UserNotFoundException** – when no user for the given email exists

Returns the recovery token that can be used for resetting the password

passwordRecoveryRequest

String **passwordRecoveryRequest** (String *email*, DateTime *expiration*, boolean *notify*)

Creates password recovery for the user with the given email address, with an optional email notification. The recovery will expire on the given date.

Parameters

- **email** – address of the user
- **expiration** – date of recovery, it shouldn't be a past date
- **notify** – about the recovery

Throws

- **UserNotFoundException** – when no user for the given email exists

Returns the recovery token that can be used for resetting the password

resetPassword

void **resetPassword** (String *token*, String *password*, String *passwordConfirmation*)

Sets new password for user from token

Parameters

- **token** – for `org.motechproject.security.domain.PasswordRecovery`
- **password** – to be set for user
- **passwordConfirmation** – to check is password is correct

Throws

- **InvalidTokenException** – when `org.motechproject.security.domain.PasswordRecovery` as a null, recovery is already expired or when user for name from token doesn't exists

validateToken

boolean **validateToken** (String *token*)

Checks if there's a not expired `org.motechproject.security.domain.PasswordRecovery` for given token

Parameters

- **token** – to validate

Returns true if recovery exists, otherwise false

validateTokenAndLoginUser

void **validateTokenAndLoginUser** (String *token*, HttpServletRequest *request*, HttpServletResponse *response*)

Creates new openId Token for user from token as long as there's a `org.motechproject.security.domain.PasswordRecovery` for that token and redirect to home page. If there's no such recovery then redirect to login page

Parameters

- **token** – for password recovery
- **request** – for session
- **response** – for session

Throws

- **IOException** – when response cannot redirect to given URL (home or login page)

12.88.8 SecurityRoleLoader

public class **`SecurityRoleLoader`**

Helper class that scans an application context for Motech roles

Constructors

SecurityRoleLoader

public **`SecurityRoleLoader`** (MotechRoleService *roleService*, MotechPermissionService *permissionService*)

Methods

loadRoles

public void **`loadRoles`** (ApplicationContext *applicationContext*)

Loads from roles.json file and adds or update them using `org.motechproject.security.service.MotechRoleServ`

Parameters

- **applicationContext** – in which file with roles can be found

12.88.9 SecurityRuleLoaderService

public interface **`SecurityRuleLoaderService`**

Service that scans an application context for security rules and re-initializes the MotechProxyManager security chain.

Methods

loadRules

void **`loadRules`** (ApplicationContext *applicationContext*)

Attempts to load rules from the application context, if rules are found, the security configuration is updated.

12.88.10 SecurityRuleLoaderServiceImpl

public class **`SecurityRuleLoaderServiceImpl`** implements SecurityRuleLoaderService

Methods

loadRules

public synchronized void **loadRules** (ApplicationContext *applicationContext*)

setAllSecurityRules

public void **setAllSecurityRules** (AllMotechSecurityRules *allSecurityRules*)

setProxyManager

public void **setProxyManager** (MotechProxyManager *proxyManager*)

12.88.11 UserContextService

public interface **UserContextService**
Interface to refresh user context (all or specified username)

Methods

logoutUser

void **logoutUser** (String *userName*)

refreshAllUsersContextIfActive

void **refreshAllUsersContextIfActive** ()
Refreshes context of all users as long as they're active

refreshUserContextIfActive

void **refreshUserContextIfActive** (String *userName*)
Refreshes context of user with given name

Parameters

- **userName** – name of user

12.89 org.motechproject.security.validator

12.89.1 PasswordValidator

public interface **PasswordValidator**
Service interface that validates password

Methods

getName

String **getName** ()

Returns the name of the validator used for retrieval. Must match the value from the configuration in order to be used.

Returns the name of this validator

getValidationError

String **getValidationError** (Locale *locale*)

Returns the error message for the validator. Should explain what is expected of the password. The message should be treated as a literal, meaning localization is left to the validator implementation.

Parameters

- **locale** – the locale for which the error message should be returned

Returns the localized error message

validate

void **validate** (String *password*)

Validates password.

Parameters

- **password** – the password to check.

Throws

- **org.motechproject.security.ex.PasswordValidatorException** – signals an issue with the validation

12.89.2 ValidatorNames

public final class **ValidatorNames**

A collection of constants representing the names of the validators registered by Motech.

Fields

LOWERCASE_UPPERCASE

public static final String **LOWERCASE_UPPERCASE**

LOWERCASE_UPPERCASE_DIGIT

public static final String **LOWERCASE_UPPERCASE_DIGIT**

LOWERCASE_UPPERCASE_DIGIT_SPECIAL

public static final String **LOWERCASE_UPPERCASE_DIGIT_SPECIAL**

MIN_PASS_LENGTH

public static final String **MIN_PASS_LENGTH**

NONE

public static final String **NONE**

12.90 org.motechproject.server.api

12.90.1 BundleIcon

public class **BundleIcon**

Represents an icon of a bundle. It will be displayed next to module name in the Manage Modules Section in the Admin panel.

Fields

ICON_LOCATIONS

public static final String[] **ICON_LOCATIONS**

Constructors

BundleIcon

public **BundleIcon** (byte[] *icon*, String *mime*)

Constructor.

Parameters

- **icon** – the icon to be stored as a byte array
- **mime** – the mime type of an icon

Methods

getContentLength

public int **getContentLength** ()

Returns size of the icon.

Returns the size of stored icon (in bytes)

getIcon

public byte[] **getIcon** ()

getMime

public String **getMime** ()

12.90.2 BundleInformation

public class **BundleInformation**

Class acting as a DTO for a `Bundle` in the system. Aggregates information about a single bundle.

Fields

BUNDLE_NAME

protected static final String **BUNDLE_NAME**

DOC_URL

public static final String **DOC_URL**

Constructors

BundleInformation

public **BundleInformation** (Bundle *bundle*)

Constructor.

Parameters

- **bundle** – the bundle which this BundleInformation instance will represent

Methods

equals

public boolean **equals** (Object *arg0*)

getAngularModule

public String **getAngularModule** ()

Returns the name of the angular module

getBundleId

public long **getBundleId**()

Returns the id of the bundle

getDocURL

public String **getDocURL**()

Returns the documentation URL for this bundle

getLocation

public String **getLocation**()

Returns the bundle's location identifier

getModuleName

public String **getModuleName**()

Returns the module name

getName

public String **getName**()

Returns the name of the bundle

getSettingsURL

public String **getSettingsURL**()

Returns the url to the settings page of the bundle

getState

public State **getState**()

Returns a string representation of the state of the bundle

getSymbolicName

public String **getSymbolicName**()

Returns the symbolic name of the bundle

getVersion

public Version **getVersion** ( )

Returns the version of the bundle

hasStatus

public boolean **hasStatus** (int *status*)

hashCode

public int **hashCode** ( )

setAngularModule

public void **setAngularModule** (String *angularModule*)
Sets the angular module name.

Parameters

- **angularModule** – the name of the angular module

setModuleName

public void **setModuleName** (String *moduleName*)
Sets the module name.

Parameters

- **moduleName** – the name of the module

setSettingsURL

public void **setSettingsURL** (String *settingsURL*)
Sets the url to the settings page.

Parameters

- **settingsURL** – the url to the settings page

12.90.3 BundleInformation.State

public enum **State**
Represents the bundle state.

Enum Constants

ACTIVE

public static final BundleInformation.State **ACTIVE**

INSTALLED

public static final BundleInformation.State **INSTALLED**

RESOLVED

public static final BundleInformation.State **RESOLVED**

STARTING

public static final BundleInformation.State **STARTING**

STOPPING

public static final BundleInformation.State **STOPPING**

UNINSTALLED

public static final BundleInformation.State **UNINSTALLED**

UNKNOWN

public static final BundleInformation.State **UNKNOWN**

12.90.4 BundleLoader

public interface **BundleLoader**

Interface for custom bundle loading processes

Author Ricky Wang

Methods

loadBundle

void **loadBundle** (Bundle *bundle*)

Parameters

- **bundle** – the bundle to process

Throws

- **BundleLoadingException** – if there were issues while loading the bundle

12.90.5 BundleLoadingException

public class **BundleLoadingException** extends Exception

This exception is thrown when a problem occurs during bundle loading.

Constructors

BundleLoadingException

public **BundleLoadingException** (String *message*)

BundleLoadingException

public **BundleLoadingException** (String *message*, Throwable *cause*)

BundleLoadingException

public **BundleLoadingException** (Throwable *cause*)

12.90.6 JarInformation

public class **JarInformation**
Holds all important information about JAR.

Fields

BUNDLE_SYMBOLIC_NAME

public static final String **BUNDLE_SYMBOLIC_NAME**

BUNDLE_VERSION

public static final String **BUNDLE_VERSION**

EXTRACTION_FAILED

public static final String **EXTRACTION_FAILED**

IMPLEMENTATION_TITLE

public static final String **IMPLEMENTATION_TITLE**

IMPLEMENTATION_VENDOR_ID

public static final String **IMPLEMENTATION_VENDOR_ID**

IMPLEMENTATION_VERSION

public static final String **IMPLEMENTATION_VERSION**

Constructors

JarInformation

public **JarInformation** (File *file*)
Constructor,

Parameters

- **file** – the file representation of a jar file or directory containing extracted jar

Throws

- **IOException** – if an I/O error has occurred

Methods

getBundleSymbolicName

public String **getBundleSymbolicName** ()

getBundleVersion

public String **getBundleVersion** ()

getDependencies

public List<Dependency> **getDependencies** ()

getFilename

public String **getFilename** ()

getImplementationTitle

public String **getImplementationTitle** ()

getImplementationVendorID

public String **getImplementationVendorID** ()

getImplementationVersion

public String **getImplementationVersion** ()

getPath

public String **getPath** ()

getRepositories

public List<RemoteRepository> **getRepositories** ()

isMotechPlatformBundle

public boolean **isMotechPlatformBundle** ()

readPOMInformation

public void **readPOMInformation** (File *file*)

Reads information from pom file and stores information about repositories and dependencies in this object.

Parameters

- **file** – the file representation of a jar file or directory containing extracted jar

12.90.7 JarInformationHandler

public class **JarInformationHandler**

Stores information about a jar.

Fields

JAR_FILE_EXTENSION

public static final String **JAR_FILE_EXTENSION**

Constructors

JarInformationHandler

public **JarInformationHandler** (String *path*)

Constructor.

Parameters

- **path** – the path to the jar file or directory containing extracted jar

Methods

extractJarInformationFromPath

public void **extractJarInformationFromPath** ()

getJarList

public List<JarInformation> **getJarList** ()

getPath

public String **getPath**()

initHandler

public void **initHandler**()
Initializes this handler object.

12.91 org.motechproject.server.config

12.91.1 SettingsFacade

public class **SettingsFacade**
SettingsFacade provides an interface to access application configuration present in files or database.

Methods

afterPropertiesSet

public void **afterPropertiesSet**()

areConfigurationSettingsRegistered

public boolean **areConfigurationSettingsRegistered**()
Checks if configuration settings have been registered.

Returns true if setting have been registered, false otherwise

asProperties

public Properties **asProperties**()
Converts stored configuration to `Properties`.

Returns the configuration as `Properties`

findFilename

protected String **findFilename**(String *key*)
Returns a name of a file containing given property.

Parameters

- **key** – the property name

Returns the name of a file

getBundleSymbolicName

public String **getBundleSymbolicName**()

getBundleVersion

public String **getBundleVersion**()

getPlatformSettings

public MotechSettings **getPlatformSettings**()

getProperties

public Properties **getProperties**(String *filename*)

Returns properties from a resource with given filename.

Parameters

- **filename** – the resource filename

Returns properties stored in the file

getProperty

public String **getProperty**(String *key*)

getProperty

public String **getProperty**(String *key*, String *filename*)

Returns a value of a property with given key, stored in a resource with given filename.

Parameters

- **key** – the name of the property
- **filename** – the resource filename

Returns property value as `String`

getRawConfig

public InputStream **getRawConfig**(String *filename*)

Allows to retrieve raw JSON data either from the database or file.

Parameters

- **filename** – Resource filename

Throws

- **org.motechproject.commons.api.MotechException** – when I/O error occurs

Returns Raw JSON data as InputStream

getResourceFileName

protected static String **getResourceFileName** (Resource *resource*)

Returns a name of resource file

Parameters

- **resource** – the resource file

Returns the file name of the resource

registerAllProperties

protected void **registerAllProperties** ()

Registers all the properties to the configuration service.

registerAllRawConfig

protected void **registerAllRawConfig** ()

Registers all raw configurations to the configuration service.

registerProperties

protected void **registerProperties** (String *filename*, Properties *properties*)

Registers properties from file with given name to the configuration service.

Parameters

- **filename** – the name of the file with properties
- **properties** – properties to be registered

saveConfigProperties

public void **saveConfigProperties** (String *filename*, Properties *properties*)

Saves given properties and resource filename to the configuration. If configuration properties stored in this object were already registered to the configuration service, the given properties and resource filename will also be added there.

Parameters

- **filename** – the resource filename
- **properties** – the properties to be saved

Throws

- **org.motechproject.commons.api.MotechException** – when I/O error occurs

savePlatformSettings

public void **savePlatformSettings** (MotechSettings *settings*)

Saves given MOTECH settings to the configuration service.

Parameters

- **settings** – the `MotechSettings` to be saved

saveRawConfig

public void **`saveRawConfig`** (String *filename*, Resource *resource*)

Allows persisting of raw JSON properties either in the database or file.

Parameters

- **filename** – resource filename
- **resource** – resource data to persist

Throws

- **org.motechproject.commons.api.MotechException** – when I/O error occurs

saveRawConfig

public void **`saveRawConfig`** (String *filename*, String *jsonText*)

Allows persisting of raw JSON properties either in the database or file.

Parameters

- **filename** – json filename
- **jsonText** – json data to persist

Throws

- **org.motechproject.commons.api.MotechException** – when I/O error occurs

setBundleContext

public void **`setBundleContext`** (BundleContext *bundleContext*)

setConfigFiles

public void **`setConfigFiles`** (List<Resource> *resources*)

setProperty

public void **`setProperty`** (String *key*, String *value*)

setRawConfigFiles

public void **`setRawConfigFiles`** (List<Resource> *resources*)

unregisterProperties

public void **unregisterProperties** (String *symbolicName*)
Unregisters properties of the bundle with given symbolic name.

Parameters

- **symbolicName** – the symbolic name of the bundle

12.92 org.motechproject.server.config.domain

12.92.1 LoginMode

public final class **LoginMode**
Encapsulates the operations on login mode.

Fields

OPEN_ID

public static final LoginMode **OPEN_ID**

REPOSITORY

public static final LoginMode **REPOSITORY**

Methods

getName

public String **getName** ()

isOpenId

public boolean **isOpenId** ()
Checks if this login mode is set to "Open ID".

Returns true if this login mode is set to "Open ID", false otherwise

isRepository

public boolean **isRepository** ()
Checks if this login mode is set to "Repository".

Returns true if this login mode is set to "Repository", false otherwise

valueOf

public static LoginMode **valueOf** (String *loginMode*)

Creates proper login mode from given `String`, which can be either "repository" or "openId".

Parameters

- **loginMode** – the login mode to be created, must be either "repository" or "openId", other values will return null

Returns the proper object of `LoginMode`, null if given value was neither "repository" nor "openId"

12.92.2 MotechSettings

public interface **MotechSettings**

Interface for main MOTECH settings management.

Methods

asProperties

Properties **asProperties** ()

Converts this MOTECH setting to `Properties`.

Returns this object as `Properties`

getConfigFileChecksum

String **getConfigFileChecksum** ()

getEmailRequired

boolean **getEmailRequired** ()

getFailureLoginLimit

int **getFailureLoginLimit** ()

Gets the failure login limit.

Returns the failure login limit

getFilePath

String **getFilePath** ()

getJmxBroker

String **getJmxBroker** ()

getJmxHost

String **getJmxHost**()

getLanguage

String **getLanguage**()

getLastRun

DateTime **getLastRun**()

getLoginMode

LoginMode **getLoginMode**()

getMinPasswordLength

Integer **getMinPasswordLength**()

Returns the minimal length of user passwords in MOTECH.

Returns the minimal length of the password, 0 or less means no minimal length

getPasswordValidator

String **getPasswordValidator**()

Returns the name of the password validator. The validator with that name will be retrieved by web-security for validation of new password.

Returns the name of the validator

getProviderName

String **getProviderName**()

getProviderUrl

String **getProviderUrl**()

getServerHost

String **getServerHost**()

getServerUrl

String **getServerUrl**()

getSessionTimeout

Integer **getSessionTimeout**()

Gets the http session timeout for Motech users. Users will be logged out after reaching this timeout. This value is specified in seconds. A negative value specifies that sessions should never time out.

Returns the http session timeout, in seconds

getStatusMsgTimeout

String **getStatusMsgTimeout**()

getUploadSize

String **getUploadSize**()

isPlatformInitialized

boolean **isPlatformInitialized**()

Checks whether platform is initialized.

Returns true if platform is initialized, false otherwise

load

void **load**(DigestInputStream *dis*)

Loads the properties from given stream and stores them withing this object.

Parameters

- **dis** – the source stream

Throws

- **IOException** – when I/O error occurs

savePlatformSetting

void **savePlatformSetting**(String *key*, String *value*)

Adds or updates given key-value pair within this object.

