OWSI-Core doc Documentation

Release 1.0

Laurent Almeras

Oct 27, 2017

Release notes

CHAPTER 1

Migrating to 0.11

This guide aims at helping OWSI-Core users migrate an application based on OWSI-Core 0.9 to OWSI-Core 0.11.

OWSI-Core 0.10 was never released, so you should not have to migrate from this version, but if you did, then you could use this guide. Some parts would be irrelevant, but nothing should be missing.

Tools

Maven

Use the latest version of maven (at least Maven 3.3.1).

Java

~~Use JDK8. Earlier versions won't work.~~ Actually, use JDK7. A frequent VM crash was spotted when using Java 8 on a 32-bit OS and a bug report is currently pending review.

Bindings & code processors

You may have trouble with your bindings. Know that:

1. QueryDSL completely revamped its bindings; you'll need to delete them completely and regenerate them.
2. QueryDSL bindings generation will now fail if your project does not compile. Worse, not having QueryDSL bindings will add an enormous amount of build errors, which will make it much harder to spot actual errors that you must fix. It might be easier to first override QueryDSL's version and the processor used for QueryDSL bindings generation in your project (while sticking with OWSI-Core 0.9), and only then attempt migrating to OWSI-Core 0.11. The new processor is `com.querydsl:querydsl-apt` version 4.0.7. See below for changes in the new version of QueryDSL (`com.querydsl:querydsl-jpa` v4.0.7)
3. This page might help if you encounter errors when generating bindings.

A new feature was introduced in OWSI-Core 0.11 that allows the bindings to be completely wiped before generation, so that you won't need to manually delete them after a refactoring, for instance. In order to benefit from this feature, you must change your `eclipse/*.launch` files so that the invoked goals are simply `generate-sources`, but with the `eclipse-processor` profile enabled. On Linux, you may use the following command line from your project root:

```
find eclipse -name '*.launch' -print0 | xargs -0 sed -r -i 's/org.bsc.maven:maven-
↪processor-plugin:process( org.bsc.maven:maven-processor-plugin:process-test)?/
↪generate-sources/;s,<stringAttribute key="M2_PROFILES" value=""/>,<stringAttribute␣
↪key="M2_PROFILES" value="eclipse-processor"/>,'
```

External changes (libraries)

OpenCSV

- The maven artifact has changed. `net.sf.opencsv:opencsv` is now `com.opencsv:opencsv`.
- The main package changed, too. `au.com.bytecode.opencsv` is now `com.opencsv`.

FlyingSaucer (XHtmlRenderer)

- The renderer now supports border-radius. Please check that the rendering output in your project still suits you.

QueryDsl

- the root package has changed
- the general logic is now `query.select(...).from(...).fetch()`
- `count()` -> `fetchCount()`
- `uniqueResult()` -> `fetchOne()`
- `singleResult()` -> `fetchFirst()`
- `fetch().fetchAll()` -> `fetchJoin().fetchAll()`
- `query.map(keyExpression, valueExpression)` becomes `query.transform(GroupBy.groupBy(key).as(value))`
- Be careful with `mapExpression.containsKey`. From QueryDSL 4 on (maybe before?), QueryDSL will *not* create a subquery, but instead will try doing a join in the main query, with mixed results (especially considering that Hibernate is notoriously buggy when it comes to cross-joins mixed with left/right/inner joins). To be safe, do not do this:

```
new JPAQuery<MyEntity>(getEntityManager())
    .select(qMyEntity)
    .from(qMyEntity)
    .from(qMyOtherEntity)
    .where(qMyOtherEntity.mapProperty.containsKey(qMyEntity))
    .orderBy(qMyEntity.id.asc())
    .fetch();
```

  But instead do this:

```
new JPAQuery<MyEntity>(getEntityManager())
.select(qMyEntity)
    .from(qMyEntity)
    .where(
            JPAExpressions.selectOne()
            .from(qMyOtherEntity)
            .where(qMyOtherEntity.mapProperty.containsKey(qMyEntity))
            .exists()
    )
    .orderBy(qMyEntity.id.asc())
    .fetch();
```

Hibernate

- When using `javax.persistence.Index` with Hibernate, now the column names in columnList are *really* column names, not some mix-up of physical and logical names: if you had written myEmbeddable.myProperty_id, it becomes myEmbeddable_myProperty_id (or whatever name your column has)
- If you were using the deprecated datatype `org.hibernate.type.StringClobType`, then be sure to switch to `fr.openwide.core.jpa.hibernate.usertype.StringClobType` (as the former has been removed). Note that you may now use `fr.openwide.core.jpa.hibernate.usertype.StringClobType.TYPENAME` instead of duplicating the same inline String constant each time you need it.
- The naming strategy system changed in Hibernate 5. You must now add this to your `configuration-private.properties`

```
hibernate.implicit_naming_strategy=fr.openwide.core.jpa.hibernate.model.naming.
↪ImplicitNamingStrategyLegacyJpaComponentPathImpl
hibernate.physical_naming_strategy=fr.openwide.core.jpa.hibernate.model.naming.
↪PostgreSQLPhysicalNamingStrategyImpl
hibernate.id.new_generator_mappings=false
```

Hibernate Search & Lucene

- `SortField.STRING` is now `SortField.Type.STRING`
- Dates are now stored as `Long` so you need to sort them using `Sort.Type.LONG`
- Lucene has been upgraded. You will have to wipe clean your indexes and reindex everything
- For any field on which you perform a sort, you should now use the `@SortableField` (or `@SortableFields`) annotation. This is not mandatory, but will offer better performance and avoid annoying logs.
  - Note that, to sort GenericEntities by ID, you should now use the `GenericEntity.ID_SORT` field (or you'll get annoying warnings in your logs).
- The use of Lucene's `TokenStream` is now more safeguarded; this may lead to exceptions where you were not using it properly. Make sure that:
  - You always instantiate them in a try-with-resource (`try (TokenStream = /* ... */) { /* ... */ }`)
  - You always call `.reset()` before use
  - You always call `.end()` after use

WiQuery

The project has been moved to wicketstuff. Thus:

- The maven artifacts have changed:
- `org.odlabs.wiquery:wiquery-core` is now `org.wicketstuff.wiquery:wiquery-core`
- `org.odlabs.wiquery:wiquery-jquery-ui` is now `org.wicketstuff.wiquery:wiquery-jquery-ui`
- The main package changed, too. `org.odlabs.wiquery` is now `org.wicketstuff.wiquery`. If you're running an Unix-like OS, you may fix this in your project automatically with this command (to be run from the root of your project): `find . -name '*.java' | xargs sed -i 's/^import org.odlabs.wiquery/import org.wicketstuff.wiquery/'`

Wicket

- You no longer need to depend on `fr.openwide.core.components:000-owsi-core-component-wicket-overri`. Plus, it will probably harm to do so. Just remove this dependency.
- `StringResourceModel` now has a fluid API: you should use `setModel`, `setParameters` and `setDefaultValue`
- You need to look for Ajax links whose markup is an `<a>` the click event is not blocked by Wicket anymore, which will result in a scroll to the top of the page each time the link is clicked. This is always true for Google Chrome, but only if there is a `href` attribute for Firefox. To avoid any kind of trouble, just follow the guidelines detailed here.
- There is now a high-level integration of JQPlot built in OWSI-Core. See the docs or the pull request for more information.

Spring & Spring Security

- in `security-http`, `use-expressions` is now `true` by default. Thus, you have to use expressions like `hasRole('xxx')` and `permitAll` or define it explicitely to `false`. Be careful that the error is triggered only when you effectively access a secured page.
- change the `-4.0.xsd` schema to `-4.2.xsd` (Spring namespaces)
- change the `-3.2.xsd` schema to `-4.0.xsd` (Spring Security namespaces) - obviously, you need to follow the order
- in **every** `security:http` (even those related to simple REST API calls), you need to add:

```
<security:headers disabled="true"/>
<security:csrf disabled="true"/>
```

Internal changes

Core

- @PermissionObject has been moved to package `fr.openwide.core.commons.util.security`. Run the following command from the root of your project to update your imports: `find . -type`

```
f -name '*.java' -print0 | xargs -0 sed -r -i 's/fr.openwide.core.jpa.
business.generic.annotation.PermissionObject/fr.openwide.core.commons.
util.security.PermissionObject/g'
```

- `TransactionSynchronizationTaskManagerServiceImpl` now executes `afterRollback` on tasks implementing it in **reverse** order. See https://github.com/openwide-java/owsi-core-parent/commit/b431545ce20c8c5a182617e7e93b9f044086d4b1

Webapp

- The JQPlot/WQPlot dependency has been moved to a separate module. If you were using JQPlot/WQPlot, add this to your webapp's dependencies:

```
    <dependency>
        <groupId>fr.openwide.core.components</groupId>
        <artifactId>owsi-core-component-wicket-more-jqplot</artifactId>
        <version>${owsi-core.version}</version>
    </dependency>
```

- The `FormErrorDecoratorListener` has been pulled from various projects to OWSI-Core. Use OWSI-Core's version.
- The `DataTableBuilder` and related classes have moved. You may use the following sed script to convert your source code. Just create a file, put the following snippet in there, then run `find . -type f -name '*.java' -print0 | xargs -0 sed -r -i -f ./thescriptfile`. Here's the content of this file:

```
s/fr.openwide.core.wicket.more.markup.html.repeater.data.table.
↪DecoratedCoreDataTablePanel/fr.openwide.core.wicket.more.markup.repeater.table.
↪DecoratedCoreDataTablePanel/g
s/fr.openwide.core.wicket.more.markup.html.repeater.data.table.
↪DecoratedCoreDataTablePanel.AddInPlacement/fr.openwide.core.wicket.more.markup.
↪repeater.table.DecoratedCoreDataTablePanel.AddInPlacement/g
s/fr.openwide.core.wicket.more.markup.html.repeater.data.table.builder.
↪DataTableBuilder/fr.openwide.core.wicket.more.markup.repeater.table.builder.
↪DataTableBuilder/g
s/fr.openwide.core.wicket.more.markup.html.repeater.data.table.AbstractCoreColumn/
↪fr.openwide.core.wicket.more.markup.repeater.table.column.AbstractCoreColumn/g
s/fr.openwide.core.wicket.more.markup.html.repeater.data.table.CoreDataTable/fr.
↪openwide.core.wicket.more.markup.repeater.table.CoreDataTable/g
s/fr.openwide.core.wicket.more.markup.html.repeater.data.table.util.DataTableUtil/
↪fr.openwide.core.wicket.more.markup.repeater.table.util.DataTableUtil/g
s/fr.openwide.core.wicket.more.markup.html.repeater.data.table.util.
↪IDataTableFactory/fr.openwide.core.wicket.more.markup.repeater.table.builder.
↪IDataTableFactory/g
```

- The `DataTableBuilder` and related classes are now based on the `ISequenceProvider` instead of `IDataProvider`. You may still use `IDataProvider` as an input to the `DataTableBuilder` (it will be wrapped).
- A new interface was introduced in order to address code execution in a wicket context: `IWicketContextExecutor`. Here are the main consequences to existing applications:
  - An object of type `IWicketContextExecutor` is now available in the Spring context. You may `@Autowire` it in your own beans, or redefine it by overriding `fr.openwide.core.wicket.more.config.spring.AbstractWebappConfig.wicketContextExecutor(WebApplication)` in your own webapp configuration.

  - Classes extending `AbstractWicketRendererServiceImpl`, `AbstractNotificationContentDescriptor` `AbstractNotificationUrlBuilderServiceImpl`, `AbstractNotificationPanelRendererServiceI` must now provide a `IWicketExecutor` to their super constructor and must not override `getApplicationName()` anymore.
  - Classes extending `AbstractBackgroundWicketThreadContextBuilder` should instead rely on a `IWicketContextExecutor`.

External link checker

The external link checker now has its own Maven module. See ExternalLinkChecker if you use it in your app.

Related to the new `PropertyService`: you also have to use `JpaExternalLinkCheckerConfig` (import) in your app.

Properties

Both immutable and mutable properties are now handled by `PropertyService`. See PropertyService to use it in your app.

- `CoreConfigurer`: getter methods are deprecated and redirect to `propertyService`. Utility methods are also deprecated.
- `AbstratParameterServiceImpl`: getter and setter methods are deprecated and redirect to `propertyService`. Utility methods are also deprecated.

Important notes

- Properties wrapping a date (or a date time) and registered in `PropertyService` must respect the following format 'yyyy-MM-dd' (or 'yyyy-MM-dd HH:mm(:ss)'). See `StringDateConverter` and `StringDateTimeConverter`.
- `ConfigurationLogger`: As previously it uses *propertyNamesForInfoLogLevel* property but it is based now on `PropertyService`. That's why all the properties you want to display must be registered in the `PropertyService`.
- To display a warning message in case of `null` value while retrieving a property, add the following entry in your log4j file: `log4j.logger.fr.openwide.core.spring.property.service.PropertyServiceImpl=DEBUG`.

1st option: keeping the old school properties management

This case is not tested yet and is **not recommended**. Please, as much as possible, migrate to the `PropertyService`.

Get the latest version of both `CoreConfigurer` and `AbstractParameterServiceImpl` (+ `IAbstractParameterService`) from the previous version of OWSI-Core and bring back all methods and attributes needed in your own `YourAppConfigurer` and `ParameterServiceImpl` (+ `IParameterService`).

Also, in `YourAppCorePropertyConfig`, make sure the `mutablePropertyDao` method returns a `IParameterDao` and not simply a `IMutablePropertyDao`.

2nd (better) option: migrating to PropertyService

See PropertyService

- Create a `YourAppCorePropertyIds` and a `YourAppApplicationPropertyConfig` in your `core` module.
- Create a `YourAppWebappPropertyIds` and a `YourAppApplicationPropertyRegistryConfig` in your `webapp` module.
- Register your properties.
- Deprecate everything in `YourAppConfigurer` and `ParameterServiceImpl`
- Fix all deprecated warnings caused by the configurer and the parameter service. See Javadoc on deprecated methods in OWSI-Core to make it easier.
- Remove `ParameterServiceImpl` and `IParameterService`.
- Remove everything from `YourAppConfigurer`.

Audit

The audit classes have been removed.

You should either:

- Copy the old Audit base classes in your own project
- Or (better) use the brand-new HistoryLog framework. See HistoryLog & Audit

PasswordEncoder

From now on, we use bcrypt method to encode new passwords. However, old passwords / hashes still use previous encryption method.

SecurityPasswordRules

SecurityPasswordRules is now a builder and provide a `Set<Rule>`.

```
SecurityPasswordRules
    .builder()
    .minMaxLength(..., ...)
    .forbiddenUsername()
    .rule(YourCustomRule())
    .build();
```

Also, replace `SecurityPasswordRules.DEFAULT`:

```
SecurityPasswordRules
    .builder()
    .minMaxLength(User.MIN_PASSWORD_LENGTH, User.MAX_PASSWORD_LENGTH)
    .build();
```

Configuration

- Ensure to give a value to notification.mail.recipientsFiltered property (true or false). If true, mail's recipients are replaced by notification.test.emails property's content

- Replace this : `hibernate.search.analyzer=org.hibernate.search.util.impl.PassThroughAnalyzer` with this `hibernate.search.analyzer=org.apache.lucene.analysis.core.KeywordAnalyzer`
- The content of configuration-private.properties should be:

```
hibernate.implicit_naming_strategy=fr.openwide.core.jpa.hibernate.model.naming.
↪ImplicitNamingStrategyLegacyJpaComponentPathImpl
hibernate.physical_naming_strategy=fr.openwide.core.jpa.hibernate.model.naming.
↪PostgreSQLPhysicalNamingStrategyImpl
hibernate.id.new_generator_mappings=false
```

Database

- You may have missing columns in the tables mapped to your `GenericLocalizedGenericListItem` entities. Please check them out.
- The position in `GenericLocalizedGenericListItems` is not nullable anymore. Execute this for each table:

```
update XXX set position=0 where position is null;
```

- The hash generated for foreign key constraints name has changed. Therefore, you will probably end up with duplicate foreign keys. After checking that this is effectively the case, you can use the following query to generate a cleanup script:

```
SELECT
    'ALTER TABLE ' || pclsc.relname || ' DROP CONSTRAINT ' || pc.conname || ';'
FROM
    (
    SELECT
         connamespace,conname, unnest(conkey) as "conkey", unnest(confkey)
          as "confkey" , conrelid, confrelid, contype
     FROM
        pg_constraint
    ) pc
    JOIN pg_namespace pn ON pc.connamespace = pn.oid
    -- and pn.nspname = 'panmydesk4400'
    JOIN pg_class pclsc ON pc.conrelid = pclsc.oid
    JOIN pg_class pclsp ON    pc.confrelid = pclsp.oid
    JOIN pg_attribute pac ON pc.conkey = pac.attnum and pac.attrelid =     pclsc.
↪oid
    JOIN pg_attribute pap ON pc.confkey = pap.attnum and pap.attrelid = pclsp.oid
WHERE pc.conname ilike 'fk\_%' or pc.conname ilike '%_fkey'
ORDER BY pclsc.relname;
```

- The hash generated for unique constraints name has changed when using table level annotation (`uk_mykeyhash` becomes `ukmykeyhash`). Therefore, you will probably end up with duplicate unique constraints. After checking that this is effectively the case, you will need to identify them and create a cleanup script. To identify these constraints, you should search for `@UniqueConstraint` annotation references in your project.
- If the application is old, you might even have a third naming scheme which you can detect with the following query:

```
SELECT
    'ALTER TABLE ' || pclsc.relname || ' DROP CONSTRAINT ' || pc.conname || ';'
FROM
    (
    SELECT
         connamespace,conname, unnest(conkey) as "conkey", unnest(confkey)
          as "confkey" , conrelid, confrelid, contype
     FROM
        pg_constraint
    ) pc
    JOIN pg_namespace pn ON pc.connamespace = pn.oid
    -- and pn.nspname = 'panmydesk4400'
    JOIN pg_class pclsc ON pc.conrelid = pclsc.oid
    JOIN pg_class pclsp ON  pc.confrelid = pclsp.oid
    JOIN pg_attribute pac ON pc.conkey = pac.attnum and pac.attrelid = pclsc.oid
    JOIN pg_attribute pap ON pc.confkey = pap.attnum and pap.attrelid = pclsp.oid
WHERE char_length(pc.conname) = 18 and pc.conname ilike 'fk%'
ORDER BY pclsc.relname;
```

Wicket Resource Security

Until now security context was not set in Wicket Resource because we used this snippet:

```
<security:http pattern="/wicket/resource/**" security="none" />
```

However, since `DropDownChoice` may now use Wicket Resources to fetch data:

1. We need a security context for some of the resources (e.g. to retrieve current authenticated user, or to prevent some users to access that resource)
2. We need to take care of which resources are publicly accessible

That's why you should now use `intercept-url` to protect resources. Add something like this before your default `security:http`:

```
    <!-- An entry point to respond with a 403 error if Spring Security wants the user␣
↪to log in.
        Useful in situations where loging in is not an option, such as when serving␣
↪CSS.
     -->
    <bean id="entryPoint403" class="org.springframework.security.web.authentication.
↪Http403ForbiddenEntryPoint"/>

    <security:http request-matcher="regex"
            pattern="^/wicket/resource/.*"
            create-session="never" entry-point-ref="entryPoint403" authentication-
↪manager-ref="authenticationManager"
            auto-config="false" use-expressions="true">
        <security:headers disabled="true"/>
        <security:csrf disabled="true"/>

        <security:intercept-url pattern="^/wicket/resource/fr.openwide.core.basicapp.
↪web.application.common.template.js.[^/]+.*" access="hasRole('ROLE_ANONYMOUS')" />
        <security:intercept-url pattern="^/wicket/resource/fr.openwide.core.basicapp.
↪web.application.common.template.styles.[^/]+.*" access="hasRole('ROLE_ANONYMOUS')" /
↪>
```

```
        <security:intercept-url pattern="^/wicket/resource/fr.openwide.core.basicapp.
↪web.application.common.template.images.[^/]+.*" access="hasRole('ROLE_ANONYMOUS')" /
↪>
        <security:intercept-url pattern="^/wicket/resource/fr.openwide.core.basicapp.
↪web.application.[^/]+.*" access="hasRole('ROLE_AUTHENTICATED')" />
        <security:intercept-url pattern="^/wicket/resource/.*" access="hasRole('ROLE_
↪ANONYMOUS')" />
    </security:http>
```

Please note that, if you have to make some other resources publicly available (for example on the login page), you should change the above to suit your needs. As is, only JS files, CSS files, static image files and Resources defined in packages other than those of your app (OWSI-Core, various dependencies like Select2) are made publicly available.

Ajax confirm link builder

- `AjaxConfirmLink#build(String)` and `AjaxConfirmLink#build(String, IModel<O>)` no longer exist. Use `AjaxConfirmLink#build()` instead.
- `AjaxConfirmLinkBuilder#create()` no longer exists. Use `AjaxConfirmLinkBuilder#create(String)` or `AjaxConfirmLinkBuilder#create(String, IModel<O>)`.
- `AjaxConfirmLinkBuilder#onClick(SerializableFunction<AjaxRequestTarget, Void>)` and `AjaxConfirmLinkBuilder#onClick(AjaxResponseAction)` no longer exist. Use `AjaxConfirmLinkBuilder#onClick(IOneParameterAjaxAction<IModel<O>>)` or `AjaxConfirmLinkBuilder#onClick(IAjaxAction)` (no parameters) instead. You can use `AbstractAjaxAction` or `AbstractOneParameterAjaxAction`.

CHAPTER 2

Migrating to 0.12

This guide aims at helping OWSI-Core users migrate an application based on OWSI-Core 0.11 to OWSI-Core 0.12.

In order to migrate from an older version of OWSI-Core, please refer to Migrating to 0.11 first.

Tools

Animal sniffer

JDK level validation using Animal sniffer is now enabled by default. This will probably force you to use JDK7 to build your project.

If the default JDK version (1.7) does not suit you, you should:

- change the value of the `jdk.version` property (as usual)
- and change the value of the `jdk.signature.artifactId` property to match one of the artifacts found here: http://search.maven.org/#search|gal|1|g%3A%22org.codehaus.mojo.signature%22

If this is somehow impossible and you want to disable these checks completely, you should disable the Animal Sniffer execution with id "check-java-version".

Bindings & code processors

There has been some changes regarding code processors. You will have to replace the goals of your `*.launch` files: instead of `generate-sources`, use `generate-sources generate-test-sources`.

