Squill metadata generator documentation

Squill metadata can be generated using an Ant target. First you should define the task:

```
<taskdef
   name="squill-gen"
   classpathref="classpath"
   classname="squill.mgen.SquillMappingsTask"/>
```

Note that Squill, ASM and Apache Velocity and your database vendor's JDBC driver should be in the classpath. Make sure there are no same libraries with different version number on the classpath, because e.g. ASM 3 is not backward compatible.

Then use it:

```
<squill-gen
   driver="org.postgresql.Driver"
   url="jdbc:postgresql://localhost:5432"
   user="username"
   password="password"
   schema="myschema"
   packageName="com.example.model"
   outputPath="src"
   namingstrategy="CamelCaseNaming"
   namingoverride="namingoverride.properties"
   excludeRules="excludeRules.properties">

  <fileset dir="war/WEB-INF/classes/com/example/model/" includes="*.class"/>

</squill-gen>
```

Your model classes should be compiled before running the task, so it's a nice idea for the target to depend on the compilation target.

Here is the description of all possible arguments of the Ant task:

driver
JDBC Driver, e.g
Oracle: oracle.jdbc.OracleDriver (oracle.jdbc.driver.OracleDriver is the deprecated one)
HSQLDB: org.hsqldb.jdbcDriver
Postgres: org.postgresql.Driver
MySQL: com.mysql.jdbc.Driver

url
Database JDBC URL, e.g. jdbc:postgresql://localhost:5432

user
The one you use to log in to your database.

password
The one you use to log in to your database.

schema
Owner of the tables needed to examined by mgen. Note that schema is case-sensitive.

packageName
The package name used for generated classes.

outputPath
The directory that will be used to save the generated classes (the package directories are included in the resulting path and will be created if necessary).

namingstrategy
The strategy for translating database table, view or column names into names of corresponding Squill mappings. Choose amongst 4 different implementations or provide your own.

SameNaming - Naming strategy that does not change table / column names.

CamelCaseNaming - Naming strategy where words are separated with underscores in database and upper case letters in Java mappings. This is the default naming strategy in case you omit the given argument.

LowerCaseNaming - Naming strategy where Java type names are converted into lowercase and names of database tables and columns are held out in upper case.

PluralCamelCaseNaming - Naming strategy where Java type names are in singular, but database table names are in plural. Words are separated with underscores in database and upper case letters in Java mappings.

Own implementation - Class should implement the squill.mgen.naming.NamingStrategy interface.
Note that fully qualified class name should be provided if you choose own implementation.

namingoverride
Path to a properties file which contains an optional mapping from the table/column names to class/filed names. If this argument is defined then the PropertyOverrideNamingStrategy will be used. This strategy looks in the property file first and in case nothing was found it delegates the naming to the strategy you defined using namingstrategy argument. Note that you shouldn't define the PropertyOverrideNamingStrategy explicitly. Entries in the property file should look like this:

```
MY_WEIRD_TABLE=MyObject # different name for a table/data class
MY_WEIRD_TABLE.MY_WEIRD_FIELD=myField # Different name for a column/field
```

There should definitely be an entry in the namingoverride property file in case:

- Database table has a column of the same name
- Table or column name have special characters that are not allowed in class and/or field names

templateFile

Path to the Velocity template file used for generating classes. In general, you will use the template, provided by Squill.

excludeRules
Path to a properties file which contains exclude rules to exclude some tables/columns from generating mappings to them. Entries in the property file should look like this:

```
MY_TABLE # exclude table
.MY_COLUMN # exclude column from all possible tables
MY_OTHER_TABLE.MY_OTHER_COLUMN # exclude column from a specific table
```

Table/column names are compared ignoring the case. Usage of exclude rules will also exclude all kinds of foreign keys that depend on the excluded tables/columns.

Inside the task you define a fileset, where all model classes that should correspond to the database tables can be found. In case some of your model class' superclass is not in this fileset, it should be on classpath, see taskdef.