Parameters

- **key** – the key of the pair
- **value** – the value of the pair

setConfigFileChecksum

void **setConfigFileChecksum**(String *configFileChecksum*)

setEmailRequired

void **setEmailRequired** (String *emailRequired*)

setFailureLoginLimit

void **setFailureLoginLimit** (int *limit*)

Sets the failure login limit. After reaching this limit user will be blocked. If 0 then blocking will be disabled.

Parameters

- **limit** – the failure login limit

setFilePath

void **setFilePath** (String *filePath*)

setJmxBroker

void **setJmxBroker** (String *jmxBroker*)

setJmxHost

void **setJmxHost** (String *jmxHost*)

setLanguage

void **setLanguage** (String *language*)

setLastRun

void **setLastRun** (DateTime *lastRun*)

setLoginModeValue

void **setLoginModeValue** (String *loginMode*)

setMinPasswordLength

void **setMinPasswordLength** (Integer *minPasswordLength*)

Sets the minimal length of user passwords in MOTECH.

Parameters

- **minPasswordLength** – the minimal length of the password, 0 or less means no minimal length

setPasswordValidator

void **setPasswordValidator** (String *validator*)

Sets the name of the password validator. The validator with that name will be retrieved by web-security for validation of new password.

Parameters

- **validator** – the name of the validator

setPlatformInitialized

void **setPlatformInitialized** (boolean *platformInitialized*)

setProviderName

void **setProviderName** (String *providerName*)

setProviderUrl

void **setProviderUrl** (String *providerUrl*)

setServerUrl

void **setServerUrl** (String *serverUrl*)

setSessionTimeout

void **setSessionTimeout** (Integer *sessionTimeout*)

Sets the http session timeout for Motech users. Users will be logged out after reaching this timeout. This value is specified in seconds. A negative value specifies that sessions should never time out.

Parameters

- **sessionTimeout** – the http session timeout, in seconds

setStatusMsgTimeout

void **setStatusMsgTimeout** (String *statusMsgTimeout*)

setUploadSize

void **setUploadSize** (String *uploadSize*)

updateFromProperties

void **updateFromProperties** (Properties *props*)
Updates this object with given properties.

Parameters

- **props** – properties to be applied

updateSettings

void **updateSettings** (String *configFileChecksum*, String *filePath*, Properties *platformSettings*)
Updates settings with given information.

Parameters

- **configFileChecksum** – the configuration file checksum to be set
- **filePath** – the file path to be set
- **platformSettings** – the platform settings to be add

12.92.3 MotechURL

public class **MotechURL**
A MOTECH class representing URL using "protocol://host" pattern.

Fields

URL_PATTERN

public static final String **URL_PATTERN**

Constructors

MotechURL

public **MotechURL** (String *url*)
Constructor.

Parameters

- **url** – the URL to be stored, if it doesn't include protocol, "http://" will be added in front

Methods

getHost

public String **getHost** ()
Returns host of the stored URL.

Returns the host of stored URL

toString

public String **toString**()

12.92.4 SettingsRecord

public class **SettingsRecord** implements MotechSettings

Class for storing settings values.

Constructors

SettingsRecord

public **SettingsRecord**()

Methods

asProperties

public Properties **asProperties**()

getConfigFileChecksum

public String **getConfigFileChecksum**()

getEmailRequired

public boolean **getEmailRequired**()

getFailureLoginLimit

public int **getFailureLoginLimit**()

getFilePath

public String **getFilePath**()

getJmxBroker

public String **getJmxBroker**()

getJmxHost

public String **getJmxHost**()

getLanguage

public String **getLanguage**()

getLastRun

public DateTime **getLastRun**()

getLoginMode

public LoginMode **getLoginMode**()

getLoginModeValue

public String **getLoginModeValue**()

getMinPasswordLength

public Integer **getMinPasswordLength**()

getPasswordValidator

public String **getPasswordValidator**()

getPlatformSettings

public Map<String, String> **getPlatformSettings**()

getProviderName

public String **getProviderName**()

getProviderUrl

public String **getProviderUrl**()

getServerHost

public String **getServerHost**()

getServerUrl

public String **getServerUrl**()

getSessionTimeout

public Integer **getSessionTimeout**()

getStatusMsgTimeout

public String **getStatusMsgTimeout**()

getUploadSize

public String **getUploadSize**()

isPlatformInitialized

public boolean **isPlatformInitialized**()

load

public synchronized void **load**(DigestInputStream *dis*)

mergeWithDefaults

public void **mergeWithDefaults**(Properties *defaultConfig*)

Merges given default configuration into existing platform settings. Keys that already exists won't be overwritten.

Parameters

- **defaultConfig** – the default configuration to be merged.

removeDefaults

public void **removeDefaults**(Properties *defaultConfig*)

Removes settings specified in defaultConfig.

Parameters

- **defaultConfig** –

savePlatformSetting

public void **savePlatformSetting**(String *key*, String *value*)

setConfigFileChecksum

public void **setConfigFileChecksum**(String *configFileChecksum*)

setEmailRequired

public void **setEmailRequired** (String *emailRequired*)

setFailureLoginLimit

public void **setFailureLoginLimit** (int *limit*)

setFilePath

public void **setFilePath** (String *filePath*)

setJmxBroker

public void **setJmxBroker** (String *jmxBroker*)

setJmxHost

public void **setJmxHost** (String *jmxHost*)

setLanguage

public void **setLanguage** (String *language*)

setLastRun

public void **setLastRun** (DateTime *lastRun*)

setLoginModeValue

public void **setLoginModeValue** (String *loginMode*)

setMinPasswordLength

public void **setMinPasswordLength** (Integer *minPasswordLength*)

setPasswordValidator

public void **setPasswordValidator** (String *validator*)

setPlatformInitialized

public void **setPlatformInitialized** (boolean *platformInitialized*)

setPlatformSettings

public void **setPlatformSettings** (Map<String, String> *platformSettings*)

setProviderName

public void **setProviderName** (String *providerName*)

setProviderUrl

public void **setProviderUrl** (String *providerUrl*)

setServerUrl

public void **setServerUrl** (String *serverUrl*)

setSessionTimeout

public void **setSessionTimeout** (Integer *sessionTimeout*)

setStatusMsgTimeout

public void **setStatusMsgTimeout** (String *statusMsgTimeout*)

setUploadSize

public void **setUploadSize** (String *uploadSize*)

updateFromProperties

public void **updateFromProperties** (Properties *props*)

updateSettings

public void **updateSettings** (String *configFileChecksum*, String *filePath*, Properties *platformSettings*)

12.93 org.motechproject.server.config.service

12.93.1 ConfigLoader

public class **ConfigLoader**

Config loader used to load the platform core settings.

Methods

findExistingConfigs

public List<File> **findExistingConfigs**()
Finds all configurations from the configuration location.

Throws

- **IOException** – If there is any error while handling the files.

loadDefaultConfig

public SettingsRecord **loadDefaultConfig**()
Loads default MOTECH settings.

Returns the {SettingsRecord} object

loadMotechSettings

public SettingsRecord **loadMotechSettings**()
Loads MOTECH settings containing core platform settings.

Returns the {SettingsRecord} object

setCoreConfigurationService

public void **setCoreConfigurationService**(CoreConfigurationService *coreConfigurationService*)

setResourceLoader

public void **setResourceLoader**(ResourceLoader *resourceLoader*)

12.93.2 SettingService

public interface **SettingService** extends MotechDataService<SettingsRecord>
Interface for settings service. Its implementation is injected by the MDS.

12.94 org.motechproject.server.osgi.event

12.94.1 OsgiEventProxy

public interface **OsgiEventProxy**
This service allows sending Motech events without having a direct dependency on the event. This is achieved by sending OSGi events, which are then relayed as Motech events by the event module. This mechanism is used by MDS in order to avoid a dependency on the event module. In normal use, using this service should be avoided, as using the event system directly should be cleaner and more efficient.

Fields

BROADCAST_PARAM

String **BROADCAST_PARAM**

PARAMETERS_PARAM

String **PARAMETERS_PARAM**

PROXY_EVENT_TOPIC

String **PROXY_EVENT_TOPIC**

PROXY_ON_RECEIVING_END_PARAM

String **PROXY_ON_RECEIVING_END_PARAM**

SUBJECT_PARAM

String **SUBJECT_PARAM**

Methods

broadcastEvent

void **broadcastEvent** (String *subject*, boolean *proxyHandledEventInOSGi*)

Calling this method will result in sending an OSGi event that will be then relayed by the event module as a Motech Event through the event topic - all Motech instances will receive the event. Note that broadcast events can be relayed as OSGi events upon being received - if that's the case, their subject must conform to the rules for OSGi event topic (i.e. not contain dots).

Parameters

- **subject** – the subject of the event
- **proxyHandledEventInOSGi** – if true, the event will be also sent as an OSGi event upon being received by the event system

broadcastEvent

void **broadcastEvent** (String *subject*, Map<String, Object> *parameters*, boolean *proxyHandledEventInOSGi*)

Calling this method will result in sending an OSGi event that will be then relayed by the event module as a Motech Event through the event topic - all Motech instances will receive the event. Note that broadcast events can be relayed as OSGi events upon being received - if that's the case, their subject must conform to the rules for OSGi event topic (i.e. not contain dots).

Parameters

- **subject** – the subject of the event

- **parameters** – the parameters map which will act as the payload of the event
- **proxyHandledEventInOSGi** – if true, the event will be also sent as an OSGi event upon being received by the event system

sendEvent

void **sendEvent** (String *subject*)

Calling this method will result in sending an OSGi event that will be then relayed by the event module as a Motech Event through the event queue - only one Motech instance will receive the event.

Parameters

- **subject** – the subject of the event

sendEvent

void **sendEvent** (String *subject*, Map<String, Object> *parameters*)

Calling this method will result in sending an OSGi event that will be then relayed by the event module as a Motech Event through the event queue - only one Motech instance will receive the event.

Parameters

- **subject** – the subject of the event
- **parameters** – the parameters map which will act as the payload of the event

12.95 org.motechproject.server.osgi.status

12.95.1 PlatformStatus

public class **PlatformStatus** implements Serializable

Represents the status of the platform startup. It contains information about which bundles were started by Gemini Blueprint, it also carries information about both OSGi level and Spring context level errors that occurred in the system.

Fields

REQUIRED_FOR_STARTUP

public static final int **REQUIRED_FOR_STARTUP**

Methods

addBundleError

void **addBundleError** (String *bundleSymbolicName*, String *error*)

addContextError

void **addContextError** (String *bundleSymbolicName*, String *error*)

addOSGiStartedBundle

void **addOSGiStartedBundle** (String *symbolicName*)

addStartedBundle

void **addStartedBundle** (String *bundleSymbolicName*)

errorsOccurred

public boolean **errorsOccurred** ()

Returns true if we faced any errors context/bundle. This doesn't necessarily mean a startup failure.

Returns true if errors occurred, false otherwise

getBundleErrorsByBundle

public Map<String, String> **getBundleErrorsByBundle** ()

Returns bundles errors that occurred in the system in a form of a map. The keys in the map are bundle symbolic names. The values are error messages. Bundle errors are errors that occurred on the OSGi level, and prevented the bundle itself from starting.

Returns bundle errors that occurred in the system

getContextErrorsByBundle

public Map<String, String> **getContextErrorsByBundle** ()

Returns context errors that occurred in the system in a form of a map. The keys in the map are bundle symbolic names. The values are error messages. Context errors are errors that occurred during the creation of the Blueprint context.

Returns context errors that occurred in the system

getOsgiStartedBundles

public List<String> **getOsgiStartedBundles** ()

Returns bundles started in the OSGi meaning of the term. Although context startup is not tied to this, it requires the bundle to be started first.

Returns bundles started in the OSGi framework

getStartedBundles

public List<String> **getStartedBundles** ()

Returns started bundles. To be considered started a bundle must had its Spring context successfully created by Gemini Blueprint. We do not track non-blueprint enabled bundles here.

Returns the started bundles.

getStartupProgressPercentage

public int **getStartupProgressPercentage** ()

Returns the startup progress in percent. The startup progress represents the number of started bundles in relation to the number of bundles that is required for the server to be fully started. This is capped at 100%.

Returns the startup progress in percent

inFatalError

public boolean **inFatalError** ()

Returns true if we faced a fatal error during startup, meaning a platform bundle failed to start. This means a startup failure.

Returns true if we occurred such an error, false otherwise

removeOSGiStartedBundle

void **removeOSGiStartedBundle** (String *symbolicName*)

removeStartedBundle

void **removeStartedBundle** (String *bundleSymbolicName*)

setBundleErrorsByBundle

public void **setBundleErrorsByBundle** (Map<String, String> *bundleErrorsByBundle*)

Sets the bundles errors that occurred in the system in a form of a map. The keys in the map are bundle symbolic names. The values are error messages. Bundle errors are errors that occurred on the OSGi level, and prevented the bundle itself from starting. Failed bundles will be removed from the started bundle list.

Parameters

- **bundleErrorsByBundle** – bundle errors that occurred in the system

setContextErrorsByBundle

public void **setContextErrorsByBundle** (Map<String, String> *contextErrorsByBundle*)

Sets the context errors that occurred in the system in a form of a map. The keys in the map are bundle symbolic names. The values are error messages. Context errors are errors that occurred during the creation of the Blueprint context. Failed bundles will be removed from the started bundle list.

Parameters

- **contextErrorsByBundle** – context errors that occurred in the system

setOsgiStartedBundles

public void **setOsgiStartedBundles** (List<String> *osgiStartedBundles*)

Returns bundles started in the OSGi meaning of the term. Although context startup is not tied to this, it requires the bundle to be started first.

Parameters

- **osgiStartedBundles** – started in the OSGi framework

setStartedBundles

public void **setStartedBundles** (List<String> *startedBundles*)

Sets the started bundles. To be considered started a bundle must had its Spring context successfully created by Gemini Blueprint. We do not track non-blueprint enabled bundles here.

Parameters

- **startedBundles** – the started bundles.

12.95.2 PlatformStatusManager

public interface **PlatformStatusManager**

This is an interface for the manager of the platform status. The manager should keep track of the status and return it to callers.

Methods

getCurrentStatus

PlatformStatus **getCurrentStatus** ()

Used to fetch the current status of the platform.

Returns the current status of the platform, never null

12.95.3 PlatformStatusManagerImpl

public class **PlatformStatusManagerImpl** implements PlatformStatusManager, OsgiBundleApplicationContextListener, Bundle

PlatformStatusManager implementation. Acts as an listener for Blueprint events to get notified about modules being started or failing. It also exposes a method used by PlatformActivator for notifying about OSGi (not blueprint) bundle errors. It also acts as an OSGi event listener for keeping track of bundles that were started by OSGi (includes all bundles in the system). It keeps a single platform status instance, that it keeps updating.

Methods

bundleChanged

public void **bundleChanged** (BundleEvent *bundleEvent*)

getCurrentStatus

public PlatformStatus **getCurrentStatus** ()

onOsgiApplicationEvent

public void **onOsgiApplicationEvent** (OsgiBundleApplicationContextEvent *event*)

registerBundleError

public void **registerBundleError** (String *bundleSymbolicName*, String *error*)

Used for registering an OSGi error. This is not part of the interface and is used only by the PlatformActivator.

Parameters

- **bundleSymbolicName** – the symbolic name of the bundle which failed to start
- **error** – the actual error

12.96 org.motechproject.server.osgi.util

12.96.1 BundleType

public enum **BundleType**

Represents a logical bundle type. Used for determining startup order.

Enum Constants

FRAGMENT_BUNDLE

public static final BundleType **FRAGMENT_BUNDLE**

A fragment bundle, this should not be started(per OSGi spec), they attach themselves to the host.

FRAMEWORK_BUNDLE

public static final BundleType **FRAMEWORK_BUNDLE**

The OSGi framework bundle

HTTP_BUNDLE

public static final BundleType **HTTP_BUNDLE**

The HTTP bridge bundle, required for HTTP access to MOTECH.

MDS_BUNDLE

public static final BundleType **MDS_BUNDLE**

The Motech DataServices bundle. Required special treatment due to its nature of changing class definitions on the fly.

MOTECH_MODULE

public static final BundleType **MOTECH_MODULE**
A regular Motech module, starts after the platform.

PLATFORM_BUNDLE_POST_WS

public static final BundleType **PLATFORM_BUNDLE_POST_WS**
All platform bundles not included in the other platform bundle types.

PLATFORM_BUNDLE_PRE_MDS

public static final BundleType **PLATFORM_BUNDLE_PRE_MDS**
Bundles that MDS depends on - commons bundles, osgi-web-util, server-api and config-core.

PLATFORM_BUNDLE_PRE_WS

public static final BundleType **PLATFORM_BUNDLE_PRE_WS**
Bundles required for Web-security to start. These are event and server-config.

THIRD_PARTY_BUNDLE

public static final BundleType **THIRD_PARTY_BUNDLE**
This a 3rd party bundle, a library.

WS_BUNDLE

public static final BundleType **WS_BUNDLE**
The web-security bundle. Gets special treatment due to its crucial nature.

12.96.2 PlatformConstants

public final class **PlatformConstants**
Collection of constants related to the Platform startup and bundle management.