External changes (libraries)

Spring & Spring Security

- change the `-4.0.xsd` schema to `-4.1.xsd` (Spring Security namespaces)

Wicket

- It seems like Wicket changed the implementation of URL mapping. There isn't many side effects, but one of them is the following: if you have mounted your two-parameter page twice, once with a trailing "/${param1}/" and once with a trailing "/${param1}/#{param2}", then Wicket will fail miserably and perform infinite redirections.
  - That's why it is now recommended, **for every page whose URL ends with a path parameter**, to mount the page with **no trailing slash**. Then the case mentioned above will work as a charm, and this will have the added benefit of allowing clients to use both versions of the URL: with or without a trailing slash.
  - Please note that if you skip the trailing slash for a page whose URL **does not** end with a path parameter, then Wicket will not allow accessing this page with a trailing slash (which is probably a bug). So **do not do this for pages whose URL does not end with a path parameter**.

Internal changes

Core

- Some classes' attributes have been renamed:
- `GenericEntityReference.getEntityId()` became `GenericEntityReference.getId()`
- `GenericEntityReference.getEntityClass()` became `GenericEntityReference.getType()`
- `HistoryValue.getEntityReference()` became `HistoryValue.getReference()`
- This could cause column name changes in your schema
- See https://github.com/openwide-java/owsi-core-parent/commit/33173617a905bdb110ee840acdad46fd20127b7f for details
- There's been some changes around notification descriptors in order to allow for applications to define user-specific context (`fr.openwide.core.spring.notification.model.INotificationContentDescriptor.withContext(INotificationRecipient)`). Thus:
- You're encouraged to use `NotificationContentDescriptors.explicit("defaultSubject", "defaultTextBody", "defaultHtmlBody")` as your default descriptor in your `EmptyNotificationContentDescriptorFactoryImpl`.
- Your notification content descriptor factories should not have a generic return type anymore, they should simply return `INotificationContentDescriptor`. Check out `NotificationDemoPage` and `ConsoleNotificationDemoIndexPage` from the basic application and use similar code in your own `NotificationDemoPage` and `ConsoleNotificationDemoIndexPage` in order to not depend on `IWicketNotificationDescriptor` anymore.
- JPAModelGen support is dropped in IGenericEntityDao/GenericEntityDaoImpl (*ByField) ; queries should be written with QueryDSL API, or compatibility layer may be extracted from GenericEntityDaoImpl 0.11.

Security

The unauthorized access mechanisms have been revamped, for more consistency:

- `AccessDeniedPage` is now accessed whenever a Wicket authorization error occurs.
- It is now clearer that `AccessDeniedPage` is **not** used when an anonymous user tries to access a protected resource.
- OWSI-Core's own redirection mechanism has been deprecated in favor of more standard ones (Wicket's and Spring Security's). On this particular subject, see UI Redirecting.

In order for your application to continue to work properly:

- You will need to add both `REQUEST` and `FORWARD` dispatchers to your Wicket application's filter mapping, so that Spring Security may forward requests when an access is denied.

  This:

```
<filter-mapping>
    <filter-name>MyApplication</filter-name>
    <url-pattern>/*</url-pattern>
</filter-mapping>
```

  will have to become this:

```
<filter-mapping>
    <filter-name>MyApplication</filter-name>
    <url-pattern>/*</url-pattern>
    <dispatcher>REQUEST</dispatcher>
    <dispatcher>FORWARD</dispatcher>
</filter-mapping>
```

- If you overrode the default exception mapper, and you did not extend `CoreDefaultExceptionMapper` you may want to add this at the very top of your `map` method, outside of any `try` block:

```
    if (e instanceof AuthorizationException) {
        throw new AccessDeniedException("Access denied by Wicket's security layer
↪", e);
    }
```

  This will translate a wicket exception into something Spring Security can understand.

- If you overrode the default exception mapper, and you **did** extent `CoreDefaultExceptionMapper`, beware that your call to `super.map(e)` may now throw an `org.springframework.security.access.AccessDeniedException` which should not be caught. Ensure this call is made outside of a `try` block.

Webapp

- Some more logs have been added to `GenericEntityModel` and `AbstractThreadSafeLoadableDetachableModel`. See https://github.com/openwide-java/owsi-core-parent/wiki/UI-Models#debugging for more information.
- `DynamicImages`, obtained through `IImageResourceLinkGenerators`, now have their anticache parameter disabled by default. This may increase performance in Ajax refreshes where the same image appears multiple times. But it also means you will have to add a sensible anticache parameter to your image resources, such as `?t=<the last time your image was changed>`. You may do this when building your link

descriptor, for instance with `fr.openwide.core.wicket.more.link.descriptor.builder.state.parameter.chosen.common.IOneChosenParameterState.renderInUrl(String, AbstractBinding<? super TChosenParam1, ?>)`.

- `IFormModelValidator` now extends `IDetachable`. You should implement `detach` as necessary.
- `ModelValidatingForm.addFormModelValidator(IFormModelValidator, IFormModelValidator ...)` has been renamed to simply `add`.
- `ModelValidatingForm.addFormModelValidator(Collection)` has been removed.
- `GenericEntityCollectionView` has been deprecated in favor of the more generic `CollectionView`. See `GenericEntityCollectionView`'s javadoc for information on migrating existing code.
- `SerializedItemCollectionView` has been deprecated in favor of the more generic `CollectionView`. See `SerializedItemCollectionView`'s javadoc for information on migrating existing code.
- `GenericEntity` collection models (`GenericEntityArrayListModel`, `GenericEntityTreeSetModel`, ...) have been deprecated in favor of the more generic `CollectionCopyModel`. See each older model's javadoc for information on migrating existing code.
- `IWicketContextExecutor` has been deprecated in favor of the more flexible `IWicketContextProvider`. Here are the main consequences to existing applications:
  - An object of type `IWicketContextProvider` is now available in the Spring context. You may `@Autowire` it in your own beans, or redefine it by overriding `fr.openwide.core.wicket.more.config.spring.AbstractWebappConfig.wicketContextProvider(WebApplication)` in your own webapp configuration.
  - The signature of `fr.openwide.core.wicket.more.config.spring.AbstractWebappConfig.wicketContextExecutor(WebApplication)` has changed and is now `fr.openwide.core.wicket.more.config.spring.AbstractWebappConfig.wicketContextExecutor(IWicketContextProvider)`. It cannot be overridden anymore. Please override `fr.openwide.core.wicket.more.config.spring.AbstractWebappConfig.wicketContextProvider(WebApplication)` instead.
  - Classes extending `AbstractWicketRendererServiceImpl`, `AbstractNotificationContentDescriptor` `AbstractNotificationUrlBuilderServiceImpl`, `AbstractNotificationPanelRendererServiceI` must now provide a `IWicketContextProvider` to their super constructor instead of a `IWicketContextExecutor`.
  - Classes extending `AbstractBackgroundWicketThreadContextBuilder` should instead rely on a `IWicketContextProvider`.
  - Classes relying on a `IWicketContextExecutor` should instead rely on a`IWicketContextProvider`. Here are a few examples of code refactoring:

```
String result = wicketExecutor.runWithContext(
    new Callable<String>() {
        public String call() throws Exception {
            return doSomethingThatRequiresAWicketContext();
        }
    },
    locale
);
```

becomes

```
String result;
try (ITearDownHandle handle = wicketContextProvider.context(locale).open()) {
```

```
    result = doSomethingThatRequiresAWicketContext();
}
```

```
And if you really must use a `Callable`:
```

```
String result = wicketExecutor.runWithContext(someCallable, locale);
```

becomes

```
String result = wicketContextProvider.context(locale).run(someCallable);
```

- `IOneParameterConditionFactory`, `IOneParameterModelFactory`, `AbstractOneParameterConditionFactory` and `AbstractOneParameterModelFactory` have been deprecated and replaced by `IDetachableFactory` and `AbstractDetachableFactory`. See their respective Javadoc for more information on migrating existing code.
- `LinkDescriptorBuilder`'s syntax changed slightly.

Previously, the entry point to building a link descriptor was `LinkDescriptorBuilder#LinkDescriptorBuilder()`, which was followed by a call to determine the type of the target, then various stuff around parameters, and finally a call to the `build()` method.

Now, the entry point is `LinkDescriptorBuilder#start()`, followed by various stuff around parameters, and finally a call to one of the build methods: `page`, `resource`, or `imageResource`.

The old syntax is still valid, but has been deprecated and will be removed in the future.

So something that we previously wrote this way:

```
    public static final IOneParameterLinkDescriptorMapper<IPageLinkDescriptor, User>␣
↪MAPPER =
            new LinkDescriptorBuilder().page(MyPage.class)
                    .model(User.class).map("id").permission(READ)
                    .build();
```

... will now have to be written this way:

```
    public static final IOneParameterLinkDescriptorMapper<IPageLinkDescriptor, User>␣
↪MAPPER =
            LinkDescriptorBuilder.start()
                    .model(User.class).map("id").permission(READ)
                    .page(MyPage.class);
```

For existing projects, this perl script may help. Execute it this way from your project root:

```
find . -name '*.java' | xargs -n 1 -I{} bash -c 'TMP="$(mktemp)" ; perl -- ../
↪linkdescriptorbuilder_migrate.pl "{}" > $TMP ; diff -Zq "$TMP" {} >/dev/null || mv "
↪$TMP" {}'
```

It will try its best to convert most uses of `new LinkDescriptorBuilder` to `LinkDescriptorBuilder.start()`, converting early target definitions to late target definitions in the process. It should leave compilation errors wherever the conversion was not easy enough, so you can detect the places where you should edit code manually.

- **IPageLinkGenerator** implementations enforce permission checking in `getValidPageClass()`, hence in `fullUrl()` ; if you used `fullUrl()` to bypass permission checking (for example for email notification sent to another user than the one connected), replace `fullUrl()` by `bypassPermissions().fullUrl()`. NOTE: this backward compatibility is available only on former implementations,

`CorePageInstanceLinkGenerator` and `CorePageLinkDescriptorImpl`; if you use newer implementions, you already should conform to the new behavior.

- **Ajax confirm link builder**: Ajax confirm link builder is now « form submit » aware ; current AjaxSubmitLink may be rewritten with `AjaxConfirmLink.build().[...].submit(form)`. AjaxSubmitLink still available.
- **Confirm link builder**: introduced `ConfirmLink.build()` builder. Unified syntax with ajax confirm link and ajax confirm submit. Confirm submit not supported (it was already a missing functionnality).
- **Confirm link**: introduced custom styles for yes / no buttons. Default values constructors were added to enable back-compatibility.
- **Condition and behavior**: EnclosureBehavior and PlaceholderBehavior are deprecated and replaced by behavior generation's methods on `Condition` object. This pattern allows to use more easily and consistently any `Condition` to control component's visibility or enabled property. More documentation on this pattern and the way to rewrite your code UI Placeholder and Enclosure

CHAPTER 3

Migrating to 0.13

This guide aims at helping OWSI-Core users migrate an application based on OWSI-Core 0.12 to OWSI-Core 0.13.

In order to migrate from an older version of OWSI-Core, please refer to Migrating to 0.12 first.

owsi-core version numbering policy

owsi-core version 0.12.5 is the last version published exclusively for jdk 7. From owsi-core 0.13, vanilla versions will be built for java 1.8, and jdk 8 dependant starting owsi-core 0.14.

owsi-core 0.13.0 is planned to be a 0.12.5 isofunctionnal release, but published both for jdk 7 and 8.

Knowing this, you have two solutions for migrating from 0.12 to 0.13 : migrate to jdk 8 at the same time or keep a jdk 7 environment.

Solution 1: continue to use jdk 7

This solution allows you to upgrade project towards post or equals 0.13 release in a jdk 7 environment.

Please note that this version can be run in a java 8 environment.

Note: If your code base is already upgraded toward jdk7 version, you can switch directly to step 3 to upgrade your Eclipse's configuration.

Step 1 · udpate dependencies

Change project parent and owsi-core version to switch to jdk 7 dependency:

```
<parent>
    <groupId>fr.openwide.core.parents</groupId>
    <artifactId>owsi-core-parent-core-project-jdk7</artifactId>
    <version>0.13.jdk7-SNAPSHOT</version>
</parent>

[...]

<properties>
    <owsi-core.version>0.13.jdk7-SNAPSHOT</owsi-core.version>
</properties>
```

Note: owsi-core 0.13 codebase and all its upgrades will continue to support the use of java 1.7. You just have to add the **.jdk7** modifier after the version. Therefore you'll need to use **0.13.0.jdk7**, **0.13.1.jdk7**... owsi-core SNAPSHOT version will be flagged **0.13.jdk7-SNAPSHOT**.

Step 2 · clean your codebase

Perform a global maven clean so that generated source code is cleaned.

No more steps are needed to enable maven build.

Step 3 (for Eclipse IDE) · install m2e integration and reconfigure m2e-apt

Code generation, configured with maven-processor-plugin, is modified to facilitate m2e plugin integration. It is now recommended to use jboss m2e maven-processor-plugin integration.

Plugin intallation's instructions are available here: https://github.com/jbosstools/m2e-apt

It is easily installable via Eclipse Marketplace: m2e-apt (at least from Eclipse 4.5.2)

It is installed by default in Eclipse's team bundles from version 4.6.0

In *Windows → Preferences → Maven → Annotation Processing*, choose *Experimental: Delegate annotation processing to maven plugins...* (this option is known to work correctly for our use-cases)

If not done automatically, you need to reconfigure Maven projects (right-click on parent project, *Maven → Update project... → OK*

Note: Nothing to do with jdk version, but you may need to pay attention to the following setting : Windows → Preferences → Team → Git → Projects → **Uncheck** Automatically ignore derived resources by adding theme to .gitignore ; this setting prevents Eclipse to alter .gitignore configurations

Step 4 · optimization

If source code generation is to heavy on your project, you can restrain regeneration on project reconfiguration by adding the following m2e's configuration in your parent pom.xml (*dependencyManagement* section)

```
<pluginsManagement>
  <plugins>
    <plugin>
```

```
      <groupId>org.eclipse.m2e</groupId>
      <artifactId>lifecycle-mapping</artifactId>
      <version>1.0.0</version>
      <configuration>
             <lifecycleMappingMetadata>
               <pluginExecutions>
                 <pluginExecution>
                   <pluginExecutionFilter>
                     <groupId>org.bsc.maven</groupId>
                     <artifactId>maven-processor-plugin</artifactId>
                     <versionRange>[0,)</versionRange>
                     <goals>
                       <goal>process</goal>
                       <goal>process-test</goal>
                     </goals>
                   </pluginExecutionFilter>
                   <action>
                     <execute>
                       <runOnConfiguration>true</runOnConfiguration>
                       <runOnIncremental>false</runOnIncremental>
                     </execute>
                   </action>
                 </pluginExecution>
               </pluginExecutions>
             </lifecycleMappingMetadata>
      </configuration>
    </plugin>
  </plugins>
</pluginManagement>
```

Solution 2 · switch to jdk 8 version

This version use the same code base than jdk 7 (for 0.13 versions), but use a jdk 8 runtime. As jdk 7 version is compatible with jdk 8, this version is mainly provided to prepare your migration to jdk 8.

Step 1 · update dependencies

Simply make sure you use a post or equals 0.13 owsi-core version (without the .jdk7 modifier).

Step 2 to 4

Follow the same steps 2 to 4 than jdk 7 version.

CHAPTER 4

Migrating to 0.14

This guide aims at helping OWSI-Core users migrate an application based on OWSI-Core 0.13 to OWSI-Core 0.14.

In order to migrate from an older version of OWSI-Core, please refer to Migrating to 0.13 first.

Java

This version only supports **Java 8**.

External changes (libraries)

Poi

- `HSSFColor.WHITE.index` is now `HSSFColorPredefined.WHITE.getIndex()`.

Spring & Spring Security

- `isTrue(boolean)` from the type Assert is now `isTrue(boolean, String)` with String = the exception message to use if the assertion fails.
- `notNull(boolean)` from the type Assert is now `notNull(boolean, String)` with String = the exception message to use if the assertion fails.

For the upgrade of Spring Security we had to update the schema from `spring-security-4.1.xsd` to `spring-security-4.2.xsd`.

Guava

- `CharMatcher.WHITESPACE` is now `CharMatcher.whitespace()`.

Hibernate

- `session.setFlushMode(FlushMode)` is now `session.setHibernateFlushMode(FlushMode)`.
- `SessionImplementor` class is replaced by `SharedSessionContractImplementor` class.

The AvailableSettings libray now is `org.hibernate.cfg.AvailableSettings` instead of `org.hibernate.jpa.AvailableSettings`.

- `AvailableSettings.SHARED_CACHE_MODE` is now `AvailableSettings.JPA_SHARED_CACHE_MODE`.
- `AvailableSettings.VALIDATION_MODE` is now `AvailableSettings.JPA_SHARED_CACHE_MODE`.

The EmbeddableTypeImpl library is now `org.hibernate.metamodel.internal.EmbeddableTypeImpl` instead of `org.hibernate.jpa.internal.metamodel.EmbeddableTypeImpl`.

The upgrade of hibernate-core forced us to explicitly specify the **default_schema** for the database. Every tables are created in this schema and it is no longer based on the search_path from PostgreSQL configuration.

By default, default_schema = db_user. If you need to change it, you have to add the variable `hibernate.defaultSchema` in `owsi-core-component-jpa.properties` and its value will override the default value.

Class `PostgresqlSequenceStyleGenerator` is renamed `PerTableSequenceStyleGenerator` as it is not postgresql-related; class content is unchanged. If you use it, just retarget the new class.

Hibernate Search

Hibernate Search & Lucene

We have upgraded Hibernate Search to the 5.7.0.Final which is not yet compatible with Lucene 6 but requires at least Lucene 5.5.X so we have upgraded Lucene to the 5.5.4 version.

The utilization of `setBoost(float)` and `getBoost()` directly to a Query is now deprecated. Instead we use the type `BoostQuery` to apply boost.

Configuration

- The `ExplicitJpaConfigurationProvider` class no longer exists, all the configuration is now exclusively provided by the `DefaultJpaConfigurationProvider` class.

Behavior checking

Some structural changes are done so that old applications are not broken. Make sure that expected behavior is still here:

- **Hibernate:** database's sequence is now handled with the *new-style* hibernate configuration. Verify that the sequence are style named *table_pk_seq*. Give a special attention to your specialized configurations:
  - ensure that *hibernate.id.new_generator_mappings=true* (if you do not override this setting, it is fine)
  - custom *@GeneratedValue.strategy()*
  - custom *@GeneratedValue.generator()*
  - custom *@SequenceGenerator*

  - custom @*GenericGenerator*

Hibernate Search & ElasticSearch

You can now choose between Lucene and ElasticSearch for your Hibernate Search requests. In order to do use ElasticSearch, you have first to install ElasticSearch 2.4.

Secondly, you have to specify 3 things in the file **app-core/configuration.properties** :

```
##
## Hibernate search Elasticsearch
##
hibernate.search.elasticsearch.enabled=true
hibernate.search.default.elasticsearch.host=http://127.0.0.1:9310
hibernate.search.default.elasticsearch.index_schema_management_strategy=CREATE
```

You have to set the first line value to `true` to enable ElasticSearch. The second line is the address of your installed ElasticSearch, and finally the third line is schema management strategy.

Lucene and ElasticSearch analyzers

Now that the analyzers are changing when you switch between Lucene and ElasticSearch, they are no longer in the annotation form in the class `Parameter.java`. You can find them respectively in `CoreLuceneAnalyzersDefinitionProvider.java` and `CoreElasticSearchAnalyzersDefinitionProvider.java`.

Due the exportation of analyzers definitions in external separate classes, you can add your own analyzers definitions by extending one of these two classes and override the function `register`. After that, you have to add a property in the file `hibernate-extra.properties` (create this file if it doesn't exists). If you want to use your own ElasticSearch analyzers add this line :

```
hibernate.search.elasticsearch.analyzer_definition_provider=package.to.yourclass.
↪ClassName
```

If you want to use your own Lucene analyzers add this line :

```
hibernate.search.lucene.analyzer_definition_provider=package.to.yourclass.ClassName
```

Note that when you choose to use ElasticSearch, Lucene's analyzers definitions are still instanciated but only used internally.

Date SortField and ElasticSearch

related commit

In ElasticSearch, Date SortField is of type STRING, but with Lucene, it is of type LONG. If you perform sort with `FullTextQuery.setSort(Sort sort)` with a Date field configured for one of the backends, it'll throw an exception with the other backend.

- **Solution 1:** Use only one backend, and initialize correctly and statically needed SortFields
- **Solution 2:** Use QueryBuilder to build your Sort object. QueryBuilder use field metadata to determine the right type to use.
  - `fr.openwide.core.jpa.search.util.SortFieldUtil` provides examples on the ways to obtain a Sort object or to perform a setSort(...) that use QueryBuilder and circumvent this issue.

- replacing FullTextQuery.setSort(Sort sort) by SortFieldUtil.setSort(...) can be done quickly
- beware that this workaround use field metadata to determine the right type; not deterministic and silent errors may become fatal errors with this workaround.

Wicket

ConsoleConfiguration.build()

`ConsoleConfiguration.build()` parameters are modified; you now need to provide a `IPropertyService`. This method call is generally done in you `<MyApplication>Application.java`. Just add IPropertyService as a `@SpringBean` field, and add it to the method call.

CHAPTER 5

Migrating to 0.15

This guide aims at helping OWSI-Core users migrate an application based on OWSI-Core 0.14 to OWSI-Core 0.15.

In order to migrate from an older version of OWSI-Core, please refer to Migrating to 0.14 first.

Mail header

To migrate from 0.14 to 0.15, you don't need to change a lot of things. In fact, the only issue you may have is if you send mail to users in your application.

In this case, we have made some changes on the mail header.

You will only have to add a line in your configuration.properties which will inquire the sender :

```
notification.mail.sender=my.mail@mail.com
```

CHAPTER 6

Tools

Maven Archetype (TODO)

Code processors

About

Projects based on OWSI-Core, and OWSI-Core itself, use automatically generated code to manipulate data metamodels.

At the moment, there are two metamodels in an application:

- The bindgen metamodel, which allows us to manipulate objects representing bean properties, and use it to access said properties in java code.
- The QueryDSL metamodel, which allows us to manipulate objects representing entity properties, and use it to build JPA queries.

Troubleshooting

Bindgen's bindings code won't compile

In some cases, bindgen will generate code that won't compile. You may ignore code generation for selected attributes by adding `skipAttribute.your.package.YourClass.yourAttribute` to `bindgen.properties`.

