Test project in Marushka environment

CONTENTS

1 The Guide to Test Project in MarushkaDesign

1.1 The Objective of Test Project

The aim of the test project in MarushkaDesign is demonstration of functionality and options that MarushkaDesign offers. In the test project we will demonstrate step by step how to create a fully functional project of map composition and how to create Web map publication from available data. The result will be an interactive web composition in Marushka environment, which will include basic vicinity map that will allow basic viewing, inserting and editing of database data. The Web Marushka in this example will be able to display information about elements, will be able to localize them, draw new graphical elements or erase user-drawn elements from the data store. For the individual group of elements is displayed the legend, there is of course an option to print to PDF or PNG file or by printer.

The main data store will be stored in **SQLite** environment, for the fulfillment of which will serve us drawings of ESRI shape files of fictional water network data, which was drawn over the real cadastral map of district Pardubice – Pardubičky. Database environment of SQLite assumes slightly advanced user knowledge of SQL.

"Sharp projects" from this test can differ – for example, using the main source Oracle database and vector data in WKB format, it is not necessary to deal with complications related to limitations of ESRI shape files, for which user need to cope with problems associated with e.g. cells. The existing GIS projects have available cell libraries, user styles libraries, etc. and the user does not have to deal with them no more. The basic principles, however, are in ESRI shape files, that most users are familiar with, well explicable.

The objective of this tutorial is not to create a detailed project description of MarushkaDesign environment, but to provide only the basic, yet the widest range of options that MarushkaDesign offers. To make the complex picture of functions and work in MarushkaDesign environment you can read the manual, which is included in the installation package of MarushkaDesign.

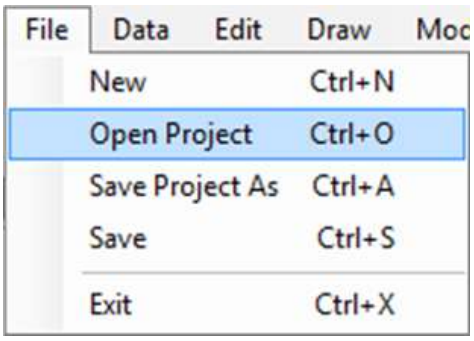
1.2 The Files of Test Project

Besides the already created main file `project.xml` and database file `test_project.db3` are in the Tutorial folder two other folders: SHP and Files. Files located in these folders will be used for creating the project itself. In the folder SHP are ESRI shape file sources and in folder Files are mainly two xml files – `Cells_EN.xml` – library of vector cells needed to view the cells of the items stored in the shape files and `NewWMS.xml` – the sub-project containing merged layers from cadastral map from WMS ČÚZK source. There are also included png files with an underscore at the beginning (to serve as raster cells to create a legend), gif files starting with "rc_" – also raster legend cells and file connection.jpg, which can be used as sample image when working with documents. Ultimately, it is SQL_Create.sql file that contain the basic script necessary to create needed database tables.

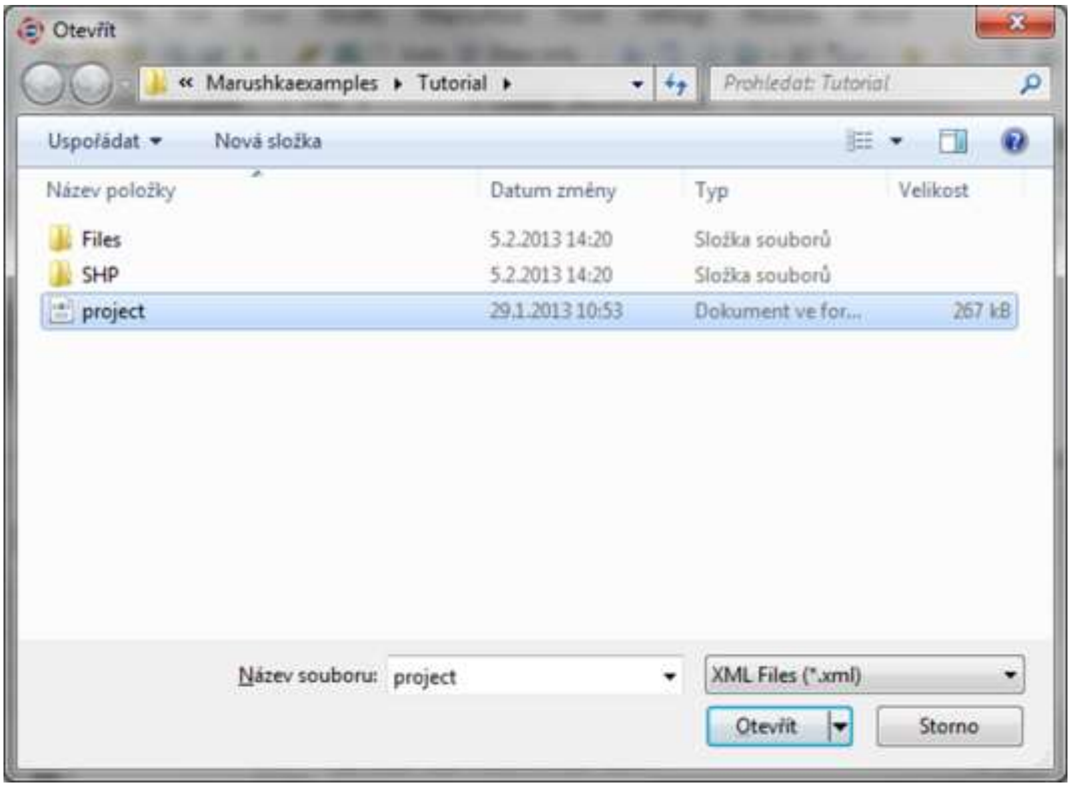
1.3 Opening the Created Project

Unzip the package into **C:\MarushkaExamples\Tutorial\**, after starting MarushkaDesign (in Windows Vista and higher is recommended to set MarushkaDesign.exe "run as administrator" so that log entries will be saved into Program Files folder) open the project – '**C:\MarushkaExamples\Tutorial\project.xml**' file. In an opened project, you can try for yourself the features that were created in this project and which we will try to create in the MarushkaDesign environment.