Fields

FELIX_FRAMEWORK_BUNDLE

public static final String **FELIX_FRAMEWORK_BUNDLE**

HTTP_BRIDGE_BUNDLE

public static final String **HTTP_BRIDGE_BUNDLE**

MDS_BUNDLE_PREFIX

public static final String **MDS_BUNDLE_PREFIX**

MDS_ENTITIES_BUNDLE

public static final String **MDS_ENTITIES_BUNDLE**

MDS_STARTUP_TOPIC

public static final String **MDS_STARTUP_TOPIC**

MOTECH_PACKAGE

public static final String **MOTECH_PACKAGE**

PAX_IT_SYMBOLIC_NAME

public static final String **PAX_IT_SYMBOLIC_NAME**

PLATFORM_BUNDLE_PREFIX

public static final String **PLATFORM_BUNDLE_PREFIX**

PLATFORM_BUNDLE_SYMBOLIC_NAME

public static final String **PLATFORM_BUNDLE_SYMBOLIC_NAME**

SECURITY_SYMBOLIC_NAME

public static final String **SECURITY_SYMBOLIC_NAME**

SERVER_SYMBOLIC_NAME

public static final String **SERVER_SYMBOLIC_NAME**

STARTUP_TOPIC

public static final String **STARTUP_TOPIC**

12.97 org.motechproject.server.startup

12.97.1 MotechPlatformState

public enum **MotechPlatformState**
Defines the different states of the MOTECH system.

Enum Constants

DB_ERROR

public static final MotechPlatformState **DB_ERROR**

FIRST_RUN

public static final MotechPlatformState **FIRST_RUN**

NEED_BOOTSTRAP_CONFIG

public static final MotechPlatformState **NEED_BOOTSTRAP_CONFIG**

NEED_CONFIG

public static final MotechPlatformState **NEED_CONFIG**

NORMAL_RUN

public static final MotechPlatformState **NORMAL_RUN**

NO_DB

public static final MotechPlatformState **NO_DB**

STARTUP

public static final MotechPlatformState **STARTUP**

12.97.2 StartupManager

public class **StartupManager**
StartupManager controlling and managing the application loading

Methods

canLaunchBundles

public boolean **canLaunchBundles**()

getDefaultSettings

public SettingsRecord **getDefaultSettings**()

This function is only called when the default configuration is loaded and is no config in the database or external files

isBootstrapConfigRequired

public boolean **isBootstrapConfigRequired**()

isConfigRequired

public boolean **isConfigRequired**()

startup

public void **startup**()

12.98 org.motechproject.server.ui.ex

12.98.1 AlreadyRegisteredException

public class **AlreadyRegisteredException** extends RuntimeException

Constructors

AlreadyRegisteredException

public **AlreadyRegisteredException**(String *msg*)

12.99 org.motechproject.server.web.controller

12.99.1 Constants

public final class **Constants**

Class that has all the common UI constants.

Fields

REDIRECT_BOOTSTRAP

public static final String **REDIRECT_BOOTSTRAP**

REDIRECT_HOME

public static final String **REDIRECT_HOME**

REDIRECT_STARTUP

public static final String **REDIRECT_STARTUP**

12.99.2 DashboardController

public class **DashboardController**

Main application controller. Responsible for retrieving information shared across the UI of different modules. The view returned by this controller will embed the UI of the currently requested module.

Methods

accessdenied

public ModelAndView **accessdenied**(HttpServletRequest *request*)

getNodeName

public String **getNodeName**()

getTime

public DateTime **getTime**()

getUptime

public DateTime **getUptime**()

getUser

public UserInfo **getUser**(HttpServletRequest *request*)

index

public ModelAndView **index**(HttpServletRequest *request*)

setBundleContext

public void **setBundleContext** (BundleContext *bundleContext*)

setLocaleService

public void **setLocaleService** (LocaleService *localeService*)

setStartupManager

public void **setStartupManager** (StartupManager *startupManager*)

12.99.3 ForgotController

public class **ForgotController**
Forgot Controller for reset password.

Methods

forgotPost

public String **forgotPost** (String *email*)

getForgotViewData

public ForgotViewData **getForgotViewData** (HttpServletRequest *request*)

login

public ModelAndView **login** (HttpServletRequest *request*)

12.99.4 LocaleController

public class **LocaleController**
The `LocaleController` class is responsible for handling requests connected with internationalization

Methods

getAvailableLocales

public Map<String, String> **getAvailableLocales** (HttpServletRequest *request*)

getLangLocalisation

public Map<String, String> **getLangLocalisation** (HttpServletRequest *request*)

getSupportedLanguages

public NavigableMap<String, String> **getSupportedLanguages** ( )

getUserLang

public String **getUserLang** (HttpServletRequest *request*)

setSessionLang

public void **setSessionLang** (HttpServletRequest *request*, HttpServletResponse *response*, LocaleDto *localeDto*)

setUserLang

public void **setUserLang** (HttpServletRequest *request*, HttpServletResponse *response*, LocaleDto *localeDto*)

12.99.5 LoginController

public class **LoginController**

Login Controller for user authentication.

Methods

getLoginViewData

public LoginViewData **getLoginViewData** (HttpServletRequest *request*)

login

public ModelAndView **login** (HttpServletResponse *response*)

12.99.6 ModuleController

public class **ModuleController**

Methods

getConfig

public List<ModuleConfig> **getConfig** ( )

getCriticalMessage

public String **getCriticalMessage** (String *moduleName*)

getMenu

public ModuleMenu **getMenu** (HttpServletRequest *request*)

getRestDocsUrl

public String **getRestDocsUrl** (String *moduleName*)

Returns the url for rest documentation spec of the given module

Parameters

- **moduleName** – the name of the module

Returns the url at which the REST API spec can be accessed

getUser

public UserInfo **getUser** (HttpServletRequest *request*)

setBundleContext

public void **setBundleContext** (BundleContext *bundleContext*)

setLocaleService

public void **setLocaleService** (LocaleService *localeService*)

setMenuBuilder

public void **setMenuBuilder** (MenuBuilder *menuBuilder*)

setUiFrameworkService

public void **setUiFrameworkService** (UIFrameworkService *uiFrameworkService*)

12.99.7 ResetController

public class **ResetController**

Methods

getResetViewData

public ResetViewData **getResetViewData** (HttpServletRequest *request*)

reset

public ResetViewData **reset** (ResetForm *form*, HttpServletRequest *request*)

resetView

public ModelAndView **resetView** (HttpServletRequest *request*)

12.99.8 StartupController

public class **StartupController**

StartupController that manages the platform system start up and captures the platform core settings and user information.

Methods

getStartupViewData

public StartupViewData **getStartupViewData** (HttpServletRequest *request*)

setStartupFormValidatorFactory

public void **setStartupFormValidatorFactory** (StartupFormValidatorFactory *validatorFactory*)

startup

public ModelAndView **startup** ()

submitForm

public List<String> **submitForm** (StartupForm *startupSettings*)

12.99.9 StatusController

public class **StatusController**

Methods

status

public String **status** ()

12.100 org.motechproject.server.web.form

12.100.1 LoginForm

public class **LoginForm**

Methods

getPassword

public String **getPassword**()

getUserName

public String **getUserName**()

setPassword

public void **setPassword**(String *password*)

setUserName

public void **setUserName**(String *userName*)

12.100.2 ResetForm

public class **ResetForm**

Fields

PASSWORD

public static final String **PASSWORD**

PASSWORD_CONFIRMATION

public static final String **PASSWORD_CONFIRMATION**

Methods

getPassword

public String **getPassword**()

getPasswordConfirmation

public String **getPasswordConfirmation**()

getToken

public String **getToken**()

setPassword

public void **setPassword**(String *password*)

setPasswordConfirmation

public void **setPasswordConfirmation**(String *passwordConfirmation*)

setToken

public void **setToken**(String *token*)

12.100.3 StartupForm

public class **StartupForm**

Fields

ADMIN_CONFIRM_PASSWORD

public static final String **ADMIN_CONFIRM_PASSWORD**

ADMIN_LOGIN

public static final String **ADMIN_LOGIN**

ADMIN_PASSWORD

public static final String **ADMIN_PASSWORD**

LANGUAGE

public static final String **LANGUAGE**

LOGIN_MODE

public static final String **LOGIN_MODE**

PROVIDER_NAME

public static final String **PROVIDER_NAME**

PROVIDER_URL

public static final String **PROVIDER_URL**

Methods

getAdminConfirmPassword

public String **getAdminConfirmPassword**()

getAdminEmail

public String **getAdminEmail**()

getAdminLogin

public String **getAdminLogin**()

getAdminPassword

public String **getAdminPassword**()

getLanguage

public String **getLanguage**()

getLoginMode

public String **getLoginMode**()

getProviderName

public String **getProviderName**()

getProviderUrl

public String **getProviderUrl**()

getSchedulerUrl

public String **getSchedulerUrl**()

setAdminConfirmPassword

public void **setAdminConfirmPassword** (String *adminConfirmPassword*)

setAdminEmail

public void **setAdminEmail** (String *adminEmail*)

setAdminLogin

public void **setAdminLogin** (String *adminLogin*)

setAdminPassword

public void **setAdminPassword** (String *adminPassword*)

setLanguage

public void **setLanguage** (String *language*)

setLoginMode

public void **setLoginMode** (String *loginMode*)

setProviderName

public void **setProviderName** (String *providerName*)

setProviderUrl

public void **setProviderUrl** (String *providerUrl*)

setSchedulerUrl

public void **setSchedulerUrl** (String *schedulerUrl*)

12.100.4 StartupSuggestionsForm

public class **StartupSuggestionsForm**

Constructors

StartupSuggestionsForm

public **StartupSuggestionsForm** ( )

Methods

addDatabaseSuggestion

public void **addDatabaseSuggestion** (String *suggestion*)

addQueueSuggestion

public void **addQueueSuggestion** (String *suggestion*)

addSchedulerSuggestion

public void **addSchedulerSuggestion** (String *suggestion*)

getDatabaseUrls

public List<String> **getDatabaseUrls** ()

getQueueUrls

public List<String> **getQueueUrls** ()

getSchedulerUrls

public List<String> **getSchedulerUrls** ()

12.100.5 UserInfo

public class **UserInfo**

Constructors

UserInfo

public **UserInfo** (String *userName*, boolean *securityLaunch*, String *lang*)

Methods

equals

public boolean **equals** (Object *obj*)

getLang

public String **getLang** ()

getUserName

public String **getUserName** ( )

hashCode

public int **hashCode** ( )

isSecurityLaunch

public boolean **isSecurityLaunch** ( )

toString

public String **toString** ( )

12.101 org.motechproject.server.web.helper

12.101.1 Header

public final class **Header**

Methods

generateHeader

public static String **generateHeader** (Bundle *bundle*)

12.101.2 Header.ElementOrder

public static class **ElementOrder**

Methods

getAfter

public String **getAfter** ( )

getBefore

public String **getBefore** ( )

getOrder

public String **getOrder** ( )

getPath

public String **getPath** ( )

setAfter

public void **setAfter** (String *after*)

setBefore

public void **setBefore** (String *before*)

setOrder

public void **setOrder** (String *order*)

setPath

public void **setPath** (String *path*)

12.101.3 Header.HeaderOrder

public static class **HeaderOrder**

Methods

getCss

public List<ElementOrder> **getCss** ( )

getJs

public List<ElementOrder> **getJs** ( )

getLib

public List<ElementOrder> **getLib** ( )

setCss

public void **setCss** (List<ElementOrder> *css*)

setJs

public void **setJs** (List<ElementOrder> *js*)

setLib

public void **setLib** (List<ElementOrder> *lib*)

12.101.4 MenuBuilder

public class **MenuBuilder**

Helper class for building the modules menu view(left-hand side nav). Modules to display are retrieved from the `UIFrameworkService`. Filters entries based on user permissions.

See also: `UIFrameworkService`

Methods

buildMenu

public ModuleMenu **buildMenu** (String *username*)

Builds the menu for the given user. Modules are retrieved from `UIFrameworkService` and filtered based on permissions.

Parameters

- **username** – username of the user for which the menu should be built.

Returns the built menu object.

12.101.5 SuggestionHelper

public class **SuggestionHelper**

Helper class for creating UI suggestions for the user. Checks default locations for available services.

Fields

DEFAULT_ACTIVEMQ_URL

public static final String **DEFAULT_ACTIVEMQ_URL**

Methods

suggestActivemqUrl

public String **suggestActivemqUrl** ()

Suggests the ActiveMQ url.

Returns The suggested url, or an empty string if the instance is not found.

12.102 org.motechproject.server.web.validator

12.102.1 AbstractValidator

public interface **AbstractValidator**

Basic interface which startup settings validators implement

Methods

validate

void **validate** (StartupForm *target*, List<String> *errors*, ConfigSource *configSource*)

12.102.2 OpenIdUserValidator

public class **OpenIdUserValidator** implements AbstractValidator

Validates presence of OpenId related field values Also validates provider URL

Constructors

OpenIdUserValidator

public **OpenIdUserValidator** (UrlValidator *urlValidator*)

Methods

validate

public void **validate** (StartupForm *target*, List<String> *errors*, ConfigSource *configSource*)

12.102.3 PersistedUserValidator

public class **PersistedUserValidator** implements AbstractValidator

Validates presence of admin registration fields. Checks existence of user with identical name Checks existence of user with identical email Checks that password and confirmed password field are same.

Constructors

PersistedUserValidator

public **PersistedUserValidator** (MotechUserService *userService*)

Methods

validate

public void **validate** (StartupForm *target*, List<String> *errors*, ConfigSource *configSource*)

12.102.4 RequiredFieldValidator

public class **RequiredFieldValidator** implements AbstractValidator

Generic validator class that validates presence of a given field

Fields

ERROR_REQUIRED

public static final String **ERROR_REQUIRED**

Constructors

RequiredFieldValidator

public **RequiredFieldValidator** (String *fieldName*, String *fieldValue*)

Methods

equals

public boolean **equals** (Object *o*)

hashCode

public int **hashCode** ()

validate

public void **validate** (StartupForm *target*, List<String> *errors*, ConfigSource *configSource*)

12.102.5 ResetFormValidator

public class **ResetFormValidator**

Methods

validate

public List<String> **validate** (ResetForm *target*)

12.102.6 StartupFormValidator

public class **StartupFormValidator**

StartupFormValidator validates user information during registration process

Constructors

StartupFormValidator

public **StartupFormValidator** ( )

Methods

add

public void **add** (AbstractValidator *validator*)

getValidators

public List<AbstractValidator> **getValidators** ( )

validate

public List<String> **validate** (StartupForm *target*, ConfigSource *configSource*)

12.102.7 StartupFormValidatorFactory

public class **StartupFormValidatorFactory**

Factory to create startUpFormValidator with requisite validators. If Admin User exists,the admin user is not created so the relevant validators are not added.

Methods

getStartupFormValidator

public StartupFormValidator **getStartupFormValidator** (StartupForm *startupSettings*, MotechUserService *userService*)

12.102.8 UserRegistrationValidator

public class **UserRegistrationValidator** implements AbstractValidator

Validator to validate user registration details. Delegates to either @OpenIdUserValidator or @UserRegistrationValidator depending on login mode preference.

Constructors

UserRegistrationValidator

public **UserRegistrationValidator** (PersistedUserValidator *persistedUserValidator*, OpenIdUserValidator *openIdUserValidator*)

Methods

validate

public void **validate** (StartupForm *target*, List<String> *errors*, ConfigSource *configSource*)

12.102.9 ValidationUtils

public final class **ValidationUtils**

Validation utils that consists of common validations that can be used across multiple controllers.

Methods

validateEmptyOrWhitespace

public static void **validateEmptyOrWhitespace** (Errors *errors*, String *errorMessageFormat*, String... *fields*)

12.103 org.motechproject.tasks.annotations

12.103.1 TaskAction

public @interface **TaskAction**

Marks methods that should be treated as task actions. Methods marked with this annotation must be placed in a class annotated with `@TaskChannel` annotation.

12.103.2 TaskActionParam

public @interface **TaskActionParam**

Marks method parameter to be treated as action parameter.

Each parameter in the given method has to have this annotation otherwise it will be a problem with the proper execution of the channel action.

See also: `TaskAction`, `TaskChannel`, `TaskAnnotationBeanPostProcessor`

12.103.3 TaskAnnotationBeanPostProcessor

public class **TaskAnnotationBeanPostProcessor** implements BeanPostProcessor

Factory class which is looking for classes with `TaskChannel` annotation to add them to the channel definition as channel action.

See also: `TaskAction`, `TaskActionParam`, `TaskChannel`

Constructors

TaskAnnotationBeanPostProcessor

public **TaskAnnotationBeanPostProcessor** (BundleContext *bundleContext*, ChannelService *channelService*)

Class constructor.

Parameters

- **bundleContext** – the bundle context, not null
- **channelService** – the channel service, not null

Methods

postProcessAfterInitialization

public Object **postProcessAfterInitialization** (Object *bean*, String *beanName*)

postProcessBeforeInitialization

public Object **postProcessBeforeInitialization** (Object *bean*, String *beanName*)

processAnnotations

public void **processAnnotations** (ApplicationContext *applicationContext*)

Searches through the given application context and processes annotations used by task module.

Parameters

- **applicationContext** – the context of the application, not null

12.103.4 TaskChannel

public @interface **TaskChannel**

Marks classes destined to represent module task channels. Methods annotated with `@TaskAction` annotation should be placed in class annotated with this annotation.

