MOLGENIS 3

application developers guide

Title: MOLGENIS 3 application developers guide

Author: Morris Swertz, please send comments to *m.a.swertz@rug.nl*

Updated: May, 2009 for NBIC BioWise course on Information Management

Table of contents

1. Introduction

This document is a hands-on guide for MOLGENIS application development.

1.1. Why MOLGENIS?

Relational (SQL) databases[1] are the workhorses of most structured data management around the world. However it still takes surprisingly amounts of effort to design and implement a full database application. The MOLGENIS[2] platform allows you to automatically generate rich database software to your specifications, including web user interfaces to manage and query your data, various database back ends to store your data, and programmatic interfaces to the R language and web services.

You tell MOLGENIS what to generate using an data model and user interface model described in XML; at the push of a button MOLGENIS translates this model into SQL, Java and R program files. Also documentation is generated. While the standard generated MOLGENIS is sufficient for most data management needs, MOLGENIS also allows you to plug in handwritten software components that build on the auto-generated software platform.

1.2. What will you achieve with this guide?

This guide can be used in a walk-through fashion to learn how:

- To model rich data models using MOLGENIS data definition language
- To generate your own customized MOLGENIS databases from scratch
- To generate a MOLGENIS to access existing databases
- To enhance the standard generated MOLGENIS with your own UI plug-ins
- And how to automatically manage and retrieve your data using the Java, R and SOAP interfaces

This guide assumes minimal Eclipse, Java and database experience; if not we suggest to team up with someone who does.

---

[1] For example MySQL http://www.mysql.org, Postgresql http://www.postgresql.org, Microsoft SQL server http://www.microsoft.com/sqlserver/, Oracle http://www.oracle.com/database

[2] Swertz & Jansen:Beyond standardization: dynamic software infrastructures for systems biology. Nature Reviews Genetics 8: 235-243; Swertz et al: Molecular Genetics Information System (MOLGENIS) alternatives in developing local experimental genomics databases. Bioinformatics 20: 2075-83; http://www.molgenis.org

2. Getting started

This section explains what you need to get started from a clean OS install up to the point where you can generate, run and browse the MOLGENIS example that is shipped with the MOLGENIS distribution (in the next chapter).

2.1. Install the necessary software (only once)

MOLGENIS is known to run happily at Windows, Linux and Mac. We here assume the procedure for the windows installation; for other operating system distributions we refer to respective documentations. To get started, download and install the most recent:

1. Java 6 JDK http://java.sun.com/javase/downloads/

   Just install the most recent standard JDK with standard options.

2. Tomcat >=5 web server http://tomcat.apache.org/

   Rember the root password you are asked.

   Don't run Tomcat as a service; we will start and stop it ourself.

3. Mysql >=5.1 database http://dev.mysql.com/downloads/mysql/5.1.html

   During installation of the windows version you need to tick 'innodb'.

   Remember your root password.

4. Eclipse Integrated Development Environment http://www.eclipse.org/downloads/

   You need to install the J2EE version (which allows you to start/stop tomcat)

5. **(Optional)** MOLGENIS autogenerates documentation for your application. This requires the GraphViz package from *http://www.graphviz.org/Download.php*

   Make sure your PATH is set to run 'dot.exe' from the commandline. In windows this is set in the control panel -> System -> Environment variables.

2.2. Setup a MySQL database (once per MOLGENIS instance)

MOLGENIS typically uses MySQL as data storage back-end although also PostgreSQL may be used. Here the procedure for MySQL is described to create a database and give MOLGENIS permission to populate it.

6. Open mysql.exe as root user.

   Under windows in "C:\Program Files\MySQL Server 5.1\bin\mysql.exe - u root -p

   Type your mysql root password remembered from step 3!

7. Create a database 'molgenis' and provide rights to the molgenis generator to change it using password 'molgenis'. Type:

```
create database molgenis;
grant all privileges on molgenis.* to molgenis@localhost identified by 'molgenis';
flush privileges;
```

2.3. Open the MOLGENIS workspace using Eclipse

Most application developers use Eclipse to develop MOLGENIS applications. Therefore a ready made Eclipse project is the standard MOLGENIS distribution.

8. Download the latest 'molgenis_workspace.zip' from http://molgenis.sourceforge.net

   Just download and unzip. A directory 'molgenisXX_workspace' is created.

3. Generating the distro MOLGENIS application

In this chapter you will learn to generate a MOLGENIS database application using Eclipse. First you will check the MOLGENIS distribution workspace, then you will get acquainted with the generator tools and finally you will generate, compile and run your first MOLGENIS application.

Before you start *– ensure that you have at least from the previous section:*

Java Development Kit 5 or higher

A Tomcat server instance

A MySQL or Postgresql database

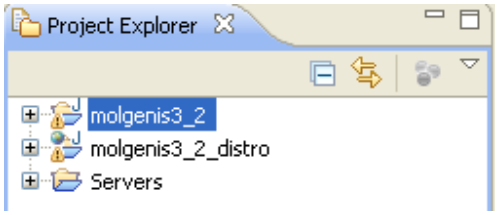
Eclipse IDE including J2EE extension

MOLGENIS workspace

3.1. Checking the MOLGENIS workspace

In this section you will veryify the MOLGENIS workspace:

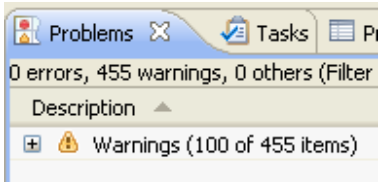
1. Start Eclipse and browse to MOLGENIS `molgenis3X_workspace` you downloaded and unzipped before. Open. You will see in the left pane:



   The `molgenis3X_workspace` contains two projects. (a) `molgenis3_x` contains all code of the MOLGENIS framework (b) the project `molgenis3_x_distro` contains an application template which you will use to create the new MOLGENIS application with.

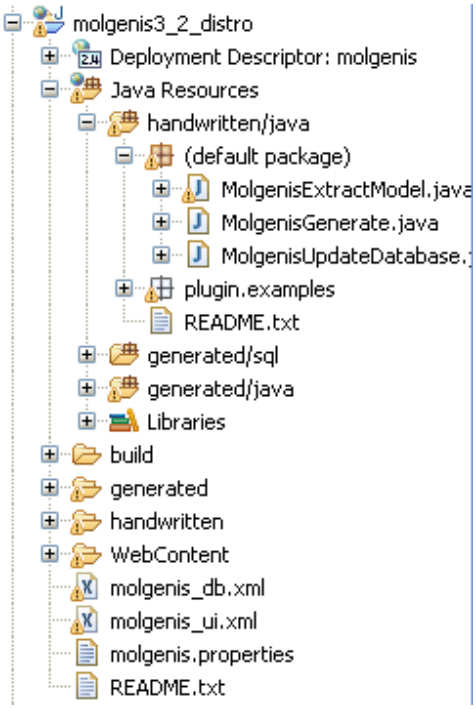
2. Check whether MOLGENIS is properly linked with Java: Right-click on the `molgenis3_x` project and choose [refresh]. Then right-click on the `molgenis3_x_distro` project and choose [refresh]. This will force Eclipse to rebuild the code for your platform.

If everything is well then there will be no errors listed in the problems pane on the bottom of your screen.



3. (Skip step unless library issues). To link missing system libraries choose [Window] -> [Preferences]. Then browse to Java -> Installed JREs. Finally add your JDK and tick to be used as default.

4. Open the `molgenis3_x_distro` project and explore. This unveils the basic MOLGENIS application structure:

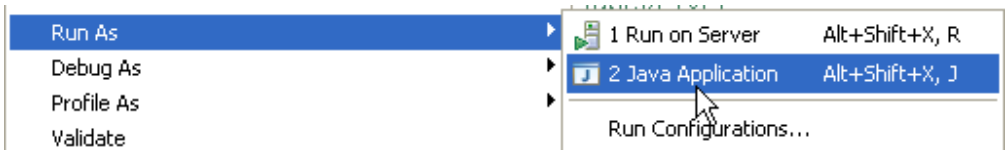


   - `molgenis.properties` contains the settings of your applications.
   - The *.xml files contain your MOLGENIS models
   - `WebContent` contains web resources (images, javascript, etc)
   - `generated/java` and `generated/sql` will contain the output of the MOLGENIS generator.
   - `handwritten/java` contains the MOLGENIS generator tools and any programs you will add.