- Opening the project:



- From MarushkaExamples\Tutorial folder open the file project.xml:

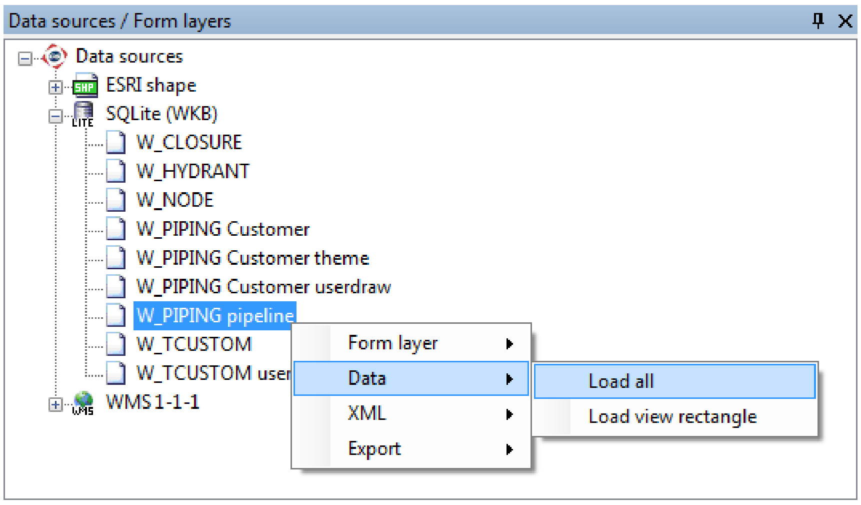


Název položky	Datum změny	Typ	Velikost
Files	5.2.2013 14:20	Složka souborů	
SHP	5.2.2013 14:20	Složka souborů	
project	29.1.2013 10:53	Dokument ve for...	267 kB

Název souboru: project
XML Files (*.xml)

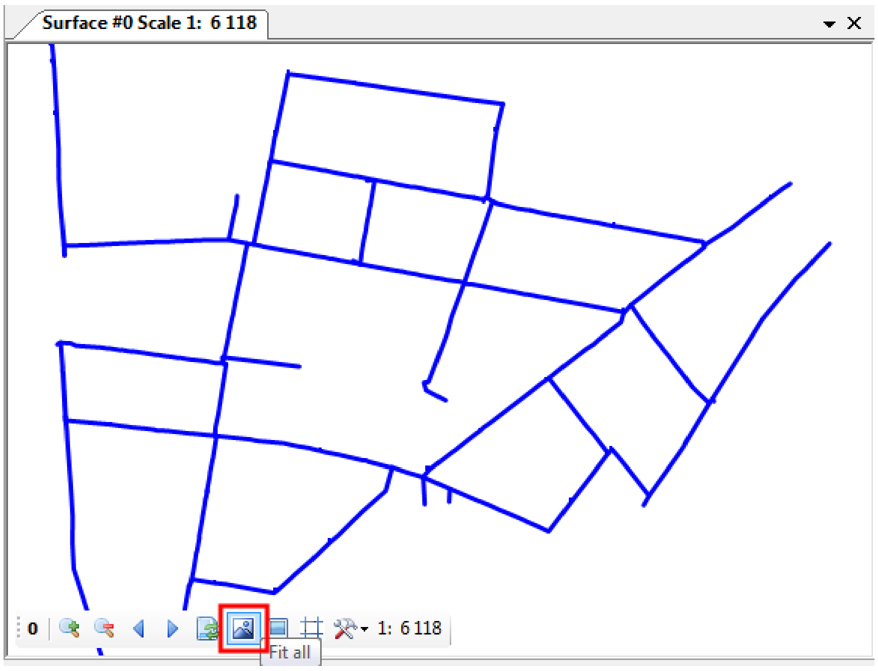
After opening the project, all the data stored within it will be retrieved. All the *Data sources* contained in the project will appear. For immediate view of the contained data is recommended to follow next procedure:

- In Data sources select from source SQLite layer "*W_PIPING pipeline*", which is after clicking the left mouse button highlighted, and right-click will display context menu, from which choose "*Data – Load all*".



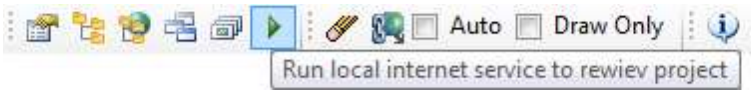
Marushka will be working for us for several moments and it will display all the elements, stored in the form layer *W_PIPING pipeline*. In the case that you do not see any element, Marushka did display data outside the visible area of this map window.

- Displaying of all the data in the map window is executed by clicking the left mouse button on the icon highlighted in the following picture and we will get the appropriate cut-out data:

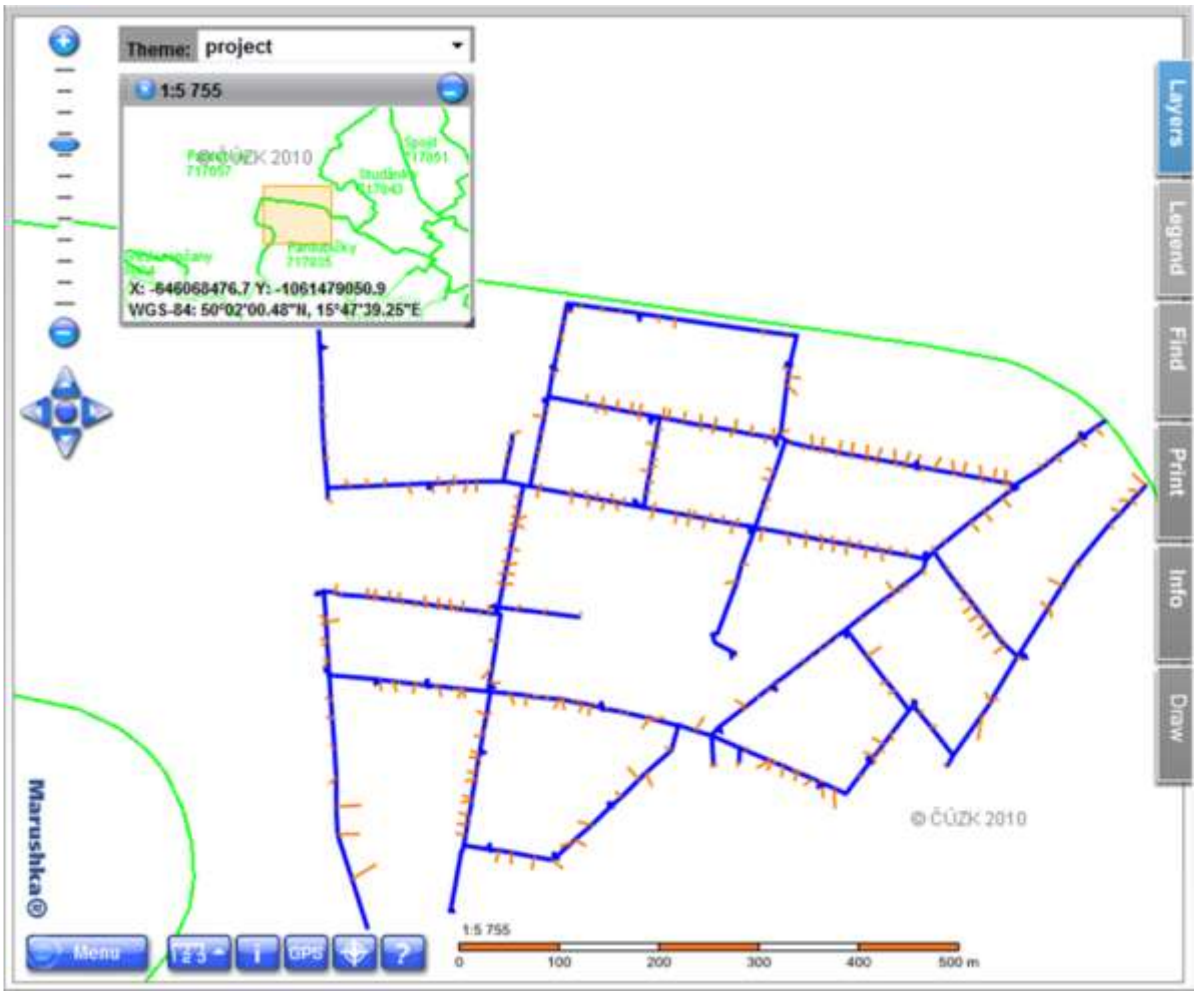


Thanks to such a defined map window cut-out the user can display data in a local web server in MarushkaDesign environment.

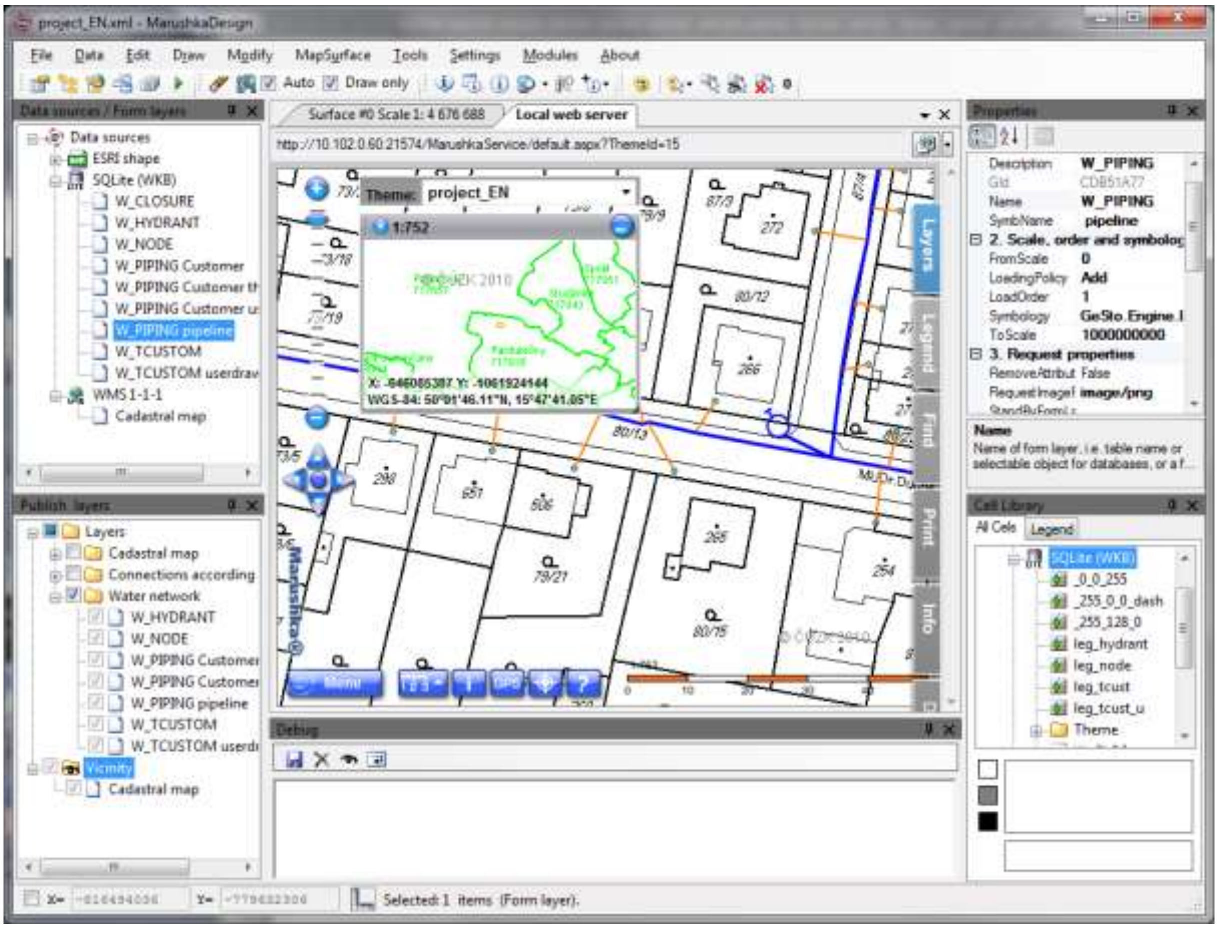
- Launch the Local WEB server by clicking icon showed in the picture below (from toolbar *Layers and mapsurfaces*)



- After launching the Local WEB server we will get the following picture:



The next figure shows the whole MarushkaDesign environment with opened project and a local WEB server preview:



And now it is just up to user how much he will play with the existing project and how he will examine existing options. In the following chapters you will be showed how to create this project by yourself.