There are known issues with bindgen code generation of some classes in OWSI-Core. You may use the following lines to work around these issues. If those are not up-to-date, check out the basic application's bindgen.properties

```
skipAttribute.fr.openwide.core.jpa.business.generic.model.GenericEntityReference.
↪entityClass=true
skipAttribute.fr.openwide.core.commons.util.fieldpath.FieldPath.root=true
```

"Cannot find symbol"

If you spot errors like this in your maven build:

```
[ERROR] diagnostic: /data/home/ANONYMIZED/Documents/ANONYMIZED/livraison/tmp/
↪ANONYMIZED-core/src/main/java/com/ANONYMIZED/core/business/document/dao/FileDaoImpl.
↪java:17: error: cannot find symbol
import com.ANONYMIZED.core.business.document.model.QDocument;
                                                  ^
  symbol:   class QDocument
  location: package com.ANONYMIZED.core.business.document.model
```

... then just ignore these errors. You are in one of these two situations:

- You generated the bindings for the very first time. In that case, the errors are false positives, and if no other error occurred, the bindings should be generated anyhow.
- Another error occurred during the generation of bindings. In that case, you should check out the very last error, which should be different and is the real cause of your generation failure.

Maven

JDK level validation

JDK level validation using Animal sniffer is enabled by default from OWSI-Core 0.12 on.

If the default JDK version (1.7 at the time of this writing) does not suit you, you should:

- change the value of the `jdk.version` property to whatever suits you
- and change the value of the `jdk.signature.artifactId` property to match one of the artifacts found here: http://search.maven.org/#search|ga|1|g%3A%22org.codehaus.mojo.signature%22

If this is somehow impossible and you want to disable these checks completely, you should disable the Animal Sniffer execution with id `check-java-version`.

Deploy to several servers using the maven-deploy-plugin

```
<build>
    <plugins>
        <plugin>
            <groupId>org.codehaus.mojo</groupId>
            <artifactId>wagon-maven-plugin</artifactId>
            <executions>
                <execution>
                    <id>upload-war-to-front1</id>
                    <phase>deploy</phase>
                    <goals>
                        <goal>upload-single</goal>
                    </goals>
                    <configuration>
                        <fromFile>${project.build.directory}/${project.artifactId}-$
↪{project.version}-${assembly.environment}.tar.gz</fromFile>
                        <url>${front1-deployment-url}</url>
                    </configuration>
                </execution>
```

```
                <execution>
                    <id>upload-war-to-front2</id>
                    <phase>deploy</phase>
                    <goals>
                        <goal>upload-single</goal>
                    </goals>
                    <configuration>
                        <fromFile>${project.build.directory}/${project.artifactId}-$
↪{project.version}-${assembly.environment}.tar.gz</fromFile>
                        <url>${front2-deployment-url}</url>
                    </configuration>
                </execution>
            </executions>
        </plugin>
    </plugins>
</build>
```

CHAPTER 7

Security

Securing accesses

This page explains how to ensure that parts of your application (pages, buttons, resources, but also Spring services) are only accessible to entitled users.

Principles

Here are some basic principles. These are not really formal security science, but are just intended to provide readers with enough understanding of OWSI-Core's security to get started.

Security model

Here are the concepts used in OWSI-Core's security model :

- A *user* is a user of your application.
- A *user group* is a business-level category of users.
- An *object*, or *resource*, is the thing whose access is to be secured. It's generally a business domain object (such as a "customer", or a "deal", and so on).
- A *role* is a functional-level category of users. One or many roles may be attributed to a user or to a user group.
- A *global permission* is an approval of a mode of access to a class of objects, an authorization which is not tied to a an object in particular (such as "write to customer contact details"). One or many global permissions may be attributed to a role.
- An *object permission* is an approval of a mode of access to an object in particular, an authorization which **is** tied to a an object in particular (such as "write to the contact details of customer Initech, Inc"). An object permission is never attributed, it is computed in a fully qualified context: given a user, an object and an object permission, the security system will compute the answer to the question "does the user have this permission on this object?".

You may notice that, depending on your point of view, some concepts seem to have the same purpose: either role and global permission or role and user group. The developers are aware of this issue and it will be addressed in a future version of OWSI-Core.

Architecture

OWSI-Core's security layer is powered by Spring Security, and most concepts detailed here come directly from Spring Security.

Here are the main components of a secured application based on OWSI-Core:

- The `ISecurityService` is an API, the main entry point for security-related queries ("does this user have this permission on this object") and operations (user authentication, authentication invalidation, ...).
- The `PermissionEvaluator` is a SPI, the way for the developer to programmatically define code that will determine whether a user has a given permission on a given object. This code is not just a mapping, as it may use business object's properties in order to answer queries: for instance "is the given user this customer's account manager?". **Think of permissions evaluators as a way to extract business information for security purposes**. The permission evaluator is generally implemented through a subclass of `AbstractCorePermissionEvaluator` which delegates its calls to various `IGenericPermissionEvaluator`, one for each type of object.
- The `UserDetailsService` is a SPI, the way for the developer to programmatically define code that will determine whether a user has a given role or global permission. It's generally a subclass of `CoreJpaUserDetailsServiceImpl`. **Think of the user details service as a way to extract user role and permission attributions.** This code consists, most of the time, in:
- extracting role and global permission attributions from the database
- expanding the results to extensive lists with the help of a role hierarchy and a permission hierarchy
- and optionally inferring hard-coded global permissions based on on attributed roles

Defining your own permissions

Defining role constants

Create a class named `<YourApplication>AuthorityConstants`, extending `CoreAuthorityConstants`, and put it in `<Your main package>.core.security.model`. In this class, add one string constant for each role, making sure that each constant has a unique value:

```
public final class MyApplicationAuthorityConstants extends CoreAuthorityConstants {

    public static final String ROLE_INTRANET_USER = "ROLE_INTRANET_USER";
    public static final String ROLE_EXTRANET_USER = "ROLE_EXTRANET_USER";

}
```

Defining permission constants

Create a class named `<YourApplication>PermissionConstants`, extending `CorePermissionConstants`, and put it in `<Your main package>.core.security.model`. In this class, add one string constant for each global permission or object permission, making sure that each constant has a unique value:

```
public final class MyApplicationPermissionConstants extends CorePermissionConstants {
    public static final String CUSTOMER_WRITE_CONTACT_DETAILS = "CUSTOMER_WRITE_
↪CONTACT_DETAILS";
}
```

Then, create another class named `<YourApplication>Permission`, extending NamedPermission, and with the following implementation:

```
public final class MyApplicationPermission extends NamedPermission {

    private static final long serialVersionUID = 8541973919257428300L;

    public static final Collection<MyApplicationPermission> ALL;
    static {
        ImmutableSet.Builder<SIPermission> builder = ImmutableSet.builder();
        Field[] fields = MyApplicationPermissionConstants.class.getFields();
        for (Field field : fields) {
            try {
                Object fieldValue = field.get(null);
                if (fieldValue instanceof String) {
                    builder.add(new MyApplicationPermission((String)fieldValue));
                }
            } catch (IllegalArgumentException|IllegalAccessException ignored) { //␣
↪NOSONAR
            }
        }
        ALL = builder.build();
    }

    private MyApplicationPermission(String name) {
        super(name);
    }

}
```

And finally, override permissionFactory in your security configuration class, which extends `AbstractJpaSecurityConfig`. Here is an implementation example:

```
@Configuration
public class MyApplicationCoreSecurityConfig extends AbstractJpaSecuritySecuredConfig
↪{

    /** ... other stuff ... */

    @Override
    public PermissionFactory permissionFactory() {
        return new NamedPermissionFactory(MyApplicationPermission.ALL);
    }

    /** ... other stuff ... */
}
```

Defining role and permission hierarchies

This is done by overriding `roleHierarchyAsString` and `permissionHierarchyAsString` in your security configuration class, which extends `AbstractJpaSecurityConfig`. Here is an implementation example:

```
import static my.application.core.security.model.MyApplicationAuthorityConstants.*;
import static my.application.core.security.model.MyApplicationPermissionConstants.*;

@Configuration
public class MyApplicationCoreSecurityConfig extends AbstractJpaSecuritySecuredConfig
↪{

    /** ... other stuff ... */

    @Override
    public String roleHierarchyAsString() {
        return defaultRoleHierarchyAsString() + hierarchyAsStringFromMap(
                ImmutableMultimap.<String, String>builder()
                .putAll(
                        ROLE_ADMIN,
                        ROLE_INTRANET_USER,
                        ROLE_TECHNICAL_ADMIN
                )
                .putAll(
                        ROLE_INTRANET_USER,
                        ROLE_AUTHENTICATED
                )
                .putAll(
                        ROLE_EXTRANET_USER,
                        ROLE_AUTHENTICATED
                )
                .putAll(
                        ROLE_SYSTEM,
                        ROLE_ADMIN,
                        ROLE_MAIN_USER,
                        ROLE_EXTRANET_USER
                )
                .build()
        );
    }

    @Override
    public String permissionHierarchyAsString() {
        return defaultPermissionHierarchyAsString() + hierarchyAsStringFromMap(
                ImmutableMultimap.<String, String>builder()
                .put(CUSTOMER_WRITE, CUSTOMER_READ)
                .build()
        );
    }

    /** ... other stuff ... */
}
```

Defining permission evaluators

Then, go to your permission evaluator. You may find a reference to this class in your configuration class that extends `AbstractJpaSecurityConfig`, in the `permissionEvaluator` method.

In this permission evaluator, you will have to dispatch security queries to various permission evaluators, one for each object type. This will look like this:

```
public class MyApplicationPermissionEvaluator extends AbstractCorePermissionEvaluator
↪<User> {

    @Autowired
    private ICustomerPermissionEvaluator customerFormationPermissionEvaluator;

    @Autowired
    private IDealPermissionEvaluator dealPermissionEvaluator;

    @Autowired
    private IInvoicePermissionEvaluator invoicePermissionEvaluator;

    public MyApplicationPermissionEvaluator() {
        // nothing to do
    }

    @Override
    protected boolean hasPermission(User user, Object targetDomainObject, Permission␣
↪permission) {
        if (targetDomainObject != null) {
            targetDomainObject = HibernateUtils.unwrap(targetDomainObject); // NOSONAR
        }

        if (user != null) {
            user = HibernateUtils.unwrap(user); // NOSONAR
        }

        if (targetDomainObject instanceof Customer) {
            return customerPermissionEvaluator.hasPermission(user, (Customer)␣
↪targetDomainObject, permission);
        } else if (targetDomainObject instanceof Deal) {
            return dealPermissionEvaluator.hasPermission(user, (Deal)␣
↪targetDomainObject, permission);
        } else if (targetDomainObject instanceof Invoice) {
            return invoicePermissionEvaluator.hasPermission(user, (Invoice)␣
↪targetDomainObject, permission);
        }


        return false;
    }
}
```

For each type-bound permission evaluator, you will define an interface (which extends `IGenericPermissionEvaluator`) and an implementation. Here is an example of implementation (you are, of course, totally free of which permissions you will or will not handle):

```
@Service
public class CustomerPermissionEvaluatorImpl extends␣
↪AbstractMyApplicationGenericPermissionEvaluator<Customer>
        implements ICustomerPermissionEvaluator {

    @Autowired
    private IUserService userService;

    @Autowired
    private IParticipationPermissionEvaluator participationPermissionEvaluator;
```

```
    @Override
    public boolean hasPermission(User user, Customer customer, Permission permission)
↪{
        if (is(permission, READ)) {
            return hasPermission(user, CUSTOMER_READ);
        } else if (is(permission, CREATE)) {
            return hasPermission(user, CUSTOMER_CREATE);
        } else if (is(permission, WRITE)) {
            return user.equals(customer.getAccountManager());
        }
        return false;
    }
}
```

Restricting accesses

Service layer

General configuration

In order to enable security checks upon method calls, you will need to make sure that your security configuration class does not extend `AbstractJpaSecuritySecuredConfig` directly, but its subclass, `AbstractJpaSecuritySecuredConfig`.

Service access

You will need to add annotations on your services' methods. For instance: s

```
public interface ICustomerService extends IGenericEntityService<Long, Customer> {

    @Override
    @PreAuthorize(value = MyAppSecurityExpressionConstants.CREATE)
    void create(@PermissionObject Customer entity) throws ServiceException,␣
↪SecurityServiceException;

}
```

`@PreAuthorize` will perform a security check before executing the method. Other, more exotic annotations exist in package `org.springframework.security.access.prepost`.

It's better to define your security expressions in a separate constants class, such as `MyAppSecurityExpressionConstants` in this example. This class will look something like that:

```
import static fr.openwide.core.commons.util.security.PermissionObject.DEFAULT_
↪PERMISSION_OBJECT_NAME;

public final class SISecurityExpressionConstants {

    public static final String READ = "hasPermission(#" + DEFAULT_PERMISSION_OBJECT_
↪NAME + ", '" + SIPermissionConstants.READ + "')";
    public static final String CREATE = "hasPermission(#" + DEFAULT_PERMISSION_OBJECT_
↪NAME + ", '" + SIPermissionConstants.CREATE + "')";
    public static final String WRITE = "hasPermission(#" + DEFAULT_PERMISSION_OBJECT_
↪NAME + ", '" + SIPermissionConstants.WRITE + "')";
```

```
}
```

Note that annotating the method's main parameter with `@PermissionObject` and using `PermissionObject.DEFAULT_PERMISSION_OBJECT_NAME` in your security expressions will ensure that changing the name of this parameter will not break your security expressions.

UI layer

General configuration

The general web application security configuration is generally located in a class named `<YourApp>WebappSecurityConfig`.

This class generally refers to an XML file (`security-web-context.xml`) whose content defines:

- security-related beans
- required roles for each page (or set of pages, by using regular expressions)
- login workflow (login page, login failure page, login success page)
- denied access behavior
- session restrictions (such as a maximum number of simultaneous sessions)

The official documentation about the format of this file may be found there: http://docs.spring.io/spring-security/site/docs/4.0.x/reference/html/ns-config.html

Page and resource access

Simple, coarse-grained configuration

You may define, for a given page or resource, which roles or global permissions are required in order to access it.

This is simply done by adding the `@AuthorizeInstantiation` (for roles) or `@AuthorizeInstantiationIfPermission` (for global permissions) on the page's class. For resources, you must use `@AuthorizeResource` instead, and you may not rely on permissions (only roles).

Be aware that, while more annotations are available (`@AuthorizeAction` and `@AuthorizeActionIfPermission` in particular), their use is discouraged because they add restrictions which cannot be checked until the very last moment. This prevents in particular from disabling links to inaccessible pages (because, when rendering the link, the page is not yet instantiated and thus we can't check action permissions on this page).

Advanced, fine-grained configuration

Most of the time, you will use link descriptors (see UI-Links) in order to provide access to pages or resources.

Link descriptors allow to define arbitrary access restrictions (based on model objects, or on anything you want), and this includes in particular authorization restrictions.

For instance, this allows to make a link accessible only when the users has the "READ" permission on the parameter:

```
public static final IOneParameterLinkDescriptorMapper<IPageLinkDescriptor, MyObject>␣
↪MAPPER =
        new LinkDescriptorBuilder()
        .page(MyObjectPage.class)
        .model(MyObject.class)
                .permission(CorePermissionConstants.READ)
                .map(CommonParameters.ID).mandatory()
        .build();
```

You may also enforce checks on global permissions, by calling the `.permission` method before defining any parameter:

```
public static final IOneParameterLinkDescriptorMapper<IPageLinkDescriptor, MyObject>␣
↪MAPPER =
        new LinkDescriptorBuilder()
        .page(MyObjectPage.class)
        .permission(MyApplicationPermissionConstants.ACCESS_MY_OBJECT_PAGE)
        .model(MyObject.class)
                .map(CommonParameters.ID).mandatory()
        .build();
```

Please note that all of this also applies to resource link descriptors.

With this configuration, checks will be performed upon link rendering and upon page/resource instantiation:

- when links are rendered, they will be automatically disabled or hidden if the user misses some roles or permissions
- when the a page or resource is instantiated, it will use the link descriptor to extract parameters, which will trigger an exception and abort the page instantiation if the user misses some roles or permissions.

Buttons/links access

When using links created from a link descriptor, if this link descriptor has been properly configured as explained above, the link will automatically be disabled whenever the user hasn't the required permissions.

For other links (external links for instance) or for buttons, ajax links, and so on, you may hide or disable these components using enclosure behaviors:

```
final IModel<T> model = /*...*/;
add(
        new AjaxLink("link", model) {
            /* ... */
        }
                .add(
                        new EnclosureBehavior().condition(
                                Condition.permission(model,␣
↪MyApplicationPermissionConstants.MY_PERMISSION)
                        )
                )
);
```

This will trigger server-side hiding, which will prevent users to trigger the server-side code even if they can guess and call the URL for each button: Wicket refuses to execute code on components that were hidden on the server side.

Popups/modals access

For modals which require initialization before showing them, you should add an enclosure behavior on the opening link:

```
MyModal editPopup = new MyModal("popup");
add(editPopup);

// The following code is potentially executed multiple times, for different models
final IModel<T> itemModel = /*...*/;
add(
        new BlankLink("edit")
                .add(
                    new AjaxModalOpenBehavior(editPopup, MouseEvent.CLICK) {
                        private static final long serialVersionUID = 1L;
                        @Override
                        protected void onShow(AjaxRequestTarget target) {
                            editPopup.init(itemModel.getObject());
                        }
                    }
                )
                .add(
                        new EnclosureBehavior().condition(
                                Condition.permission(model,␣
↪MyApplicationPermissionConstants.MY_PERMISSION)
                        )
                )
);
```

This ensures that the modal will be initially visible, but unusable (because it's not initialized), and that it will be "openable" if and only if at least one button is visible.

For modals whose content is fully determined by their main model, and which do not require initialization upon showing them, it is recommended to apply an enclosure behavior on the modal itself:

```
IModel<T> model = /*...*/;
MyModal editPopup = new MyModal("popup", model);
add(
        editPopup
                .add(
                        new EnclosureBehavior().condition(
                                Condition.permission(model, SIPermissionConstants.
↪WRITE)
                        )
                )
        new BlankLink("edit")
                .add(new AjaxModalOpenBehavior(editPopup, MouseEvent.CLICK))
);
```

This ensures that when the use has no access to the modal, even if the client tries to execute a manually-crafted ajax call to open the modal, the modal will be hidden on the server-side and wicket will thus trigger an error.

CHAPTER 8

Backend

Querying

This page explains how to query data using OWSI-Core.

How to expose queries to the web application

Through an `ISearchQuery`

Use case

`ISearchQuery` should be used when providing a search form to users. It makes it easy to define a search query with numerous search criteria that are independent from each other, the ability to sort the result, and the ability to retrieve the paginated result or the result count.

`ISearchQuery` is also commonly used to implement "autocomplete" queries, i.e. the queries behind Select2 select boxes.

Description

`ISearchQuery` is an interface that provides read access to the data while hiding the implementation details, as does a DAO. But on contrary of a DAO:

- Each interface extending `ISearchQuery` provides one method for each search criterion, which will be kept in memory until data retrieval (`list`, `count`). Note that this "keeping in memory" might not be done explicitly by implementors, but just by starting the query building with the query framework under the hood.
- For this reason, an `ISearchQuery` instance is stateful and can only be used for **one** query (i.e. search criteria may be added, but not removed nor cleared).
- `ISearchQuery`s are expected to be used directly from the UI layer

Here is an example of `ISearchQuery`-extending interface:

```
public interface IPersonSearchQuery extends ISearchQuery<Person, PersonSort> {

    IPersonSearchQuery quickSearch(String filter);

    IPersonSearchQuery lastName(String lastName);

    IPersonSearchQuery firstName(String firstName);

    IPersonSearchQuery company(Company company);
}
```

And here is an example of use (from inside a Wicket DataProvider):

```
return createSearchQuery(IPersonSearchQuery.class) // Some Spring magic (beanFactory.
↪getBean(...))
        .lastName(lastNameModel.getObject())
        .firstName(firstNameModel.getObject())
        .company(companyModel.getObject())
        .sort(sortModel.getObject())
        .list(offset, limit)
```

Through a service

Use case

Queries should be exposed to the web application through services (a simple method in a service) when:

- counting the results is not necessary (within a service, that would involve providing two service methods with the same parameters and implementing them, which is a bit of a pain)
- **and**
- paging is not necessary
- **or** criteria are strongly interrelated (they can't be implemented by separate criteria in the underlying query)
- **or** the query is really too simple to justify the overhead of creating an interface and an implementation class

Note that in any of these case, `ISearchQuery` *could* still be used. It's just a matter of guessing whether using `ISearchQuery` would help in implementing your query or not (spoiler: it probably does).

Also, know that in the case of complex queries (reporting for instance), a `ISearchQuery` is still (and maybe more) relevant, since you may just store arguments passed to criteria methods in attributes and use those attributes later when asked for the results. This brings the advantage of consistency with little implementation cost.

Exposing sort selection

Whatever the solution you choose among the two above, you may have to provide clients a way to tune the sorting of retrieved data.

In OWSI-Core, this is generally done by adding a parameter to your query that is a `Map<S, SortOrder>` with `S extends ISort<F>` and with `F` being implementation-dependent. `ISort` will allow the implementor to convert the business-level sort definitions into an internal list of fields on which to sort.

`ISorts` are simple business wrappers. Each `ISort` instance by provides a list of sort "fields" (whose type is implementation-dependent).

The implementations are generally an enum type:

```
public enum PersonSort implements ISort<SortField> {

    SCORE {
        @Override
        public List<SortField> getSortFields(SortOrder sortOrder) {
            return GenericEntitySort.SCORE.getSortFields(sortOrder);
        }
        @Override
        public SortOrder getDefaultOrder() {
            return GenericEntitySort.SCORE.getDefaultOrder();
        }
    },
    ID {
        @Override
        public List<SortField> getSortFields(SortOrder sortOrder) {
            return GenericEntitySort.ID.getSortFields(sortOrder);
        }
        @Override
        public SortOrder getDefaultOrder() {
            return GenericEntitySort.ID.getDefaultOrder();
        }
    },
    LAST_NAME {
        @Override
        public List<SortField> getSortFields(SortOrder sortOrder) {
            return ImmutableList.of(
                    SortUtils.luceneSortField(
                            this, sortOrder, SortField.Type.STRING,
                            Ressortissant.LAST_NAME_SORT
                    )
            );
        }
        @Override
        public SortOrder getDefaultOrder() {
            return SortOrder.ASC;
        }
    },
    FIRST_NAME {
        @Override
        public List<SortField> getSortFields(SortOrder sortOrder) {
            return ImmutableList.of(
                    SortUtils.luceneSortField(
                            this, sortOrder, SortField.Type.STRING,
                            Ressortissant.FIRST_NAME_SORT
                    )
            );
        }
        @Override
        public SortOrder getDefaultOrder() {
            return SortOrder.ASC;
        }
    },
    FULL_NAME {
        @Override
        public List<SortField> getSortFields(SortOrder sortOrder) {
            return ImmutableList.of(
                    SortUtils.luceneSortField(
```

```
                        this, sortOrder, SortField.Type.STRING,
                        Ressortissant.LAST_NAME_SORT
                    ),
                    SortUtils.luceneSortField(
                        this, sortOrder, SortField.Type.STRING,
                        Ressortissant.FIRST_NAME_SORT
                    )
            );
        }
        @Override
        public SortOrder getDefaultOrder() {
            return SortOrder.ASC;
        }
    };

    @Override
    public abstract List<SortField> getSortFields(SortOrder sortOrder);

    @Override
    public abstract SortOrder getDefaultOrder();

}
```

Note that, on the UI side, an utility exists to easily manage a sort selection: `CompositeSortModel`. See UI-Models for more information.