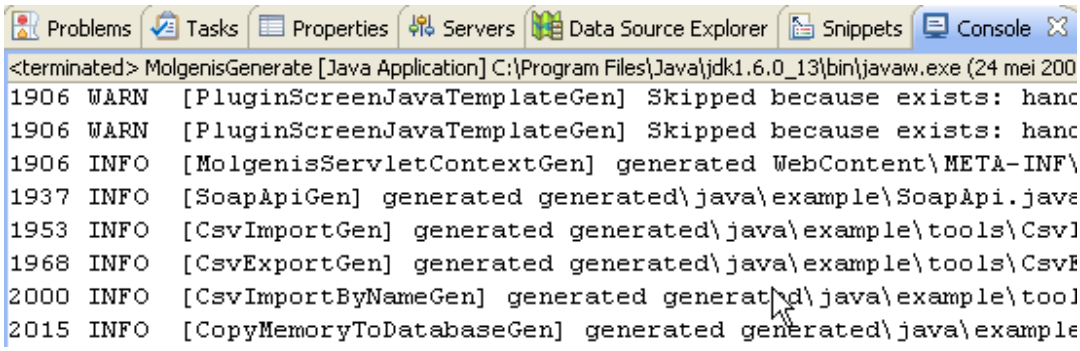
3.2. Running the MOLGENIS generator

In this section you go through the basic procedure of generation, compilation and running of a MOLGENIS application.

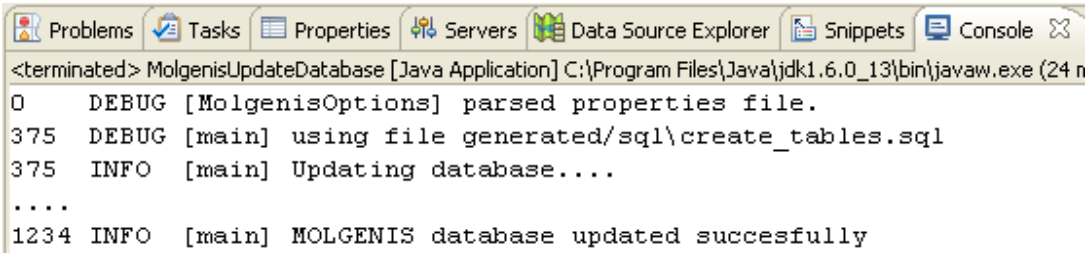
1. Run the MOLGENIS generator: **right-click** on the `MolgenisGenerate.java` program and choose 'Run as….run as Java Application:



   The MOLGENIS generator parses the *.xml files to read in the model and subsequently translates the model in many Java, SQL and R files that end up in the folder generated/java and generated/sql. All this will be printed to the console. Scroll through the console to get a feel for what has just happened:



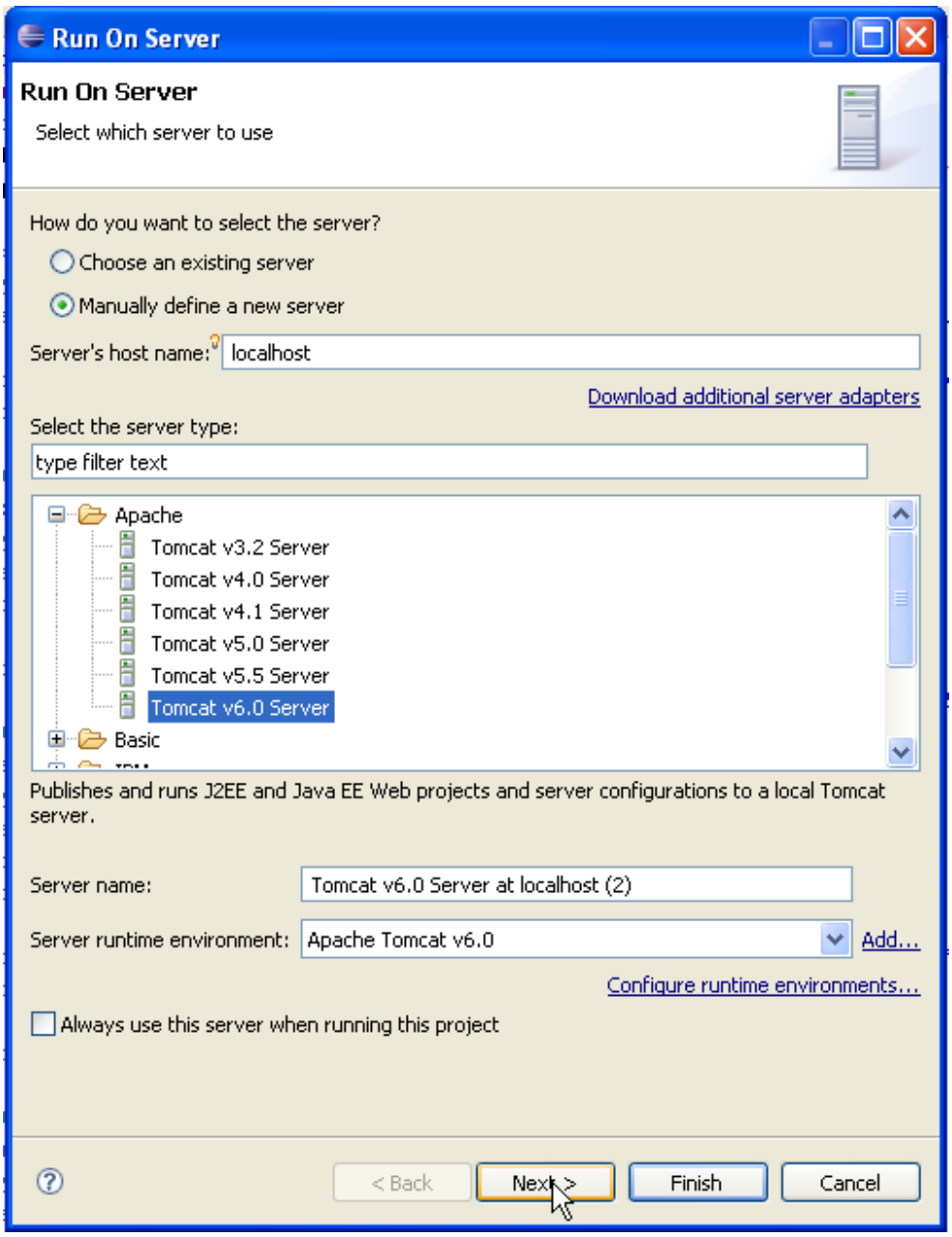
2. Run the MOLGENIS database updater: **right-click** the `MolgenisUpdateDatabase.java` program and choose **Run as**….**Java Application**. This will load `generated/java/create_tables.sql` into mysql database based on the settings in `molgenis.properties`.