2 Data Structure and a Project Composition

The main carriers of data structure of the project are:

1) Simplified fictitious data of water distribution network drawn over the real cadastral map (they will be imported from source ESRI shape files into SQLite data store).
2) Public WMS service from ČÚZK (Czech Office for Surveying, Mapping and Cadastre) allowing displaying cadastral map under the water supply network vector data.

2.1 Description of the Data Model of Water Distribution Network

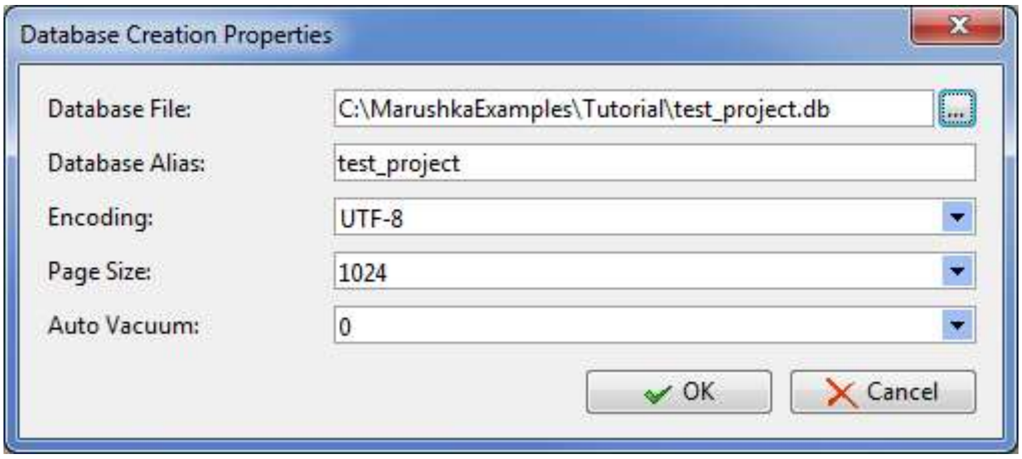
Water distribution network in our case consists of linear **route objects** of water supply network (pipelines and connections) defining the course of our engineering networks. Another vector data, that occurs on the water supply route are **cell of devices**. They represent **fictitious nodes** in crossing multiple pipelines or pipelines with connection, **hydrants** and particularly **connection terminations**, which are in our case also bearers of non-graphical data about the customer. In the resulting map publication are not displayed connection closures (eventually pipeline closures).

3 Creation of Data Structure in SQLite

As a source data were used ESRI shape files. Working with data stored in such a data store in WEB publication would be considerably limited and it would not be possible to use the database properties of individual elements, so it is convenient to convert the source data into the database structure. For simplicity, effectiveness and because of this solution is available to every user, was chosen SQLite database environment, which is fully open. The database structure of SQLite environment is of file character, so it is very easily transferable between different computer stations. For managing the database entities was chose database client SQLite Expert Personal (http://www.sqliteexpert.com/download.html). It is a fully sufficient SQL manager, working with it is fairly intuitive, but for work outside the scope of this tutorial it requires knowledge of SQL.

3.1 Database Creation

The new database is in the client SQLite Expert Personal created using the menu *File – New database*. A dialog box will display with properties of the database. Using the ... button, select the path and name for the new database file and keep other properties unchanged.


Database Creation Properties

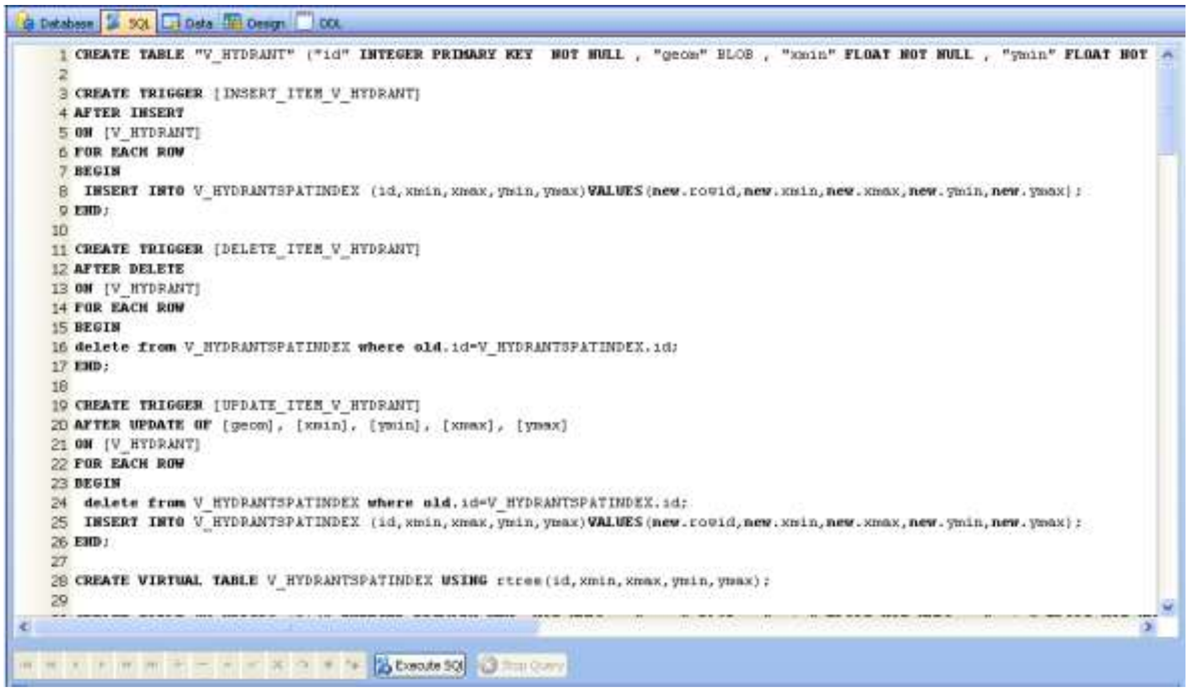
Database File:	C:\MarushkaExamples\Tutorial\test_project.db
Database Alias:	test_project
Encoding:	UTF-8
Page Size:	1024
Auto Vacuum:	0

3.2 Constitutive Scripts of Database Tables

Each graphic table consists of several mandatory columns and there must be also a virtual table for spatial indexing. All the tables have a similar structure. They must contain the columns ID, GEOM, XMIN, YMIN, XMAX, YMAX. Additional columns in created tables were created for the needs of our project. The importance of the tables that are not mandatory in terms of the structure of any GIS project will be explained later in the actual configuration of the project.