12.104 org.motechproject.tasks.contract

12.104.1 ActionEventRequest

public class **ActionEventRequest**

Service layer object denoting a `org.motechproject.tasks.domain.ActionEvent`. It is a part of the `org.motechproject.tasks.contract.ChannelRequest` and is used by `org.motechproject.tasks.service.ChannelService` for adding new or updating already existent action events.

Constructors

ActionEventRequest

public **ActionEventRequest** ()

Constructor.

ActionEventRequest

public **ActionEventRequest** (String *name*, String *displayName*, String *subject*, String *description*, String *serviceInterface*, String *serviceMethod*, String *serviceMethodCallManner*, SortedSet<ActionParameterRequest> *actionParameters*)

Constructor.

Parameters

- **name** – the event name
- **displayName** – the event display name
- **subject** – the event subject
- **description** – the event description
- **serviceInterface** – the event service interface
- **serviceMethod** – the event service method
- **serviceMethodCallManner** – the event service method call manner, for supported values check {@see org.motechproject.tasks.domain.MethodCallManner}
- **actionParameters** – the action parameters

Methods

addParameter

public void **addParameter** (ActionParameterRequest *parameter*, boolean *changeOrder*)

Adds the given parameter request to the list of stored parameter requests.

Parameters

- **parameter** – the action parameter request
- **changeOrder** – defines if order of the given parameter should continue numeration of the stored list

equals

public boolean **equals** (Object *obj*)

getActionParameters

public SortedSet<ActionParameterRequest> **getActionParameters** ()

Returns the action parameters.

Returns the action parameters

getDescription

public String **getDescription** ()

Returns the description of the action event.

Returns the action event description

getDisplayName

public String **getDisplayName** ()

Returns the display name of the action event.

Returns the action event display name

getName

public String **getName** ()

Returns the name of the action event.

Returns the action event name

getServiceInterface

public String **getServiceInterface** ()

Returns the service interface of the action event.

Returns the action event service interface

getServiceMethod

public String **getServiceMethod** ()

Returns the service method of the action event.

Returns the action event service method

getServiceMethodCallManner

public String **getServiceMethodCallManner**()

Returns the service method call manner of the action event.

Returns the action event service method call manner

getSubject

public String **getSubject**()

Returns the subject of the action event.

Returns the action event subject

hasService

public boolean **hasService**()

Checks if this action event request has service interface and method specified.

Returns true if action has service interface and method specified, false otherwise

hasSubject

public boolean **hasSubject**()

Checks if this action event request has subject.

Returns true if action event has subject, false otherwise

hashCode

public int **hashCode**()

isValid

public boolean **isValid**()

Checks if this action event request has subject or service defined.

Returns true if this has subject or service set, false otherwise

toString

public String **toString**()

12.104.2 ActionEventRequestBuilder

public class **ActionEventRequestBuilder**

The `ActionEventRequestBuilder` class provides methods for constructing action event requests.

See also: `org.motechproject.tasks.contract.ActionEventRequest`

Methods

createActionEventRequest

public ActionEventRequest **`createActionEventRequest`**()

Builds an object of the `ActionEventRequest` class.

Returns the created instance

setActionParameters

public ActionEventRequestBuilder **`setActionParameters`**(SortedSet<ActionParameterRequest> *actionParameters*)

Sets the parameters of the action event to be built.

Parameters

- **actionParameters** – the action event parameter

Returns the reference to this object

setDescription

public ActionEventRequestBuilder **`setDescription`**(String *description*)

Sets the description of the action event to be built.

Parameters

- **description** – the action event description

Returns the reference to this object

setDisplayName

public ActionEventRequestBuilder **`setDisplayName`**(String *displayName*)

Sets the display name of the action event to be built.

Parameters

- **displayName** – the action event display name

Returns the reference to this object

setName

public ActionEventRequestBuilder **`setName`**(String *name*)

Sets the name of the action event to be built.

Parameters

- **name** – the action event name

Returns the reference to this object

setServiceInterface

public ActionEventRequestBuilder **setServiceInterface** (String *serviceInterface*)

Sets the service interface of the action event to be built.

Parameters

- **serviceInterface** – the action event service interface

Returns the reference to this object

setServiceMethod

public ActionEventRequestBuilder **setServiceMethod** (String *serviceMethod*)

Sets the service method of the action event to be built.

Parameters

- **serviceMethod** – the action event service method

Returns the reference to this object

setServiceMethodCallManner

public ActionEventRequestBuilder **setServiceMethodCallManner** (String *serviceMethodCallManner*)

Sets the service method call manner of the action event to be built.

Parameters

- **serviceMethodCallManner** – the action event service method call manner, for supported values see {@see org.motechproject.tasks.domain.MethodCallManner}

Returns the reference to this object

setSubject

public ActionEventRequestBuilder **setSubject** (String *subject*)

Sets the subject of the action event to be built.

Parameters

- **subject** – the action event subject

Returns the reference to this object

12.104.3 ActionParameterRequest

public class **ActionParameterRequest** implements Comparable<ActionParameterRequest>

Object representation of a parameter in the channel action request definition.

See also: `ActionEventRequest`

Constructors

ActionParameterRequest

public **ActionParameterRequest** ()
Constructor.

ActionParameterRequest

public **ActionParameterRequest** (Integer *order*, String *key*, String *value*, String *displayName*, String *type*, boolean *required*, boolean *hidden*, SortedSet<String> *options*)
Constructor.

Parameters

- **order** – the parameter order
- **key** – the parameter key
- **value** – the parameter value
- **displayName** – the parameter display name
- **type** – the parameter type
- **required** – defines if the parameter is required
- **hidden** – defines if the parameter is hidden on the UI
- **options** – the parameter options for select parameter type

Methods

compareTo

public int **compareTo** (ActionParameterRequest *o*)

equals

public boolean **equals** (Object *obj*)

getDisplayName

public String **getDisplayName** ()
Returns the display name of the parameter.

Returns the parameter display name

getKey

public String **getKey** ()
Returns the key of the parameter.

Returns the parameter key

getOptions

public SortedSet<String> **getOptions**()

Returns the options of the parameter.

Returns options of the parameter order

getOrder

public Integer **getOrder**()

Returns the order of the parameter.

Returns the parameter order

getType

public String **getType**()

Returns the type of the parameter.

Returns the parameter type

getValue

public String **getValue**()

Returns the value of the parameter.

Returns the parameter value

hashCode

public int **hashCode**()

isHidden

public boolean **isHidden**()

Returns whether this action parameter should be hidden on the UI.

Returns true if this action parameter should be hidden on the UI, false otherwise

isRequired

public boolean **isRequired**()

Returns whether this action parameter is required.

Returns true if this action parameter is required, false otherwise

setOptions

public void **setOptions** (SortedSet<String> *options*)

Sets the options of the parameter.

Parameters

- **options** – of the parameter order

setOrder

public void **setOrder** (int *order*)

Sets the order of the parameter.

Parameters

- **order** – the parameter order

toString

public String **toString** ()

12.104.4 ActionParameterRequestBuilder

public class **ActionParameterRequestBuilder**

The `ActionParameterRequestBuilder` class provides methods for constructing action parameter requests.

See also: `org.motechproject.tasks.contract.ActionParameterRequest`

Methods

createActionParameterRequest

public ActionParameterRequest **createActionParameterRequest** ()

Builds an object of the `ActionParameterRequest` class.

Returns the created instance

setDisplayName

public ActionParameterRequestBuilder **setDisplayName** (String *displayName*)

Sets the display name of the action parameter to build the request for.

Parameters

- **displayName** – the action parameter display name

Returns the reference to this object

setHidden

public ActionParameterRequestBuilder **setHidden** (boolean *hidden*)

Defines whether the action parameter, that request will be build for, should be hidden on the UI.

Parameters

- **hidden** – defines if the action parameter should be hidden on the UI

Returns the reference to this object

setKey

public ActionParameterRequestBuilder **setKey** (String *key*)

Sets the key of the action parameter to build the request for.

Parameters

- **key** – the action parameter key

Returns the reference to this object

setOptions

public ActionParameterRequestBuilder **setOptions** (SortedSet<String> *options*)

Sets the options of the action parameter to build the request for.

Parameters

- **options** – the action parameter options

Returns the reference to this object

setOrder

public ActionParameterRequestBuilder **setOrder** (Integer *order*)

Sets the order of the action parameter to build the request for.

Parameters

- **order** – the action parameter order

Returns the reference to this object

setRequired

public ActionParameterRequestBuilder **setRequired** (boolean *required*)

Defines whether the action parameter, that request will be build for, should be required.

Parameters

- **required** – defines if the action parameter should be required

Returns the reference to this object

setType

public ActionParameterRequestBuilder **setType** (String *type*)

Sets the type of the action parameter to build the request for.

Parameters

- **type** – the action parameter type

Returns the reference to this object

setValue

public ActionParameterRequestBuilder **setValue** (String *value*)

Sets the value of the action parameter to build the request for.

Parameters

- **value** – the action parameter value

Returns the reference to this object

12.104.5 ChannelRequest

public class **ChannelRequest**

Service layer object denoting a `org.motechproject.tasks.domain.Channel`. Used by `org.motechproject.tasks.service.ChannelService`. It is used for registering new and updating already existent channels.

Constructors

ChannelRequest

public **ChannelRequest** (String *displayName*, String *moduleName*, String *moduleVersion*, String *description*, List<TriggerEventRequest> *triggerTaskEvents*, List<ActionEventRequest> *actionTaskEvents*)

Constructor.

Parameters

- **displayName** – the channel display name
- **moduleName** – the module symbolic name
- **moduleVersion** – the module version
- **description** – the channel description
- **triggerTaskEvents** – the triggers definitions
- **actionTaskEvents** – the actions definitions

Methods

equals

public boolean **equals** (Object *obj*)

getActionTaskEvents

public List<ActionEventRequest> **getActionTaskEvents** ()

Returns the task action events for this channel.

Returns the task action events

getDescription

public String **getDescription** ()

Returns the description of the channel.

Returns the channel description

getDisplayName

public String **getDisplayName** ()

Returns the display name of the channel.

Returns the channel display name

getModuleName

public String **getModuleName** ()

Returns the symbolic name of the module.

Returns the module symbolic name

getModuleVersion

public String **getModuleVersion** ()

Returns the version of the module.

Returns the module version

getTriggerTaskEvents

public List<TriggerEventRequest> **getTriggerTaskEvents** ()

Returns the task trigger events for this channel.

Returns the task trigger events

hashCode

public int **hashCode** ()

setModuleName

public void **setModuleName** (String *moduleName*)
Sets the module name of this channel.

Parameters

- **moduleName** – the channel module name

setModuleVersion

public void **setModuleVersion** (String *moduleVersion*)
Sets the module version of this channel.

Parameters

- **moduleVersion** – the module version

toString

public String **toString** ()

12.104.6 EventParameterRequest

public class **EventParameterRequest**
Service layer object denoting a `org.motechproject.tasks.domain.EventParameter`. It is a part of the `org.motechproject.tasks.contract.TriggerEventRequest` and is used by `org.motechproject.tasks.service.ChannelService` for adding new or updating already existent trigger event parameters.

Constructors

EventParameterRequest

public **EventParameterRequest** (String *displayName*, String *eventKey*, String *type*)
Constructor.

Parameters

- **displayName** – the event parameter display name
- **eventKey** – the event key
- **type** – the event parameter type

EventParameterRequest

public **EventParameterRequest** (String *displayName*, String *eventKey*)
Constructor.

Parameters

- **displayName** – the event parameter display name
- **eventKey** – the event key

Methods

equals

public boolean **equals** (Object *obj*)

getDisplayName

public String **getDisplayName** ()

Returns the display name of the event.

Returns the event parameter display name

getEventKey

public String **getEventKey** ()

Returns the key of the event.

Returns the event key

getType

public String **getType** ()

Returns the type of the event.

Returns the event parameter type

hashCode

public int **hashCode** ()

toString

public String **toString** ()

12.104.7 TriggerEventRequest

public class **TriggerEventRequest**

Service layer object denoting a `org.motechproject.tasks.domain.TriggerEvent`. It is a part of the `org.motechproject.tasks.contract.ChannelRequest` and is used by `org.motechproject.tasks.service.ChannelService` for adding new or updating already existent trigger events.

Constructors

TriggerEventRequest

public **TriggerEventRequest** (String *displayName*, String *subject*, String *description*, List<EventParameterRequest> *eventParameters*)

Constructor. The given subject will be used as the actual subject the listener will listen for.

Parameters

- **displayName** – the trigger event display name
- **subject** – the event subject
- **description** – the event description
- **eventParameters** – the trigger event parameters

TriggerEventRequest

public **TriggerEventRequest** (String *displayName*, String *subject*, String *description*, List<EventParameterRequest> *eventParameters*, String *triggerListenerSubject*)

Constructor. The given `triggerListenerSubject` will be used as the actual subject the listener will listen for. If it is not specified subject will be used instead.

Parameters

- **displayName** – the trigger event display name
- **subject** – the event subject
- **description** – the event description
- **eventParameters** – the trigger event parameters
- **triggerListenerSubject** – the listener event subject

Methods

equals

public boolean **equals** (Object *obj*)

getDescription

public String **getDescription** ()

Returns the description of the trigger event.

Returns the trigger event description

getDisplayName

public String **getDisplayName** ()

Returns the display name of the trigger event.

Returns the trigger event display name

getEventParameters

public List<EventParameterRequest> **getEventParameters** ()

Returns the parameters of the trigger event.

Returns the trigger event parameters

getSubject

public String **getSubject** ()

Returns the subject of the trigger event.

Returns the trigger event subject

getTriggerListenerSubject

public String **getTriggerListenerSubject** ()

Returns the trigger listener subject of the trigger event.

Returns the trigger event listener subject

hashCode

public int **hashCode** ()

toString

public String **toString** ()

12.105 org.motechproject.tasks.domain

12.105.1 ActionEvent

public class **ActionEvent** extends TaskEvent

Represents an action from a channel. An action is taken once a task is triggered. This class is the representation of the definition from the channel, not the representation of an usage within task. An action can be represented as an event, but also as a direct OSGi message(or both - a service call with the event acting as a fallback way of executing the action).

Constructors

ActionEvent

public **ActionEvent** ()

Constructor.

ActionEvent

public **ActionEvent** (String *name*, String *description*, String *displayName*, String *subject*, String *serviceInterface*, String *serviceMethod*, MethodCallManner *serviceMethodCallManner*, SortedSet<ActionParameter> *actionParameters*)

Constructor.

Parameters

- **name** – the action name
- **description** – the action description
- **displayName** – the action display name
- **subject** – the action event subject
- **serviceInterface** – the event service interface
- **serviceMethod** – the event service method
- **serviceMethodCallManner** – the event service call manner, for supported values see {@see MethodCallManner}
- **actionParameters** – the action parameters

Methods

accept

public boolean **accept** (TaskActionInformation *info*)

addParameter

public void **addParameter** (ActionParameter *parameter*, boolean *changeOrder*)

containsParameter

public boolean **containsParameter** (String *key*)

equals

public boolean **equals** (Object *obj*)

getActionParameters

public SortedSet<ActionParameter> **getActionParameters** ()

getServiceInterface

public String **getServiceInterface** ()

getServiceMethod

public String **getServiceMethod**()

getServiceMethodCallManner

public MethodCallManner **getServiceMethodCallManner**()

hasService

public boolean **hasService**()

hashCode

public int **hashCode**()

setActionParameters

public void **setActionParameters**(SortedSet<ActionParameter> *actionParameters*)

setServiceInterface

public void **setServiceInterface**(String *serviceInterface*)

setServiceMethod

public void **setServiceMethod**(String *serviceMethod*)

setServiceMethodCallManner

public void **setServiceMethodCallManner**(MethodCallManner *serviceMethodCallManner*)

toString

public String **toString**()

12.105.2 ActionEventBuilder

public class **ActionEventBuilder**

The `ActionEventBuilder` class provides methods for constructing action events.

See also: `org.motechproject.tasks.domain.ActionEvent`

Methods

createActionEvent

public ActionEvent **`createActionEvent`**()

Builds an object of the `ActionEvent` class.

Returns the created instance

fromActionEvent

public static ActionEventBuilder **`fromActionEvent`**(ActionEvent *actionEventRequest*)

Builds an object of the `ActionEventBuilder` class based on the passed `ActionEvent` instance.

Parameters

- **actionEventRequest** – the action event, not null

Returns the instance of the `ActionEventBuilder` class ready to build instance of the `ActionEvent` class

fromActionEventRequest

public static ActionEventBuilder **`fromActionEventRequest`**(ActionEventRequest *actionEventRequest*)

Builds an object of the `ActionEventBuilder` class based on the passed `ActionEventRequest` instance.

Parameters

- **actionEventRequest** – the action event request, not null

Returns the instance of the `ActionEventBuilder` class ready to build instance of the `ActionEvent` class

setActionParameters

public ActionEventBuilder **`setActionParameters`**(SortedSet<ActionParameter> *actionParameters*)

setDescription

public ActionEventBuilder **`setDescription`**(String *description*)

setDisplayName

public ActionEventBuilder **`setDisplayName`**(String *displayName*)

setName

public ActionEventBuilder **`setName`**(String *name*)

setServiceInterface

public ActionEventBuilder **setServiceInterface** (String *serviceInterface*)

setServiceMethod

public ActionEventBuilder **setServiceMethod** (String *serviceMethod*)

setServiceMethodCallManner

public ActionEventBuilder **setServiceMethodCallManner** (MethodCallManner *serviceMethodCallManner*)

setSubject

public ActionEventBuilder **setSubject** (String *subject*)

12.105.3 ActionParameter

public class **ActionParameter** extends Parameter implements Comparable<ActionParameter>

Represents a single parameter of an action in the channel definition.