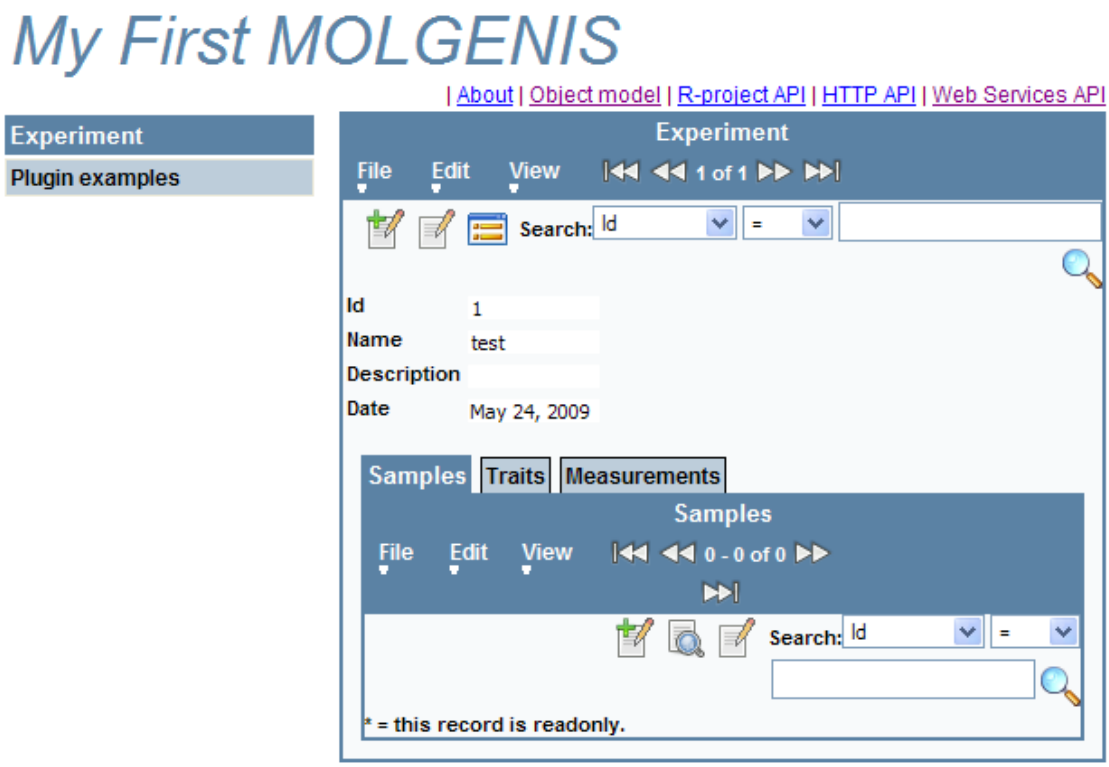
3. Compile the generated application: right-click on the `molgenis3_x_distro` project and choose [refresh]. This will force Eclipse to discover and compile the code you just generated.

4. Run the application: right-click on the `molgenis3_x_distro` project and choose [Run as…] -> [Run on server].

   Notice: on the first run this will in a dialog asking you to define a new server. Choose Tomcat of your version and choose [next] to point it to the right directory.



5. Finally browse the generated result and test the result by adding data:



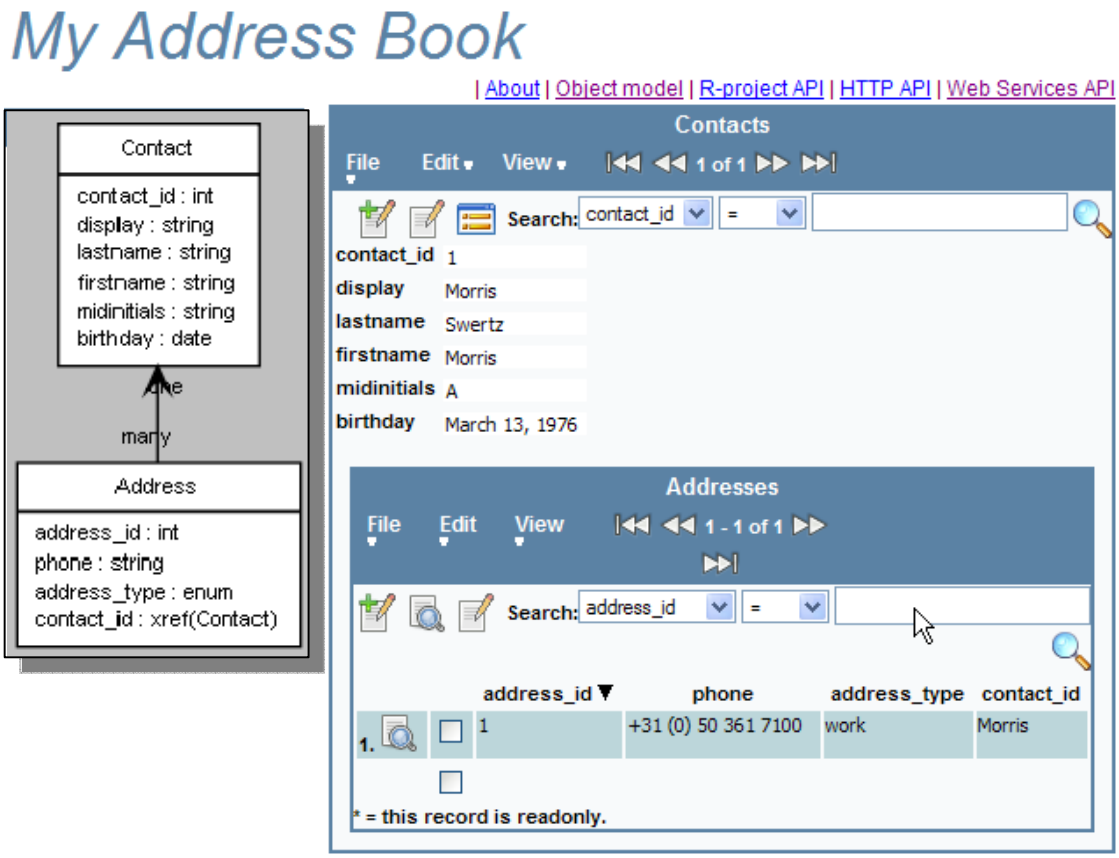
See that you get a rich set of user tools ‘for free’:

- change the view from listview to recordview to editview
- add/update/remove one or, in listview, multiple records
- search for particular elemements
- view the object model under the ‘object model link’
- view the R and SOAP interfaces (to be discussed later)

4. Creating a MOLGENIS from scratch

In this chapter you will create a new MOLGENIS application from scratch using Eclipse. You will use the example of a simple Address Book application: a simple database having “Contacts” entities that each can have zero or more “Addresses”.

The data model and user interface produced will look like:



Before you start *– ensure that you have at least from the previous sections:*

Learned how to compile, generate and run a MOLGENIS application

Learned how to setup a MySQL database with proper privileges

We also assume basic knowledge of XML and relational database modeling.

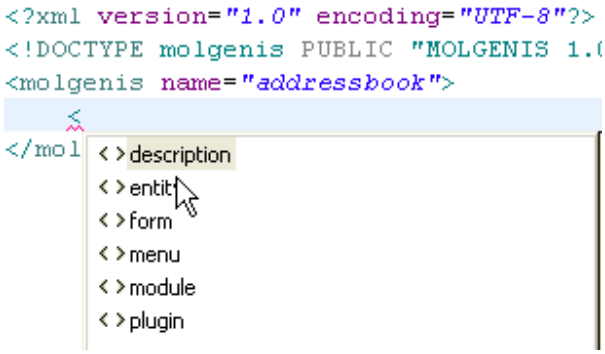
4.1. Create the data model

1. Create a new xml file '`addressbook_db.xml`' in the root of you application. This file will contain the MOLGENIS model. Use the following template:

```
<?xml version="1.0" encoding="UTF-8"?>
<!DOCTYPE molgenis PUBLIC "MOLGENIS 1.0"
"http://molgenis.sourceforge.net/dtd/molgenis_v_1_0.dtd">
<molgenis name="addressbook">

</molgenis>
```

Now we will step-by-step build the XML model. But first two tips:

- If you type a '<' and wait a second then a menu will pop-up with valid elements to choose from. This is because the <!DOCTYPE element tells Eclipse what is allowed and what not in a MOLGENIS file.
- If you use the key combination 'ctrl+f' then Eclipse will automatically layout the XML file.



For a complete description of all XML elements allowed we refer to section 10.