The only non-graphical table that we use in our case is the table `"demo_doc"`. This is a table of documents, which will store any files belonging to specific graphic elements.

Constitutive scripts to all necessary tables can be found in the file `SQL_Create.sql` which can be found in the package Tutorial Files folder. It is sufficient to copy the text contained in it to clipboard, paste it into SQLite expert manager and run the Execute SQL button.



```
CREATE TABLE "V_HYDRANT" ("id" INTEGER PRIMARY KEY  NOT NULL , "geom" BLOB , "xmin" FLOAT NOT NULL , "ymin" FLOAT NOT

CREATE TRIGGER [INSERT_ITEM_V_HYDRANT]
AFTER INSERT
ON [V_HYDRANT]
FOR EACH ROW
BEGIN
 INSERT INTO V_HYDRANTSPATINDEX (id,xmin,xmax,ymin,ymax)VALUES(new.rowid,new.xmin,new.xmax,new.ymin,new.ymax);
END;

CREATE TRIGGER [DELETE_ITEM_V_HYDRANT]
AFTER DELETE
ON [V_HYDRANT]
FOR EACH ROW
BEGIN
delete from V_HYDRANTSPATINDEX where old.id=V_HYDRANTSPATINDEX.id;
END;

CREATE TRIGGER [UPDATE_ITEM_V_HYDRANT]
AFTER UPDATE OF [geom], [xmin], [ymin], [xmax], [ymax]
ON [V_HYDRANT]
FOR EACH ROW
BEGIN
 delete from V_HYDRANTSPATINDEX where old.id=V_HYDRANTSPATINDEX.id;
 INSERT INTO V_HYDRANTSPATINDEX (id,xmin,xmax,ymin,ymax)VALUES(new.rowid,new.xmin,new.xmax,new.ymin,new.ymax);
END;

CREATE VIRTUAL TABLE V_HYDRANTSPATINDEX USING rtree(id,xmin,xmax,ymin,ymax);

```

3.2.1 Graphical Tables

Hydrant

```
CREATE TABLE "W_HYDRANT" ("id" INTEGER PRIMARY KEY  NOT NULL , "geom" BLOB
, "xmin" FLOAT NOT NULL , "ymin" FLOAT NOT NULL , "xmax"FLOAT NOT NULL ,
"ymax" FLOAT NOT NULL, "rc" VARCHAR(255), `angle`  float);

CREATE TRIGGER [INSERT_ITEM_W_HYDRANT]
AFTER INSERT
ON [W_HYDRANT]
FOR EACH ROW
BEGIN
 INSERT INTO W_HYDRANTSPATINDEX
(id,xmin,xmax,ymin,ymax)VALUES(new.rowid,new.xmin,new.xmax,new.ymin,new.yma
x);
END;

CREATE TRIGGER [DELETE_ITEM_W_HYDRANT]
AFTER DELETE
ON [W_HYDRANT]
FOR EACH ROW
BEGIN
delete from W_HYDRANTSPATINDEX where old.id=W_HYDRANTSPATINDEX.id;
END;

CREATE TRIGGER [UPDATE_ITEM_W_HYDRANT]
AFTER UPDATE OF [geom], [xmin], [ymin], [xmax], [ymax]
ON [W_HYDRANT]
FOR EACH ROW
BEGIN
 delete from W_HYDRANTSPATINDEX where old.id=W_HYDRANTSPATINDEX.id;
 INSERT INTO W_HYDRANTSPATINDEX
(id,xmin,xmax,ymin,ymax)VALUES(new.rowid,new.xmin,new.xmax,new.ymin,new.yma
x);
END;
```

```
CREATE VIRTUAL TABLE W_HYDRANTSPATINDEX USING
rtree(id,xmin,xmax,ymin,ymax);
```

Connection Termination

```
CREATE TABLE "W_TCUSTOM" ("id" INTEGER PRIMARY KEY  NOT NULL , "geom" BLOB
, "xmin" FLOAT NOT NULL , "ymin" FLOAT NOT NULL , "xmax" FLOATNOT NULL ,
"ymax" FLOAT NOT NULL, "rc" VARCHAR(255), `name`  text, `surname`  text,
`city`  text, `userdraw`  INTEGER);

CREATE TRIGGER [INSERT_ITEM_W_TCUSTOM]
AFTER INSERT
ON [W_TCUSTOM]
FOR EACH ROW
BEGIN
 INSERT INTO W_TCUSTOMSPATINDEX
(id,xmin,xmax,ymin,ymax)VALUES(new.rowid,new.xmin,new.xmax,new.ymin,new.yma
x);
END;

CREATE TRIGGER [DELETE_ITEM_W_TCUSTOM]
AFTER DELETE
ON [W_TCUSTOM]
FOR EACH ROW
BEGIN
delete from W_TCUSTOMSPATINDEX where old.id=W_TCUSTOMSPATINDEX.id;
END;

CREATE TRIGGER [UPDATE_ITEM_W_TCUSTOM]
AFTER UPDATE OF [geom], [xmin], [ymin], [xmax], [ymax]
ON [W_TCUSTOM]
FOR EACH ROW
BEGIN
 delete from W_TCUSTOMSPATINDEX where old.id=W_TCUSTOMSPATINDEX.id;
 INSERT INTO W_TCUSTOMSPATINDEX
(id,xmin,xmax,ymin,ymax)VALUES(new.rowid,new.xmin,new.xmax,new.ymin,new.yma
x);
END;

CREATE VIRTUAL TABLE W_TCUSTOMSPATINDEX USING
rtree(id,xmin,xmax,ymin,ymax);
```