How to implement queries

Search queries (`ISearchQuery`)

You may always use your own implementation. But in most cases, extending one of the two provided abstract classes is the way to go.

WARNING: always think to add `@Scope("prototype")` to your implementation, else you will experience very disturbing concurrent modification issues.

`AbstractHibernateSearchSearchQuery`

`AbstractHibernateSearchSearchQuery` provides sensible protected methods that allow you to stack criteria on each call of a criterion method. For convenience, most of those utility methods have no effect when given `null` parameters. This allow clients to skip null-checks entirely and to call your criteria methods regardless of whether or not the users provided a value for each parameter.

Some full implementations already exist in OWSI-Core (most notably for `fr.openwide.core.jpa.more.business.generic.query.ISimpleGenericListItemSearchQuery<T, S>`).

The following assumes that Lucene field have already been defined on your entities. If not, see Hibernate Search & Lucene.

Simple match

```
    @Override
    public IPersonSearchQuery company(Company company) {
```

```
        must(matchIfGiven(Person.COMPANY /* Lucene field name for field "company" */,␣
↪company));
        return this;
    }
```

Presence of a single item in a collection field

```
    @Override
    public IPersonSearchQuery company(Company company) {
        must(beIncludedIfGiven(Person.COMPANIES /* Lucene field name for field
↪"companies" */, company));
        return this;
    }
```

Presence of at least one item from a set in a collection field

```
    @Override
    public IPersonSearchQuery company(Set<Company> companies) {
        must(matchOneIfGiven(Person.COMPANY /* Lucene field name for field "company"␣
↪*/, companies));
        return this;
    }
```

Presence of all items from a set in a collection field

```
    @Override
    public IPersonSearchQuery companies(Set<Company> companies) {
        must(matchAllIfGiven(Person.COMPANIES /* Lucene field name for field
↪"companies" */, companies));
        return this;
    }
```

Range query

```
    @Override
    public IPersonSearchQuery modificationDate(Date dateMin, Date dateMax) {
        must(matchRange(
                Person.MODIFICATION_DATE,
                dateMin,
                dateMax
        ));

        return this;
    }
```

"OR" operator

WARNING: If you're ORing multiple criterion, the default mechanisms of not applying null criteria may not be enough. You'd better wrap your code in a `if` checking for the presence of arguments.

```
    @Override
    public IPersonSearchQuery modificationDate(Date dateMin, Date dateMax) {
        if (dateMin != null || dateMax != null) { // BEWARE!
            must(
                any( // = "OR"
                    matchRange(
                        Person.MODIFICATION_DATE,
                        dateMin,
                        dateMax
                    ),
                    matchNull(Person.MODIFICATION_DATE)
                )
            );
        }

        return this;
    }
```

"AND" operator

If you don't have to nest the "AND" in another "OR", you may simply leverage the fact that criteria are ANDed by default:

```
    @Override
    public IPersonSearchQuery noDateInfo() {
        // Implicit "AND"
        must(matchNull(Person.MODIFICATION_DATE));
        must(matchNull(Person.CREATION_DATE));

        return this;
    }
```

Otherwise:

```
    @Override
    public IPersonSearchQuery modificationDate(Date dateMin, Date dateMax) {
        if (dateMin != null || dateMax != null) {
            must(
                any( // = "OR"
                    matchRange(
                        Person.MODIFICATION_DATE,
                        dateMin,
                        dateMax
                    ),
                    all( // = "AND"
                        matchNull(Person.MODIFICATION_DATE),
                        matchNull(Person.CREATION_DATE),
                    )
                )
            );
        }
```

```
        return this;
    }
```

Other criteria

Many more utility methods are provided in `fr.openwide.core.jpa.more.business.search.query.AbstractHibernateSearchSearchQuery<T, S>`. If what you're looking for wasn't above, check out the code.

Overriding utility methods or extending them

If you feel the need to extend this class with additional utility methods, or to override existing utility methods, know that you may do this simply by overriding `fr.openwide.core.jpa.more.config.spring.AbstractJpaMoreJpaConfig.hibernateSearchLuceneQueryFactory()` to return your own query factory.

```
    @Override
    public IMyHibernateSearchLuceneQueryFactory hibernateSearchLuceneQueryFactory() {
        return new MyHibernateSearchLuceneQueryFactoryImpl();
    }
```

Then in any search query implementation, the utility methods will be those defined in your own query factory. You may access additional methods with this snippet of code:

```
    @Override
    protected IMyHibernateSearchLuceneQueryFactory getFactory() {
        return (IMyHibernateSearchLuceneQueryFactory) super.getFactory();
    }

    @Override
    public IMySearchQuery label(String label) throws SearchException {
        must(getFactory().myAdditionalUtilityMethod(
                /* ... */
        ));
        return this;
    }
```

`AbstractJpaSearchQuery`

TODO

Lower-level solutions (service and DAO methods)

JPA querying

TODO QueryDSL-JPA

Native SQL querying

TODO QueryDSL-SQL, Hibernate native SQL

QueryDSL tips

Generating maps and tables

In order to generate a map, use this syntax:

```
return new JPAQuery<>(getEntityManager())
        .from(QUser.user)
        .groupBy(QUser.user.gender)
        .orderBy(QUser.user.gender.asc())
        .transform(GroupBy2.transformer(GroupBy.sortedMap(QUser.user.gender, QUser.
↪user.count().intValue())));
```

If you need a `com.google.common.collect.Table<R, C, V>` instead of a `Map`, you may use `GroupBy2.table` or `GroupBy2.sortedTable` instead of `GroupBy.sortedMap`.

If the keys in database are too precise, and you want to perform another aggregation on the Java side (for instance turning day-precise dates into weeks), you may use the following syntax:

```
return new JPAQuery<>(getEntityManager())
        .from(QUser.user)
        .groupBy(QUser.user.gender, QUser.user.creationDate)
        .orderBy(QUser.user.gender.asc(), QUser.user.creationDate.asc())
        .transform(GroupBy2.transformer(GroupBy2.table(
                QUser.user.gender,
                new MappingProjection<Date>(Date.class, QUser.user.creationDate) {
                    private static final long serialVersionUID = 1L;
                    @Override
                    protected Date map(Tuple row) {
                        return DateDiscreteDomain.weeks().alignPrevious(row.get(0,␣
↪Date.class));
                    }
                },
                /**
                  * We sum twice: once in the SQL query (for each date) and once in␣
↪Java (for each week).
                  * We could have summed only in Java, but it would be sub-optimal if
                  * many user are created each day.
                  * The even better solution would have been to group by week in the␣
↪SQL query,
                  * but unfortunately it's not easy to do with JPQL.
                  */
                GroupBy.sum(QUser.user.count().intValue())
        )));
```

Lucene (Hibernate Search) querying

TODO Hibernate Search DSL

TODO QueryDSL-HibernateSearch?

Hibernate mappings (TODO)

TODO: some advice about when to use:

- `Enum`s or `GenericListItem`s
- `@ElementCollection`
- `AbstractHibernateMapBugWorkaroundValueHolder`
- `AbstractMaterializedPrimitiveValue`
- ...

Hibernate Interceptors

You can declare Spring managed Hibernate interceptors by adding an `hibernateInterceptor()` method in `YourAppCoreCommonJpaConfig`:

```
        @Bean
        public Interceptor hibernateInterceptor() {
                return new ChainedInterceptor()
                                .add(new YourInterceptor());
        }
```

The `ChainedInterceptor` is a class we provide to be able to chain multiple Hibernate interceptors.

For an example of implementation, see:

- https://github.com/openwide-java/owsi-core-parent/blob/master/owsi-core/owsi-core-components/owsi-core-component-jpa-externallinkchecker/src/main/java/fr/openwide/core/jpa/externallinkchecker/business/interceptor/ExternalLinkWr
- https://github.com/openwide-java/owsi-core-parent/blob/master/owsi-core/owsi-core-components/owsi-core-component-jpa-externallinkchecker/src/main/java/fr/openwide/core/jpa/externallinkchecker/business/model/ExternalLinkWrappe

Hibernate Search & Lucene (TODO)

TODO:

- Base
- Explain how we should deal with @IndexedEmbedded/@ContainedIn
- Explain how we should deal with sorts

Sorting

Sorting behavior depends on the data type:

- to sort by `id`, you have to use the field `GenericEntity.ID_SORT`
- to sort by string, you should define an additional field with the `TEXT_SORT` analyzer:

```
    @Column
    @Fields({
        @Field(analyzer = @Analyzer(definition = HibernateSearchAnalyzer.TEXT_
↪STEMMING)),
        @Field(name = EN_SORT, analyzer = @Analyzer(definition =␣
↪HibernateSearchAnalyzer.TEXT_SORT))
    })
    @SortableField(forField = EN_SORT)
    private String en;
```

- to sort by date, you don't need an additional field *BUT* you need to sort using `SortField.Type.LONG` (starting from 0.11)

Note that, you need to add a `@SortableField(forField = "fieldName")` annotation for each field used for sorting.

Cronjobs

Cronjobs tasks can be defined in the `YourAppCoreSchedulingConfig` class.

We use the `@Scheduled` annotation to define the cronjob expression: `@Scheduled(cron = "...")`.

The syntax for the cron expression respects the following rules: https://docs.spring.io/spring/docs/current/javadoc-api/org/springframework/scheduling/support/CronSequenceGenerator.html .

CHAPTER 9

UI

UI Links

This page explains various ways of creating bookmarkable links to pages or resources using Wicket and OWSI-Core.

What is a bookmarkable link?

We'll define bookmarkable links as any non-ajax link that points directly to a HTML page or to a user download (XLS file, JPEG image, ...).

This excludes in particular the subclasses of `org.apache.wicket.markup.html.link.AbstractLink` that implement event handlers (`onClick` or `onSubmit` methods): they may *redirect* to a HTML page or to a user download, but they don't point to it *directly*. We'll name those *action* links.

The main differences between bookmarkable links and action links are that:

- on the user side, the action link will render as a URL tied to the page from where the link originates, whereas bookmarkable links will renders as a URL tied to the target.
- on the server side, bookmarkable link are a bit lighter to execute than event handlers implementing a redirection (since they require one request instead of two).

For more information on action links, see UI User Actions

Link descriptor

Link descriptors are an addition from OWSI-Core. They offer several advantages over traditional Wicket linking:

- They enforce type-safety: users provide business object models only, and they do not need to perform manual conversion to strings each and every time they create a link.
- They enforce consistency: link generation *and* parameter extraction is done by the same object, which only has to be defined once.

- They provide dynamic link generation. When you add an `AbstractLink` generated by a link descriptor to your page, you are guaranteed that if an underlying model has its value updated, the link will also be updated the next time it is rendered.

Link descriptors are in fact two things: link *generators* and link *parameter extractors*.

As link generators, they allow to generate an URL, or even a Wicket `AbstractLink` tied to predefined models and that will automatically render with an up-to-date URL with each page render.

As link parameters extractors, they allow to take the content of Wicket `PageParameters`, convert it to actual objects (not just primitive types) and store it in predefined models.

Those "predefined models" are in fact models that were *mapped* to HTTP query parameters. See below for details about link descriptor mappers.

Interfaces

The terms "link descriptor" refer to several interfaces:

- `ILinkGenerator` (see above)
- `ILinkParameterExtractor` (see above)
- `ILinkDescriptor` (an extension of both `ILinkGenerator` and `ILinkParameterExtractor`)
- `IPageLinkGenerator`, an extension of `ILinkGenerator` that provides some features relevant only to pages
- `IImageResourceLinkGenerator`, an extension of `ILinkGenerator` that provides some features relevant only to resources providing image files
- `IPageLinkParametersExtractor`, an extension of `ILinkParameterExtractor` that provides some features relevant only to pages

Examples of use

The following examples are about *using* an already-created link descriptor. For information about *creating* a link descriptor, see the section about link descriptor builders below.

URL generation

```
ILinkGenerator linkGenerator = /* ... */;
String relativeOrAbsoluteUrl = linkGenerator.url();
String absoluteUrl = linkGenerator.fullUrl();
```

Wicket link generation

Note: links generated using this method are automatically disabled (no `href`) when they render and their parameters fail validation. You may hide them instead by calling `hideIfInvalid` as below.

```
// Inside a Wicket component's constructor
ILinkGenerator linkGenerator = /* ... */;
add(
        linkGenerator.link("linkWicketId")
                .hideIfInvalid()
```

```
                .add(new TargetBlankBehavior())
);
```

Redirection

```
IPageLinkGenerator linkGenerator = /* ... */;
throw linkGenerator.newRestartResponseException();
```

<img> markup

```
// Inside a Wicket component's constructor
IImageResourceLinkGenerator linkGenerator = /* ... */;
add(
        linkGenerator.image("linkWicketId")
                .hideIfInvalid()
                .add(new TargetBlankBehavior())
);
IPageLinkGenerator linkGenerator = /* ... */;
```

Validity check

Important note: validity check is normally unnecessary, as it will be performed automatically and an exception will be thrown if the link is invalid. Generally, this is want you want, because an invalid link simply should not have been used (Wicket links obtained through `ILinkGenerator#link(String)`, for instance, are automatically disabled when invalid, so the user cannot click them).

If invalid links are a possibility that you want to handle as part of your business code, though, you may use code similar to the following snippet. **This should be exceptional**: if you're doing this extensively in your code, you probably missed something.

```
ILinkGenerator linkGenerator = /* ... */;
if (linkGenerator.isAccessible()) {
    throw linkGenerator.newRestartResponseException();
} else {
    // Fallback code
}
```

Link descriptor mappers

Link descriptor mappers are factories that take models as arguments and map them to previously incomplete link definitions in order to create a link descriptor.

They are primarily useful to separate the definition of links (list of parameters types on the Java, mapping of those Java parameters to HTTP query parameters, validations, ...) from the actual parameter definition. The link descriptor mapper will then represent the incomplete link definition that only lacks parameter models in order to provide a full link descriptor.

Examples of use

Note: the result of a link descriptor mappers' `map` method is a link descriptor that may be used in each and every way described above in the "Link descriptor" section. We only provide one such example here to avoid unnecessary repetitions.

Wicket link generation

```
// Inside a Wicket component's constructor
IModel<User> userModel = /* ... */;
IOneParameterLinkDescriptorMapper<IPageLinkGenerator, User> mapper = /* ... */;
add(
        mapper.map(userModel).link("linkWicketId")
                .hideIfInvalid()
                .add(new TargetBlankBehavior())
);
```

Data table link declaration

See UI-Displaying Collections for some context about `DataTableBuilder`.

```
IOneParameterLinkDescriptorMapper<IPageLinkGenerator, User> mapper = /* ... */;
CoreDataTablePanel<?, ?> results =
        DataTableBuilder.start(dataProvider, dataProvider.getSortModel())
        .addLabelColumn(new ResourceModel("business.customer.lastName"), Bindings.
↪customer().lastName())
                .withLink(mapper) // <= USE THE MAPPER HERE
                .withSort(CustomerSort.LASTNAME, SortIconStyle.ALPHABET, CycleMode.
↪NONE_DEFAULT_REVERSE)
                .withClass("text text-sm")
        /** Add some more columns... */
        .build("results");
```

Link descriptor builder

The link descriptor builder allows to build link descriptors or link descriptor mappers. It provides methods to define the mappable Java-side parameters, the mappings between those parameters and HTTP query parameters, the validations around those parameters and the target of the link.

Examples of use

The following sections provide some examples of use. This is not an exhaustive reference, so if those examples do not match exactly your need, you may start from the closest one and use the builder's Javadoc to find what you're looking for.

Simple link descriptor

This is especially useful for pages with no parameters (home page, lists, ...).

```
@AuthorizeInstantiationIfPermission(permissions = {MyPermissionConstants.READ_
↪CUSTOMER})
public class CustomerListPage extends MainTemplate {

    public static final IPageLinkDescriptor linkDescriptor() {
        return LinkDescriptorBuilder.start()
                .page(CustomerListPage.class);
    }

    public CustomerListPage(PageParameters parameters) {
        super(parameters);
        /* ... */
    }
}
```

Link descriptor mapper

```
public class CustomerDescriptionPage extends MainTemplate {

    public static final IOneParameterLinkDescriptorMapper<IPageLinkDescriptor,␣
↪Customer> MAPPER =
            LinkDescriptorBuilder.start()
            .model(Customer.class).map(CommonParameters.ID).mandatory()
            .permission(MyPermissionConstants.READ)
            .page(CustomerDescriptionPage.class);

    public CustomerDescriptionPage(PageParameters parameters) {
        super(parameters);

        IModel<Customer> customerModel = new GenericEntityModel<Long, Customer>();

        MAPPER.map(customerModel)
                .extractSafely(
                        parameters,
                        CustomerListPage.linkDescriptor(),
                        getString("common.error.unexpected")
                );

        /* ... */
    }
}
```

Link descriptor mapper with multiple parameters

```
public class CustomerDescriptionPage extends MainTemplate {

    public static final ITwoParameterLinkDescriptorMapper<IPageLinkDescriptor,␣
↪Customer, String> MAPPER_TAB =
            LinkDescriptorBuilder.start()
            .model(Customer.class)
            .model(String.class)
            .pickFirst().map(CommonParameters.ID).mandatory()
            .pickSecond().map("tab").optional()
            .pickFirst().permission(MyPermissionConstants.READ)
```

```
            .page(CustomerDescriptionPage.class);

    public static final IOneParameterLinkDescriptorMapper<IPageLinkDescriptor,␣
↪Customer> MAPPER =
            MAPPER_TAB.ignoreParameter2();

    public CustomerDescriptionPage(PageParameters parameters) {
        super(parameters);

        IModel<Customer> customerModel = new GenericEntityModel<Long, Customer>();
        IModel<String> selectedTabNameModel = new Model<>();

        MAPPER_TAB.map(customerModel, selectedTabNameModel)
                .extractSafely(
                        parameters,
                        CustomerListPage.linkDescriptor(),
                        getString("common.error.unexpected")
                );

        /* ... */
    }
}
```

Link descriptor mapper with a collection parameter

```
public class MyPage extends MainTemplate {

    public static final IOneParameterLinkDescriptorMapper<IPageLinkDescriptor, List
↪<Customer>> MAPPER =
            LinkDescriptorBuilder.start()
            .<List<Customer>>model(List.class).mapCollection("list", Customer.class).
↪mandatory()
            .permission(MyPermissionConstants.READ)
            .page(CustomerDescriptionPage.class);

    public MyPage(PageParameters parameters) {
        super(parameters);

        IModel<Customer> customerListModel = CollectionCopyModel.custom(
                Suppliers2.<Customer>arrayListAsList(), GenericEntityModel.<Customer>
↪factory()
        );

        MAPPER.map(customerListModel)
                .extractSafely(
                        parameters,
                        CustomerListPage.linkDescriptor(),
                        getString("common.error.unexpected")
                );

        /* ... */
    }
}
```

Link descriptor mapper with custom validation condition

```
public class CustomerDescriptionPage extends MainTemplate {

    public static final IOneParameterLinkDescriptorMapper<IPageLinkDescriptor,␣
↪Customer> MAPPER =
            LinkDescriptorBuilder.start()
            .model(Customer.class).map(CommonParameters.ID).mandatory()
            .permission(MyPermissionConstants.READ)
            .validator(DetachableFactories.forUnit(
                    new AbstractDetachableFactory<IModel<Customer>, Condition>() {
                        private static final long serialVersionUID = 1L;
                        @Override
                        public Condition create(IModel<Customer> parameter) {
                            return new MyCondition(parameter);
                        }
                    }
            ))
            .page(CustomerDescriptionPage.class);

    public CustomerDescriptionPage(PageParameters parameters) {
        super(parameters);

        IModel<Customer> customerModel = new GenericEntityModel<Long, Customer>();

        MAPPER.map(customerModel)
                .extractSafely(
                        parameters,
                        CustomerListPage.linkDescriptor(),
                        getString("common.error.unexpected")
                );

        /* ... */
    }
}
```

Other examples

See OWSI-Core's tests, in particular the test methods in `fr.openwide.core.test.wicket.more.link.descriptor.AbstractAnyTargetTestLinkDescriptor` and `fr.openwide.core.test.wicket.more.link.descriptor.AbstractAnyTargetTestLinkDescriptorMapper`.

Other links

EmailLink

`EmailLinks` is a `mailto:` link that automatically defines its body as the email it points to.

```
IModel<String> emailModel = /* ... */
add(new EmailLink("email", emailModel));
```

UI Redirecting

This page explains various methods for redirecting from one page to another in you web application.

Please note that we're talking about redirection as part of a server-side process, such as a form that redirects to a different page based on the user input. If you just want a link in your HTML page, please see UI-Links.

Redirecting as part of the authentication/authorization process

Basics

When some page is accessed, but the current user has no right to access it (either because the user is not authenticated or he hasn't got the proper authorization), OWSI-Core throws a `org.springframework.security.access.AccessDeniedException`, which is caught by Spring Security's servlet filter. Spring Security then handles this exception with whatever behavior you configured; by default in OWSI-Core, it's a redirection to "/access-denied/", on which the `AccessDeniedPage` is mapped.

`AccessDeniedException`s are thrown:

- When Spring Security detects an unauthenticated access (access to a page without authorization while being unauthenticated).
- When a Wicket's `AuthorizationException` is caught by the `CoreDefaultExceptionMapper`.

Customizing the general behavior

If you want to customize the behavior when an access is denied, you should either:

- change Spring Security's configuration to customize the access denied handler
- map your own page to "/access/denied/"

Customizing the behavior for specific pages

If you want to customize the behavior for specific pages, you may do so:

- at the Wicket level, by declaring your own exception mapper by overriding `org.apache.wicket.Application.getExceptionMapperProvider()` and defining a specific behavior when an `AuthorizationException` is caught. Be aware that this will not cover cases when an access is denied by Spring Security, though, only cases when an access is denied by Wicket itself (due to an annotation on a page, for instance).