2. Add a "Contact" data type. Data types are called 'entity' in MOLGENIS. Type:

```
<molgenis name="addressbook">
   <entity name="Contact">

   </entity>
</molgenis>
```

3. Add properties to the "Contact" data type. These are called 'field' in MOLGENIS. Add within the entity element:

   - A unique, automatic numeric identifier field that MOLGENIS will use to refer contacts.

```
<field name="contact_id" type="autoid"/>
```

   - A required and unique string field 'displayname' to have a unique label to find Contacts by. Note that if you provide no 'type' then type="string" is implied.

```
<field name="displayname" unique="true"/>
```

- Optional (‘nillable’) string fields for firstname, lastname and midinitials:

```
<field name="firstname" nillable="true"/>
<field name="midinitials" nillable="true"/>
<field name="lastname" nillable="true"/>
```

- An optional field of type date for birthday:

```
<field name="birthday" type="date" nillable="true"/>
```

4. Similarly create a “Address” data type (to keep it simple we only do phone numbers):

```
<entity name="Address">
```

- A unique, automatic numeric identifier field that MOLGENIS will use to refer addresses.

```
<field name="address_id" type="autoid"/>
```

- Add the phone number field:

```
<field name="phone"/>
```

- Add a field where the user can choose from an enumeration of options whether the address is ‘home’,’work’,or ‘mobile’:

```
<field name="address_type"
       type="enum" enum_options="[home,work,mobile]"/>
```

5. Finally we need to link the Addresses to each Contact. In relational databases this is done using the mechanism of a ‘foreign key’. In MOLGENIS fkeys can be defined as follows:

```
 <field name="contact" type="xref"
        xref_field="Contact.contact_id"
        xref_label="displayname"/>
</entity>
```

Note that the xref_field refers to unique field ‘contact_id’ in entity ‘Contact’. The xref_label attribute indicates that we want to view references to Contact by ‘displayname’ name instead of the numeric id.

Use of a meaningless numeric ‘id’ next to a meaningful ‘label’ enables us to change the ‘displayname’ name on Contact without the problem of also having to change this on each Address that refers to this contact.

4.2. Create the user interface model

6. Create a new xml file 'addressbook_ui.xml' in the root of you application by copy-pasting the db.xml file. This file will contain the MOLGENIS user interface model.

7. We want to show the list of Addresses nested per Contact; for MOLGENIS you simply express this by nesting form elements. If suitable 'xrefs' exists these will be used to tie container and nested form together:

```
<?xml version="1.0" encoding="UTF-8"?>
<!DOCTYPE molgenis PUBLIC "MOLGENIS 1.0"
"http://molgenis.sourceforge.net/dtd/molgenis_v_1_0.dtd">
<molgenis name="addressbook">
   <form entity="Contact" name="Contacts">
         <form entity="Address" name="Addresses" view="list"/>
   </form>
</molgenis>
```

4.3. Generate the new MOLGENIS application

8. Open the molgenis.properties file. The model_database and model_userinterface options point to the XML files where a data model and user interface is specified. Change to point to our newly created addressbook_db.xml and addressbook_ui.xml files

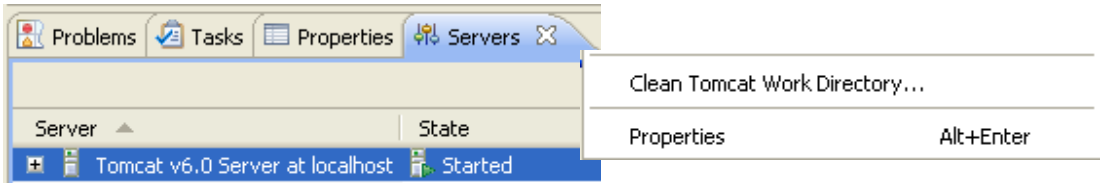
```
#####################################################################
# 1. FILES DESCRIBING YOUR DESIGN USING MOLGENIS XML LANGUAGE
# can be multiple files ',' separated
#####################################################################

# xml file with data model in terms of 'entity' descriptions
model_database = addressbook_db.xml

# xml file with user screen descriptions in terms of 'form',
'menu',..
model_userinterface = addressbook_ui.xml
```

9. Create a new mysql database instance named 'addressbook' as you learned in section 2.2. Then edit molgenis.properties file to connect to this new database instance (we assume you used the molgenis username/password again:

```
#####################################################################
# 2. DATABASE SETTINGS
#####################################################################

# MySQL:
# jdbc compatible connection parameters to a database (see doc
of database supplier)
db_driver = com.mysql.jdbc.Driver
db_user = molgenis
db_password = molgenis
```

```
db_uri= jdbc:mysql://localhost/addressbook
```

10. Finally, generate, compile, load database and run your application as you learned in section 3.2.



NOTICE: if you don't see your newly generated MOLGENIS but instead a previous iteration, go to the Servers screen and 'clean tomcat work directory': s user session was still in place.

4.4. Optional exercises

4.1 Expand the address book: add fields to register email address, physical addresses (organization, street, zip code, city, country) and for city and country add additional entities 'Country' and 'City' and link these to the Address using xref fields.

4.2 Improve the address book application by using advanced features such as 'hidden' (for automatic ids) and 'label' to create prettier field labels (for address_type). See reference section 10.

5. Creating a MOLGENIS for existing data

There is legacy of existing databases of which many provide SQL dumps/TEXT downloads or you may have a local database already of your own that you want to 'molgenize'. In this section you will learn how to generate a MOLGENIS for an existing database by automatically extracting the MOLGENIS data model from it. Second you will learn how to batch-load existing data sets from delimited text files.

Before you start *– ensure that you have at least from the previous sections:*

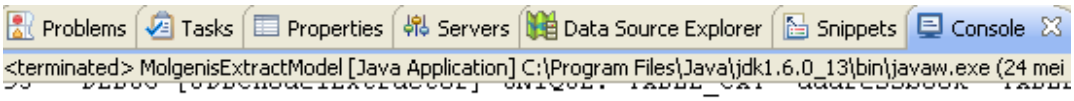
Learned how to compile, generate and run a MOLGENIS application

Learned how to edit molgenis.properties to connect to a particular database

Loaded the MOLGENIS distro database in section 3.

5.1. Generating a MOLGENIS for an existing database

1. Edit the database connection settings in the `molgenis.properties` file to refer to an existing database. See section 4.3, step 9. In this example it is assumed you connect to the Address Book database from the previous section. Alternatively you can download a database dump of your choice or connect to a remote database (assuming proper rights).
2. Run `handwritten/java/MolgenisExtractModel.java.` The extraction tool will connect to your selected database and extract a model from its table structure. An example output for the Address Book Application:




```
<molgenis name="addressbook">
    <entity name="address">
        <field name="address_id" type="autoid"/>
        <field name="phone"/>
        <field name="address_type"/>
        <field name="contact_id" type="xref" xref_field="contact.cont
    </entity>
    <entity name="contact">
        <field name="contact_id" type="autoid"/>
        <field name="display" unique="true"/>
        <field name="lastname"/>
        <field nillable="true" name="firstname"/>
        <field nillable="true" name="midinitials"/>
        <field nillable="true" name="birthday" type="date"/>
    </entity>
</molgenis>
```

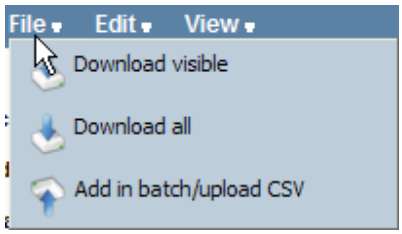
3. Copy-paste the parts of XML you want to use in your `*_db.xml` file. Then add a suitable `*_ui.xml`. Then adapt the `molgenis.properties` file accordingly and generate,compile,run to view the results (as learned in the previous section).

5.2. Batch loading of data

MOLGENIS also provides methods to quickly load large data sets from comma or tab separated data, both in the UI as well as in the Java API (as we will see in section 8). This works if:

- The column headers match the entity field definitions (order doesn't matter)
- A special case are 'xref' data that either can use the xref_field or the xref_label. For example in the case of address book 'contact_id' and 'contact_displayname'.

4. Load data for Contact. Within the MOLGENIS user interface for Contacts choose 'File' and then 'Add in batch'.



In the CSV data dialog paste the following[3]:

```
displayname,lastname,firstname,mid_initials,birthday
Prof.dr. R Bischoff,Bischoff,Rainer,,
Dr. R Breitling,Breitling,Rainer,,
Prof.dr. RC Jansen,Ritsert,Jansen,C,
```

5. Load data for Address. In the dialog for Example data for Address again choose ‘File’ and then ‘Add in batch’. Set the ‘address_type’ to ‘work’ as constant for all addresses to be loaded (otherwise you get an error!). Then paste in the CSV data dialog:

```
phone,contact_displayname
+31 (0)50 3633338,Prof.dr. R Bischoff
+31 (0)50-3638088,Dr. R Breitling
+31 (0)50-3638089,Prof.dr. RC Jansen
```

Notice the cross reference by ‘xref_label’ using ‘contact_displayname’!