Connection Closure

```
CREATE TABLE "W_CLOSURE" ("id" INTEGER PRIMARY KEY  NOT NULL , "geom" BLOB
, "xmin" FLOAT NOT NULL , "ymin" FLOAT NOT NULL , "xmax" FLOATNOT NULL ,
"ymax" FLOAT NOT NULL, "rc" VARCHAR(255));

CREATE TRIGGER [INSERT_ITEM_W_CLOSURE]
AFTER INSERT
ON [W_CLOSURE]
FOR EACH ROW
BEGIN
 INSERT INTO W_CLOSURESPATINDEX
(id,xmin,xmax,ymin,ymax)VALUES(new.rowid,new.xmin,new.xmax,new.ymin,new.yma
x);
END;

CREATE TRIGGER [DELETE_ITEM_W_CLOSURE]
```

```
AFTER DELETE
ON [W_CLOSURE]
FOR EACH ROW
BEGIN
delete from W_CLOSURESPATINDEX where old.id=W_CLOSURESPATINDEX.id;
END;

CREATE TRIGGER [UPDATE_ITEM_W_CLOSURE]
AFTER UPDATE OF [geom], [xmin], [ymin], [xmax], [ymax]
ON [W_CLOSURE]
FOR EACH ROW
BEGIN
 delete from W_CLOSURESPATINDEX where old.id=W_CLOSURESPATINDEX.id;
 INSERT INTO W_CLOSURESPATINDEX
(id,xmin,xmax,ymin,ymax)VALUES(new.rowid,new.xmin,new.xmax,new.ymin,new.yma
x);
END;

CREATE VIRTUAL TABLE W_CLOSURESPATINDEX USING
rtree(id,xmin,xmax,ymin,ymax);
```

Fictitious Node

```
CREATE TABLE "W_NODE" ("id" INTEGER PRIMARY KEY  NOT NULL , "geom" BLOB ,
"xmin" FLOAT NOT NULL , "ymin" FLOAT NOT NULL , "xmax" FLOATNOT NULL ,
"ymax" FLOAT NOT NULL, "rc" VARCHAR(255));

CREATE TRIGGER [INSERT_ITEM_W_NODE]
AFTER INSERT
ON [W_NODE]
FOR EACH ROW
BEGIN
 INSERT INTO W_NODESPATINDEX
(id,xmin,xmax,ymin,ymax)VALUES(new.rowid,new.xmin,new.xmax,new.ymin,new.yma
x);
END;

CREATE TRIGGER [DELETE_ITEM_W_NODE]
AFTER DELETE
ON [W_NODE]
FOR EACH ROW
BEGIN
delete from W_NODESPATINDEX where old.id=W_NODESPATINDEX.id;
END;

CREATE TRIGGER [UPDATE_ITEM_W_NODE]
AFTER UPDATE OF [geom], [xmin], [ymin], [xmax], [ymax]
ON [W_NODE]
FOR EACH ROW
BEGIN
 delete from W_NODESPATINDEX where old.id=W_NODESPATINDEX.id;
 INSERT INTO W_NODESPATINDEX
(id,xmin,xmax,ymin,ymax)VALUES(new.rowid,new.xmin,new.xmax,new.ymin,new.yma
x);
END;

CREATE VIRTUAL TABLE W_NODESPATINDEX USING rtree(id,xmin,xmax,ymin,ymax);
```

Pipeline

```
CREATE TABLE [W_PIPING] (
  [id] INTEGER NOT NULL PRIMARY KEY,
  [geom] BLOB,
  [xmin] FLOAT NOT NULL,
  [ymin] FLOAT NOT NULL,
  [xmax] FLOAT NOT NULL,
  [ymax] FLOAT NOT NULL,
  [rc] VARCHAR(255),
  [Datum_vystavby] DATE,
  [dim_prip] INTEGER,
  [zakazka] integer,
  [userdraw] INTEGER);

CREATE TRIGGER [INSERT_ITEM_W_PIPING]
AFTER INSERT
ON [W_PIPING]
FOR EACH ROW
BEGIN
 INSERT INTO W_PIPINGSPATINDEX
(id,xmin,xmax,ymin,ymax)VALUES(new.rowid,new.xmin,new.xmax,new.ymin,new.yma
x);
END;

CREATE TRIGGER [DELETE_ITEM_W_PIPING]
AFTER DELETE
ON [W_PIPING]
FOR EACH ROW
BEGIN
delete from W_PIPINGSPATINDEX where old.id=W_PIPINGSPATINDEX.id;
END;

CREATE TRIGGER [UPDATE_ITEM_W_PIPING]
AFTER UPDATE OF [geom], [xmin], [ymin], [xmax], [ymax]
ON [W_PIPING]
FOR EACH ROW
BEGIN
 delete from W_PIPINGSPATINDEX where old.id=W_PIPINGSPATINDEX.id;
 INSERT INTO W_PIPINGSPATINDEX
(id,xmin,xmax,ymin,ymax)VALUES(new.rowid,new.xmin,new.xmax,new.ymin,new.yma
x);
END;

CREATE VIRTUAL TABLE W_PIPINGSPATINDEX USING rtree(id,xmin,xmax,ymin,ymax);
```