Constructors

ActionParameter

public **ActionParameter** ()

Constructor.

ActionParameter

public **ActionParameter** (String *displayName*, ParameterType *type*, Integer *order*, String *key*, String *value*, Boolean *required*, Boolean *hidden*, SortedSet<String> *options*)

Constructor.

Parameters

- **displayName** – the parameter display name
- **type** – the parameter type
- **order** – the parameter order
- **key** – the parameter key
- **value** – the parameter value
- **required** – defines whether the parameter is required
- **hidden** – defines whether the parameter is hidden
- **options** – the parameter options for select parameter type

Methods

compareTo

public int **compareTo** (ActionParameter *o*)

equals

public boolean **equals** (Object *obj*)

getKey

public String **getKey** ()

getOptions

public SortedSet<String> **getOptions** ()

getOrder

public Integer **getOrder** ()

getValue

public String **getValue** ()

hashCode

public int **hashCode** ()

isHidden

public boolean **isHidden** ()

isRequired

public boolean **isRequired** ()

setHidden

public void **setHidden** (Boolean *hidden*)

setKey

public void **setKey** (String *key*)

setOptions

public void **setOptions** (SortedSet<String> *options*)

setOrder

public void **setOrder** (Integer *order*)

setRequired

public void **setRequired** (Boolean *required*)

setValue

public void **setValue** (String *value*)

toString

public String **toString** ()

12.105.4 ActionParameterBuilder

public class **ActionParameterBuilder**

The `ActionParameterBuilder` class provides methods for constructing action parameters.

See also: `org.motechproject.tasks.domain.ActionParameter`

Methods

createActionParameter

public ActionParameter **createActionParameter** ()

Builds an object of the `ActionParameter` class.

Returns the created instance

fromActionParameter

public static ActionParameterBuilder **fromActionParameter** (ActionParameter *actionParameter*)

Builds an object of the `ActionParameterBuilder` class based on the passed `ActionParameter` instance.

Parameters

- **actionParameter** – the action parameter request, not null

Returns a builder ready to build a new instance of the `ActionParameter` class

fromActionParameterRequest

public static ActionParameterBuilder **fromActionParameterRequest** (ActionParameterRequest *actionParameterRequest*)

Builds an object of the `ActionParameterBuilder` class based on the passed `ActionParameterRequest` instance.

Parameters

- **actionParameterRequest** – the action parameter request, not null

Returns a builder ready to build a new instance of the `ActionParameter` class

setDisplayName

public ActionParameterBuilder **setDisplayName** (String *displayName*)

setHidden

public ActionParameterBuilder **setHidden** (boolean *hidden*)

setKey

public ActionParameterBuilder **setKey** (String *key*)

setOptions

public ActionParameterBuilder **setOptions** (SortedSet<String> *options*)

setOrder

public ActionParameterBuilder **setOrder** (Integer *order*)

setRequired

public ActionParameterBuilder **setRequired** (boolean *required*)

setType

public ActionParameterBuilder **setType** (ParameterType *type*)

setValue

public ActionParameterBuilder **setValue** (String *value*)

12.105.5 Channel

public class **Channel**

Represents a single task channel. Channel contains the list of triggers from the given module and the list of actions that can be taken by that module.

Constructors

Channel

public **Channel** ( )

Constructor.

Channel

public **Channel** (String *displayName*, String *moduleName*, String *moduleVersion*)

Constructor.

Parameters

- **displayName** – the channel display name
- **moduleName** – the channel module name
- **moduleVersion** – the module version

Channel

public **Channel** (String *displayName*, String *moduleName*, String *moduleVersion*, String *description*, List<TriggerEvent> *triggerTaskEvents*, List<ActionEvent> *actionTaskEvents*)

Constructor.

Parameters

- **displayName** – the channel display name
- **moduleName** – the channel module name
- **moduleVersion** – the module version
- **description** – the channel description
- **triggerTaskEvents** – the list of events for provided triggers
- **actionTaskEvents** – the list of events for provided actions

Channel

public **Channel** (ChannelRequest *channelRequest*)

Constructor.

Parameters

- **channelRequest** – the channel request, not null

Methods

addActionTaskEvent

public void **addActionTaskEvent** (ActionEvent *actionEvent*)

containsAction

public boolean **containsAction** (TaskActionInformation *actionInformation*)

containsTrigger

public boolean **containsTrigger** (TaskTriggerInformation *triggerInformation*)

equals

public boolean **equals** (Object *obj*)

getAction

public ActionEvent **getAction** (TaskActionInformation *actionInformation*)

getActionTaskEvents

public List<ActionEvent> **getActionTaskEvents** ()

getDescription

public String **getDescription** ()

getDisplayName

public String **getDisplayName** ()

getModuleName

public String **getModuleName** ()

getModuleVersion

public String **getModuleVersion** ()

getTrigger

public TriggerEvent **getTrigger** (TaskTriggerInformation *triggerInformation*)

getTriggerTaskEvents

public List<TriggerEvent> **getTriggerTaskEvents** ()

hashCode

public int **hashCode** ()

setActionTaskEvents

public void **setActionTaskEvents** (List<ActionEvent> *actionTaskEvents*)

setDescription

public void **setDescription** (String *description*)

setDisplayName

public void **setDisplayName** (String *displayName*)

setModuleName

public void **setModuleName** (String *moduleName*)

setModuleVersion

public void **setModuleVersion** (String *moduleVersion*)

setTriggerTaskEvents

public void **setTriggerTaskEvents** (List<TriggerEvent> *triggerTaskEvents*)

toString

public String **toString** ()

12.105.6 ChannelRegisterEvent

public class **ChannelRegisterEvent**

A wrapper over a `MotechEvent` with subject {@value EventSubjects#CHANNEL_REGISTER_SUBJECT}. Raised when a channel is registered with the tasks module

Constructors

ChannelRegisterEvent

public **ChannelRegisterEvent** (String *moduleName*)
Constructor.

Parameters

- **moduleName** – the module name

ChannelRegisterEvent

public **ChannelRegisterEvent** (MotechEvent *motechEvent*)
Constructor.

Parameters

- **motechEvent** – the motech event

Methods

getChannelModuleName

public String **getChannelModuleName** ()

toMotechEvent

public MotechEvent **toMotechEvent** ()
Convert this event to the instance of `MotechEvent`.

Returns the instance of `MotechEvent`

12.105.7 DataSource

public class **DataSource** extends TaskConfigStep
Represents a single data source object used by a task. This class is part of the task itself and does not describe the data source itself. This object translates to retrieving a data source object during task execution.

Constructors

DataSource

public **DataSource** ()
Constructor.

DataSource

public **DataSource** (Long *providerId*, Long *objectId*, String *type*, String *name*, List<Lookup> *lookup*, boolean *failIfDataNotFound*)

Constructor.

Parameters

- **providerId** – the provider ID
- **objectId** – the object ID
- **type** – the data source object
- **name** – the data source name
- **lookup** – the lookup name
- **failIfDataNotFound** – defines if task should fail if no data was found

DataSource

public **DataSource** (String *providerName*, Long *providerId*, Long *objectId*, String *type*, String *name*, List<Lookup> *lookup*, boolean *failIfDataNotFound*)

Constructor.

Parameters

- **providerName** – the provider name
- **providerId** – the provider ID
- **objectId** – the object ID
- **type** – the data source type
- **name** – the data source name
- **lookup** – the lookup name
- **failIfDataNotFound** – defines if task should fail if no data was found

Methods

equals

public boolean **equals** (Object *obj*)

getLookup

public List<Lookup> **getLookup** ()

getName

public String **getName** ()

getObjectId

public Long **getObjectId**()

getProviderId

public Long **getProviderId**()

getProviderName

public String **getProviderName**()

getType

public String **getType**()

hashCode

public int **hashCode**()

isFailIfDataNotFound

public boolean **isFailIfDataNotFound**()

objectEquals

public boolean **objectEquals**(Long *providerId*, Long *objectId*, String *type*)

setFailIfDataNotFound

public void **setFailIfDataNotFound**(boolean *failIfDataNotFound*)

setLookup

public void **setLookup**(Object *lookup*)

setName

public void **setName**(String *name*)

setObjectId

public void **setObjectId**(Long *objectId*)

setProviderId

public void **setProviderId** (Long *providerId*)

setProviderName

public void **setProviderName** (String *providerName*)

setType

public void **setType** (String *type*)

toString

public String **toString** ()

12.105.8 EventParameter

public class **EventParameter** extends Parameter

Represents a parameter of a trigger event. These parameters can be dragged and dropped within tasks. This class is part of the channel model.

Constructors

EventParameter

public **EventParameter** ()

Constructor.

EventParameter

public **EventParameter** (String *displayName*, String *eventKey*)

Constructor.

Parameters

- **displayName** – the parameter display name
- **eventKey** – the event key

EventParameter

public **EventParameter** (String *displayName*, String *eventKey*, ParameterType *type*)

Constructor.

Parameters

- **displayName** – the parameter display name
- **eventKey** – the event key

- **type** – the parameter type

EventParameter

public **`EventParameter`** (EventParameterRequest *eventParameterRequest*)
Constructor.

Parameters

- **eventParameterRequest** – the request for event parameter, not null

Methods

equals

public boolean **`equals`** (Object *obj*)

getEventKey

public String **`getEventKey`** ()

hashCode

public int **`hashCode`** ()

setEventKey

public void **`setEventKey`** (String *eventKey*)

toString

public String **`toString`** ()

12.105.9 FieldParameter

public class **`FieldParameter`** extends Parameter
Represents a single field of the `TaskDataProviderObject` that is part of the `TaskDataProvider`.

Constructors

FieldParameter

public **`FieldParameter`** ()
Constructor.

FieldParameter

public **FieldParameter** (String *displayName*, String *fieldKey*)

Constructor.

Parameters

- **displayName** – the parameter display name
- **fieldKey** – the field key

FieldParameter

public **FieldParameter** (String *displayName*, String *fieldKey*, ParameterType *type*)

Constructor.

Parameters

- **displayName** – the parameter display name
- **fieldKey** – the field key
- **type** – the parameter type

Methods

equals

public boolean **equals** (Object *obj*)

getFieldKey

public String **getFieldKey** ()

hashCode

public int **hashCode** ()

setFieldKey

public void **setFieldKey** (String *fieldKey*)

toString

public String **toString** ()

12.105.10 Filter

public class **Filter** implements Serializable

Represents a single filter. A filter is a part of the `FilterSet` and represents a single condition that task must meet before being executed. If that condition is not met the task execution will be stopped. It is an optional part of a task.

Constructors

Filter

public **Filter** ()

Constructor.

Filter

public **Filter** (EventParameter *eventParameter*, boolean *negationOperator*, String *operator*, String *expression*)

Constructor.

Parameters

- **eventParameter** – the event parameter
- **negationOperator** – defines if the represented operator should be negated
- **operator** – the filter operator
- **expression** – the filter operator

Filter

public **Filter** (String *displayName*, String *key*, ParameterType *type*, boolean *negationOperator*, String *operator*, String *expression*)

Constructor.

Parameters

- **displayName** – the filter display name
- **key** – the filter key
- **type** – the filter type
- **negationOperator** – defines if the represented operator should be negated
- **operator** – the filter operator
- **expression** – the filter exception

Methods

equals

public boolean **equals** (Object *obj*)

getDisplayName

public String **getDisplayName**()

getExpression

public String **getExpression**()

getKey

public String **getKey**()

getOperator

public String **getOperator**()

getType

public ParameterType **getType**()

hashCode

public int **hashCode**()

isNegationOperator

public boolean **isNegationOperator**()

setDisplayName

public void **setDisplayName**(String *displayName*)

setExpression

public void **setExpression**(String *expression*)

setKey

public void **setKey**(String *key*)

setNegationOperator

public void **setNegationOperator**(boolean *negationOperator*)

setOperator

public void **setOperator** (String *operator*)

setType

public void **setType** (ParameterType *type*)

toString

public String **toString** ()

12.105.11 FilterSet

public class **FilterSet** extends TaskConfigStep

Represents a set of `Filter`s. Those are conditions that task must meet before being executed. If the conditions are not met the task execution will be stopped. Joining those conditions can be done by using logical "and" or logical "or" as an operator and can be set by `setOperator(LogicalOperator)` method.

Constructors

FilterSet

public **FilterSet** ()

Constructor.

FilterSet

public **FilterSet** (List<Filter> *filters*)

Constructor. The operator will be set to "AND".

Parameters

- **filters** – the filters

FilterSet

public **FilterSet** (List<Filter> *filters*, LogicalOperator *operator*)

Constructor.

Parameters

- **filters** – the filters
- **operator** – the operator, can be "AND" or "OR

Methods

addFilter

public void **addFilter** (Filter *filter*)

equals

public boolean **equals** (Object *obj*)

getFilters

public List<Filter> **getFilters** ()

getOperator

public LogicalOperator **getOperator** ()

hashCode

public int **hashCode** ()

setFilters

public void **setFilters** (List<Filter> *filters*)

setOperator

public void **setOperator** (LogicalOperator *operator*)

toString

public String **toString** ()

12.105.12 KeyInformation

public final class **KeyInformation**

Object representation of dragged field from trigger or data source.

This class represents a single dragged field from trigger or data source. This class does not expose a public constructor. You have to use `parse(String)` method if you want to parse single field or `parseAll(String)` if you want ot get all fields from a given string.

Fields

ADDITIONAL_DATA_PREFIX

public static final String **ADDITIONAL_DATA_PREFIX**
Prefix which is used for data source fields.

TRIGGER_PREFIX

public static final String **TRIGGER_PREFIX**
Prefix which is used for trigger fields.

Methods

equals

public boolean **equals** (Object *obj*)

fromAdditionalData

public boolean **fromAdditionalData** ()
Check if the field is from the data source.

Returns true if the field is from the data source otherwise false

fromTrigger

public boolean **fromTrigger** ()
Check if the field is from the trigger.

Returns true if the field is from the trigger otherwise false

getDataProviderId

public Long **getDataProviderId** ()

getKey

public String **getKey** ()

getManipulations

public List<String> **getManipulations** ()
Get manipulations assigned to the field.

Returns list of manipulations

getObjectId

public Long **getObjectId**()

getObjectType

public String **getObjectType**()

getOriginalKey

public String **getOriginalKey**()

Get original representation of the dragged field.

Returns string representation of the field

getPrefix

public String **getPrefix**()

hasManipulations

public boolean **hasManipulations**()

Check if the field has any manipulations.

Returns true if the field has manipulations otherwise false

hashCode

public int **hashCode**()

parse

public static KeyInformation **parse**(String *input*)

Parse given string to instance of `KeyInformation`.

This method should be used to convert string representation of dragged field to instance of `KeyInformation`.

Argument has adhere to one of the following format:

- trigger field format: **trigger.eventKey**
- data source format: **ad.dataProviderId.objectType#objectId.fieldKey**

Argument can also contain list of manipulation which should be executed on field before it will be used by `org.motechproject.tasks.service.TaskTriggerHandler` class. Manipulations should be connected together by the **?** character.

Example of input argument:

- ad.279f5fdf60700d9717270b1ae3011eb1.CaseInfo#0.fieldValues.phu_id
- trigger.message?format(Ala,cat)?capitalize

Parameters

- **input** – string representation of a dragged field from trigger or data source

Throws

- **IllegalArgumentException** – exception is thrown if format for data source field is incorrect or if the dragged field is not from trigger or data source.

Returns Object representation of a dragged field

parseAll

public static List<KeyInformation> **parseAll** (String *input*)

Find all fields from given input and convert them to the instance of `KeyInformation`.

This method should be used to find and convert all of string representation of the field from trigger and/or data sources. Fields in input have to adhere to one of the following formats:

•trigger field format: **{{trigger.eventKey}}**

•data source format: **{{ad.dataProviderId.objectType#objectId.fieldKey}}**

To find fields in the input argument this method uses regular expression. When field is found it is converted to an instance of `KeyInformation` by using the `parse(String)` method.

Fields are found by the following regular expression: **{{((.*?))(}})(?![^(]*)**. The expression searches for strings that start with *{{* and end with *}}* and are not within *(* and *)*. Because of manipulations which contain additional data in *(...)* needed to execute manipulation on the field (e.g.: join needs to have the join character) and the text in *(...)* can be another string representation of the dragged field, the expression has to check if the field has this kind of manipulation.

Example of input argument:

•{{trigger.message?format(Ala,cat)?capitalize}}

•You get the following message: {{trigger.message}}

Parameters

- **input** – string with one or more string representation of dragged fields from trigger and/or data sources

Throws

- **IllegalArgumentException** – in the same situations as the `parse(String)` method.

Returns list of object representation of dragged fields.

12.105.13 LogicalOperator

public enum **LogicalOperator**

Enumerates logical operators that can be used on `Filter`s in the task `FilterSet`s.