  This can be done this way (for instance) in the `map` method of a class extending `CoreDefaultExceptionMapper`:

```
    try {
        if (
                Exceptions.findCause(e, AccessDeniedException.class) != null
            ||  Exceptions.findCause(e, AuthorizationException.class) != null
        ) {
            IPageRequestHandler handler = PageRequestHandlerTracker.
↪getFirstHandler(RequestCycle.get());
            Class<? extends IRequestablePage> pageClass = handler.getPageClass();
            PageParameters pageParameters = handler.getPageParameters();
            Component component = null;
            if (handler instanceof IComponentRequestHandler &&␣
↪((IComponentRequestHandler) handler).getComponent() instanceof Component) {
```

```
                component = (Component) ((IComponentRequestHandler) handler).
↪getComponent();
            }
            if (
                    IMyPageWhoseAccessMayBeDenied.class.
↪isAssignableFrom(pageClass)
                ||  (component != null &&␣
↪IMyPageWithAPopupWhoseAccessMayBeDenied.class.isAssignableFrom(pageClass))
            ) {
                Session.get().error(rendererService.localize("access.denied.
↪customMessage", Session.get().getLocale()));
                PageParameter parameters = /* ... */;
                return new RenderPageRequestHandler(new␣
↪PageProvider(MyRedirectPage.class, ));
            }
    } catch (RuntimeException e2) {
        if (LOGGER.isDebugEnabled()) {
            LOGGER.error("An error occurred while handling a previous error: " +␣
↪e2.getMessage(), e2);
        }

        // We were already handling an exception! give up
        LOGGER.error("unexpected exception when handling another exception: " +␣
↪e.getMessage(), e);
        return new ErrorCodeRequestHandler(500);
    }

    return super.map(e);
```

- or at the Spring Security level by defining your own access denied handler in Spring Security's configuration

Redirecting to the current page

Full refresh (keeping the same page instance)

If you're in a non-Ajax component, know that handling the request will automatically trigger a full-page refresh. No need for you to do anything.

In the context of an Ajax component you may use this snippet in order to fully refresh the current page:

```
target.add(getPage());
```

Or if you want to completely abort your currently executing code, you may throw an exception:

```
throw new RestartResponseException(getPage());
```

Redirecting to another instance of the same page

In some cases, you will want to redirect to another instance of the same page with the same parameters. This is mostly used when a fatal error occurs.

```
throw new RestartResponseException(getPage().getClass(), getPage().
↪getPageParameters());
```

For any other redirection (most cases)

Redirection is mainly done through exceptions. These come in various flavors, depending on your redirection target.

Please note that `IPageLinkGenerators` (see UI-Links) offer methods for easily generating the exception of your choice. This is the recommended way of redirecting.

Here are the main exception types:

- `RestartResponseException` when you simply want to redirect to another page in your Wicket application.
- `RestartResponseAtInterceptPageException` when you want to redirect to another page which will later trigger another redirection to the current page (mainly used for sign-in pages).
- `RedirectToUrlException` when you want to redirect to an external URL (outside of your Wicket application).

You may also encounter the following patterns in Wicket components or pages. These should be avoided, as they only throw an exception but they do not make it clear, neither to you nor to the compiler. Thus you may end up with dead code after your `redirect` call.

```
// AVOID THIS
redirect(MyPage.class);
```

```
// AVOID THIS
redirectToInterceptPage(MyPage.class);
```

Adding an anchor

If you want to point to an anchor on the target page, then you must use a `RedirectToUrlException`. This feature is built in the `IPageLinkGenerator`.

UI IE 8 Support

A reasonable, transparent IE8 support can be achieved in a Wicket application the following way.

Obviously, you'll still run into slowdowns, bugs and limitations due to IE8 being IE8. But the most visible issues will be gone.

1. Add IE8-specific CSS

Put these files beside your `StylesLessCssResourceReference.java`:

`IE8AndLesserLessCssResourceReference.java`:

```
public final class IE8AndLesserLessCssResourceReference extends␣
↪LessCssResourceReference {

    private static final long serialVersionUID = 4656765761895221782L;

    private static final IE8AndLesserLessCssResourceReference INSTANCE = new␣
↪IE8AndLesserLessCssResourceReference();

    private IE8AndLesserLessCssResourceReference() {
```

```
        super(IE8AndLesserLessCssResourceReference.class, "ie8-and-lesser.less");
    }

    public static IE8AndLesserLessCssResourceReference get() {
        return INSTANCE;
    }

}
```

ie8-and-lesser.less:

```
@import "@{scope-core-bs3}bootstrap/variables.less";
@import "@{scope-core-bs3}bootstrap/override/variables.less";
@import "variables.less";

.placeholder { // Added by the "placeholder" jquery polyfill plugin. See jquery.
↪placeholder.js
    color: @input-color-placeholder;
}

.btn-icon-only.btn-placeholder, .btn-placeholder {
    visibility: hidden;
}

/* Put any IE8-specific CSS here */
```

2. Add global listeners

Add this in your Wicket Application's `init` method:

```
IELegacySupport.init(this);
```

With `IELegacySupport.java` being:

```
public final class IELegacySupport {

    private IELegacySupport() {
    }

    public static final String IE_LEGACY_CONDITION = "lte IE 8";

    public static final String IE_UNSUPPORTED_CONDITION = "lt IE 8";

    public static final String IE_8_CONDITION = "IE 8";

    public static final int IE_LEGACY_VERSION = 8;

    public static void init(WebApplication webApplication) {
        webApplication.getComponentInstantiationListeners().add(new␣
↪InstantiationListener());
        webApplication.getAjaxRequestTargetListeners().add(new␣
↪AjaxRequestTargetListener());
    }

    private static class IELegacyHeaderItemsContributorBehavior extends Behavior {
```

```
        private static final long serialVersionUID = -1441191136903604013L;

        private final Iterable<HeaderItem> headerItems;

        public IELegacyHeaderItemsContributorBehavior(HeaderItem ... headerItems) {
            this(ImmutableList.copyOf(headerItems));
        }

        public IELegacyHeaderItemsContributorBehavior(Iterable<HeaderItem>␣
↪headerItems) {
            super();
            this.headerItems = headerItems;
        }

        @Override
        public void renderHead(Component component, IHeaderResponse response) {
            WebClientInfo clientInfo = (WebClientInfo) Session.get().getClientInfo();
            ClientProperties properties = clientInfo.getProperties();
            if (properties.isBrowserInternetExplorer() && properties.
↪getBrowserVersionMajor() <= IE_LEGACY_VERSION) {
                for(HeaderItem headerItem : headerItems) {
                    response.render(headerItem);
                }
            }
        }
    }

    private static class InstantiationListener implements␣
↪IComponentInstantiationListener {
        @Override
        public void onInstantiation(Component component) {
            if (component instanceof Page) {
                Page page = (Page) component;

                // Support for the placeholder text of input fields in IE8 and lesser
                page.add(new PlaceholderPolyfillBehavior());

                page.add(new IELegacyHeaderItemsContributorBehavior(
                        // Support for media queries in IE8 and lesser
                        JavaScriptHeaderItem.
↪forReference(RespondJavaScriptResourceReference.get()),
                        // IE8 and lesser specific CSS
                        CssHeaderItem.
↪forReference(IE8AndLesserLessCssResourceReference.get())
                ));
            }
        }
    }

    private static class AjaxRequestTargetListener extends AjaxRequestTarget.
↪AbstractListener {
        @Override
        public void updateAjaxAttributes(AbstractDefaultAjaxBehavior behavior,␣
↪AjaxRequestAttributes attributes) {
            WebClientInfo clientInfo = (WebClientInfo) Session.get().getClientInfo();
            ClientProperties properties = clientInfo.getProperties();
            if (properties.isBrowserInternetExplorer() && properties.
↪getBrowserVersionMajor() <= IE_LEGACY_VERSION) {
```

```
                attributes.getAjaxCallListeners().add(
                        new AjaxCallListener().onBefore(
                                // Prevents placeholder text from being submitted
                                PlaceholderPolyfillBehavior.disable().render()
                        )
                );
            }
        }

        @Override
        public void onBeforeRespond(Map<String, Component> map, AjaxRequestTarget␣
↪target) {
            // Refresh the placeholder text (for instance when rendering a popup)
            target.appendJavaScript(PlaceholderPolyfillBehavior.statement().render());
        }

        @Override
        public void onAfterRespond(Map<String, Component> map, IJavaScriptResponse␣
↪response) {
            // Nothing to do
        }
    }

}
```

UI Models (TODO)

This page explains the Wicket concept of models and details various types of models that you might use in your applications.

What is an `IModel`?

TODO

Special cases

Collections

For details about how to display collections, and some tips about how to choose the correct interface for accessing you collection data, see UI-Displaying Collections.

`IDataProvider`, `ISequenceProvider`

Both `IDataProvider` and `ISequenceProvider` are interfaces designed to provide access to large datasets, with built-in paging.

They

They mainly differ in the way they wrap elements into models. `IDataProvider` exposes a `Iterator<T> iterator(long, long)` method and a `IModel<T> model(T)` method that must be called by clients. This

works, but causes trouble when `IDataProvider` implementor wants to always return the same model for the same element (for example because the model carries more mutable information than just a reference to the element).

`ISequenceProvider` solves the issue by directly returning a `Iterator<IModel<T>> iterator(long, long)`. This gives more flexibility to implementors and changes next to nothing (save the interface) for clients.

Please note that in most cases, you should not have to implement an `ISequenceProvider` yourself: simply implementing an `IDataProvider` should do the job. This interface is available for very specific features, such as those implemented in `CollectionCopyModel`.

`ICollectionModel`, `IMapModel`

Those interfaces are implemented by models that:

- provide access to a collection or a map
- implement `ISequenceProvider` so as to always return the same model to wrap the same collection/map element
- optionally, provide write operations (`add`/`put`, `clear`, ...)

Those implementations are noteworthy:

- `CollectionCopyModel`
- `CollectionMapModel`
- `ReadOnlyCollectionModel`
- `ReadOnlyMapModel`

See below on this page for details.

Main use cases

`GenericEntityModel`

TODO

`BindingModel`

TODO

`IBindableModel` et al.

The use case: inter-dependent form components

When designing complex forms, often we have to update some parts of the form whenever a given field changes, even before the form was submitted. This may happen for instance:

- Because some field's proposed values depend on another field's value
- Because of some read-only panel must dynamically display detailed information about a selected value

In any case, you've got what we will call "source" form components (the ones whose value another component depends on) and "target" components (the ones which depend on another form component's value).

That kind of feature is generally achieved by adding an Ajax behavior on the source component that will update the underlying model whenever a client-side change occurs, and refresh the target components.

Why caches are needed

Most of the time, the underlying model is a `BindingModel`, and its root model is a `GenericEntityModel`, which means that the updated value may not be correctly saved for the next requests:

- If the root object (the one we extract the property value from) is an unpersisted entity, then the updated value will be serialized with this unpersisted entity, which might not be a good idea (for instance if the value is itself an entity).
- If the root object is a persisted entity, then on the next request, the root will be loaded from the database and will thus have its properties reset.

In the particular case where each "target" component depends on only one "source" component, and no other Ajax refreshes are performed, then it's fine, because we won't need the updated value again.

But let's say one "target" component depends on multiple "source" components, say `C` depends on `A` and `B`. We've got the following code:

```
// DON'T DO THIS, IT WON'T WORK AS EXPECTED

IModel<MyEntity> rootModel = /* ... */;
IModel<MyEntity2> propertyAModel = BindingModel.of(rootModel, Bindings.myEntity().
↪propertyA());
IModel<MyEntity3> propertyBModel = BindingModel.of(rootModel, Bindings.myEntity().
↪propertyB());

Form<?> form = new Form<>("form");
final MyDependingComponent depending = new MyDependingComponent("depending",␣
↪propertyAModel, propertyBModel);
form.add(
        new MyEntity2DropDownChoice("propertyA", propertyAModel)
                .add(new AjaxFormComponentUpdatingBehavior() {
                    protected void onUpdate(AjaxRequestTarget target) {
                        // THIS MAY FAIL, because B's value may not be up-to-date
                        target.add(depending);
                    }
                }),
        new MyEntity3DropDownChoice("propertyB", propertyBModel, String.class),
                .add(new AjaxFormComponentUpdatingBehavior() {
                    protected void onUpdate(AjaxRequestTarget target) {
                        // THIS MAY FAIL, because A's value may not be up-to-date
                        target.add(depending);
                    }
                }),
        depending
);
```

Then the following scenario may fail:

- `A` is modified by the user
- The Ajax behavior updates `A`'s model and refreshes `C`, which will use `A`'s *updated* value and `B`'s *initial* value. So far so good.

- `B` is modified by the user
- The Ajax behavior updates `B`'s model and refreshes `C`. **`C` will use a wrong value for `A`:**
  - If `A`'s model is a `BindingModel` for an entity property whose root model is a `GenericEntityModel` holding an *unpersisted* entity, **`C` will use a serialized entity for `A`'s value, which may throw `LazyInitializationException`s whenever we try to access its properties**.
  - If `A`'s model is a `BindingModel` for an entity property whose root model is a `GenericEntityModel` holding a *persisted* entity, **`C` will use `A`'s initial value**.

How `IBindableModel`s solve the problem

Enter `IBindableModel`. The idea is to wrap the root model in a `IBindableModel`, and then only use this model's methods to access property models, which will have their values cached transparently.

So instead of the snippet of code above, we will do this:

```
IModel<MyEntity> rootModel = /* ... */;
IBindableModel<MyEntity> bindableRootModel = BindableModel.of(rootModel);
IModel<MyEntity2> propertyAModel = bindableRootModel.bindWithCache(Bindings.
↪myEntity().propertyA(), new GenericEntityModel<Long, MyEntity2>());
IModel<MyEntity3> propertyBModel = bindableRootModel.bindWithCache(Bindings.
↪myEntity().propertyB(), new GenericEntityModel<Long, MyEntity2>());

Form<?> form = new CacheWritingForm<>("form", bindableRootModel); // Necessary so the␣
↪caches are written to the object when submitting
final MyDependingComponent depending = new MyDependingComponent("depending",␣
↪propertyAModel, propertyBModel);
form.add(
        new MyEntity2DropDownChoice("propertyA", propertyAModel)
                .add(new AjaxFormComponentUpdatingBehavior() {
                    protected void onUpdate(AjaxRequestTarget target) {
                        target.add(depending);
                    }
                }),
        new MyEntity3DropDownChoice("propertyB", propertyBModel, String.class),
                .add(new AjaxFormComponentUpdatingBehavior() {
                    protected void onUpdate(AjaxRequestTarget target) {
                        target.add(depending);
                    }
                }),
        depending
);
```

That way, the values used by `depending` are those in `propertyAModel` and `propertyBModel`'s caches, and those are always clean and up-to-date.

If you must make `MyDependingComponent` use a `MyEntity` model instead of a `MyEntity2` model and a `MyEntity3` model, and only use these properties indirectly (for instance because you must call a service wich accept a `MyEntity` parameter), then you can make use of the `IBindingModel#writeAll()` method, which forces the writing of caches to the underlying entity:

```
IModel<MyEntity> rootModel = /* ... */;
final IBindableModel<MyEntity> bindableRootModel = BindableModel.of(rootModel);
IModel<MyEntity2> propertyAModel = bindableRootModel.bindWithCache(Bindings.
↪myEntity().propertyA(), new GenericEntityModel<Long, MyEntity2>());
IModel<MyEntity3> propertyBModel = bindableRootModel.bindWithCache(Bindings.
↪myEntity().propertyB(), new GenericEntityModel<Long, MyEntity2>());
```

```
Form<?> form = new CacheWritingForm<>("form", bindableRootModel); // Necessary so the␣
↪caches are written to the object when submitting

// MyDependingComponent depends on a IModel<MyEntity>, and only indirectly uses␣
↪propertyA and propertyB
final MyDependingComponent depending = new MyDependingComponent("depending",␣
↪bindableRootModel);
form.add(
        new MyEntity2DropDownChoice("propertyA", propertyAModel)
                .add(new AjaxFormComponentUpdatingBehavior() {
                    protected void onUpdate(AjaxRequestTarget target) {
                        bindableRootModel.writeAll();
                        target.add(depending);
                    }
                }),
        new MyEntity3DropDownChoice("propertyB", propertyBModel, String.class),
                .add(new AjaxFormComponentUpdatingBehavior() {
                    protected void onUpdate(AjaxRequestTarget target) {
                        bindableRootModel.writeAll();
                        target.add(depending);
                    }
                }),
        depending
);
```

Do's and don'ts

Declare your caches

Caches must be declared explicitly:

- call `IBindableModel.bindCollectionWithCache()` on any collection property whose elements may have their properties written to.
- call `IBindableModel.bindMapWithCache()` on any map property whose keys or values may have their properties written to.
- call `IBindableModel#bindWithCache(<binding>, <cache model>)` on any property which both read from (by depending components) and written to (by a `FormComponent` for instance).

Don't mix `BindingModels` with `IBindableModels`

Using a `BindingModel` with a `IBindableModel` as root model will result in bypassing the `IBindableModel`'s cache (if any). You may then witness some strange behaviors due to your `BindableModel` returning a stale value.

Thus, if you use `IBindableModel`, stick with it. If you must pass a model to a child component, check that this child component doesn't use `BindingModels`.

If you really must use a component that uses `BindingModels` internally, you can, but only if it's read-only (i.e. it doesn't wraps `FormComponents`). **Keep in mind**, though, that you must explicitly write the caches to your business objects whenever you do an Ajax refresh.

Write caches to your business objects before using them

Caches are not written magically to your business objects. Thus:

- When your form is being submitted and after it has written the submitted values to your models (to your caches, actually), you must ensure that the caches are actually written to the actual properties so that the root object is fully updated.

  Luckily for you, `CacheWritingForm` does exactly that. Just use it as your root form and, if all of your `IBindableModel`s are children of your root model, then they will all be updated upon submit.

- Whenever you do handle events (`Link`s, `AjaxLink`s, ajax behaviors, ...), if any treatment bypasses the `IBindableModel` and reads directly from the business objects (e.g. `bindableRootModel.getObject().getPropertyA()` instead of `bindableRootModel.bind(Bindings.myEntity().propertyA()).getObject()`), then you should call `IBindableModel#writeAll()` beforehand.

Read caches from your business objects when you modify objects directly

Caches are not updated magically when you bypass the `IBindableModel` and write to the properties directly (e.g. `bindableRootModel.getObject().setPropertyA(<something>)` instead of `bindableRootModel.bind(Bindings.myEntity().propertyA()).setObject(<something>)`).

If you have to do such things, make sure that you call `IBindableModel#readAllUnder()` afterwards.

`CollectionCopyModel` and `MapCopyModel`

`CollectionCopyModel` and `MapCopyModel` are simply put, models to store your collections or maps directly in your page, with no persistence.

They provide two main features:

- they always copy the collection/map when `setObject` is called (hence the name). So even if some one calls `setObject` with an immutable collection as a parameter, the collection returned by `getObject` will still be mutable.
- upon detach, they do not reference the collection or its elements directly, but they wrap the elements in models so that each element is detached correctly. This is especially useful when handling collections of `GenericEntity`, that should never be serialized with the page.

Both models require two things when they're created: a mean of instantiating a new empty collection/map (a `Supplier`), and a mean of instantiating the model that will wrap an element (a `Function<T, IModel<T>>`).

Here are some examples:

```
ICollectionModel<?, Set<MyEntity>> myEntitySetModel =
        CollectionCopyModel.custom(Suppliers2.<MyEntity>hashSetAsSet(),␣
↪GenericEntityModel.<MyEntity>factory())
```

```
ICollectionModel<?, SortedSet<MyEnum>> myEnumSortedSetModel =
        CollectionCopyModel.serializable(Suppliers2.
↪treeSetAsSortedSet(MyEnumComparator.get()));
```

```
IMapModel<?, ?, Map<MyEntity, String>> myEntityToStringModel = MapCopyModel.custom(
        Suppliers2.<MyEntity, String>hashMapAsMap(),
```

```
        GenericEntityModel.<MyEntity>factory(),
        Models.<String>serializableModelFactory()
);
```

AbstractReadOnlyModel

TODO

LoadableDetachableModel and LoadableDetachableDataProvider

`LoadableDetachableModel` and `LoadableDetachableDataProvider` are two abstract classes that provide a caching feature, so that the data they give access to is "loaded" on the first access, cached, and then returned as it was retrieved on the first access on every subsequent call, until `detach` is called.

This avoids repeated calls to the database during a single request/response cycle.

Caching

As said above, `LoadableDetachableDataProvider` and `LoadableDetachableModel` will only execute the underlying query once per HTTP request, even if their data-access methods (`count`, `iterator`, `getObject`) are called multiple times. This is, in most cases, what you want.

However, in some very particular cases, you may have to first access the data source (`LoadableDetachableDataProvider` or `LoadableDetachableModel`), then change the underlying data (through a service call), then render the page. **Be warned** that in this case, the rendered data will be the data loaded before your change. If it's not what you want, then you should "refresh" the `LoadableDetachableXXX` explicitly by calling `detach()`.

Modifying data

As usual, modifying the entities retrieved from the `LoadableDetachableDataProvider` or `LoadableDetachableModel` won't alter the database: you need to make service calls in order for these changes to be persisted.

Another thing that might be obvious: be aware that calls to `LoadableDetachableModel.setObject()` will, by default, only change the model value for the current request/response cycle. This is normal, because only you know what to do in order to persist this change.

If you want to persist your changes in database, then you should provide a method in your service layers that will do the work.

If you just want a cache that spans multiple requests, then `CollectionCopyModel` or `MapCopyModel` might be what you're looking for. See further down this page for more information.

Utilities

CompositeSortModel

TODO

StreamModel

`StreamModel` is made to manage Wicket models wrapping `Iterable`. A `StreamModel` is a read-only `IModel<Iterable<T>>`.