3.2.2 Table of Documents

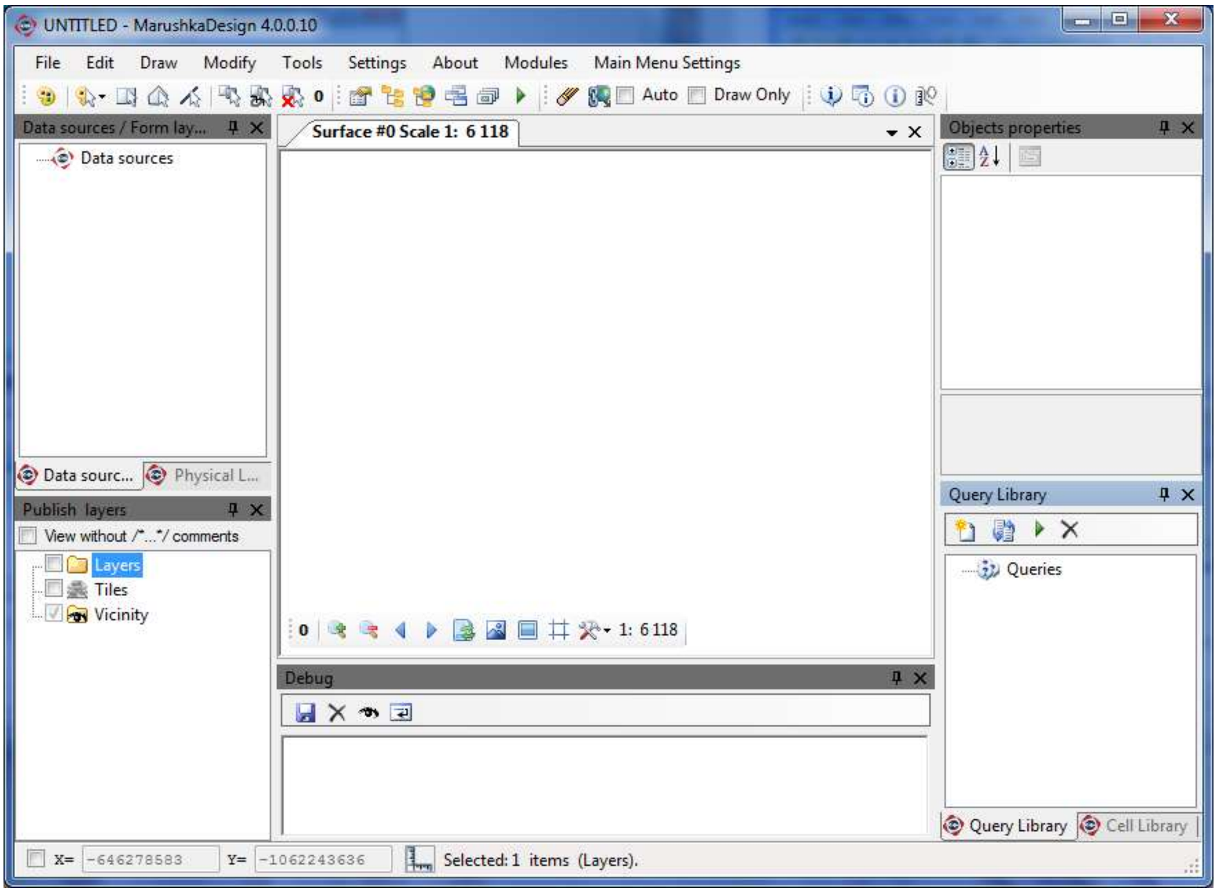
```
CREATE TABLE "demo_doc" ("ID" INTEGER NOT NULL PRIMARY KEY AUTOINCREMENT,
BLOB_TYPE VARCHAR(50),LABEL TEXT,BIRTH_DATE DATETIME,DOC
BLOB,ID_GRAPHICS_ELEMENT INTEGER, `STRING_ID`  varchar(256));

CREATE TRIGGER demo_doc_insert_date AFTER INSERT ON demo_doc
BEGIN
UPDATE demo_doc SET BIRTH_DATE = DATETIME('NOW')  WHERE rowid = new.rowid;
UPDATE demo_doc SET STRING_ID = 'ID'||new.rowid WHERE rowid = new.rowid;
```

After successfully creating all the tables you are ready to move on to the next step, which is an actual import of data in the data store.

4 Teaching Marushka to read and write

To be able to read from the database data store, the user need to write data from source ESRI shape files into the database.

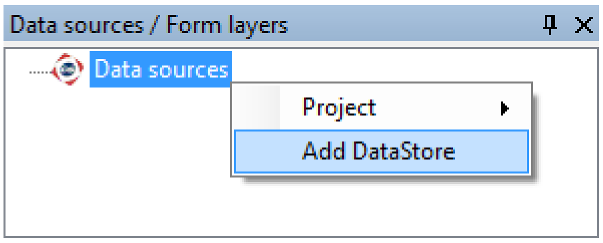
In our new project you can see only blank MarushkaDesign environment at the moment, which may look like this (depending on individual user configuration, how he organizes dialog boxes – you can read more about control and work in MarushkaDesign environment in the manual):



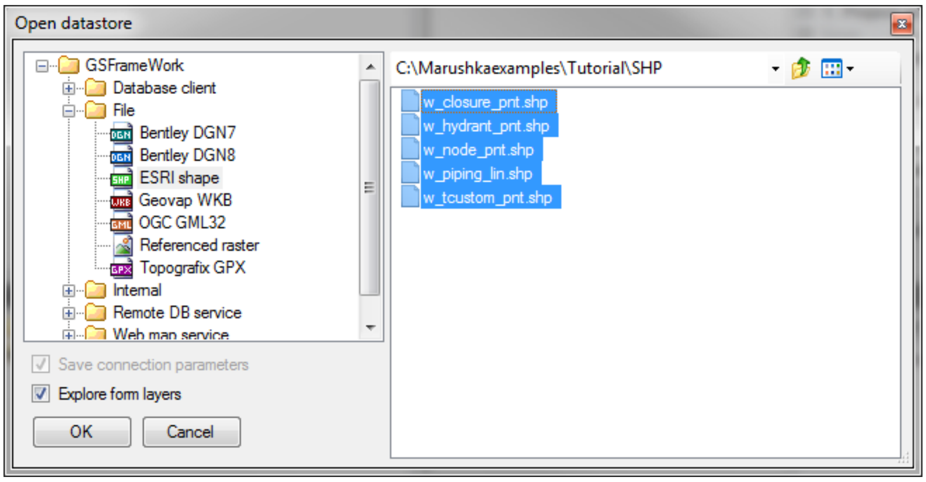
X= -646278583 Y= -1062243636 Selected: 1 items (Layers).