5.3. Optional exercises

5.1 Connect MOLGENIS one of your own datasets or to one of the publicly available databases.

5.2 (For programmers) Write a simple script that given a set of csv files produces a MOLGENIS data model definition in XML.

[3] These example data was copied from http://www.nbic.nl/nbic/network/?city=Groningen

6. Creating a user interface plug-in

MOLGENIS plug-in mechanism allows you to add custom pieces of code to the generated user interface. Here you will create a simple 'Literature' plug-in for the Address Book application that shows this years list of publications from Pubmed for each contact.

Before you start *– ensure that you have at least from the previous sections:*

Learned how to compile, generate and run a MOLGENIS application

Completed the Address Book example as that is reused here

6.1. Adding the plug-in to the user interface

First we will add the plugin to the model:

1. Edit `addressbook_ui.xml` to add your plugin to the XML model; note how you can use menu to group the Address next to the new 'Publications' plugin.

```
<?xml version="1.0" encoding="UTF-8"?>
<molgenis name="addressbook">
    <form entity="Contact" name="Contacts">
        <menu name="ContactMenu">
            <form entity="Address" name="Addresses" view="list" />
            <plugin name="Publications"
                    type="plugin.pubmed.PubmedPlugin"/>
        </menu>
    </form>
</molgenis>
```

When you now run the generator an 'empty' screen plugin class will be generated inside the folder handwritten/java (unless the file allready consists). In this case a class file called `plugin.pubmed.PubmedPlugin`*. Next to that also a layout template is generated to layout the contents of your plugin in '* `plugin.pubmed.PubmedPlugin.ftl`*. This template is generated in the Freemarker template language, http://freemarker.sourceforge.net*

Second, add the 'logic' to the Java part of the plugin:

2. Open the `plugin.pubmed.PubmedPlugin.java` file from the handwritten/java folder. You see it has four methods including:

   - *'handleRequest' to process user requests (updates, interactive features)*
   - *'reload' to implement what needs to be done if the page is reloaded (data loading)*

3. Create the logic to produce a pubmed query string. Edit the ‘reload’ section as follows:

Above, create properties to store the lastname and current year to be filtered on

```
String lastname;
String year = "2009";
```

On reload take the lastname from the Contact currently shown in the parent from:

```
public void reload()
{
	//get the author name from parent from
	MenuScreen parentMenu = (MenuScreen)this.getParent();
	FormScreen<Contact> parentForm =
			(FormScreen)parentMenu.getParent();
	List<Contact> visibleContacts =
				parentForm.getRecords();

	this.lastname= visibleContacts.get(0).getLastname();
}
```

Add an additional method to produce the query string for pubmed.

```
public String getQueryString() {
	String url ="http://www.ncbi.nlm.nih.gov/pubmed?term="
			+lastname+"[au] ";
	if(year != null)
		url += year+"[Publication Date]";
	return url;

}
```

Finally we will edit the Freemarker template (ftl) in order add a window to the user interface layout in order to show oubmed results based on our query string:

4. Optional step: install the Freemarker plugin for Eclipse in order to make Eclipse show colorful code highlighting which greatly eases editing of the *.ftl file. Follow the instructions at http://freemarker.sourceforge.net/eclipse.html

5. Open the `plugin/pubmed/PubmedPlugin.ftl` file from the handwritten/java folder. You see it already is a complete web page including a form. Edit the section between ‘begin’ and ‘end’ to create a view to pubmed as ‘iframe’ and using the querystring defined in the java plugin. Note this java object is refered to as ‘screen’:

```
<#--begin your plugin-->

<iframe src="${screen.getQueryString()}"
	width="100%" height="800px">

<#--end of your plugin-->
```

6.2. Handling user requests

In this section we will enhance the plugin with some interactive element.

6. In the *.ftl file add inputs for the user to enter a ‘year’ and a ‘submit’ button to send this input to the plugins’ Java class. NOTICE: within MOLGENIS it has become practice to always use a field named ‘action’ to transfer the user intension. Hence the ‘onclick=”__action.value=’setYear’” mechanism.

```
<#--begin your plugin-->

<input type="submit" value="set year:"
onclick="__action.value='setYear'">
<input name="year" value="${screen.getYear()}">
<iframe src="${screen.getQueryString()}" width="100%" height="800px">


<#--end of your plugin-->
```

7. In the *.java class edit the handleRequest method to handle these new action input:

```
   public void handleRequest(Tuple request)
   {
         if("setYear".equals(request.getAction()))
         {
               if(request.getString("year") != null)
                     this.year = request.getString("year");
         }
   }
```

6.3. Exercises

6.1 Extend the pubmed plugin with (a) include use of firstname and initials to improve filtering and (2) create a maximization or popup button using javascript on the iframe

6.2 First walkthrough section 9.1 to learn using the Java Database interface. Then create a new plugin called ‘VCardAdd’ that enable you to add a new contact based on a vcard snippet. See http://en.wikipedia.org/wiki/VCard.

7. Creating a data validation plug-in

MOLGENIS allows you to add custom pieces of code to the database mapping components (that is the piece of software that talks between MOLGENIS and the SQL database back-end). This type of plug-in is called a 'decorator'. You will use this decorator mechanism to validate the input of email addresses and provide the user with an error message if needed.

Before you start *– ensure that you have at least from the previous sections:*

Learned how to compile, generate and run a MOLGENIS application

Completed the Address Book example as that is reused here

7.1. A data validator for email addresses

1. Extend the `addressbook_db.xml` model with an additional 'email' field on 'Address' and add the decorator. Then regenerate and the decorator class `handwritten/java/decorators.AddressDecorator.java` is auto-generated.

```
<entity name="Address" decorator="decorators.AddressDecorator">
      <field name="email"/>…

</entity>
```

2. Edit the decorator class `handwritten/java/decorators.AddressDecorator.java` for email address validation on 'add' and 'update'. We naively say that an email address should have a '@' in the middle and if not we throw a 'DatabaseException':

```
@Override
public int add(List<addressbook.data.types.Address> entities)
throws SQLException, DatabaseException
{
       for (addressbook.data.types.Address e : entities)
       {
            if(e.getEmail() != null &&
                  e.getEmail().split("@").length != 2)
                  throw new DatabaseException(
                        "Email address '"+
                        e.getEmail()+"' is invalid");
       }

      // else we call the generated 'add'
      int count = super.add(entities);

      return count;
}
```

3. Restart your application to test your decorator. You don't need to regenerate/recompile: just select the 'Servers' pane and righ-click -> clean tomcat work directory.

8. Intermezzo: learning SQL

MOLGENIS is built on top of SQL. For practical purposes it is good to learn SQL. Below there is a series of exercises to get started.

Before you start *– ensure that you have at least from the previous sections:*

Learned how to compile, generate and run a MOLGENIS application

8.1. SQL quick reference

8.1.1. General format

```
SELECT what_to_select
FROM which_table
WHERE conditions_to_satisfy;
```

8.1.2. Example Select:

```
SELECT * FROM pet;

SELECT name, birth FROM pet;

SELECT * FROM pet WHERE name = 'Bowser';

SELECT * FROM pet WHERE birth >= ‘1998-1-1’

SELECT * FROM pet WHERE species = 'dog' AND sex = 'f';
```

Note: 'yyyy-m-d’ is ANSI SQL format for date.