Use `StreamModel<T> mySteamModel = StreamModel.of(IModel<? extends Iterable<T>> model)` to get started. From there, you can :

- Use it as a classic Wicket model : `mySteamModel.getObject()`.
- Concatenate multiple models : `mySteamModel.concat(IModel<? extends Iterable<? extends T>> firstModel, IModel<? extends Iterable<? extends T>>... otherModels)`.
- Transform (map) elements of the collection : `mySteamModel.map(Function<T, S> function)`.
- Get a `IModel` which provides the elements in a specific collection type: `mySteamModel.collect(Supplier<? extends C> supplier)`
- Combine all of the above : `mySteamModel.concat(IModel<? extends Iterable<? extends T>> firstModel, IModel<? extends Iterable<? extends T>>... otherModels).map(Function<T, S> function).collect(Supplier<? extends C> supplier)`

WorkingCopyModel, CollectionWorkingCopyModel and MapWorkingCopyModel

These model wrap two other models: a reference model and a "copy" model. They delegate read and write access to the copy model, while providing additional methods to *write* from the copy to the reference and *read* from the reference to the copy.

These models should not be used directly as a more high-level feature is available with the `BindableModel` described above.

Troubleshooting

Sometimes, you've got models that are not detached properly, but you simply don't know which ones. You just know that, on the next rendering of you page, everything explodes with a `org.hibernate.LazyInitializationException`. In that case, you've got to dig up a bit, and this chapter aims at helping you doing just that.

Built-in logs

`GenericEntityModel` and `AbstractThreadSafeLoadableDetachableModel` (plus its subclasses) provide built-in logging when attached values are suspiciously serialized.

They show:

- The currently attached value at WARN level
- A stacktrace of the latest attach operation on this model at DEBUG level (don't use this level in production environment: it involves aggressive stacktrace recording)

Breakpoints

If the above logs are not enough (and they should), you may still use breakpoints.

Just put your breakpoints inside the `if` in `GenericEntityModel#writeObject` or `AbstractThreadSafeLoadableDetachableModel#writeObject`. In the stack will appear several `writeObject0` methods: inspect those and the `arg0` parameter to determine the chain of objects that lead to the incorrect serialization of your model. You will then probably have to fix one of this object by adding a missing `detach` somewhere.

UI Displaying Collections

This page explains how to display collections using OWSI-Core. Several methods are provided, ordered from easiest to the hardest: use the first that fits, so as to avoid unnecessary complexity.

Data sources

Choosing your implementation

There are several types of objects you may use to build an object that will make up the interface between your service/data layers and your view.

When getting data from an entity

If you want to retrieve the data directly from an entity attribute (`myEntity.getMyCollection()`) you may use a `BindingModel`. See UI-Models for more information.

When getting data from a service or `IQuery`

If you're not familiar with data querying in OWSI-Core, you probably should read Querying before going on.

Special case: `ISearchQuery`

If your query is an `ISearchQuery` (`AbstractHibernateSearchSearchQuery` or `AbstractJpaSearchQuery`), you may simply extend `AbstractSearchQueryDataProvider`. The typical implementation will:

- define some `IModel` attributes and getters, for the search parameters. These models will be used in a form, so that the user may alter them.
- implement `getSearchQuery` by:
  - calling `createSearchQuery()` with the query interface type (`IMyQuery.class`) as a parameter ;
  - and then calling methods on the resulting query in order to set the search parameters.

Other cases (service method call or non-search `IQuery`)

- If your query uses paging (with an offset and a limit), you'd better define a `IDataProvider`. A good place to start is `LoadableDetachableDataProvider`, which you should try extending. Also, see UI-Models for more information on `LoadableDetachableDataProvider` and its caveats.

- If your query has no paging feature, you may simply define your own `IModel<WhateverCollectionType<T>>`. A good place to start is `LoadableDetachableModel`, which you should try extending. Also, see UI-Models for more information on `LoadableDetachableModel` and its caveats.

Renderers

Renderers offer a way to build a non-HTML (plain text) string from a given `Collection<T>` or `IModel<? extends Collection<T>>`.

When to use it

You should use renderers when:

- Your expected output has a very simple structure
- Your expected output does not require HTML (be warned that this excludes any kind of line break, since this would require a `<br>` or `<p>`)
- Requirements are very unlikely to change in the future to require HTML inside the output

If you need more complex output, go for the *DataTableBuilder* or *RefreshingViews*.

Examples

Building the renderer

```
Renderer<Iterable<? extends MyItem>> collectionRenderer = Renderer.fromJoiner(
        Joiners.Functions.onComma(),
        MyItemRenderer.get()
);
```

Using the renderer

In a label

```
IModel<Set<MyItem>> myModel = /* ... */;
add(new CoreLabel("id", collectionRenderer.asModel(myModel))); // Will display "item1,
↪ item2, item3"
```

In a StringResourceModel

`*.properties`:

```
my.resource.key=List value: {0}
```

Java:

```
IModel<Set<MyItem>> myModel = /* ... */;
IModel<String> stringModel = new StringResourceModel("my.resource.key")
        .setParameters(collectionRenderer.asModel(myModel));
add(new CoreLabel("id", stringModel))); // Will display "List value: item1, item2,␣
↪item3"
```

In an error message

Just use `Component.getString()` as follows:

```
IModel<Set<MyItem>> myModel = /* ... */;
component.error(component.getString("my.resource.key", collectionRenderer.
↪asModel(myModel)); // Will display "List value: item1, item2, item3"
```

And define your properties as follows:

```
my.resource.key=List value: ${}
```

DataTableBuilder

The DataTableBuilder offers the simplest way to build a HTML table, quick & clean.

When to use it?

In order to use DataTableBuilers, the component you want to build must meet the following requirements:

- The expected output must be a HTML table
- The data source must be some kind of collection of elements (a `IDataProvider`, a `IModel<? extends Collection<?>>` or a `ISequenceProvider`)
- There must be one row in the table's body for each element in the data model (paging aside)
- There must be a pre-defined, static maximum number of columns. Some columns may get hidden dynamically. For instance, you can't have one column for each element of an `IModel<? extends Collection<?>>` if this model's content may change between ajax refreshes.

If all of the above seems fine to you, then go ahead with the `DataTableBuilder`. Otherwise, you may still use *RefreshingViews*.

Overview

The general pattern for building a data table is as follows:

- create a builder through one of the static `start` methods
- add a column though one of the `.add*Column` methods, defining in particular the data to be displayed (with a binding or a `Function`)
- customize the column though the various methods allowing to add CSS classes on cells, to add a link for each row, to add a sort-switching link in the header, and so on
- repeat the same operations for each column

- optionally, call `.decorate` in order to create a table with an upper title and pagers, or `.bootstrapPanel` to the the same in a Bootstrap panel
- call `.build("wicketId")` in order to retrieve the resulting component.

Here is a (simple) example of use of `DataTableBuilder`:

```
DecoratedCoreDataTablePanel<?, ?> results =
        DataTableBuilder.start(dataProvider, dataProvider.getSortModel())
        .addLabelColumn(new ResourceModel("business.customer.lastName"), Bindings.
↪customer().lastName())
                .withLink(CustomerDescriptionPage.MAPPER)
                .showPlaceholder()
                .withSort(CustomerSort.LASTNAME, SortIconStyle.ALPHABET, CycleMode.
↪NONE_DEFAULT_REVERSE)
                .withClass("text text-sm")
        .addLabelColumn(new ResourceModel("business.customer.firstName"), Bindings.
↪customer().firstName())
                .withLink(CustomerDescriptionPage.MAPPER)
                .showPlaceholder()
                .withSort(CustomerSort.FIRSTNAME, SortIconStyle.ALPHABET, CycleMode.
↪NONE_DEFAULT_REVERSE)
                .withClass("text text-sm")
        .addLabelColumn(new ResourceModel("business.customer.birthdate.short"),␣
↪Bindings.customer().birthdate(), DatePattern.REALLY_SHORT_DATE)
                .showPlaceholder()
                .withSort(CustomerSort.BIRTHDATE, SortIconStyle.DEFAULT, CycleMode.
↪NONE_DEFAULT_REVERSE)
                .withClass("date date-xs")
                .withClass(ResponsiveHidden.XS_AND_LESS)
        .addBootstrapLabelColumn(new ResourceModel("business.customer.status"),␣
↪Bindings.customer().status(), CustomerStatusRenderer.get())
                .withClass("statut statut-md")
                .withClass(ResponsiveHidden.XS_AND_LESS)
        .addLabelColumn(new ResourceModel("business.customer.sector.short"), Bindings.
↪customer().sector())
                .showPlaceholder()
                .withClass("code code-sm")
                .withClass(ResponsiveHidden.XS_AND_LESS)
        .decorate()
                .count("customer.list.result.count")
                .ajaxPagers()
        .build("results");
```

Data source

You may provide either a `ISequenceProvider` or a `IDataProvider` to the `start` method as a data source. The resulting data table will contain exactly one row for each element provided by your data source.

Supported column types

Here is a list of the built-in column types:

- Label columns (`addLabelColumn`), which display a simple textual label derived from the underlying value (through the use of a Renderer). Optionally, the label may be wrapped in a link, or have a side link (a link on a side button) appended.

- Bootstrap label columns (`addBootstrapLabelColumn`), which display a textual label with a background color and prepended icon that all depend on the underlying value. Optionally, the label may be wrapped in a link, or have a side link (a link on a side button) appended.
- Bootstrap badge columns (`addBootstrapBadgeColumn`), which display a badge with a background color and an icon that depend on the underlying value. Optionally, the label may be wrapped in a link, or have a side link (a link on a side button) appended.
- Action columns (`addActionColumn`), which display one or more buttons, each button being either:
- A link to a bookmarkable page
- An action link: a link which will trigger execution of arbitrary code (with or without a confirmation popup)
- An **ajax** action link (with or without a confirmation popup)

If none of the above suits your needs, keep in mind that you may simply use the `fr.openwide.core.wicket.more.markup.repeater.table.builder.DataTableBuilder.addColumn(ICoreColumn<T, S>)` method and pass your own column implementation as a parameter. Most of the time, you will simply have to extend `fr.openwide.core.wicket.more.markup.repeater.table.column.AbstractCoreColumn<T, S extends ISort<?>>` and implement `populateItem(Item<ICellPopulator<T>>, String, IModel<T>)` so as to add a `Fragment` defined in your own component.

Adding components around the table

Super headers

You may add arbitrary rows above or below the data table by calling `addTopToolbar` or `addBottomToolbar` and then adding components, optionally attributing a colspan to each of them. This is great in particular if you want to add headers that span multiple columns above your column headers.

Simple title and pager

You may create a "decorated" table, with a top title and pagers, by calling `decorate` after having defined your columns. You may then define the title (optionally making it dependent on the result count, by calling `count`), add top and/or bottom pagers (`.pagers`, `.ajaxPagers`, ...), or even add arbitrary add-ins (`.addIn`).

Bootstrap panel

You may create a "decorated" table as above, but with Bootstrap styling, wrapped in a Bootstrap panel. Just call `bootstrapPanel` instead of `decorate`, and proceed the same as with `decorate`.

RefreshingViews

Compared to the `DataTableBuilder`, the `RefreshingViews` are a lower-level way of displaying collections.

When to use it?

Whenever you can't use the `DataTableBuilder`:

- You don't want a HTML table, but just some repeating `divs` or `lis` (or any other markup, really)

- You want a HTML table, but it's too complex and can't be built using the `DataTableBuilder`. For example you may need multiple `<tr>` for each element in your collection, or you may need to repeat columns instead of rows.

Overview

`RefreshingViews` are generally used this way:

Panel's HTML:

```
...

<div wicket:id="item">
    <span wicket:id="content1" />
    <wicket:container wicket:id="content2" />
</div>
...
```

Panel's Java:

```
add(new SubclassOfRefreshingView<MyItem>("item", dataProvider) {
    @Override
    public void populateItem(final Item<MyItem> item) {
        item.add(new Label("content1", new ResourceModel("some.resource.key")));
        item.add(new SomePanel("content2", item.getModel());
    }
})
```

Which view to use?

It depends on you data source:

- for a `IDataProvider`, use a `SequenceView` (or wicket's `DataView`, which for now should do the same job, but may not gain as much features as SequenceView in the future).
- for a `ISequenceProvider`, use a `fr.openwide.core.wicket.more.markup.repeater.sequence.SequenceView`
- for a `ICollectionModel<T, ?>`, use a `fr.openwide.core.wicket.more.markup.repeater.collection.CollectionView`.
- for a `IMapModel<K, V, ?>`, use a `fr.openwide.core.wicket.more.markup.repeater.map.MapView`.
- for a `IModel<? extends Collection<T>>`, use a `fr.openwide.core.wicket.more.markup.repeater.collection.CollectionView`. You will need to provide a factory for the collection item models. (see *below*).
- for a `IModel<? extends Map<K, V>>`, use a `fr.openwide.core.wicket.more.markup.repeater.map.MapView`. You will need to provide a factory for the map key models (see *below*).

Please note that all of the above provide a paging mechanism, but it will only be efficient if your data source is a properly implemented `IDataProvider` or `ISequenceProvider`. Otherwise, the whole data set will be loaded, and then reduced to the current page's data.

Item models

The `RefreshingViews` need to obtain a reference to the collection's item model, in order to handle manipulations of this item (for instance, a click on a button in a table row mapped to an item).

The way you will implement the "item model factory" will depend on your data source:

- With Wicket's built-in `IDataProvider`, the `IDataProvider` itself will provide the item model through its `model(T)` method. This method is the "item model factory".
- With OWSI-Core's `ISequenceProvider` the provided items are already wrapped in models: the `iterator(long, long)` method returns an `Iterator<? extends IModel<T>>`. The `ISequenceProvider` itself is the "item model factory". This also applies to `ICollectionModel` and `IMapModel`.
- With an `IModel<? extends Collection<T>>` (be it a `LoadableDetachableModel`, a `BindableModel`, or anything else), nothing in the data source itself allows to build item models. That's why most views defined above require you to provide a `Function<? super T, ? extends IModel<T>>` that will serve as an "item model factory".

When a `Function<? super T, ? extends IModel<T>>` is required, you may:

- Use `GenericEntityModel.factory()` if your items are `GenericEntitys`
- Use `Models.serializableModelFactory()` if your items are serializable (`Integer`, `String`, enums, ...)
- Define your own function if none of the above suits your needs. Take care to make the `Function` also implement `Serializable`, since it will be serialized with the page after the response.

For more advanced needs

Wicket also offers various types of built-in `RepeatingViewss`, but the above should encompass most common needs. Only use these views as a fallback if the above clearly won't do.

`ListView` and `IndexedItemListView`

The main advantage of `ListView` is that you don't need to define a specific model for the collection items: they are (by default) mapped by their index to the collection model.

Be warned, though, that with this mechanism, you may run into issues if you implement user operations on the element's models (like opening a modal, or removing an element) and if your underlying collection's content changes between the initial page rendering and the user request: the operation may end up being executed on the wrong item (because the index may not point the the same collection element anymore).

`ListView` and `IndexedItemListView` should be used very rarely, and only if you really know what you're doing.

Adding or removing items using Ajax without refreshing the whole collection view

Wicket, by default, only allows to add or remove items using Ajax to a collection view by refreshing the whole view.

If, for some reason, you don't want to refresh pre-existing, unremoved items, you may use `fr.openwide.core.wicket.more.ajax.AjaxListeners.refreshNewAndRemovedItems(IRefreshableOnDemandRepeater)`:

```
AjaxListeners.add(
        target,
        AjaxListeners.refreshNewAndRemovedItems(repeater)
);
```

There are some constraints, though:

- `repeater` must implement `IRefreshableOnDemandRepeater` (that's the case for most view provided in OWSI-Core)
- both the repeater's parent and the repeater's items must have `setOutputMarkupId` set to `true`
- newly added items will be added at the end of the repeater's parent (the Wicket parent). If there is some HTML between the repeater and the end of the parent, you'll probably want to wrap your repeater in an `WebMarkupContainer`.
- only some classes that implement `IRefreshableOnDemandRepeater` allow to detect removed elements, so only these will see their removed items removed from the HTML. `RepeatingView` and its subclasses, in particular, will not have their removed elements removed from the HTML.

~~AbstractGenericItemListPanel~~ and ~~GenericPortfolioPanel~~ (don't use this)

These classes should not be used anymore. Anything you can do with a `GenericPortfolioPanel`, you can also do it with a `DataTableBuilder` or (worst case) with `RefreshingViews`.

These classes are kept as-is in order to avoid major refactorings in older projects.

UI User Actions (TODO)

TODO:

- request type (Ajax/non ajax)
- button appearance (bootstrap button with or without label, non-bootstrap button, ...)
- button position (in a form, outside a form, in a table, ...)
- data (submits a form or not)
- redirections (or not)

HTML Markup

The following guidelines should be followed every time you add a button to a page:

- if a click on your button launches javascript code, then use `<button type="button">`. This goes even if your button is outside of a form. This includes buttons triggering an ajax call (`AjaxLink`, `AjaxSubmitLink`, `AjaxButton`) as well as buttons triggering your own javascript.
- if a click on your button submits a form without using ajax, then use `<button type="submit">`.
- if a click on your button simply redirects to a bookmarkable page, then it is an actual HTTP link and you should use `<a>`.

The reason for these guidelines is that, while Wicket does support binding ajax calls to `<a>` (or even to arbitrary markup), Wicket does not, however, prevent the default event handler for this markup to execute when a user clicks. Unfortunately, that means that when a user click on a `<a>` with an ajax call bound to the `click` event, then first the ajax call will be performed, then the default action... Which is, for most browser, a scroll to the top of the page. Which probably isn't what you want.

UI Forms (TODO)

TODO:

- IndependentNestedForm
- Available form components
- How to deal with collections:
- when only adding/removing is required (no element editing)
- when only element editing is required (no add/remove)
- when adding/removing is required **and** element editing is also required
- ...

UI Placeholder and Enclosure

This pages explains how to write Enclosure and Placeholder. Enclosure and Placeholder are component which are displayed or hidden (at page-generation level) based on various input variables.

Conditions and component's visibility

OWSI-Core provides `Condition` base class which provides easy-to-use transformation of a Condition (that resolves to true or false result on an *applies* method's call) as an enclosure or placeholder behavior.

Once your `Condition` created, you can generate the following behaviors :

- thenShow(): behavior that sets `visibilityAllowed` property equal to condition's result (enclosure-like behavior)
- thenHide(): behavior that sets `visibilityAllowed` property equal to negated condition's result (placeholder-like behavior)
- thenShowInternal(): behavior that sets `visible` property equal to condition's result
- thenHideInternal(): behavior that sets `visible` property equal to negated condition's result

By convention, `visibilityAllowed` setting (thenHide(), thenShow()) must be preferred for external impact on component visibility (permission, inter-component dependencies driven by application behaviors).

On the contrary, `visible` setting is used for internal behaviors. For example, for a table + pager widget, to control the pager's visibility in relation to result's page number, as this behavior is driven by an internal state of the component.

To circumvent visibility settings when `visible` and `visibilityAllowed` properties conflict, use of an intermediate component may be a solution.

Component's *enabled* property

`thenEnable()` and `thenDisable()` methods are provided to allow `enabled` property control, based on the same mechanism than visibility control.

Deprecated patterns

Before `Condition`, the following components and behaviors were provided:

- `PlaceholderBehavior`, `EnclosureBehavior`: sets `visibilityAllowed` property (by default) or an alternate property when provided. This behavior can be replaced by a Condition method call:
  - `.add(new PlaceholderBehavior().component(component)):  .add(Condition.componentVisible(component).thenHide())`
  - `.add(new EnclosureBehavior().component(component)):  .add(Condition.componentVisible(component).thenShow())`
  - `.add(new PlaceholderBehavior(ComponentBooleanProperty.VISIBLE).component(component)):  .add(Condition.componentVisible(component).thenHideInternal())`
  - `.add(new EnclosureBehavior(ComponentBooleanProperty.VISIBLE).component(component)):  .add(Condition.componentVisible(component).thenShowInternal())`

When other method than `component()` is used on `...Behavior` object, others `Condition`'s methods or subclasses can be used to provides the right behavior (permission(), anyPermission(), role(), isTrue(), isFalse(), predicate(), ...)

UI Charts and plots

This page explains how to generate data charts using OWSI-Core.

JQPlot

About

JQplot is "a versatile and expandable JQuery plotting plugin" (see the official site). OWSI-Core uses WQPlot in order to ensure low-level integration of JQPlot into wicket, plus a specific maven module (`fr.openwide.core.components:owsi-core-component-wicket-more-jqplot`) in order to provide easy-to-use, high-level integration. Please note that WQPlot was originally a wicketstuff project, which seems no longer maintained.

High-level usage

Architecture

You will encounter four types of components when you'll set up a chart:

- The data provider, which implements `IJQPlotDataProvider`. Its role is extracting data from your service layer (or an `IModel`, or anything you want) and provide it in a standardized form.

- The data adapter, which implements `IJQPlotDataAdapter`. Its role is to take data from a data provider and transform it in the WQPlot types (`BaseSeries`, `NumberSeries`, and so on) that can be used by the panel.
- The panel itself, which extends `JQPlotPanel`. It's a Wicket component that uses the data adapter and WQPlot in order to generate the actual chart.
- The configurers, which implement `IJQPlotConfigurer`. Their role is to customize various JQPlot options.

Using it in your application

Setup global configuration

Add a dependency in your pom to `fr.openwide.core.components:owsi-core-component-wicket-more-jqplot`.

You will also need to import some Spring configuration: add an `@Import` annotation to your webapp in order to import `fr.openwide.core.wicket.more.jqplot.config.JQPlotJavaConfig`.

You may optionally, instead of importing `JQPlotJavaConfig` directly, define you own configuration that extend `JQPlotJavaConfig`. In this case, you will be able to override the default options passed to JQPlot's plots.

Setup your chart

First, setup your data source. Generally it will be a service that returns:

- A `java.util.Map<K,V>` if there's only one data series.
- A `com.google.common.collect.Table<S,K,V>` if there are multiple data series.

See Querying#generating-maps-and-tables for more information about how to easily generate a `Map` or `Table` from a QueryDSL query in your DAO.

Then, pick your data provider:

- `fr.openwide.core.wicket.more.jqplot.data.provider.JQPlotMapDataProvider<S, K, V>` if you've got a service that returns a `Map`. Then you will wrap the service call in a `LoadableDetachableModel`, and pass the model as a constructor parameter to the data provider.
- `fr.openwide.core.wicket.more.jqplot.data.provider.JQPlotTableDataProvider<S, K, V>` if you've got a service that returns a `Table`.
- Or, if for some reason the above won't do, you may implement your own.