Enum Constants

AND

public static final LogicalOperator **AND**

OR

public static final LogicalOperator **OR**

12.105.14 Lookup

public class **Lookup** implements Serializable

Represents a single lookup. Lookup is a method of retrieving an object from a data provider. It is a part of a task model.

Constructors

Lookup

public **Lookup** ( )

Constructor.

Lookup

public **Lookup** (String *field*, String *value*)

Constructor.

Parameters

- **field** – the field name
- **value** – the field value

Methods

equals

public boolean **equals** (Object *obj*)

getField

public String **getField** ( )

getValue

public String **getValue** ( )

hashCode

public int **hashCode** ( )

setField

public void **setField** (String *field*)

setValue

public void **setValue** (String *value*)

12.105.15 LookupFieldsParameter

public class **LookupFieldsParameter**

Represents a lookup fields. It is part of the `TaskDataProviderObject` and describes a single lookup with its display name and fields that are used by that lookup.

Constructors

LookupFieldsParameter

public **LookupFieldsParameter** ()

LookupFieldsParameter

public **LookupFieldsParameter** (String *displayName*, List<String> *fields*)

Methods

equals

public boolean **equals** (Object *obj*)

getDisplayName

public String **getDisplayName** ()

getFields

public List<String> **getFields** ()

hashCode

public int **hashCode** ()

setDisplayName

public void **setDisplayName** (String *displayName*)

setFields

public void **setFields** (List<String> *fields*)

toString

public String **toString** ()

12.105.16 ManipulationTarget

public enum **ManipulationTarget**
Defines the target of various manipulations used in a task for both triggers and data sources.

Enum Constants

ALL

public static final ManipulationTarget **ALL**

DATE

public static final ManipulationTarget **DATE**

STRING

public static final ManipulationTarget **STRING**

12.105.17 ManipulationType

public enum **ManipulationType**
Defines the type of various manipulations used in a task for both triggers and data sources.

Enum Constants

CAPITALIZE

public static final ManipulationType **CAPITALIZE**

DATETIME

public static final ManipulationType **DATETIME**

FORMAT

public static final ManipulationType **FORMAT**

JOIN

public static final ManipulationType **JOIN**

MINUSDAYS

public static final ManipulationType **MINUSDAYS**

MINUSHOURS

public static final ManipulationType **MINUSHOURS**

MINUSMINUTES

public static final ManipulationType **MINUSMINUTES**

PARSEDATE

public static final ManipulationType **PARSEDATE**

PLUSDAYS

public static final ManipulationType **PLUSDAYS**

PLUSHOURS

public static final ManipulationType **PLUSHOURS**

PLUSMINUTES

public static final ManipulationType **PLUSMINUTES**

SPLIT

public static final ManipulationType **SPLIT**

SUBSTRING

public static final ManipulationType **SUBSTRING**

TOLOWER

public static final ManipulationType **TOLOWER**

TOUPPER

public static final ManipulationType **TOUPPER**

UNKNOWN

public static final ManipulationType **UNKNOWN**

URLENCODE

public static final ManipulationType **URLENCODE**

12.105.18 MethodCallManner

public enum **MethodCallManner**

The `MethodCallManner` enumerates possible call manners of an `ActionEvent` service method. It also implies expected signature of this method.

See also: `org.motechproject.tasks.domain.ActionEvent`

Enum Constants

MAP

public static final MethodCallManner **MAP**

When using this call manner, the parameters are evaluated, casted to appropriate types and then wrapped with a Map in which keys corresponds to parameters names and values corresponds to parameters values. Map in this form gets passed to the service method.

NAMED_PARAMETERS

public static final MethodCallManner **NAMED_PARAMETERS**

When using this call manner, the parameters are evaluated, casted to appropriate types and then passed to the service method in specified order as a regular java method parameters.

12.105.19 OperatorType

public enum **OperatorType**

Object representation of available operators in filter definition.

Enum Constants

AFTER

public static final OperatorType **AFTER**

AFTER_NOW

public static final OperatorType **AFTER_NOW**

BEFORE

public static final OperatorType **BEFORE**

BEFORE_NOW

public static final OperatorType **BEFORE_NOW**

CONTAINS

public static final OperatorType **CONTAINS**

ENDSWITH

public static final OperatorType **ENDSWITH**

EQUALS

public static final OperatorType **EQUALS**

EQUALS_IGNORE_CASE

public static final OperatorType **EQUALS_IGNORE_CASE**

EQ_NUMBER

public static final OperatorType **EQ_NUMBER**

EXIST

public static final OperatorType **EXIST**

GT

public static final OperatorType **GT**

LESS_DAYS_FROM_NOW

public static final OperatorType **LESS_DAYS_FROM_NOW**

LESS_MONTHS_FROM_NOW

public static final OperatorType **LESS_MONTHS_FROM_NOW**

LT

public static final OperatorType **LT**

MORE_DAYS_FROM_NOW

public static final OperatorType **MORE_DAYS_FROM_NOW**

MORE_MONTHS_FROM_NOW

public static final OperatorType **MORE_MONTHS_FROM_NOW**

STARTSWITH

public static final OperatorType **STARTSWITH**

12.105.20 Parameter

public abstract class **Parameter** implements Serializable

Abstract class that stores common information about a single parameter. Serves as a base class for `FieldParameter`, `ActionParameter` and `EventParameter` classes. It is a part of the channel model.

Constructors

Parameter

protected **Parameter** ( )

Constructor.

Parameter

protected **Parameter** (String *displayName*, ParameterType *type*)

Constructor.

Parameters

- **displayName** – the parameter display name
- **type** – the parameter type

Methods

equals

public boolean **equals** (Object *obj*)

getDisplayName

public String **getDisplayName** ()

getType

public ParameterType **getType** ()

hashCode

public int **hashCode** ()

setDisplayName

public void **setDisplayName** (String *displayName*)

setType

public void **setType** (ParameterType *type*)

toString

public String **toString** ()

12.105.21 ParameterType

public enum **ParameterType**

Defines the type of various values used in a task including trigger parameters, action parameters and data source object fields.

Enum Constants

BOOLEAN

public static final ParameterType **BOOLEAN**

DATE

public static final ParameterType **DATE**

DOUBLE

public static final ParameterType **DOUBLE**

INTEGER

public static final ParameterType **INTEGER**

LIST

public static final ParameterType **LIST**

LONG

public static final ParameterType **LONG**

MAP

public static final ParameterType **MAP**

PERIOD

public static final ParameterType **PERIOD**

SELECT

public static final ParameterType **SELECT**

TEXTAREA

public static final ParameterType **TEXTAREA**

TIME

public static final ParameterType **TIME**

UNICODE

public static final ParameterType **UNICODE**

UNKNOWN

public static final ParameterType **UNKNOWN**

12.105.22 SettingsDto

public class **SettingsDto**

Methods

getTaskPossibleErrors

public String **getTaskPossibleErrors** ()

isValid

public boolean **isValid** ()

setTaskPossibleErrors

public void **setTaskPossibleErrors** (String *taskPossibleErrors*)

12.105.23 Task

public class **Task**

A task is set of actions that are executed in response to a trigger. The actions and the trigger are defined by their respective `Channel`s.

Constructors

Task

public **Task** ()

Constructor.

Task

public **Task** (String *name*, TaskTriggerInformation *trigger*, List<TaskActionInformation> *actions*)

Constructor.

Parameters

- **name** – the task name
- **trigger** – the task trigger
- **actions** – the list of related actions

Task

public **Task** (String *name*, TaskTriggerInformation *trigger*, List<TaskActionInformation> *actions*, TaskConfig *taskConfig*, boolean *enabled*, boolean *hasRegisteredChannel*)

Constructor.

Parameters

- **name** – the task name
- **trigger** – the task trigger
- **actions** – the list of related actions
- **taskConfig** – the task configuration
- **enabled** – defines if this task is enabled
- **hasRegisteredChannel** – defines if this task has a registered channel

Methods

addAction

public void **addAction** (TaskActionInformation *action*)

Stores the given action.

Parameters

- **action** – the action

addValidationErrors

public void **addValidationErrors** (Set<TaskError> *validationErrors*)

equals

public boolean **equals** (Object *obj*)

getActions

public List<TaskActionInformation> **getActions** ()

getDescription

public String **getDescription** ()

getFailuresInRow

public int **getFailuresInRow** ()

getId

public Long **getId**()

getName

public String **getName**()

getTaskConfig

public TaskConfig **getTaskConfig**()

getTrigger

public TaskTriggerInformation **getTrigger**()

getValidationErrors

public Set<TaskError> **getValidationErrors**()

hasRegisteredChannel

public boolean **hasRegisteredChannel**()

hasValidationErrors

public boolean **hasValidationErrors**()

hashCode

public int **hashCode**()

incrementFailuresInRow

public void **incrementFailuresInRow**()

Increases the counter of task execution failures that occurred since the last successful execution of this task or since the task was enabled.

isEnabled

public boolean **isEnabled**()

removeValidationError

public void **removeValidationError**(String *message*)

resetFailuresInRow

public void **resetFailuresInRow**()

Resets the counter of task execution failures that occurred since the last successful execution of this task or since the task was enabled.

setActions

public void **setActions**(List<TaskActionInformation> *actions*)

setDescription

public void **setDescription**(String *description*)

setEnabled

public void **setEnabled**(boolean *enabled*)

setFailuresInRow

public void **setFailuresInRow**(int *failuresInRow*)

setHasRegisteredChannel

public void **setHasRegisteredChannel**(boolean *hasRegisteredChannel*)

setId

public void **setId**(Long *id*)

setName

public void **setName**(String *name*)

setTaskConfig

public void **setTaskConfig**(TaskConfig *taskConfig*)

setTrigger

public void **setTrigger**(TaskTriggerInformation *trigger*)

setValidationErrors

public void **setValidationErrors**(Set<TaskError> *validationErrors*)

toString

public String **toString**()

12.105.24 TaskActionInformation

public class **TaskActionInformation** extends TaskEventInformation

Represents an action from a channel. An action is taken upon a task trigger. This class is the representation of the definition from the channel, not the representation of an usage within a task. An action can be represented as an event, but also as a direct OSGi message(or both - a service call with fallback event). It is part of the tasks model.

Constructors

TaskActionInformation

public **TaskActionInformation**()

Constructor.

TaskActionInformation

public **TaskActionInformation**(String *displayName*, String *channelName*, String *moduleName*, String *moduleVersion*, String *subject*)

Constructor.

Parameters

- **displayName** – the task display name
- **channelName** – the channel name
- **moduleName** – the module name
- **moduleVersion** – the module version
- **subject** – the task subject

TaskActionInformation

public **TaskActionInformation**(String *displayName*, String *channelName*, String *moduleName*, String *moduleVersion*, String *subject*, Map<String, String> *values*)

Constructor for an action that is an event sent by the task module.

Parameters

- **displayName** – the task display name
- **channelName** – the channel name
- **moduleName** – the module name
- **moduleVersion** – the module version
- **subject** – the task subject
- **values** – the map of values

TaskActionInformation

public **TaskActionInformation**(String *displayName*, String *channelName*, String *moduleName*, String *moduleVersion*, String *serviceInterface*, String *serviceMethod*)

Constructor for an action that is an OSGi service method call from the tasks module.

Parameters

- **displayName** – the task display name
- **channelName** – the channel name
- **moduleName** – the module name
- **moduleVersion** – the module version
- **serviceInterface** – the task service interface
- **serviceMethod** – the task service method

TaskActionInformation

public **TaskActionInformation**(String *name*, String *displayName*, String *channelName*, String *moduleName*, String *moduleVersion*, String *subject*, String *serviceInterface*, String *serviceMethod*, Map<String, String> *values*)

Constructor for a task that is an OSGi service call from the tasks module, but falls back to sending an event if the service is not present

Parameters

- **name** – the task name
- **displayName** – the task display name
- **channelName** – the channel name
- **moduleName** – the module name
- **moduleVersion** – the module version
- **subject** – the task subject
- **serviceInterface** – the task service interface
- **serviceMethod** – the task service method
- **values** – the map of values

TaskActionInformation

public **TaskActionInformation**(String *name*, String *displayName*, String *channelName*, String *moduleName*, String *moduleVersion*, String *serviceInterface*, String *serviceMethod*)

Constructor.

Parameters

- **name** – the task name
- **displayName** – the task display name
- **channelName** – the channel name

- **moduleName** – the module name
- **moduleVersion** – the module version
- **serviceInterface** – the task service interface
- **serviceMethod** – the task service method

Methods

equals

public boolean **`equals`** (Object *obj*)

getServiceInterface

public String **`getServiceInterface`** ()

getServiceMethod

public String **`getServiceMethod`** ()

getValues

public Map<String, String> **`getValues`** ()

hasService

public boolean **`hasService`** ()

hashCode

public int **`hashCode`** ()

setServiceInterface

public void **`setServiceInterface`** (String *serviceInterface*)

setServiceMethod

public void **`setServiceMethod`** (String *serviceMethod*)

setValues

public void **`setValues`** (Map<String, String> *values*)

toString

public String **toString**()

12.105.25 TaskActivity

public class **TaskActivity** implements Comparable<TaskActivity>

Represents a single task activity. Task activity is a historical entry about a task execution.

Constructors

TaskActivity

public **TaskActivity**()

Constructor.

TaskActivity

public **TaskActivity**(String *message*, Long *task*, TaskActivityType *activityType*)

Constructor.

Parameters

- **message** – the activity message
- **task** – the activity task ID
- **activityType** – the activity type

TaskActivity

public **TaskActivity**(String *message*, String *field*, Long *task*, TaskActivityType *activityType*)

Constructor.

Parameters

- **message** – the activity message
- **field** – the field name
- **task** – the activity task ID
- **activityType** – the activity type

TaskActivity

public **TaskActivity**(String *message*, List<String> *fields*, Long *task*, TaskActivityType *activityType*)

Constructor.

Parameters

- **message** – the activity message
- **fields** – the field names

- **task** – the activity task ID
- **activityType** – the activity type

TaskActivity

public **TaskActivity** (String *message*, List<String> *fields*, Long *task*, TaskActivityType *activityType*, String *stackTraceElement*)

Constructor.

Parameters

- **message** – the activity message
- **fields** – the field names
- **task** – the activity ID
- **activityType** – the activity type
- **stackTraceElement** – the stack trace that caused the task failure

Methods

compareTo

public int **compareTo** (TaskActivity *o*)

equals

public boolean **equals** (Object *obj*)

getActivityType

public TaskActivityType **getActivityType** ()

getDate

public DateTime **getDate** ()

getFields

public List<String> **getFields** ()

getMessage

public String **getMessage** ()

getStackTraceElement

public String **getStackTraceElement** ( )

getTask

public Long **getTask** ( )

hashCode

public int **hashCode** ( )

setActivityType

public void **setActivityType** (TaskActivityType *activityType*)

setDate

public void **setDate** (DateTime *date*)

setField

public void **setField** (String *field*)

setFields

public void **setFields** (List<String> *fields*)

setMessage

public void **setMessage** (String *message*)

setStackTraceElement

public void **setStackTraceElement** (String *stackTraceElement*)

setTask

public void **setTask** (Long *task*)

toString

public String **toString** ( )

12.105.26 TaskActivityType

public enum **TaskActivityType**
Enumerates all types of task activities.

Enum Constants

ERROR

public static final TaskActivityType **ERROR**

SUCCESS

public static final TaskActivityType **SUCCESS**

WARNING

public static final TaskActivityType **WARNING**

12.105.27 TaskBuilder

public class **TaskBuilder**

Constructors

TaskBuilder

public **TaskBuilder** ()

Methods

addAction

public TaskBuilder **addAction** (TaskActionInformation *action*)

addDataSource

public TaskBuilder **addDataSource** (DataSource *dataSource*)

addFilterSet

public TaskBuilder **addFilterSet** (FilterSet *filterSet*)

build

public Task **build**()

clear

public TaskBuilder **clear**()

isEnabled

public TaskBuilder **isEnabled**(boolean *enabled*)

withDescription

public TaskBuilder **withDescription**(String *description*)

withId

public TaskBuilder **withId**(Long *id*)

withName

public TaskBuilder **withName**(String *name*)

withTaskConfig

public TaskBuilder **withTaskConfig**(TaskConfig *taskConfig*)

withTrigger

public TaskBuilder **withTrigger**(TaskTriggerInformation *trigger*)

12.105.28 TaskConfig

public class **TaskConfig** implements Serializable

Represents a single task configuration. Task configuration is a list of `TaskConfigStep`s that are taken during task execution. A single step can be a filter(whose conditions must be meet) or a data store that must be fetched in order to execute the task successfully.

Methods

add

public TaskConfig **add**(TaskConfigStep... *configSteps*)

Stores the given configuration steps.

Parameters

- **configSteps** – the configuration steps, not null

Returns this object

addAll

public TaskConfig **addAll** (SortedSet<TaskConfigStep> *set*)

Stores the given configuration steps

Parameters

- **set** – the configuration steps

Returns this object

equals

public boolean **equals** (Object *obj*)

getDataSource

public DataSource **getDataSource** (Long *providerId*, Long *objectId*, String *objectType*)

Returns the data source for the given information.

Parameters

- **providerId** – the provider ID
- **objectId** – the object ID
- **objectType** – the object type

Returns the object matching the given data.

getDataSources

public List<DataSource> **getDataSources** ()

getDataSources

public SortedSet<DataSource> **getDataSources** (Long *providerId*)

getFilters

public List<FilterSet> **getFilters** ()

getSteps

public SortedSet<TaskConfigStep> **getSteps** ()

hashCode

public int **hashCode** ( )

removeAll

public TaskConfig **removeAll** ( )

Clears filter sets and data sources for this object.