8.1.3. Table joins

```
SELECT pet.name,(YEAR(date)-YEAR(birth)) -
(RIGHT(date,5)<RIGHT(birth,5)) AS age, remark FROM pet, event WHERE
pet.name = event.name AND type = 'litter';

SELECT p1.name, p1.sex, p2.name, p2.sex, p1.species FROM pet AS p1,
pet AS p2 WHERE p1.species = p2.species AND p1.sex = 'f' AND p2.sex
= 'm';
```

There are other ways of combining tables involving so-called joins. Joins are more efficient, but also somewhat more complex.

```
SELECT * FROM table1 INNER JOIN table2 ON table1.id=table2.id;
```

8.1.4. MAX(), COUNT(), AVG()

There are so-called aggregate functions available. These include: max(*fieldname*) [get het maximum], count(*fieldname*) [count the number of occurrences] and avg(*fieldname*) [compute the

average]. They work based on a selection. The first example will give the number of rows in the table. The second example shows the use of the GROUP BY statement: it gets the number of pets per owner (effectively creating selections in selections, which can be counted).

```
SELECT COUNT(*) FROM pet;

SELECT owner, COUNT(*) FROM pet GROUP BY owner

SELECT article, MAX(price) AS price
FROM   shop
GROUP BY article
```

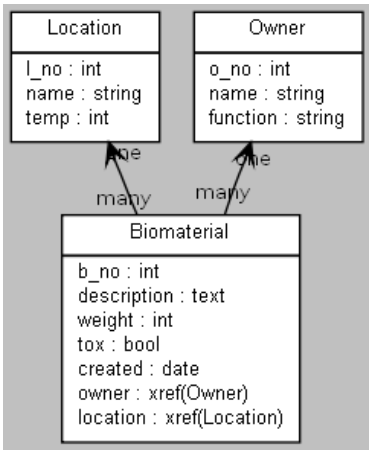
8.1.5. Distinct

SQL will return all records that match the criteria, often leading to multiple instances of the same rows. DISTINCT will filter these for you.

```
SELECT DISTINCT owner FROM pet;
```

8.2. Query exercises with expected output

An example data model ‘biomaterials’ is used, its structure summarized below.



The accompanying datasets used here are available for download as mysql ‘dump’:

1. Open mysql.exe command line interface as root user.

   Under windows in "C:\Program Files\MySQL Server 5.1\bin\mysql.exe - u root -p

   Type your mysql root password remembered from section 2.1, step 3.

2. Load and use the example data set using commands:

```
create database biomaterial;
use biomaterial;
\. <pathto>/biomaterial_dump.sql
```

Now you can start typing sql queries for the following cases:

3. Which biomaterials were created before 2003?

b_no	description	weight	tox	created	owner	location
676		1248	1	2002-12-14 00:00:00.0	10	26
687		714	1	2002-12-22 00:00:00.0	7	22
709		807	1	2002-12-28 00:00:00.0	11	29
725		1133	0	2002-12-10 00:00:00.0	7	1
774		1190	0	2002-12-14 00:00:00.0	1	5

4. Which students have been working with toxic biomaterials?

name
Sukke Lee
Boris Lee
Sarah Finn
Peter Finn

5. Which owners have biomaterials larger than 1000 grams in –80 C storage locations

name
Sukke Lee
Sarah Jones
Sukke Becker
Susan von Svede
Sukke von Svede
Boris Jones
Boris Smith
Peter Smith
Boris Finn

8.2.1. Using database functions

4. What was the total weight of the stored biomaterials created in 2003?

sum(weight)
56258

5. What was weight of the biggest sample stored in –80?

max(weight)
1471

8.2.2. Using group by/having

6. What is the total weight of biomaterials per owner?Give the average weight of toxic biomaterials per location?

name	sum(weight)
Boris Finn	3982
Boris Jones	5305
Boris Lee	5480
Boris Smith	4846
Fokke Becker	3680
Fokke Smith	5848
Fokke von Svede	3769
Martin Finn	1432
Martin Smith	1535
Martin von Svede	3295
Peter Finn	4517
Peter Smith	8091
Sarah Becker	4402
Sarah Finn	2331
Sarah Jones	10990
Sarah Smith	2837
Sukke Becker	10772
Sukke Lee	7989
Sukke von Svede	3385
Susan Becker	2490
Susan Jones	1525
Susan Lee	702
Susan Smith	6451
Susan von Svede	6239

9. Using the programmatic interfaces

The MOLGENIS framework provides programmatic access from Java, R and SOAP so you can use MOLGENIS as part of your statistical and/or work flows..

Before you start *– ensure that you have at least from the previous sections:*

Completed the Address Book example as that is reused here

9.1. Using the Java interface

Java access to MOLGENIS is via a 'Database' class. This class provides efficient methods to add, update, remove, find and count data inside your MOLGENIS either one-by-one or in an efficient batch modus:

1. Using the 'Database' interface: In the Address book application, create a new java class file in handwritten/java as follows and then Run as … Java application (watch the console)::

```
import java.io.FileNotFoundException;
import java.io.IOException;
import java.util.ArrayList;
import java.util.List;
import addressbook.data.JDBCDatabase;
import addressbook.data.types.Contact;

public class TestDatabaseApi {

      public static void main(String[] args) throws Exception {
```

Create a database from the properties file

```
            Database db =
            new addressbook.data.JDBCDatabase("molgenis.properties");
```

List all contacts:

```
            for(Contact c: db.find(Contact.class))
            {
                  System.out.println(c);
            }
```

Add one object to the database:

```
            Contact one = new Contact();
            one.setDisplayname("testone");
            db.add(one);
```

Add many objects to the database using the same method:

```
            System.out.println("add many contacts:");
            List<Contact> many = new ArrayList<Contact>();
            for(int i = 0; i < 10; i++)
            {
                  Contact c = new Contact();
                  c.setDisplayname("simulation"+i);
                  many.add(c);
            }
            db.add(many);
```

Observe that 'many' now also has the autoid column set:

```
            for(Contact c: many)
            {
                  System.out.println(c);
            }
```

Finally, remove all the data you just added:

```
            System.out.println("remove contacts:");
            db.remove(one);
            db.remove(many);
      }
}
```

2. Using the 'Query' interface. Extend your class above with the following to query. Notice you can just add more criteria using the '.' notation:

```
            Query<Address> q = db.query(Address.class);
            q.like("displayname","B");
            for(Address t: q.find())
            {
                  System.out.println(t);

            }
```

3. Using the 'sql' interface. Extend your class above to directly use sql:

```
            List<Tuple> result = db.sql(
                "select * from contact where displayname like '%B%'");
            for(Tuple t: result)
            {
                  System.out.println(t);

            }
```

9.2. Using the R interface

On the MOLGENIS user interface there is a link to the R interface. If you read this we is assume that you already installed R from the http://www.r-project.org web page.

5. Start the MOLGENIS server for your Address Book application and browse to the user interface.
6. Click the 'R-project API' pagelink on the top menu of your Address Book and view and copy the R script that is shown on this page. Notice how the R-api is simply an 'source.R' file on the MOLGENIS server.
7. Start the RGui and paste the R script you just copied. (Optional) you may get an error that the RCurl package is unavailable. In that case uncomment and paste the following lines to install this package

```
source("http://bioconductor.org/biocLite.R")
#biocLite("RCurl")
```

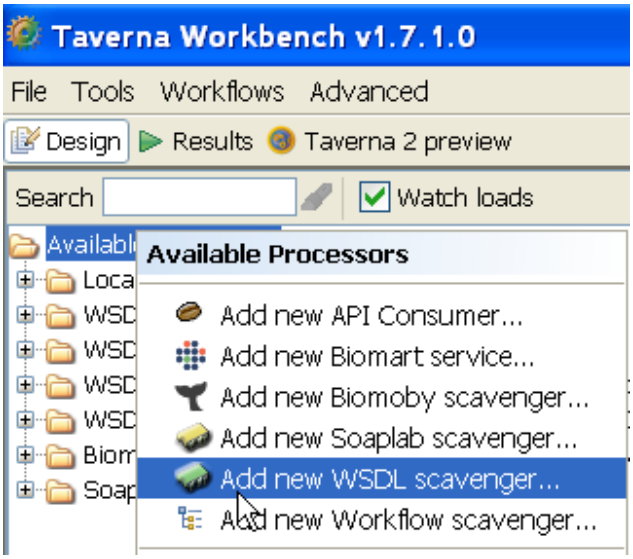
8. Finally reproduce the java script listed above. The first part is already given below:

```
find.contact()

add.contact(displayname="test")

find.contact()

remove.contact(find.contact(displayname="test"))

...
```

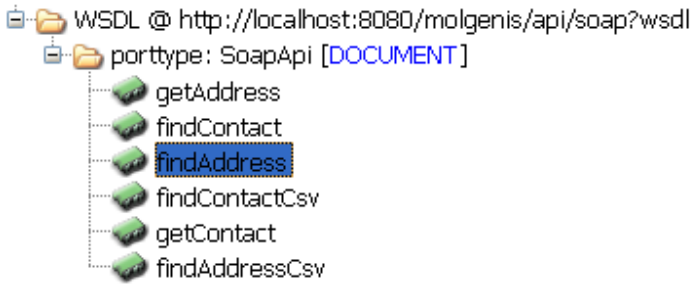
9.3. Using the SOAP interface in Taverna

On the MOLGENIS user interface there is a link to the SOAP interface for web services. Here we will show how to use web services to access the data from Taverna. If you read this we is assume that you already installed Taverna from the http://taverna.sourceforge.net .