Then, pick your chart:

- For bar charts, use `fr.openwide.core.wicket.more.jqplot.component.JQPlotBarsPanel<S, K, V>`
- For line charts, use `fr.openwide.core.wicket.more.jqplot.component.JQPlotLinesPanel<S, K, V>`
- For pie charts, use `fr.openwide.core.wicket.more.jqplot.component.JQPlotPiePanel<K, V>`
- For stacked bar charts, use `fr.openwide.core.wicket.more.jqplot.component.JQPlotStackedBarsPanel<S, K, V extends Number & Comparable<V>>`
- For stacked lines charts, use `fr.openwide.core.wicket.more.jqplot.component.JQPlotStackedLinesPanel<S, K, V>`

Then, pick your data adapter (it will be the bridge between your panel and your data provider):

- If your data references keys from a discrete domain (i.e. with a small, finite number of values in the observed range), use`fr.openwide.core.wicket.more.jqplot.data.adapter.JQPlotDiscreteKeysDataAdapter<S, K, V>`. Please keep in mind that if you want your X axis to have a linear scale, you should either ensure that all keys are represented in your data, or provide to the data adapter a model containing all of the expected keys (so that the adapter can generate a linear axis).
- If, on the other hand, your data references keys from a continuous domain (i.e. with a high or infinite number of values in the observed range), use one of these:
- `fr.openwide.core.wicket.more.jqplot.data.adapter.JQPlotContinuousDateKeysDataAdapter<S K, V>`
- `fr.openwide.core.wicket.more.jqplot.data.adapter.JQPlotContinuousNumberKeysDataAdapter K extends Number, V extends Number>`
- Or, if for some reason the above won't do, you may implement your own.

Note: pie charts are a special case: you won't need a data adapter, just a data provider.

And finally, put all of this together: you've got a basic chart. Some examples are provided in the "Statistics" page of OWSI-Core's wicket showcase (`fr.openwide.core.showcase:wicket-showcase`).

To go further:

- You may add one or several configurers to the chart in order to customize JQPlot options: either pick a configurer from `fr.openwide.core.wicket.more.jqplot.config.JQPlotConfigurers` or use your own implementations.
- You may wrap your data adapter in order to customize the data passed to JQPlot: see `fr.openwide.core.wicket.more.jqplot.data.adapter.JQPlotDataAdapters`. You may for instance add a percentage in the tooltip shown when hovering over a stacked bar.

Low-level usage

If `fr.openwide.core.components:owsi-core-component-wicket-more-jqplot` is not flexible enough for your needs, you may still use raw wqplot. Some examples are provided in the "Statistics" page of OWSI-Core's wicket showcase (`fr.openwide.core.showcase:wicket-showcase`).

Keep in mind that you'll need to explicitly update jqplot options whenever the data changes (the X axis ticks, for instance, if you declared them explicitly).

Troubleshooting

The plots are not drawn

Please be aware that the supporting markup of your plots must be initially visible (i.e. not `display: none`) if you want the plots to be drawn automatically. If, for instance, your plots are located in an initially inactive (hidden) tab, then they will not be drawn automatically.

To address the specific case of plots in an initially inactive Bootstrap tab, you may use the following snippet:

```
tabContainer.add(new JQPlotReplotBehavior("shown.bs.tab"));
```

This will ensure that plots are "replotted" each time the user switches tabs.

Other plotting libraries

No other plotting library is integrated into OWSI-Core at the moment.

CHAPTER 10

Contributing to upstream

Hibernate

Our cloned repo: https://github.com/openwide-java/hibernate-orm

Resources

- Full contribution procedure (also here, but it seems to be almost the same)
- How to develop using Eclipse (see below for more concrete explanations)

Developing

Hibernate uses Gradle. This means some pain if you haven't had to work with it in Eclipse, ever.

In order to build using gradle:

- Check that your default JRE is recent enough (tested with JRE8 on Hibernate 5.0, it should work)
- Generate the Eclipse `.project` files: `./gradlew clean eclipse --refresh-dependencies`
- Install the Gradle Eclipse plugin from this update site: `http://dist.springsource.com/release/TOOLS/gradle`
- Import the projects **as standard Eclipse projects** (Gradle import seems to mess things up, at least with Eclipse 4.3)
- Pray that everything builds right. I personally couldn't make every project compile, but what I had to work on did, so...

Testing

Running tests locally

Launch your test this way (example for a test in hibernate-core):

```
./gradlew :hibernate-core:test --tests 'MyTestClassName'
```

Running tests locally, with database vendor dependency

If your test relies on a specific database vendor, you'll need to do the following in order to run it locally (examples for PostgreSQL):

- Specify the Dialect to use with the following option `-Dhibernate.dialect=org.hibernate.dialect.PostgreSQL9Dialect`
- Specify JDBC information: `-Dhibernate.connection.url=...`, `-Dhibernate.connection.username=...`, `-Dhibernate.connection.password=...`, `-Dhibernate.connection.driver_class=...`
- Provide the vendor-specific driver jar. I couldn't find a way to do it other than changing the `hibernate-core/hibernate-core.gradle` file and adding this line in the `dependencies` block: `testCompile( 'org.postgresql:postgresql:9.4-1200-jdbc41' )`

You'll end up launching your test this way (example for a test in hibernate-core):

```
./gradlew -Dhibernate.dialect=org.hibernate.dialect.PostgreSQL9Dialect -Dhibernate.
↪connection.url=jdbc:postgresql://localhost:5432/hibernate_test -Dhibernate.
↪connection.username=hibernate -Dhibernate.connection.password=hibernate -Dhibernate.
↪connection.driver_class=org.postgresql.Driver :hibernate-core:test --tests
↪'MyTestClassName'
```

Hibernate Search

Our cloned repo: https://github.com/openwide-java/hibernate-search

Resources

- Full contribution procedure

CHAPTER 11

Assertion

Check non null

It is recommended to use `Objects.requireNonNull` (with `Objects` being the one of Java 8):

```
Objects.requireNonNull(executionResult, "executionResult must not be null");
```

More advanced conditions

As for more advanced conditions, it is recommended to use Guava's `Preconditions`.

In Wicket code

In Wicket code, you can use `Args.notNull`, `Args.notEmpty`, `Args.isTrue`, `Args.isFalse`. Be careful to use the `Args` class from Wicket.

CHAPTER 12

Predicate (TODO)

CHAPTER 13

Renderer (TODO)

CHAPTER 14

Backend

PropertyService

About

The PropertyService gets and sets the values for your application's properties in a typesafe manner. These might be stored in `*.properties` files (being "immutable" properties) or in database ("mutable" properties).

Architecture

Properties are referenced to using their *id*, which may have one of two types:

- `ImmutablePropertyId` for immutable properties that are linked to configuration.properties.
- `MutablePropertyId` for mutable properties that are stored in database in table Parameter as String. Beware that the other columns of this table are deprecated.

Here are the basic principles:

- Properties are first registered using their ID during the application initialization. The registration involves telling to the `PropertyService` how to convert the property value to and from its string representation, which is done by providing a `Converter` (a Guava type), or (for immutable properties that don't need to be converter back to string) a `Function`.
- During the execution of the application, the property values may be retrived using the service's `get` method, or altered using the `set` method.

Using it in your application

Declare the IDs

You must first declare constants for your property ids. You'd better have two sets of property ids: one for you core (non-UI) project and the other for you webapp (UI) project.

The keys passed as parameters to `AbstractPropertyIds`'s methods are the one used to retrieve the values, either in the `configuration.properties` file or in database.

```
public final class YourAppCorePropertyIds extends AbstractPropertyIds {

    public static final ImmutablePropertyId<Integer> CORE_IMMUTABLE_INTEGER_PROPERTY
↪= immutable("core.immutable.integer.property");
    public static final ImmutablePropertyId<MyType> CORE_IMMUTABLE_MYTYPE_PROPERTY =
↪immutable("core.immutable.mytype.property");

    public static final MutablePropertyId<String> CORE_MUTABLE_STRING_PROPERTY =
↪mutable("core.mutable.string.property");

}
```

```
public final class YourAppWebappPropertyIds {

    public static final ImmutablePropertyId<Long> WEBAPP_IMMUTABLE_LONG_PROPERTY =
↪immutable("webapp.immutable.long.property");
    public static final ImmutablePropertyId<MyType> WEBAPP_IMMUTABLE_MYTYPE_PROPERTY
↪= immutable("webapp.immutable.mytype.property");

    public static final MutablePropertyId<String> WEBAPP_MUTABLE_STRING_PROPERTY =
↪mutable("webapp.mutable.string.property");

}
```

Register the properties

In your core module set up a JavaConfig as follows.

Beware that `AbstractApplicationPropertyConfig` also declares the property service, which is a singleton. Thus it must only be used once (in your core module).

```
@Configuration
public class YourAppCoreApplicationPropertyConfig extends
↪AbstractApplicationPropertyConfig {

    @Override
    public IMutablePropertyDao mutablePropertyDao() {
        return new ParameterDaoImpl();
    }

    @Override
    public void register(IPropertyRegistry registry) {
        // register core properties here
    }

}
```

In your webapp module set up a JavaConfig as follows:

```
@Configuration
public class YourAppWebappApplicationPropertyRegisterConfig extends
↪AbstractApplicationPropertyRegistryConfig {

    @Override
```

```
    public void register(IPropertyRegistry registry) {
        // register webapp properties here
    }

}
```

`IPropertyRegistry` provides a bunch of methods to register properties. See details below.

Access the properties

Anywhere an `IPropertyService` is available (it can be injected), do the following:

```
@Autowired
private IPropertyService propertyService;

// Get a value
propertyService.get(YourAppCorePropertyIds.CORE_IMMUTABLE_MYTYPE_PROPERTY);

// Set a value (only for mutable properties)
propertyService.set(YourAppCorePropertyIds.CORE_MUTABLE_STRING_PROPERTY, "NewValue");
```

Details about registration

Important note

It is strongly recommended to define a default value for collection properties in order to always get a collection even if the value is `null`.

Examples

Basic immutable property

```
public static final ImmutablePropertyId<String> MAINTENANCE_URL = immutable(
↪"maintenance.url");
```

```
propertyService.registerString(MAINTENANCE_URL);
```

Immutable property with custom converter/function

```
public static final ImmutablePropertyId<Date> DATE_PICKER_RANGE_MAX_DATE = immutable(
↪"datePicker.range.max.yearsFromNow");
```

```
propertyService.registerImmutable(DATE_PICKER_RANGE_MAX_DATE, new Function<String,␣
↪Date>() {
    @Override
    public Date apply(String input) {
        Integer years = Ints.stringConverter().convert(input);
        if (years == null) {
            return null;
        }
        return DateUtils.truncate(
                DateUtils.addYears(new Date(), years),
                Calendar.DAY_OF_MONTH
```

```
        );
    }
});
```

Mutable property with dynamic key with default value

```
public static final MutablePropertyIdTemplate<Boolean> DATA_UPGRADE_DONE_TEMPLATE =␣
↪mutableTemplate("dataUpgrade.%1s");
public static final MutablePropertyId<Boolean> dataUpgrade(IDataUpgrade dataUpgrade) {
    return DATA_UPGRADE_DONE_TEMPLATE.create(dataUpgrade.getName());
}
```

```
propertyService.registerBoolean(DATA_UPGRADE_DONE_TEMPLATE, false);
```

Property - enum

```
public static final ImmutablePropertyId<Environment> ENVIRONMENT = immutable(
↪"environment");
```

```
propertyService.registerEnum(ENVIRONMENT, Environment.class, Environment.production);
```

Property - collection

```
public static final ImmutablePropertyId<List<MediaType>> FICHIER_PIECE_JOINTE_MEDIA_
↪TYPES = immutable("fichier.pieceJointe.mediaTypes");
```

```
propertyService.register(FICHIER_PIECE_JOINTE_MEDIA_TYPES, new␣
↪StringCollectionConverter<>(Enums.stringConverter(MediaType.class), Suppliers2.
↪<MediaType>arrayList()));
```

E-mail notifications (TODO)

HistoryLog & Audit

Principles

The history log machinery is designed to track the action performed on our business entities.

It can also be used to track the differences but this feature should be used with caution as it can be quite time consuming to configure and has a negative impact on performance.

Logs are performed upon commit (just before the commit), in order to:

- Remove duplicate logs
- Ensure that all edits on the entities are over, so that a diff can safely be computed

Warning: Please note that when performing a batch processing, you should take care of calling `ITransactionSynchronizationTaskManagerService.beforeClear()` before flushing the indexes and clearing the Hibernate session, so that the logs can be safely flushed too.

Using it in your application

Setting up

The basic application provide a basic (but functional) template for setting up the HistoryLog. See package `fr.openwide.core.basicapp.core.business.history` for more information.

The minimal set of classes to define includes:

- The concrete enum representing `HistoryEvents` (create, update, sign-in, ...)
- The concrete class for your `HistoryLogs`, with (optionally) custom fields.
- Your `HistoryLogDao` and `HistoryLogService`, which may simply extend the provided abstract classes
- Your own implementation of the bean that will store additional informations about log objects: `HistoryLogAdditionalInformationBean`. This bean serves as a way to pass additional parameters to the `HistoryLogService`: secondary objects (for actions performed on more than one object), contextual information (to enable later search on logs with filters on objects *linked* to the main object at the time of the action), ... **It's typically not recommended to randomly use the object1...4 fields (see `AbstractHistoryLogAdditionalInformationBean` for more information).**

You may also want to define a `HistoryLogSearchQuery` for querying you history. An example is provided in the basic application.

Recording simple logs

```
historyLogService.log(HistoryEventType.SIGN_IN, user,␣
↪HistoryLogAdditionalInformationBean.of(user));
```

Recording diff-enabled logs

Linking a full diff of the main object to the log:

```
historyLogService.logWithDifferences(HistoryEventType.UPDATE, person,␣
↪HistoryLogAdditionalInformationBean.of(person), userDifferenceService);
```

The code above will add to the log every change on every relevant fields of the given entity. This is typically what we do upon updates.

Sometimes, though, you only want to record changes on a subset of the entity fields. This is typically what's required upon create/delete (where we know that almost all fields will change), but it may be done upon updates, too.

Here's how to link a minimal diff of the main object to the log:

```
historyLogService.logWithDifferences(HistoryEventType.UPDATE, person,
                HistoryLogAdditionalInformationBean.of(person),
                userDifferenceService.getMinimalDifferenceGenerator(),
                userDifferenceService);
```

Optionnally, you may pass to those methods one or several `IDifferenceHandler<T>`, which are ways for you to inspect a diff, and do something based on that information. This is typically used to update a date on the entity when some field changed.

For more information about setting up a difference service, see DifferenceService.

Displaying the logs on the user interface

Basic stuff

The basic application contains everything needed to display the audit. See `UserHistoryLogPanel`.

Renderers/Converters

The history log machinery uses the renderers/converters infrastructure so you need to define either converters (in your `YourAppApplication.newConverterLocator()` or renderers (in `YourAppWebappConfig.rendererService()`).

Resource keys for difference display

When the label of a field needs to be displayed, several resource keys are tried, in this order:

- `history.difference<entity resource key>.<property.path>`
- `business<entity resource key>.<property path>`
- `history.difference.common.<property.path>`
- `business.<property path>`

The first key that actually exists in you properties is used. This behavior is defined in the basic application class `HistoryDifferencePathRenderer`.

The resource key for each entity is defined in `AbstractHistoryRenderer`.

Difference Service (TODO)

Principles

OWSI-Core provides a way for applications to compute a diff between two objects, i.e. of recursively computing field-by-field differences between a "working" version and a "reference" version of an object.

The diff infrastructure is powered by Java-object-diff. OWSI-Core itself provides:

- basic setup of java-object-diff
- abstract bases for your diff services, with clearly identified points of configuration
- interfaces to integrate your diff services to other parts of your application (mainly HistoryLog & Audit)
- a way for you to perform a diff between an object in your Hibernate session and the version currently in your database

Using it in your application

Setting up

TODO: this part is still evolving. See https://trello.com/c/gUTpyoMr in particular.

Performing a diff between two objects

```
// Obtain your IDifferenceGenerator
// You may also use differenceService.getMinimalDifferenceGenerator(), depending on␣
↪your use case
IDifferenceGenerator generator = differenceService.getMainDifferenceGenerator();
// Perform the  diff
Difference<T> diff = generator.diff(workingObject, referenceObject);
```

Performing a diff between your (modified) object and the version in your DB

```
// Obtain your IDifferenceGenerator
// You may also use differenceService.getMinimalDifferenceGenerator(), depending on␣
↪your use case
IDifferenceFromReferenceGenerator generator = differenceService.
↪getMainDifferenceGenerator();
// Perform the  diff
Difference<T> diff = generator.diffFromReference(workingObject);
```

Using it in the history log machinery

See HistoryLog & Audit.

Task Executor

About

OWSI-Core comes with a task service allowing to persist and execute background tasks.

Configuration

The task service can manage multiple queues (typically, a queue for short tasks and a queue for long tasks).

```
@Configuration
public class YourAppTaskManagementConfig extends AbstractTaskManagementConfig {

    @Override
    @Bean
    public Collection<? extends IQueueId> queueIds() {
        return EnumUtils.getEnumList(YourAppTaskQueueId.class);
    }

}
```

The task manager can be configured with the following properties:

- `task.startMode`: auto or manual: define if the task manager is started automatically at application startup
- `task.queues.config.<queueId>.threads`: number of execution threads for the given queue (by default 1)
- `task.stop.timeout`: timeout of a task: the task is stopped if its execution lasts longer than the timeout

Queueing a task

A task extends `AbstractTask`. It's going to be serialized as JSON (using Jackson) in order to be persisted.

A task has a name (for identification by a human beand a type.

You can define the queue used by implementing the `selectQueue()` method. It allows you to choose a different queue depending on the parameters passed (e.g. selecting a different queue based on the task type, or based on how many items you have to export).

It's quite easy to submit a new task:

```
YourTask task = new YourTask(user, parameters...);
queuedTaskHolderService.submit(task);
```

Inside a task

TODO GSM

Console

The administration console contains an administration UI for the task manager.

External Link Checker

About

OWSI-Core has a service called the external link checker. It's used to check that external links stays valid.

It's currently used in production to validate several 100k links.

It is based on the HTTPClient library. It first tries a HEAD request to limit the bandwidth usage, it tries a GET request if the HEAD request fails.

If a link is unreachable, it is marked as `OFFLINE`. After several checks being offline (to avoid transient errors), the link is marked as `DEAD_LINK`.

Model

The link must be wrapped in an `ExternalLinkWrapper`:

```
@OneToOne(fetch = FetchType.LAZY, cascade = { CascadeType.ALL }, orphanRemoval = true)
private ExternalLinkWrapper externalLinkWrapper;
```

The status of the link is available in the `ExternalLinkWrapper`. A link is considered invalid if it's in the `DEAD_LINK` status.

Usage

Maven

Starting with OWSI-Core 0.11, ExternalLinkChecker has its own Maven module:

```
<dependency>
    <groupId>fr.openwide.core.components</groupId>
    <artifactId>owsi-core-component-jpa-externallinkchecker</artifactId>
    <version>${project.version}</version>
</dependency>
```

Configuration

The configuration is as follows:

```
externalLinkChecker.timeout=30000
externalLinkChecker.userAgent=Your user agent
externalLinkChecker.batchSize=400
externalLinkChecker.retryAttemptsNumber=4
externalLinkChecker.maxRedirects=5
externalLinkChecker.cronExpression=0 0/15 3-6,18-23 * * ?
```

The cron expression must be used to configure a scheduled task which launches `externalLinkCheckerService.checkBatch()`.

You can ignore links by adding regexps using the `externalLinkCheckerService.addIgnorePattern(pattern)` method.

In `YourAppCoreCommonConfig`, you have to:

- add `JpaExternalLinkCheckerBusinessPackage` to the `basePackageClasses` of `@ComponentScan`

In `YourAppCoreCommonJpaConfig`, you have to:

- add `JpaExternalLinkCheckerBusinessPackage` to the package scanned for entities in `applicationJpaPackageScanProvider()`
- declare an Hibernate interceptor:

```
        @Bean
        public Interceptor hibernateInterceptor() {
                return new ChainedInterceptor()
                                .add(new ExternalLinkWrapperInterceptor());
        }
```

Table import

About

OWSI-Core comes with an optional feature that allows you to import data from table structured documents. Today, it means that you can import data from documents under either CSV or Excel format.

A classical case which can be treated with this feature is the following:

1. Iterate over rows in a sheet.
2. For each row, get data from some columns and check its consistency.
3. Save each row as an element of a table in your database (or in a collection, etc.).
4. At the end of the treatment, commit the transaction if no error occurs or rollback all changes if so.

The main benefit of it is to trace all errors / warnings in a unique run. It avoids a boring couple of "run - error on line 84 - run again - error on line 123 - etc.".

Include sub-module

To be able to use the classes we will mention later in this documentation, you need to include the Maven sub-module containing them.

```
<dependency>
    <groupId>fr.openwide.core.components</groupId>
    <artifactId>owsi-core-component-imports</artifactId>
    <version>${owsi-core.version}</version>
</dependency>
```

Interfaces and implementation

The `fr.openwide.core.imports.table.common` module is currently divided in 3 parts:

- `common`: all interfaces and abstract class common to all implementation ;
- `apache.poi`: implementation to read Excel format using the Apache POI API ;
- `opencsv`: implementation to read CSV format using Opencsv.

Using it in your application

Notes about the sample coming:

1. We will use the `apache.poi` implementation. However, it should be really close of it to deal with a CSV file.
2. We will have a main class `CarDataUpgrade` containing all other classes. If you project architecture needs to do something different, you are obviously free to make separate classes.

Declare you columns

First of all, you need to declare how your data file is structured. Each column defines :

- its position (0 based) ;
- its type.

```
private static final class CarSheetColumnSet extends ApachePoiImportColumnSet {
    private final Column<Long> id = withIndex(0).asLong().build();
    private final Column<String> brand = withIndex(3).asString().build();
    private final Column<String> referenceName = withIndex(4).asString().build();
    private final Column<Date> firstSellingDate = withIndex(5).asDate().build();
}
```

In the definition above, we can see that columns at positions 1, 2 and 3 are ignored. They may contains informative data, or whatever else, and don't concern the data import. We just ignore them in our structure declaration.