Returns this object

removeDataSources

public TaskConfig **removeDataSources** ( )

Clears data sources for this object.

Returns this object

removeFilterSets

public TaskConfig **removeFilterSets** ( )

Clears filter sets for this object.

Returns this object

setDataSources

public void **setDataSources** (List<DataSource> *dataSources*)

setFilters

public void **setFilters** (List<FilterSet> *filters*)

toString

public String **toString** ( )

12.105.29 TaskConfigStep

public abstract class **TaskConfigStep** implements Comparable<TaskConfigStep>, Serializable

Represents a single task configuration step. Task configuration step is an abstract class that should be extended by all steps that are taken during task execution. Currently it serves as a base class for `Filter`s and `DataSource`s.

Methods

compareTo

public int **compareTo** (TaskConfigStep *o*)

equals

public boolean **equals** (Object *obj*)

getOrder

public Integer **getOrder** ()

hashCode

public int **hashCode** ()

setOrder

public void **setOrder** (Integer *order*)

toString

public String **toString** ()

12.105.30 TaskDataProvider

public class **TaskDataProvider**

Represents a single data provider used by the task module. It provides provider objects used as data sources by the tasks.

Constructors

TaskDataProvider

public **TaskDataProvider** ()

TaskDataProvider

public **TaskDataProvider** (String *name*, List<TaskDataProviderObject> *objects*)

Methods

containsProviderObject

public boolean **containsProviderObject** (String *type*)

containsProviderObjectField

public boolean **containsProviderObjectField** (String *type*, String *fieldKey*)

containsProviderObjectLookup

public boolean **containsProviderObjectLookup** (String *type*, String *lookupField*)

equals

public boolean **equals** (Object *obj*)

getId

public Long **getId** ()

getKeyType

public String **getKeyType** (String *key*)

getName

public String **getName** ()

getObjects

public List<TaskDataProviderObject> **getObjects** ()

getProviderObject

public TaskDataProviderObject **getProviderObject** (String *type*)

hashCode

public int **hashCode** ()

setId

public void **setId** (Long *id*)

setName

public void **setName** (String *name*)

setObjects

public void **setObjects** (List<TaskDataProviderObject> *objects*)

toString

public String **toString** ()

12.105.31 TaskDataProviderObject

public class **TaskDataProviderObject** implements Serializable

Represents a single object of the task data provider. It describes fields and lookups of an entity that is used as a data store in the task module.

Constructors

TaskDataProviderObject

public **TaskDataProviderObject** ()

TaskDataProviderObject

public **TaskDataProviderObject** (String *displayName*, String *type*, List<LookupFieldsParameter> *lookupFields*, List<FieldParameter> *fields*)

Methods

containsField

public boolean **containsField** (String *fieldKey*)

equals

public boolean **equals** (Object *obj*)

getDisplayName

public String **getDisplayName** ()

getFields

public List<FieldParameter> **getFields** ()

getLookupFields

public List<LookupFieldsParameter> **getLookupFields** ()

getType

public String **getType** ()

hashCode

public int **hashCode** ()

setDisplayName

public void **setDisplayName** (String *displayName*)

setFields

public void **setFields** (List<FieldParameter> *fields*)

setLookupFields

public void **setLookupFields** (List<Object> *lookupFields*)

setType

public void **setType** (String *type*)

toString

public String **toString** ()

12.105.32 TaskError

public class **TaskError** implements Serializable

Represents a single task error. Those error are encountered during validation of a channel if some of the required fields are blank or missing.

Constructors

TaskError

public **TaskError** ( )
Constructor.

TaskError

public **TaskError** (TaskErrorType *type*, String... *args*)
Constructor.

Parameters

- **type** – the error type, not null
- **args** – the arguments

TaskError

public **TaskError** (String *message*, String... *args*)
Constructor.

Parameters

- **message** – the error message
- **args** – the arguments

Methods

equals

public boolean **equals** (Object *obj*)

getArgs

public List<String> **getArgs** ( )

getMessage

public String **getMessage** ( )

hashCode

public int **hashCode** ( )

setArgs

public void **setArgs** (List<String> *args*)

setMessage

public void **setMessage** (String *message*)

toString

public String **toString** ()

12.105.33 TaskErrorType

public enum **TaskErrorType**
Enumerates all possible causes of task failures.

Enum Constants

BLANK

public static final TaskErrorType **BLANK**

EMPTY_COLLECTION

public static final TaskErrorType **EMPTY_COLLECTION**

NULL

public static final TaskErrorType **NULL**

VERSION

public static final TaskErrorType **VERSION**

12.105.34 TaskEvent

public abstract class **TaskEvent** implements Serializable
Represents a single task event. Task event is an abstract base for events utilized in the task module. It serves as a base for both `ActionEvent`s and `TriggerEvent`s. It is a part of the channel model.

Constructors

TaskEvent

protected **TaskEvent** ()
Constructor.

TaskEvent

protected **`TaskEvent`** (String *description*, String *displayName*, String *subject*)

Constructor.

Parameters

- **description** – the event description
- **displayName** – the event display name
- **subject** – the event subject

TaskEvent

protected **`TaskEvent`** (String *name*, String *description*, String *displayName*, String *subject*)

Constructor.

Parameters

- **name** – the event name
- **description** – the event description
- **displayName** – the event display name
- **subject** – the event subject

Methods

containsParameter

public boolean **`containsParameter`** (String *key*)

equals

public boolean **`equals`** (Object *obj*)

equalsSubject

protected boolean **`equalsSubject`** (String *subject*)

getDescription

public String **`getDescription`** ()

getDisplayName

public String **`getDisplayName`** ()

getName

public String **getName** ()

getSubject

public String **getSubject** ()

hasSubject

public boolean **hasSubject** ()

hashCode

public int **hashCode** ()

setDescription

public void **setDescription** (String *description*)

setDisplayName

public void **setDisplayName** (String *displayName*)

setName

public void **setName** (String *name*)

setSubject

public void **setSubject** (String *subject*)

toString

public String **toString** ()

12.105.35 TaskEventInformation

public abstract class **TaskEventInformation** implements Serializable

Represents information about single task event. Task event is an abstract base for events utilized in the task module. It serves as a base for both `TaskActionInformation`s and `TaskTriggerInformation`s. It is a part of the task model.

Constructors

TaskEventInformation

public **TaskEventInformation** ( )
Constructor.

TaskEventInformation

public **TaskEventInformation** (String *name*, String *displayName*, String *channelName*, String *moduleName*, String *moduleVersion*, String *subject*)
Constructor.

Parameters

- **name** – the event name
- **displayName** – the event display name
- **channelName** – the event channel name
- **moduleName** – the event module name
- **moduleVersion** – the module version
- **subject** – the event subject

Methods

equals

public boolean **equals** (Object *obj*)

getChannelName

public String **getChannelName** ( )

getDisplayName

public String **getDisplayName** ( )

getModuleName

public String **getModuleName** ( )

getModuleVersion

public String **getModuleVersion** ( )

getName

public String **getName** ( )

getSubject

public String **getSubject** ( )

hasSubject

public boolean **hasSubject** ( )

hashCode

public int **hashCode** ( )

setChannelName

public void **setChannelName** (String *channelName*)

setDisplayName

public void **setDisplayName** (String *displayName*)

setModuleName

public void **setModuleName** (String *moduleName*)

setModuleVersion

public void **setModuleVersion** (String *moduleVersion*)

setName

public void **setName** (String *name*)

setSubject

public void **setSubject** (String *subject*)

toString

public String **toString** ( )

12.105.36 TaskTriggerInformation

public class **TaskTriggerInformation** extends TaskEventInformation

Represents information about a single task trigger. A task trigger is an event that triggers execution of a task. It is a part of the task model.

Constructors

TaskTriggerInformation

public **TaskTriggerInformation** ()

Constructor.

TaskTriggerInformation

public **TaskTriggerInformation** (String *displayName*, String *channelName*, String *moduleName*, String *moduleVersion*, String *subject*, String *triggerListener*)

Constructor.

Parameters

- **displayName** – the trigger display name
- **channelName** – the trigger channel name
- **moduleName** – the trigger module name
- **moduleVersion** – the module version
- **subject** – the trigger subject
- **triggerListener** – the trigger listener

TaskTriggerInformation

public **TaskTriggerInformation** (TaskTriggerInformation *other*)

The copy constructor.

Parameters

- **other** – the other `TaskTrigger` to copy, not null

Methods

getEffectiveListenerSubject

public String **getEffectiveListenerSubject** ()

Convenient method for determining effective listener subject. For tasks created prior release 0.25 the trigger listener subject will not be set in the db, therefore we have to use subject.

Returns `triggerListenerSubject` if present. Otherwise returns `subject`

getTriggerListenerSubject

public String **getTriggerListenerSubject** ()

12.105.37 TriggerEvent

public class **TriggerEvent** extends TaskEvent

Represents a single trigger event. Trigger event is an event that triggers executions of a task. It is a part of the channel model.

Constructors

TriggerEvent

public **TriggerEvent** ()

Class constructor.

TriggerEvent

public **TriggerEvent** (String *displayName*, String *subject*, String *description*, List<EventParameter> *eventParameters*, String *triggerListenerSubject*)

Class constructor.

Parameters

- **displayName** – the event display name
- **subject** – the event subject
- **description** – the event description
- **eventParameters** – the event parameters
- **triggerListenerSubject** – the subject that is wrapped by this trigger, in a simple case it is identical to the subject above, so it can be omitted

TriggerEvent

public **TriggerEvent** (TriggerEventRequest *triggerEventRequest*)

Class constructor.

Parameters

- **triggerEventRequest** – the trigger event request, not null

Methods

containsParameter

public boolean **containsParameter** (String *key*)

copy

public TriggerEvent **copy** ( )

equals

public boolean **equals** (Object *obj*)

getEventParameters

public List<EventParameter> **getEventParameters** ( )

getKeyType

public String **getKeyType** (String *key*)

getTriggerListenerSubject

public String **getTriggerListenerSubject** ( )

hashCode

public int **hashCode** ( )

setEventParameters

public void **setEventParameters** (List<EventParameter> *eventParameters*)

toString

public String **toString** ( )

12.106 org.motechproject.tasks.ex

12.106.1 ActionNotFoundException

public class **ActionNotFoundException** extends Exception

Thrown when the requested action doesn't exists.

Constructors

ActionNotFoundException

public **ActionNotFoundException** (String *message*)

12.106.2 CustomParserNotFoundException

public class **`CustomParserNotFoundException`** extends IllegalArgumentException
Indicates an error, while looking for the Custom Event Parser in the context.

Constructors

CustomParserNotFoundException

public **`CustomParserNotFoundException`** (String *parserName*)
Exception constructor.

Parameters

- **parserName** – the name of the missing parser

12.106.3 TaskHandlerException

public class **`TaskHandlerException`** extends Exception
Thrown when there were problems while handling a task.

Constructors

TaskHandlerException

public **`TaskHandlerException`** (TaskFailureCause *failureCause*, String *messageKey*)
Exception constructor.

Parameters

- **failureCause** – the failure cause
- **messageKey** – the message key

TaskHandlerException

public **`TaskHandlerException`** (TaskFailureCause *failureCause*, String *messageKey*, String... *args*)
Exception constructor.

Parameters

- **failureCause** – the failure cause
- **messageKey** – the message key
- **args** – the arguments

TaskHandlerException

public **`TaskHandlerException`** (TaskFailureCause *failureCause*, String *messageKey*, Throwable *cause*, String... *args*)
Exception constructor.

Parameters

- **failureCause** – the failure cause
- **messageKey** – the message key
- **cause** – the cause of the failure
- **args** – the arguments

Methods

getArgs

public List<String> **getArgs** ()

getFailureCause

public TaskFailureCause **getFailureCause** ()

getMessage

public String **getMessage** ()

12.106.4 TaskNotFoundException

public class **TaskNotFoundException** extends IllegalArgumentException
Thrown when task with given ID doesn't exists.

Constructors

TaskNotFoundException

public **TaskNotFoundException** (Long *taskId*)
Exception constructor.

Parameters

- **taskId** – the task ID

12.106.5 TriggerNotFoundException

public class **TriggerNotFoundException** extends Exception
Thrown when requested trigger doesn't exists.

Constructors

TriggerNotFoundException

public **TriggerNotFoundException** (String *message*)
Exception constructor.

Parameters

- **message** – the message to be passed with exception

12.106.6 ValidationException

public class **`ValidationException`** extends IllegalArgumentException

Thrown when there were problems while validating

Constructors

ValidationException

public **`ValidationException`** (String *objectType*, Set<TaskError> *taskErrors*)

Exception constructor.

Parameters

- **objectType** – the type of the object
- **taskErrors** – the set of errors

Methods

getMessage

public String **`getMessage`** ()

Generates message based on the given errors and type of the object.

Returns the message

getTaskErrors

public Set<TaskError> **`getTaskErrors`** ()

12.107 org.motechproject.tasks.json

12.107.1 ActionEventRequestDeserializer

public class **`ActionEventRequestDeserializer`** implements JsonDeserializer<ActionEventRequest>

`JsonDeserializer` for the `ActionEventRequest` class.

Fields

ACTION_PARAMETERS_FIELD

public static final String **`ACTION_PARAMETERS_FIELD`**

DESCRIPTION_FIELD

public static final String **DESCRIPTION_FIELD**

DISPLAY_NAME_FIELD

public static final String **DISPLAY_NAME_FIELD**

NAME_FIELD

public static final String **NAME_FIELD**

SERVICE_INTERFACE_FIELD

public static final String **SERVICE_INTERFACE_FIELD**

SERVICE_METHOD_CALL_MANNER_FIELD

public static final String **SERVICE_METHOD_CALL_MANNER_FIELD**

SERVICE_METHOD_FIELD

public static final String **SERVICE_METHOD_FIELD**

SUBJECT_FIELD

public static final String **SUBJECT_FIELD**

Methods

deserialize

public ActionEventRequest **deserialize**(JsonElement *element*, Type *type*, JsonDeserializationContext *context*)

12.107.2 TaskConfigDeserializer

public class **TaskConfigDeserializer** extends JsonDeserializer<TaskConfig>

`JsonDeserializer` for `TaskConfig` class.

Methods

deserialize

public TaskConfig **deserialize**(JsonParser *parser*, DeserializationContext *context*)

12.107.3 TaskDeserializer

public class **TaskDeserializer** extends JsonDeserializer<Task>

`JsonDeserializer` for the `Task` class.

Methods

deserialize

public Task **deserialize** (JsonParser *jsonParser*, DeserializationContext *deserializationContext*)

12.108 org.motechproject.tasks.repository

12.108.1 ChannelsDataService

public interface **ChannelsDataService** extends MotechDataService<Channel>

Data service for channels.

Methods

findByModuleName

Channel **findByModuleName** (String *moduleName*)

Returns the list of channels for the module with given name.

Parameters

- **moduleName** – the name of the module, null returns null

Returns the list of matching activities

12.108.2 DataProviderDataService

public interface **DataProviderDataService** extends MotechDataService<TaskDataProvider>

Data Service for data providers.

Methods

findByName

TaskDataProvider **findByName** (String *name*)

Returns the data provider with the given name.

Parameters

- **name** – the name of the data provider

Returns the provider with the given name

12.108.3 TaskActivitiesDataService

public interface **TaskActivitiesDataService** extends MotechDataService<TaskActivity>

Data service for task activities.

Methods

byTask

List<TaskActivity> **byTask** (Long *task*)

Returns the list of activities for the given task name.

Parameters

- **task** – the name of the task, null returns empty list

Returns the list of matching task activities

12.108.4 TasksDataService

public interface **TasksDataService** extends MotechDataService<Task>

Data service for tasks.

Methods

findTasksByName

List<Task> **findTasksByName** (String *name*)

Returns the list of tasks with the given name.

Parameters

- **name** – the task name, null returns empty list

Returns the list of matching tasks

12.109 org.motechproject.tasks.service

12.109.1 ChannelService

public interface **ChannelService**

Manages CRUD operations for a `Channel`.

Methods

addOrUpdate

void **addOrUpdate** (Channel *channel*)

Saves the given channel. If the channel exists it will be updated.

Parameters

- **channel** – the channel to be added, not null

delete

void **delete** (String *moduleName*)

Deletes the given module.

Parameters

- **moduleName** – the channel to be deleted

getAllChannels

List<Channel> **getAllChannels** ()

Returns the list of all registered channels.

Returns the list of channels

getChannel

Channel **getChannel** (String *moduleName*)

Returns the channel for the module with the given name.

Parameters

- **moduleName** – the name of the module, null returns null

Returns the channel for the module

getChannelIcon

BundleIcon **getChannelIcon** (String *moduleName*)

Returns the icon for the channel from module with the given name.

Parameters

- **moduleName** – the name of the module, null returns default icon

Throws

- **IOException** – when there were problems while fetching the icon

Returns the icon of the module

registerChannel

void **registerChannel** (ChannelRequest *channelRequest*)

Registers the given channel with the task module.

Parameters

- **channelRequest** – the channel request, not null

registerChannel

void **registerChannel** (InputStream *stream*, String *moduleName*, String *moduleVersion*)

Registers channel from the given stream for the given module. The input stream should contain the JSON definition of the channel.