9. Start the MOLGENIS server for your Address Book application and browse to the user interface.
10. Click the 'Web Services API' pagelink on the top menu of your Address Book and view and copy the hyperlink http://..... api/soap?wsdl. Note: a WSDL file is a standard description format of web services in the SOAP protocol.
11. Start Taverna and add the WSDL to Taverna using the WSDL scavenger as shown in the screenshot below.



12. Assuming you experienced Taverna user explore the usage of the services added. Otherwise we recommend to follow a Taverna tutorial first. Hint: use 'splitters' to ease the adding of parameters. Note: update and remove services have been disabled in the standard MOLGENIS distro.



9.4. (Experimental) generate a RDF/Sparql interface using the R2D server

13. Download and extract R2D from http://www4.wiwiss.fu-berlin.de/bizer/d2r-server/.
14. Generate a mapping file for your database schema. Change into the D2R Server directory and run:

```
generate-mapping -u molgenis -p molgenis -o mapping.n3
jdbc:mysql://localhost/addressbook
```

15. Start the server:

```
d2r-server mapping.n3
```

16. Test the Server: Open http://localhost:2020/ in a web browser.


Start Page | D2R Server
http://localhost:2020/
Google
D2R Server
Running at http://localhost:2020/
Home | conferences organizations papers persons rel_paper_topic topics
This is a database published with D2R Server. It can be accessed using
1. your plain old web browser
2. Semantic Web browsers
3. SPARQL clients.
1. HTML View
You can use the navigation links at the top of this page to explore the database.
2. RDF View
You can also explore this database with Semantic Web browsers like Tabulator. To start browsing, open this entry point URL in your Semantic Web browser:
http://localhost:2020/all
3. SPARQL Endpoint
SPARQL clients can query the database at this SPARQL endpoint:
http://localhost:2020/sparql
The database can also be explored using this AJAX-based SPARQL Explorer.
Generated by D2R Server

10. MOLGENIS XML language reference

MOLGENIS is configured using an XML based language as described above. This chapter details the semantics and use of each XML element and attribute that is defined in this XML language.

- for data model definition use `entity`, `field`, `unique`, and `module` elements.
- for user interface design use `form`, `menu`, and `plugin` elements.

Below the Document Type Definition (DTD) that summarizes MOLGENIS XML structure:

```
<?xml version="1.0" encoding="UTF-8"?>
<!ELEMENT molgenis (description? , (module | entity |form | menu | plugin)*)>
	<!ATTLIST molgenis name CDATA #REQUIRED>
	<!ATTLIST molgenis label CDATA #IMPLIED>
	<!ATTLIST molgenis version CDATA #IMPLIED>
<!ELEMENT description ANY>
<!ELEMENT module (description?, entity+)>
	<!ATTLIST module name CDATA #REQUIRED>
<!ELEMENT entity (description?,field*,unique*)>
	<!ATTLIST entity name CDATA #REQUIRED>
	<!ATTLIST entity abstract (true|false) #IMPLIED>
	<!ATTLIST entity implements CDATA #IMPLIED>
	<!ATTLIST entity extends CDATA #IMPLIED>
	<!ATTLIST entity decorator CDATA #IMPLIED>
	<!ATTLIST entity description CDATA #IMPLIED>
<!ELEMENT field EMPTY>
	<!ATTLIST field name CDATA #REQUIRED>
	<!ATTLIST field type CDATA #IMPLIED>
	<!ATTLIST field label CDATA #IMPLIED>
	<!ATTLIST field length CDATA #IMPLIED>
	<!ATTLIST field xref_field CDATA #IMPLIED>
	<!ATTLIST field xref_label CDATA #IMPLIED>
	<!ATTLIST field enum_options CDATA #IMPLIED>
	<!ATTLIST field default CDATA #IMPLIED>
	<!ATTLIST field auto (true|false) #IMPLIED>
	<!ATTLIST field nillable (true|false) #IMPLIED>
	<!ATTLIST field unique (true|false) #IMPLIED>
	<!ATTLIST field readonly (true|false) #IMPLIED>
	<!ATTLIST field hidden (true|false) #IMPLIED>
	<!ATTLIST field description CDATA #IMPLIED>
<!ELEMENT unique EMPTY>
	<!ATTLIST unique fields CDATA #REQUIRED>
	<!ATTLIST unique subclass (true|false) #IMPLIED>
	<!ATTLIST unique description CDATA #IMPLIED>
<!ELEMENT form (form*,menu*,plugin*)>
	<!ATTLIST form name CDATA #REQUIRED>
	<!ATTLIST form entity CDATA #REQUIRED>
	<!ATTLIST form label CDATA #IMPLIED>
	<!ATTLIST form view (list|record) #IMPLIED>
	<!ATTLIST form readonly (yes|no) #IMPLIED>
<!ELEMENT menu (form*,menu*,plugin*)>
	<!ATTLIST menu name CDATA #REQUIRED>
	<!ATTLIST menu label CDATA #IMPLIED>
<!ELEMENT plugin (form*,menu*,plugin*)>
	<!ATTLIST plugin name CDATA #REQUIRED>
	<!ATTLIST plugin type CDATA #REQUIRED>
	<!ATTLIST plugin label CDATA #IMPLIED>
```

10.1. <molgenis> application definition ELEMENT

The <molgenis> element is the root of each MOLGENIS application definition file and can contain data definition and/or user interface definition elements. The model can be split, i.e. there can be multiple MOLGENIS XML files for one application (see section on molgenis.properties file).

Example usage of the <molgenis> element:

```
<molgenis name="myfirstdb" label="My First Database">
  <menu name="mainmenu"> ...
  <entity name="firstentity"/>
  <module name="mymodule"/>
  ...
</molgenis>
```

Required attributes of the <molgenis> element:

- name="name": name of your MOLGENIS blueprint. This will be used by the generator to name the java packages that are generated..

Optional attributes of the <molgenis> element:

- label="your label": Label of your MOLGENIS system. This will be shown on the screen as well as heading in the generated documentation.
- Version="1.2.3": Version of your MOLGENIS system. It is recommended to use this to manage the versions of your application.

The <molgenis> element can contain other elements:

- Exactly one <menu> or one <form> as main screen.
- Zero or one <description> elements describing the application
- Many <entity> elements describing the data structure.
- Many <module> elements describing the data structure. These are containers to group <entity>.

10.2. <entity> data definition ELEMENT

The <entity> element defines the structure of one data entity and will result in a table in the database, and several Java classes.

Example usage of the <entity> element:

```
<entity name="unique_name">
```

```
    <description>This is my first entity.</description>
    <field name="name" type="string"/>
     <field name="investigation" type="string"/>
    <unique fields="name,investigation"/>
</entity>
```

Required attributes of the <entity> element:

- name="name": globally unique name for this entity (within this blueprint).