Iterate over sheets

After that, you will need to use an `IExcelImportFileScanner` to iterate over sheets in the Excel file. Our sample contains only one sheet, but you could have a more complex workbook and you could deal with it making different cases inside the `visitSheet()` method.

```
@Autowired
private ITransactionScopeIndependantRunnerService␣
↪transactionScopeIndependantRunnerService;

[...]

private final class CarExcelFileImporter {
    private final ApachePoiImportFileScanner scanner = new␣
↪ApachePoiImportFileScanner();

    private final CarSheetColumnSet carSheetColumnSet = new CarSheetColumnSet();

    public void doImportExcelFile(InputStream stream, String fileName) throws␣
↪TableImportException {
        scanner.scan(stream, fileName, SheetSelection.ALL, new IExcelImportFileVisitor
↪<Workbook, Sheet, Row, Cell, CellReference>() {
            @Override
            public void visitSheet(final ITableImportNavigator<Sheet, Row, Cell,␣
↪CellReference> navigator, Workbook workbook, final Sheet sheet)
                    throws TableImportException {
                transactionScopeIndependantRunnerService.run(false, new Callable<Void>
↪() {
                    @Override
                    public Void call() throws Exception {
                        ITableImportEventHandler eventHandler = new␣
↪LoggerTableImportEventHandler(LOGGER);
                        importCarSheet(navigator, sheet, eventHandler);
                        eventHandler.checkNoErrorOccurred();
                        return null;
                    }
                });
            }
        });
    }

    [...]
}
```

Transaction

Please note that we use an `ITransactionScopeIndependantRunnerService` to be sure that all database actions are performed in a unique transaction. It allows us to log all potential errors and rollback all changes only at the end of the Excel sheet.

Event handler

The `ITableImportEventHandler` allow the import process we build to log some messages about the treated data. We can initialize it with a `TableImportNonFatalErrorHandling` mode:

- `THROW_ON_CHECK` (default) will throw an exception when the `checkNoErrorOccurred()` is called ;

- `THROW_IMMEDIATELY` will throw an exception when the event is handle ; following rows are not treated.

Iterates overs rows

Now that we are in a sheet, we can iterate over its rows. We can do it simply like shown below.

```
private void importCarSheet(ITableImportNavigator<Sheet, Row, Cell, CellReference>␣
↪navigator,
        Sheet sheet, ITableImportEventHandler eventHandler) throws␣
↪TableImportContentException, TableImportMappingException {
    CarSheetColumnSet.TableContext sheetContext = carSheetColumnSet.map(sheet,␣
↪navigator, eventHandler);

    for (CarSheetColumnSet.RowContext rowContext : Iterables.skip(sheetContext, 1)) {
        CarSheetColumnSet.CellContext<Long> idCell = rowContext.
↪cell(carSheetColumnSet.id);
        CarSheetColumnSet.CellContext<String> brandCell = rowContext.
↪cell(carSheetColumnSet.brand);
        CarSheetColumnSet.CellContext<String> referenceNameCell = rowContext.
↪cell(carSheetColumnSet.referenceName);
        CarSheetColumnSet.CellContext<Date> firstSellingDateCell = rowContext.
↪cell(carSheetColumnSet.firstSellingDate);

        Long idFromXls = idCell.getMandatory("Car id is mandatory.");
        String brandFromXls = brandCell.getMandatory("Brand is mandatory");
        String referenceNameFromXls = referenceNameCell.get();
        Date firstSellingDateFromXls = firstSellingDateCell.get();

        Car car = carService.getById();
        if (car != null) {
            // The id cannot be found, the car will not be updated
            idCell.error("Car {} not found.", idFromXls);
            continue;
        }

        if (firstSellingDateFromXls != null && firstSellingDateFromXls.after(new␣
↪Date())) {
            // The first selling date should be wrong, but it's a secondary␣
↪information,
            // the car will be updated with this information
            firstSellingDateCell.warn("Car {} - The first selling date ({}) is in the␣
↪future.", firstSellingDateFromXls);
        }

        car.setBrand(brandFromXls);
        car.setReferenceName(brandFromXls);
        car.setFirstSellingDate(firstSellingDateFromXls);

        try {
            carService.update(car);
        } catch (ServiceException | SecurityServiceException e) {
            LOGGER.error("An error occured while updating a car.", e);
            rowContext.error("An error occured while updating a car.");
        }
    }
}
```

Getting values

You can handle some basic behavior while getting values:

- treat result with some functions at column description like `withDefault()`, `extract()` or `capitalize()`
- raise an error in case of missing value with `getMandatory()` instead of a simple `get()`

Error location

Please note that using cell or row context to record logs will produce messages with precise location details (i.e.:

```
(at TableImportLocation[fileName=my-pretty-cars.xlsx,tableName=cars,rowIndex (1-
↪based)=123,cellAddress=F123])
```

Eating the data file

Obviously, we need to give the file to our data import mechanic. Here we get this file from the project's resources, but we also could get it from the file system, from a user input, etc.

```
public class CarDataUpgrade implements IDataUpgrade {

    private static final Logger LOGGER = LoggerFactory.getLogger(CarDataUpgrade.
↪class);

    private static final String FILE_PATH = "/dataupgrade/my-pretty-cars.xlsx";

    [...]

    public void perform() throws TableImportException {
        InputStream inputStream = null;
        try {
            LOGGER.info("Car import...");
            inputStream = CarDataUpgrade.class.getResourceAsStream(FILE_PATH);
            new CarExcelFileImporter().doImportExcelFile(inputStream, FilenameUtils.
↪getName(FILE_PATH));
            LOGGER.info("Car import completed.");
        } finally {
            IOUtils.closeQuietly(inputStream);
        }
    }
}
```

CHAPTER 15

UI

UI Bootstrap

We use Bootstrap 3 as our main source for UI components.

Core

Application

Stylesheets

The basic application comes with 2 stylesheets:

- `styles.less`: the stylesheet for most of the UI
- `service.less`: the stylesheet used for all the login process (sign in, forgot password...)

Variables

You can set the value of the variables by editing the file `<yourapp>/web/common/template/styles/variables.less`.

UI Font Awesome

We use Font Awesome as our main source for icons.

Icon reference: http://fortawesome.github.io/Font-Awesome/icons/

Examples: http://fortawesome.github.io/Font-Awesome/examples/

Please be especially aware of the existence of `fa-fw` to align properly the icons in lists.

UI Plugins (JS, CSS) (TODO)

TODO list plugins and explain (briefly) why and when to use them.

CHAPTER 16

Infrastructure Apache

Default configuration for an Apache Vhost

```
<VirtualHost *:443>
        ServerName www.siteurl.com
        ServerAlias technical.alias.net

        DocumentRoot /data/services/web/<sitename>/site

        ProxyErrorOverride On
        ProxyPass /static-content/ !
        ProxyPass /errors/ !
        ProxyPass /server-status !
        ProxyPass         / ajp://localhost:8009/ keepalive=on timeout=3000 ttl=300
        ProxyPassReverse  / ajp://localhost:8009/

        ErrorDocument 403 /errors/403.html
        ErrorDocument 404 /errors/404.html
        ErrorDocument 500 /errors/500.html
        ErrorDocument 503 /errors/503.html

        AddOutputFilterByType DEFLATE application/x-javascript text/html text/xml␣
↪text/css text/javascript

        AddDefaultCharset UTF-8
        AddCharset UTF-8 .js .css

        <Directory /data/services/web/<sitename>/site>
                Require all granted
        </Directory>

        SSLEngine on

        SSLProtocol all -SSLv2 -SSLv3
```

```
        SSLCipherSuite ALL:!ADH:!EXPORT:!SSLv2:RC4+RSA:+HIGH:+MEDIUM:+LOW

        SSLCertificateFile /etc/httpd/ssl/<sitename>.crt
        SSLCertificateKeyFile /etc/httpd/ssl/<sitename>.key.unsecure
        SSLCACertificateFile /etc/httpd/ssl/ca-thawte.crt

        CustomLog /data/log/web/<sitename>-access_log combined
        ErrorLog /data/log/web/<sitename>-error_log
</VirtualHost>
```

CHAPTER 17

Infrastructure Tomcat (TODO)

CHAPTER 18

Documentation

Miscelleaneous

The documentation is located at http://owsi-core-doc.readthedocs.io/en/latest/index.html

Contributing to the doc

Install the documentation

To install the documentation on your computer, follow these steps :

First clone the git repository owsi-core-doc

```
git@github.com:openwide-java/owsi-core-doc.git
```

When the clone is over, execute the installation script :

```
cd ~/git/owsi-core-doc
./bootstrap.sh
```

When the script ends, the documentation installation is finished.

Build the documentation locally

A few commands to interact with the documentation locally :

```
invoke docs
```

The command 'docs' builds the documentation and generates the html files.

```
invoke docs-live
```

The command 'docs-live' builds the documentation and opens it in a new tab of your browser, allowing you to see your modifications as soon as you save them.

```
invoke docs-clean
```

The command 'docs-clean' cleans all the build directory and files.

Build the documentation on ReadTheDocs

To modify the online documentation, you just have to push your modifications on the owsi-core-doc git repository. A webhook is set and will automatically rebuild the documentation everytime you push something.

CHAPTER 19

Project installation

Prerequisite

To run the basic-application or any project that use the basic-application as an outline, you need to initialize a database. To do so, you need to create a user first, and then create a database with the user you have created as its owner.

```
createuser -U postgres -P basic_application
createdb -U postgres -O basic_application basic_application
psql -U postgres hello_world
#Here you are connected to the database as the user postgres
DROP SCHEMA public;
\q
psql -U hello_world hello_world
#Here you are connected to the database as the user hello_world
CREATE SCHEMA hello_world;
```

After that, you can populate your database with some date by running the class *BasicApplicationInitFromExcelMain.java* as a java application. This class is located in basic-application-init.

CHAPTER 20

Build, deploy and exploit the Maven archetype

To initialize a new project based on the basic-application, you have to follow several steps. Each steps are detailed one by one in the following sections.

Build the archetype

Place yourself in the folder owsi-core-parent/basic-application.

- to install the archetype locally:

```
./build-and-push-archetype.sh ../basic-application/ local
```

- to install the archetype on our repository:

```
./build-and-push-archetype.sh ../basic-application/ snapshot
```

Generate a new project

Place yourself in a new folder or somewhere like /tmp/. This command will generate a new folder where you are containing your new project.

using your local repository:

```
mvn archetype:generate -DarchetypeVersion=X.X -DarchetypeCatalog=local -
↪DartifactId=your-artifact-id -DgroupId=your.group.id -Dversion=0.1-SNAPSHOT -
↪Dpackage=com.your.package -DarchetypeApplicationNamePrefix="YourApplication" -
↪DarchetypeSpringAnnotationValuePrefix="yourApplication" -
↪DarchetypeFullApplicationName="Customer - Your application" -
↪DarchetypeDatabasePrefix=c_database_prefix -DarchetypeDataDirectory=your-data-
↪directory
```

using the snapshot repository:

```
mvn archetype:generate -DarchetypeCatalog=https://openwide:openwide@projects.openwide.
↪fr/services/nexus/content/repositories/owsi-core-snapshots/ -DartifactId=your-
↪artifact-id -DgroupId=your.group.id -Dversion=0.1-SNAPSHOT -Dpackage=com.your.
↪package -DarchetypeApplicationNamePrefix="YourApplication" -
↪DarchetypeSpringAnnotationValuePrefix="yourApplication" -
↪DarchetypeFullApplicationName="Customer - Your application" -
↪DarchetypeDatabasePrefix=c_database_prefix -DarchetypeDataDirectory=your-data-
↪directory
```

using the release repository:

```
mvn archetype:generate -DarchetypeCatalog=https://openwide:openwide@projects.openwide.
↪fr/services/nexus/content/repositories/owsi-core/ -DartifactId=your-artifact-id -
↪DgroupId=your.group.id -Dversion=0.1-SNAPSHOT -Dpackage=com.your.package -
↪DarchetypeApplicationNamePrefix="YourApplication" -
↪DarchetypeSpringAnnotationValuePrefix="yourApplication" -
↪DarchetypeFullApplicationName="Customer - Your application" -
↪DarchetypeDatabasePrefix=c_database_prefix -DarchetypeDataDirectory=your-data-
↪directory
```

Push the new project

Go in the newly generate folder containing your project and push it on gitlab :

```
/bin/bash init-gitlab.sh <unix name of the Wombat project>
git push --set-upstream origin master
```

/!\ After having push your project, delete the project folder and initialize a new one directly from gitlab before starting your work.

CHAPTER 21

Database Scripts (from 0.14)

Model - Database comparisons

Note: `*SqlUpdateScriptMain` is named at project generation's time: for example ProjectSqlUpdateScriptMain.

The script *<Project>SqlUpdateScriptMain.java* can generates the differences between your java model and your database and write the result as an update sql script (update of your database). It can also generate an sql script for the creation of your whole database.

To launch this script, make sure you are in the basic-application/basic-application-init directory, then execute

```
mvn exec:java -Dexec.mainClass="fr.openwide.core.basicapp.init.
↪BasicApplicationSqlUpdateScriptMain" -Dexec.args="arg0 arg1"
```

You have to provide two arguments :

- arg0 is the mode of the script, you have the choice between **create** for generating your database's creation script, and **update** for generating the update of your database.
- arg1 is the file which will contain the result script.

BasicApplicationSqlUpdateScriptMain not available

If you project does not supply `*SqlUpdateScriptMain`, you can copy and rename the file *BasicApplication-SqlUpdateScriptMain.java* in you project. Inside this file, you just have to replace **BasicApplication** by the name of your project in the line :

```
context = new AnnotationConfigApplicationContext(BasicApplicationInitConfig.class);
```

CHAPTER 22

Install an Oomph project

In the git folder, clone the owsi-tools-oomph repository

```
git clone git@github.com:openwide-java/owsi-oomph-project.git
```

Open Eclipse, use a new workspace. Select File->Import->Oomph->Projects into workspace. Click on the "+" button :

- Choose the catalog Eclipse Projects
- Browse file : choose the file git/owsi-oomph-project/fr.openwide.core.eclipse.project.setup (choose the other file if your project is not in the github group openwide-java or in the openwide gitlab)
- Check the box corresponding to the file you just add, and click next

In the window with required variables :

- Nom du clone git : the name of the file which will contain the git clone on your computer
- Nom du projet gitlab : the name of the project as it appears in gitlab or github (correspond to <project> in the url git.projects.openwide.fr:open-wide/<project>.git)
- Branche : the branch which will be checkout
- Répertoire du tomcat : the path to your tomcat folder on your computer
- Nom du projet maven : the artifactId in your pom.xml
- Nom de la webapp : the name of the webapp which will be generated
- Choix du dépôt : if your application is host in github choose Dépot Github, otherwise if your application is host on gitlab choose Dépôt Gitlab. The default value is the gitlab repository.
- If you use the other file you will have an other variable to fill named "Nom du groupe/compte. Enter here the name of the github group or account which host the repository."

Click next, then finish. The installation may take a few minutes, and Eclipse might have to restart. In this case, the installation will go back to where it stopped automatically. It will clone the application, install and set a Tomcat server.

Warning: If you perform some modifications that you have to push on the repository, don't push the file eclipse/tomcat7.launch if you didn't modify it manually.

CHAPTER 23

Use Oomph with an existing project

Files required by Oomph are already in the basic-application repository :

The file /eclipse/tomcat7.launch contains default arguments that will be passed to your Tomcat server. You will have to modify these arguments/add new arguments here to make your Tomcat server suit your project.

```
<stringAttribute key="org.eclipse.jdt.launching.VM_ARGUMENTS" value="-Xmx1g -
↪XX:MaxPermSize=256m "/>
```

The folder /eclipse/tomcat-main contains files that are necessary to Tomcat's installation and proper working. For a new project, you have to do a modification : in one of the last lines of the file /eclipse/tomcat-main/server.xml you need to put the name of your webapp instead of the two mentions to "basicapplication-webapp".

```
<Context docBase="basicapplication-webapp" path="/" reloadable="false" source="org.
↪eclipse.jst.jee.server:basicapplication-webapp"/></Host>
```

CHAPTER 24

Use data upgrades with Flyway

OWSI-Core and the basic application are programmed to be able to use Flyway for handling your data upgrades, here's how you can do it.

Activate Flyway's Spring profile

First of all, you need to activate Flyway's Spring profile. To do so, just modify this line to add flyway in the file **development.properties** :

```
maven.spring.profiles.active=flyway
```

With this simple modification, you have enabled the creation of the Flyway bean in the application and you can now start to use it.

Note: If you want to disable flyway from your application, just remove the word *flyway* from the line *maven.spring.profiles.active* in the file **development.properties**.

Create a Flyway data upgrade

The first thing to do is to is to add flyway's variables in the file **development.properties** :

```
maven.flyway.locations=fr.openwide.core.basicapp.core.config.migration.common,db/
↪migration/
maven.flyway.table=flyway_schema_version
```

You can write your data upgrades either in SQL or Java. Here we have chosen to put our upgrades .sql in the folder *src/main/resources/db/migration/* and our upgrades .java in the package *fr.openwide.core.basicapp.core.config.migration.common*. If you want to specify multiple locations, you have to separate them with a single comma.

Now you can create the data upgrades which will be applied by Flyway. If you want to be able to relaunch manually the upgrade in case it fails, you have to use the Java formatted upgrades.

Note: Flyway works with a database versioning system. The versions are based on the names of the data upgrades so be careful how you name them. The name must respect the pattern *Vversion_you_want__NameOfDataUpgrade*. For example *V1_0__ImportTable.sql* is a valid name. SQL and Java upgrades follow the same naming pattern.

Create an SQL formatted data upgrade

If you want to write a data upgrade in SQL, just write your script and place your SQL file in the folder or package you have specified earlier.

Create a Java formatted data upgrade

If you want to write a date upgrade in Java, you have to follow a particular workflow. In fact, it is not the Flyway upgrade which will contain the operations on your data/database, you will have to create an OWSI-Core data upgrade after the Flyway one.

Your Flyway data upgrade will only declare that the data upgrade exists and that the application needs to launch it. To do so, copy the existing Flyway data upgrade *V1_2__ImportExcel.java*, give it the name you want and change in the class the value of the **DATA_UPGRADE_NAME** variable :

```
private String DATA_UPGRADE_NAME = "ImportExcel";
```

Create an OWSI-Core data upgrade

If you have wrote Java formatted data upgrades, you need to create an OWSI-Core data upgrade for each one of these which is named after the **DATA_UPGRADE_NAME** value you specified earlier. For example, if the value entered in the Flyway Java data upgrade is *ImportTable*, you have to name your OWSI-Core data upgrade *ImportTable.java*.

An OWSI-Core data upgrade is a java class which implements *IDataUpgrade* and override its methods. Write all your operations in the function **perform**().

CHAPTER 25

Create, initialize and launch a project - Workflow

In this page, we will follow the complete workflow to properly create a project, starting from nothing to finally be able to run the project on a server. In the following steps, we will call the project **hello-world**.

Clone the owsi-core-parent repository

First of all, clone the owsi-core-parent project :

```
git clone git@github.com:openwide-java/owsi-core-parent.git
```

Generate the new project and push it on gitlab

You can find a more detailed documentation of this part here.

In order to generate the project, we need to build the archetype :

```
cd ~/git/owsi-core-parent/basic-application
./build-and-push-archetype.sh ../basic-application/ local
```

After that, we place ourselves in /tmp and we generate the project :

```
cd /tmp
mvn archetype:generate -DarchetypeVersion=X.X -DarchetypeCatalog=local -
↪DartifactId=hello-world -DgroupId=fr.hello.world -Dversion=0.1-SNAPSHOT -
↪Dpackage=fr.hello.world -DarchetypeApplicationNamePrefix="HelloWorld" -
↪DarchetypeSpringAnnotationValuePrefix="helloWorld" -DarchetypeFullApplicationName=
↪"Customer - Hello World" -DarchetypeDatabasePrefix=hello_world -
↪DarchetypeDataDirectory=hello-world
```

The script asks what archetype we want to use, we choose the number corresponding to local, and validate the different values we entered previously.

We go to the newly generated project folder and make an adjustment : we add a line with the version of the owsi-core in the file *hello-world/pom.xml* between the markers *properties*, just under the line for the tomcat-jdbc.version :

```
<properties>
            <!-- il est préférable de bien surcharger la version du pool jdbc␣
↪Tomcat au niveau du projet en fonction de la version de Tomcat -->
            <tomcat-jdbc.version>${owsi-core.tomcat-jdbc.version}</tomcat-jdbc.
↪version>
            <owsi-core.version>0.14-SNAPSHOT</owsi-core.version>
      </properties>
```

Note: Note that here the version is the 0.14-SNAPSHOT because it is the latest version at the time.

After that, we push the project on gitlab by executing the script located in the project folder :

```
/bin/bash init-gitlab.sh hello-world
git push --set-upstream origin master
```

After pushing the project on gitlab, we have to delete the created folder and start working with a fresh one.

```
cd ..
rm -rf hello-world/*
```

We will make a new clone of the project using Oomph in the next step.

Create a fresh clone and a properly configured workspace with Oomph

You can find a more detailed documentation of this part in the dedicated Oomph page.

Open an Eclipse Neon and select a new and clean workspace. After that, we follow the Oomph page documentation until we come to the window with multiple variables to fill. We fill the window as follow :

- Nom du clone git : hello-world
- Choix du dépôt : Dépôt Gitlab
- Branche : master
- Répertoire du tomcat : ${user.home}/Documents/apps/apache-tomcat-7.0.53
- Nom du projet maven : ${gitlab.project.name}
- Nom de la webapp : ${gitlab.project.name}-webapp
- Nom du projet gitlab : hello-world

From here, we have a new project successfully created and pushed online, and a properly configure workspace. The only thing left is the database.

Create and initialize the database

You can find a more detailed documentation of this part in the *prerequisite* part of the Project installation page.

In this part, we will create the database with the proper user and schema, and we will fill it with a script. Before performing the following commands, make sure you have PostgreSQL installed.

To create the database, we execute some commands directly in a terminal:

```
createuser -U postgres -P hello_world
createdb -U postgres -O hello_world hello_world
psql -U postgres hello_world
#Here you are connected to the database as the user postgres
DROP SCHEMA public;
\q
psql -U hello_world hello_world
#Here you are connected to the database as the user hello_world
CREATE SCHEMA hello_world;
```

Note: Use the name of the project for the password (here: hello_world)

After that we have to enable an option which will allow our the project to create new entities in the database. To do so, in the file *hello-world-core/src/main/filters/development.properties* we have to switch the line **maven.hibernate.hbm2ddl.auto=none** to :

```
maven.hibernate.hbm2ddl.auto=update
```

To make sure the new property is taken into account, we refresh the project (in Eclipse : menu Project -> Clean...).

Finally, we fill our database with the script HelloWorldInitFromExcelMain.java especially written for this. We just right click on it in Eclipse and Run as Java Application.

Launch the project

Now we have all the tools properly configurated and ready to run our project. To do that, we just start the server tomcat7 in Eclipse (if you don't have the server view : Window -> Show view -> Other -> Server/Servers). To access to our project, we can go to http://localhost:8080/ . To access the console, the address is http://localhost:8080/console/ .

Note: Until you change it, the login/password for the project and the project's console is admin/admin.