Parameters

- **stream** – the channel JSON definition as a stream, not null
- **moduleName** – the name of the module
- **moduleVersion** – the version of the module

unregisterChannel

void **unregisterChannel** (String *moduleName*)

Unregisters the given channel with the task module.

Parameters

- **moduleName** – , not null

12.109.2 DataSourceObject

public class **DataSourceObject**

DataSourceObject is the result of a `org.motechproject.commons.api.DataProvider` lookup.

Constructors

DataSourceObject

public **DataSourceObject** (String *objectId*, Object *objectValue*, boolean *failIfNotFound*)

Class constructor.

Parameters

- **objectId** – the object id
- **objectValue** – the object value
- **failIfNotFound** – defines if the task should fail if object wasn't found

Methods

equals

public boolean **equals** (Object *o*)

getObjectId

public String **getObjectId** ()

getObjectValue

public Object **getObjectValue**()

hashCode

public int **hashCode**()

isFailIfNotFound

public boolean **isFailIfNotFound**()

12.109.3 HandlerPredicates

public final class **HandlerPredicates**

Utility class defining filters over some collections.

Methods

tasksWithRegisteredChannel

public static Predicate **tasksWithRegisteredChannel**()

Returns the predicate for fetching tasks with channels that are currently registered.

Returns the predicate for fetching tasks

withServiceName

public static Predicate **withServiceName**(String *serviceName*)

Returns the predicate for fetching `MotechListenerEventProxy` with the given name.

Parameters

- **serviceName** – the name of the service

Returns the predicate for fetching `MotechListenerEventProxy`

12.109.4 KeyEvaluator

public class **KeyEvaluator**

KeyEvaluator evaluates the value of a key in the context of a task which is used to execute filters and actions.

Constructors

KeyEvaluator

public **KeyEvaluator**(TaskContext *taskContext*)

Class constructor.

Parameters

- **taskContext** – the task context, not null

Methods

evaluateTemplateString

public String **`evaluateTemplateString`** (String *template*)

Evaluates the given template by replacing the keys with their manipulated values.

Parameters

- **template** – the template to be evaluated

Throws

- **TaskHandlerException** – if there was problem while manipulating the value

Returns the evaluated template

getManipulatedValue

public Object **`getManipulatedValue`** (KeyInformation *keyInformation*)

Retrieves the value for the given key and applies all the passed manipulations.

Parameters

- **keyInformation** – the key information, not null

Throws

- **TaskHandlerException** – if there were problems while retrieving the value

Returns the manipulated value

getValue

public Object **`getValue`** (KeyInformation *keyInformation*)

Returns value for the given key.

Parameters

- **keyInformation** – the key information, not null

Throws

- **TaskHandlerException** – if there were problems while retrieving the value

Returns the value for the given key

manipulate

String **`manipulate`** (String *manipulation*, String *value*)

12.109.5 MethodHandler

class **MethodHandler**

Utility class used by `TaskTriggerHandler` to construct a list of parameter types of the method in the correct order.

See also: `TaskTriggerHandler`

Constructors

MethodHandler

public **MethodHandler** (ActionEvent *action*, Map<String, Object> *parameters*)

Class constructor.

Parameters

- **action** – the event action
- **parameters** – the action parameters, not null

Throws

- **TaskHandlerException** – if there were problems while handling the task

Methods

getClasses

public Class[] **getClasses** ()

Returns the array of the parameter classes.

Returns the array of the parameter classes

getObjects

public Object[] **getObjects** ()

Returns the array of the parameter values.

Returns the array of the parameter values

12.109.6 TaskActionExecutor

public class **TaskActionExecutor**

Builds action parameters from `TaskContext` and executes the action by invoking its service or raising its event.

Constructors

TaskActionExecutor

public **TaskActionExecutor** (TaskService *taskService*, TaskActivityService *activityService*, EventRelay *eventRelay*)

Methods

execute

public void **execute** (Task *task*, TaskActionInformation *actionInformation*, TaskContext *taskContext*)

Executes the action for the given task.

Parameters

- **task** – the task for which its action should be executed, not null
- **actionInformation** – the information about the action, not null
- **taskContext** – the context of the current task execution, not null

Throws

- **TaskHandlerException** – when the task couldn't be executed

setBundleContext

void **setBundleContext** (BundleContext *bundleContext*)

12.109.7 TaskActivityService

public interface **TaskActivityService**

Service for managing task activities. Task activities are used for storing information about past task executions.

Methods

addError

void **addError** (Task *task*, TaskHandlerException *e*)

Logs an execution error for the given task.

Parameters

- **task** – the failed task, not null
- **e** – the cause of the error, not null

addSuccess

void **addSuccess** (Task *task*)

Logs an execution success for the given task.

Parameters

- **task** – the succeeded task, not null

addWarning

void **addWarning** (Task *task*)
Logs a warning for the given task.

Parameters

- **task** – the task, not null

addWarning

void **addWarning** (Task *task*, String *key*, String *value*)
Logs a warning for the given task.

Parameters

- **task** – the task, not null
- **key** – the key of the message
- **value** – the name of the field that caused the warning

addWarning

void **addWarning** (Task *task*, String *key*, String *field*, Exception *e*)
Logs a warning for the given task.

Parameters

- **task** – the task, not null
- **key** – the key of the message
- **field** – the name of the failed that caused the warning, not null
- **e** – the exception that caused the warning, not null

deleteActivitiesForTask

void **deleteActivitiesForTask** (Long *taskId*)
Deletes all activities for the task with the given ID.

Parameters

- **taskId** – the task ID, not null

getAllActivities

List<TaskActivity> **getAllActivities** ()
Returns all activities as a list ordered by date.

Returns the list of all activities

getTaskActivities

List<TaskActivity> **getTaskActivities** (Long *taskId*)

Returns list of all activities for task with the given ID.

Parameters

- **taskId** – the task ID, null returns null

Returns the list of all activities for task with given ID

12.109.8 TaskContext

public class **TaskContext**

TaskContext holds task trigger event and data provider lookup objects that are used while executing filters/actions.

Constructors

TaskContext

public **TaskContext** (Task *task*, Map<String, Object> *parameters*, TaskActivityService *activityService*)

Class constructor.

Parameters

- **task** – the task, not null
- **parameters** – the task parameters
- **activityService** – the activity service, not null

Methods

addDataSourceObject

public void **addDataSourceObject** (String *objectId*, Object *dataSourceObject*, boolean *failIfDataNotFound*)

Adds the given data source to this task.

Parameters

- **objectId** – the ID of the object, not null
- **dataSourceObject** – the result of lookup execution, not null
- **failIfDataNotFound** – defines whether task should fail if the data wasn't found

getDataSourceObjectValue

public Object **getDataSourceObjectValue** (String *objectId*, String *field*, String *objectType*)

Returns the value of data source object based on it's field, id and type.

Parameters

- **objectId** – the id of the object, not null

- **field** – the name of the field, not null
- **objectType** – the type of the object

Throws

- **TaskHandlerException** –

Returns the value of data source object

getTask

public Task **getTask**()

getTriggerParameters

public Map<String, Object> **getTriggerParameters**()

getTriggerValue

public Object **getTriggerValue**(String *key*)

Returns the value of the trigger with the given key.

Parameters

- **key** – the key of the trigger, not null

Returns the value of the trigger with the given key

publishWarningActivity

public void **publishWarningActivity**(String *message*, String *field*)

Publishes warning activity for this task.

Parameters

- **message** – the message to be published
- **field** – the name of the field

12.109.9 TaskDataProviderService

public interface **TaskDataProviderService**

Service for managing data providers.

Methods

getProvider

TaskDataProvider **getProvider**(String *name*)

Returns the data provider with the given name.

Parameters

- **name** – the name of the data provider, null returns null

Returns the data provider with the given name, null if `name` was null

getProviderById

TaskDataProvider **`getProviderById`** (Long *providerId*)

Returns the data provider with the given ID.

Parameters

- **providerId** – the ID of the data provider, null returns null

Returns the data provider with the given ID, null if `name` was null

getProviders

List<TaskDataProvider> **`getProviders`** ()

Returns all data providers.

Returns the list of all data providers

registerProvider

void **`registerProvider`** (String *json*)

Registers the data provider defined by the given JSON `String`.

Parameters

- **json** – the data provider as JSON, not null

registerProvider

void **`registerProvider`** (InputStream *stream*)

Registers the data provider defined by the JSON represented by the given stream.

Parameters

- **stream** – the data provider as stream, not null

unregister

void **`unregister`** (String *providerName*)

Unregisters the given data provider.

Parameters

- **providerName** – the unique name of the task data provider

12.109.10 TaskFilterExecutor

public class **TaskFilterExecutor**

The `TaskFilterExecutor` applies a list of filters in a #`TaskContext`.

- **convertTo** - convert a given value to a correct type,
- **getFieldValue** - get value of a field defined in the key from the given object,
- **getTriggerKey** - get value of a trigger event parameter,
- **checkFilters** - executed defined filters for a task,
- **manipulate** - executed the given manipulation on the given string value.

Constructors

TaskFilterExecutor

public **TaskFilterExecutor** ( )

Default constructor.

Methods

checkFilters

public boolean **checkFilters** (List<Filter> *filters*, LogicalOperator *logicalOperator*, TaskContext *taskContext*)

Checks whether task with the given context matches the given filters.

Parameters

- **filters** – the filters, null returns true
- **logicalOperator** – the logical operator
- **taskContext** – the task context, not null

Throws

- **TaskHandlerException** – if there were problems while handling task

Returns true if the task matches the filters

12.109.11 TaskInitializer

class **TaskInitializer**

The `TaskInitializer` class prepares an action in the task definition to execution.

- **evalConfigSteps** - executes all config steps (load data sources, check filters) defined in the task,

See also: `TaskTriggerHandler`, `TaskActionExecutor`

Constructors

TaskInitializer

TaskInitializer (TaskContext *taskContext*)
Class constructor.

Parameters

- **taskContext** – the task context

Methods

evalConfigSteps

public boolean **evalConfigSteps** (Map<String, DataProvider> *dataProviders*)
Executes all config steps (loading data from data sources, checking filters) defined for this task.

Parameters

- **dataProviders** – the map of data providers, not null or empty

Throws

- **TaskHandlerException** – if there were error while handling task

Returns true if all steps were executed, false otherwise

12.109.12 TaskService

public interface **TaskService**
Service interface for managing tasks.

Methods

deleteTask

void **deleteTask** (Long *taskId*)
Deletes the task with the given ID.

Parameters

- **taskId** – the ID of the task, not null

exportTask

String **exportTask** (Long *taskId*)
Exports the task with the given ID as a JSON `String`.

Parameters

- **taskId** – the ID of the task, not null

Returns the JSON as a `String`

findActiveTasksForTrigger

List<Task> **findActiveTasksForTrigger** (TriggerEvent *trigger*)

Returns the list of active tasks with the given trigger. Used for retrieving tasks to execute when a given trigger fires.

Parameters

- **trigger** – the trigger, null returns empty list

Returns the list of active tasks

findActiveTasksForTriggerSubject

List<Task> **findActiveTasksForTriggerSubject** (String *subject*)

Returns the list of active tasks for the given trigger subject. Used for retrieving tasks to execute when a given trigger fires.

Parameters

- **subject** – the subject of the trigger, null returns empty list

Returns the list of active tasks

findCustomParser

TasksEventParser **findCustomParser** (String *name*)

Looks for implementations of the `org.motechproject.commons.api.TasksEventParser` that have been exposed as OSGi services by bundles. For all found implementations, this method will match names returned by the `getName()` method of the `TasksEventParser` and passed as parameter to this method. If a match is found, the implementation is returned.

Parameters

- **name** – A name of the parser, that will be matched with `getName()` of the implementations

Returns Implementation of the `org.motechproject.commons.api.TasksEventParser` that returns the same name via `getName()` method as the name passed to the method

findTasksByName

List<Task> **findTasksByName** (String *name*)

Return the list of tasks with the given name.

Parameters

- **name** – the task name, null returns null

Returns the list of tasks meeting the given condition

findTasksDependentOnModule

List<Task> **findTasksDependentOnModule** (String *moduleName*)

Returns the list of task dependent on the module with the given name.

Parameters

- **moduleName** – the name of the module, not null

Returns the list of tasks

findTrigger

TriggerEvent **findTrigger** (String *subject*)

Returns a trigger with the given subject.

Parameters

- **subject** – the trigger subject, not null

Throws

- **TriggerNotFoundException** – if the trigger for the given subject wasn't found

Returns the trigger with the given subject

getActionEventFor

ActionEvent **getActionEventFor** (TaskActionInformation *taskActionInformation*)

Returns the action event that matches the given information about the task action.

Parameters

- **taskActionInformation** – the action information, not null

Throws

- **ActionNotFoundException** – when action was not found

Returns the action event matching the given information

getAllTasks

List<Task> **getAllTasks** ()

Returns the list of all tasks.

Returns the list of all tasks

getTask

Task **getTask** (Long *taskId*)

Returns the task with the given ID.

Parameters

- **taskId** – the ID of the task, null returns null

Returns the task with the given ID, null if task with the given ID wasn't found

importTask

Task **importTask** (String *json*)

Imports the task from JSON `String`.

Parameters

- **json** – the task in JSON format

Throws

- **IOException** – when there were problems while parsing the JSON

Returns the imported task, not null

save

void **save** (Task *task*)

Saves the given task in the database.

Parameters

- **task** – the task to be saved, not null

12.109.13 TaskTriggerHandler

public class **TaskTriggerHandler** implements TriggerHandler

The `TaskTriggerHandler` receives events and executes tasks for which the trigger event subject is the same as the received event subject.

Constructors

TaskTriggerHandler

public **TaskTriggerHandler** (TaskService *taskService*, TaskActivityService *activityService*, EventListenerRegistryService *registryService*, EventRelay *eventRelay*, TaskActionExecutor *taskActionExecutor*, SettingsFacade *settings*)

Methods

addDataProvider

public void **addDataProvider** (DataProvider *provider*)

handle

public void **handle** (MotechEvent *event*)

registerHandlerFor

public final void **registerHandlerFor** (String *subject*)

removeDataProvider

public void **removeDataProvider** (String *taskDataProviderId*)

setBundleContext

public void **setBundleContext** (BundleContext *bundleContext*)

setDataProviders

void **setDataProviders** (Map<String, DataProvider> *dataProviders*)

12.109.14 TriggerHandler

public interface **TriggerHandler**

Service responsible for handling triggers. When registered for an event with a specific subject, it will act as MotechEvent listener for it. That means, when the event with the subject handled by this handler is fired, the handler will retrieve all active tasks with triggers corresponding to this event and execute them.

Methods

handle

void **handle** (MotechEvent *event*)

Handles the given event. This method is responsible for retrieving active tasks with triggers corresponding to this event and then executing them. It is called by the event system.

Parameters

- **event** – the event, not null

Throws

- **TriggerNotFoundException** – if the trigger for the given event wasn't found

registerHandlerFor

void **registerHandlerFor** (String *subject*)

Registers this handler to listen for events with the given subject. This handler will now act as a MotechEvent listener for the given subject and will get called by the event system when an event with the given subject is fired.

Parameters

- **subject** – the event subject, not null

12.110 org.motechproject.tasks.util

12.110.1 DataProviderManager

public class **DataProviderManager** implements OsgiServiceLifecycleListener

Service for managing data providers.

Constructors

DataProviderManager

public **DataProviderManager** (TaskDataProviderService *taskDataProviderService*)

DataProviderManager

public **DataProviderManager** (TaskTriggerHandler *handler*, TaskDataProviderService *taskDataProviderService*)

Service constructor.

Parameters

- **handler** – the task trigger handler
- **taskDataProviderService** – the task data provider service, not null

Methods

bind

public void **bind** (Object *service*, Map *serviceProperties*)

Checks if the given service is a data provider and, if so, registers it in the data provider service.

Parameters

- **service** – the service to be registered, null will do nothing
- **serviceProperties** – unused

unbind

public void **unbind** (Object *service*, Map *serviceProperties*)

Checks if given service is a data provider and, if so, unregisters it from the data provider service.

Parameters

- **service** – the service to be unregistered, null will do nothing
- **serviceProperties** – unused

CHAPTER 13

Acknowledgements

The MOTECH team would like to acknowledge the following companies for providing free software.

13.1 Balsamiq

Balsamiq Mockups is a rapid wireframing tool that helps you Work Faster & Smarter. It reproduces the experience of sketching on a whiteboard, but using a computer.

13.2 GitHub

GitHub. Build software better, together. Powerful collaboration, code review, and code management for open source and private projects.

13.3 JetBrains

Intellij IDEA. The Most Intelligent Java IDE. Excel at enterprise, mobile and web development with Java, Scala and Groovy, with all the latest modern technologies and frameworks available out of the box.

PyCharm. The Most Intelligent Python IDE. Enjoy productive Python, Django, and Web development with PyCharm, an intelligent Python IDE offering unique coding experience.

13.4 YourKit

YourKit supports open source projects with its full-featured Java Profiler.

YourKit, LLC is the creator of YourKit Java Profiler and YourKit .NET Profiler, innovative and intelligent tools for profiling Java and .NET applications.

A

B

C

D

E

F

G

H

I

J

K

L

M

N

O

P

Q

R

S

T

U

V

W

Y