Optional attributes of the <entity> element:

- label="Nice screen name": an user-friendly alias to show as form header (default: copied from name).
- extends="other_entity": you can use inheritance to make your entity inherit the fields of its 'superclass' using extends.
- abstract="true": you define what programmers call 'interfaces'. This are abstract objects that you can use as 'contract' for other entities to 'implement'..
- implements="abstract_entity": you can use inheritance to make your entity inherit the fields of its 'interface' using implements and refering to 'abstract' entities. The implemented fields are copied to this entity.
- decorator="package.class": you can add custom code to change the way entities are added, updated and removed. See the section on how to write a MappingDecorator plugin.

The <entity> element can contain other elements:

- Zero or one <description> to describe this entity; a description can contain xhtml.
- zero or more <field> that detail entity structure.
- Zero or more <unique> indicating unique constraints on field(s).

10.2.1. Advanced topic: using entity inheritance: extends and implements

MOLGENIS enables the use of the object-oriented mechanisms. Entities can extend on another existing entity using the ‘extends=”other_entity_name”’ notation which means that all fields from the other entity are added. Also abstract entities (interfaces) can be created using the

‘abstract=”true”’ attribute. Other entities can reuse these as reusable building blocks using the ‘implements=”other_entity_name”’ attribute.

Example usage of <entity …. extends= …> syntax:

```
<entity name="bicycle">
    <description>This my general entity</description>
    <field name="name" type="string"/>
</entity>
<entity name="mountainbike" extends="bicycle">
    <description>
      This type of bicycle has a name, but also number of gears
      </description>
    <field name="numberOfGears" type="int"/>
</entity>
```

10.3. <field> data definition ELEMENT

A field defines one property of an entity (i.e., a table column).

Example usage of the <field> element:

```
<field name="field_name"
       description="this is my first field of type string"/>

<field name="field_name" type="autoid"
       description="this is a id field, unique autonum integer"/>


<field name="field_name" type="xref"
       xref_field="other_entity.id_field"
       description="this is a crossrerence to otherentity"/>

<field name="field_name" type="enum" enum_options="[option1,option2]
       description="this is field of type enum"/>
```

Required attributes of the <field> element:

- name="name": locally unique name for this entity (within this entity).
- type="type": define the type of data that can be stored in this field, either:
    - type="string": a single line text string of variable length, max 255 chars.
    - type="int": a natural number.
    - type="boolean": a boolean.
    - type="decimal": a decimal number.
    - type="date": a date.

- type="datetime": a date that includes the time.
- type="file": attaches a file to the entity.
- type="text": a multiline textarea of max 2gb.
- type="xref": references to a field in another entity specified by xref_field attribute (required for xref). This will be shown as variable lookup-list.
- type="mref": many-to-many references to a field in another entity specified by xref_field attribute (required for mref). This will be shown as multiple select lookup-list. (Under the hood, a link table is generated)
- type="enum": references to a fixed look-up list options, specificed by enum_options attribute (required for enum)

Optional attributes of the <field> element:

- label="Nice screen name": an user-friendly alias to show as form header (default: copied from name).
- unique="true": values of this field must be unique within the entity (default: "false")
- nillable="true": this field can be left blank (default: "false")
- readonly="true": this field cannot be edited once they are saved (default: "false")
- length="n" (string only): limits the length of a string to 1<=n<=255 (default: "255").
- xref_field="entityname.fieldname": specifies a foreign key to the entity and field that this xref field must reference to, separated by '.' (default: required for xref)
- xref_label="anotherfield": makes labels for the xref options based on one field of the referenced entity (default: copied from xref_field).
- enum_options="[value1,value2]": the fixed list of options for this enum (required for enum).
- description="One line description": describes this field. This will be visibible to the user in the UI when (s)he mouses over the field or visits the documentation pages.

The <field> ELEMENT cannot contain other elements.

10.4. <unique> data definition ELEMENT

A unique defines which properties of an entity (i.e., table columns) should be unique. There are two ways to make a field unique.

1. A single column is unique. This example below shows that field "f1" is defined unique via unique="true". This means that there cannot be two entity instances - two rows in table entity1 - with the same value “x” in the f1 column.

```
<molgenis name="example">
  <entity name="entity1">
    <field name="f1" unique="true"/>
    <field name="f2" />
    <field name="f3" />
  </entity>
</molgenis>
```

2. A combination of two or more columns is unique. The example below shows that the combination of field “f1” and “f2” is defined as unique via the <unique> element. This means that there cannot be two entity instances - two rows in table entity1 - with the same value “x” in the f1 AND f2 column paired.

```
<molgenis name="example">
  <entity name="entity1">
    <field name="f1" />
    <field name="f2" />
    <field name="f3" />
    <unique fields="f1,f2"/>
  </entity>
</molgenis>
```

Required attributes of the <unique> ELEMENT:

- fields="field1,field2": comma separated enumeration of the unique fields.

The <unique> ELEMENT cannot contain other elements.

10.5. <module> data definition ELEMENT

The <module> element allows designers to group entities in packages which will show up in the generated documentation (and in future MOLGENIS also in the package structure). Example usage:

```
<molgenis name="example">
    <module name="module1">
        <description>This is my first module</description>
          <entity name="entity1">
                <field name="f1" type="string" unique="true"/>
                <field name="f2" type="string"/>
```

```
                <field name="f3" type="string"/>
          </entity>
          <entity name="entity2">
                <field name="f1" type="string" unique="true"/>
                <field name="f2" type="string"/>
                <field name="f3" type="string"/>
          </entity>
       </module>
</molgenis>
```

10.6. <form> user interface ELEMENT

The <form> element is used to define a user element that shows the records of a certain entity on screen (including insert, update, save, search, etc). A form may have tabbed (menu) or un-tabbed (form) subscreens which are defined by nesting other user interface elements.

Example usage of <form> element:

```
<form name="myname" entity="myentity">
  <form name="myname" entity="mysubentity"/>
</form>

<form name="myname" entity="myentity" viewtype="list" limit="10"/>
```

Required attributes of the <form> element:

- name="name": locally unique name for this screen element (within its container).
- entity="myentity": which data entity will be loaded (i.e., points to a <entity name="myentity"/>).

Optional attributes of the <form> element:

- label="Nice screen name": an user-friendly alias to show as form header (default: copied from name).
- viewtype="record" or viewtype="list": whether the form should start with a list or per-record (default: "record").
- limit="10": how many records must be shown in the list (default: "5").
- readonly="true": can the records be edited or is the form readonly (default: "false").

The <form> element can contain other elements:

- Zero or more <menu> elements to denote subscreen(s).
- Zero or more <form> elements to denote subscreen(s).
- Zero or more <plugin> elements to denote subscreen(s).

Nested forms are automatically linked to the containing form based on foreign key (xref) relationships.

10.7. <menu> user interface ELEMENT

The menu element allows the design of a hierarchical user interface with a menu on the left of the user interface and/or in tabs for each contained subscreen (menu, form, plugin).

Usage example of the <menu> element:

```
<molgenis>
	<menu name="my_mainmenu">
		<form name="myfirsttab" entity="an_entity1" />
		<menu name="my_submenu">
			<form name="mythirdtab" entity="an_entity2" />
			<form name="myfourthab" entity="an_entity3" />
		</menu>
	</menu>
</molgenis>
```

Required attributes of the <menu> element:

- name="menuname": locally unique name for this screen element (within its container).

Optional attributes of the <menu> element:

- startswith="mysubelement": subscreen tab that is selected when menu is first shown (default: first subscreen).

The <menu> element can contain other elements:

- Zero or more <menu> elements to denote subscreen(s).
- Zero or more <form> elements to denote subscreen(s).
- Zero or more <plugin> elements to denote subscreen(s).

10.8. <plugin> user interface ELEMENT

The <plugin> element allows to plug-in custom screen elements into the MOLGENIS user interface next to the auto-generated <form> and <menu> elements. The implementation of how to write a plugin is described elsewhere.

Example usage:

Required attributes of the <plugin> element:

- name="name": globally unique name for this entity (within this blueprint).
- Type="package.path.ClassName": reference to a java class in the 'handwritten/java' folder. Note: if this class doesn't exist than it will be automatically generated.

Optional attributes of the <plugin> element:

- label="Nice screen name": an user-friendly alias to show as form header (default: copied from name).

The <plugin> element cannot contain other elements.