4.1 Connecting data sources

- Connect the first data source, data from ESRI shape files (context menu will appear after right mouse click on the *Data sources*):



-
- A dialog box appears, allowing selecting individual shape files. You can join them individually or collectively using standard multiselect keys (CTRL or SHIFT + left mouse button):



Open datastore

- GSFrameWork
  - Database client
  - File
    - Bentley DGN7
    - Bentley DGN8
    - ESRI shape
    - Geovap WKB
    - OGC GML32
    - Referenced raster
    - Topografix GPX
  - Internal
  - Remote DB service
  - Web map service

C:\Marushkaexamples\Tutorial\SHP

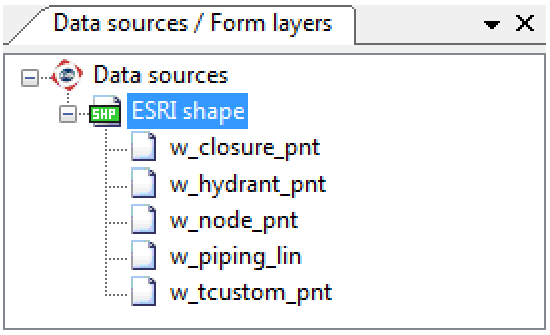
- w_closure_pnt.shp
- w_hydrant_pnt.shp
- w_node_pnt.shp
- w_piping_lin.shp
- w_tcustom_pnt.shp

☑ Save connection parameters

☑ Explore form layers

OK Cancel

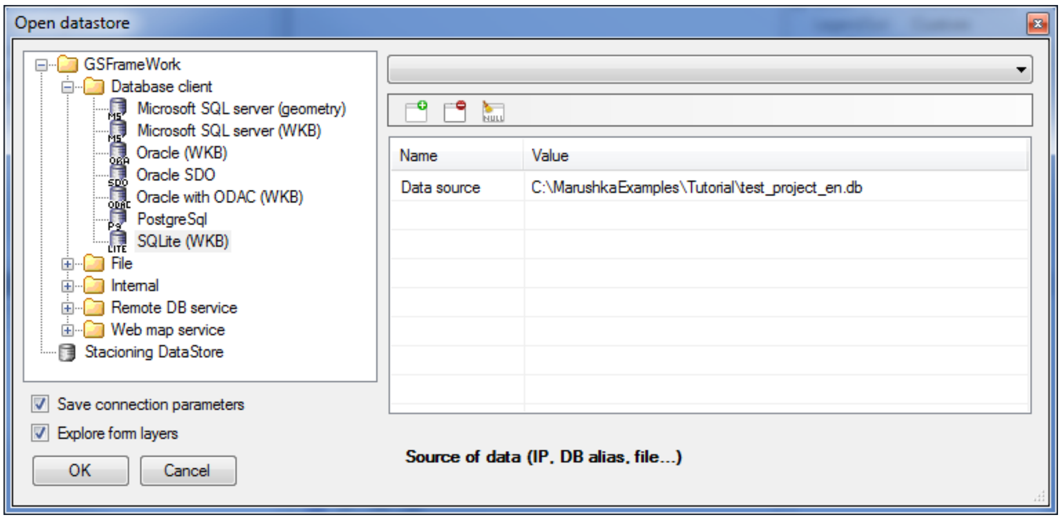
- After you click on OK, the *Data sources* dialogue box will change, a new item ESRI shape with five nodes will appear:



Data sources / Form layers

- Data sources
  - ESRI shape
    - w_closure_pnt
    - w_hydrant_pnt
    - w_node_pnt
    - w_piping_lin
    - w_tcustom_pnt

- Now we need to write the data into database data store. Therefore connect the already created SQLite database. Again using context menu of *Data sources* add a new data source, this time *Database – SQLite (WKB)*. Insert the name of your database and path to it into the field value:



Open datastore

- GSFrameWork
  - Database client
    - Microsoft SQL server (geometry)
    - Microsoft SQL server (WKB)
    - Oracle (WKB)
    - Oracle SDO
    - Oracle with ODAC (WKB)
    - PostgreSql
    - SQLite (WKB)
  - File
  - Internal
  - Remote DB service
  - Web map service
  - Stacioning DataStore

Name	Value
Data source	C:\MarushkaExamples\Tutorial\test_project_en.db

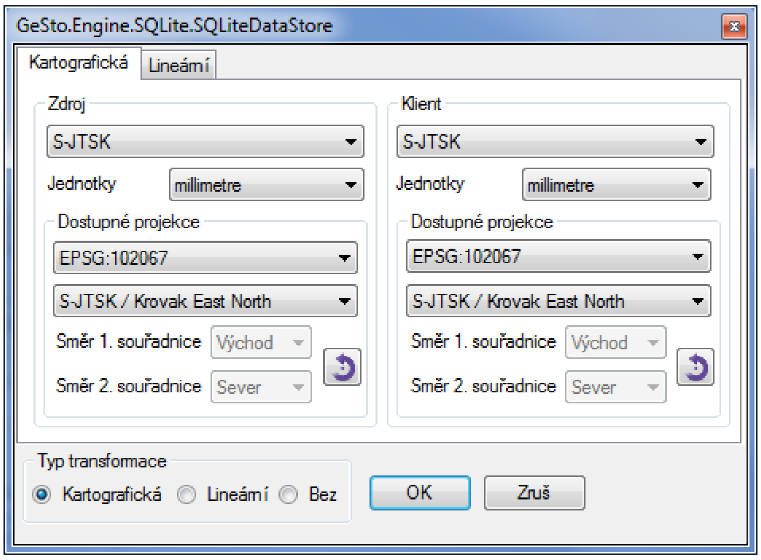
☑ Save connection parameters

☑ Explore form layers

OK Cancel

Source of data (IP, DB alias, file...)

- After confirming with OK, the dialog box for entering the source cartographic projection will appear. As a source coordinate system choose *S-JTSK* in units of millimeters. Keep projection without changes: *EPSG: 102067:*


GeSto.Engine.SQLite.SQLiteDataStore

Kartografická | Lineární

Zdroj: S-JTSK; Jednotky: millimetre; Dostupné projekce: EPSG:102067, S-JTSK / Krovak East North; Směr 1. souřadnice: Východ; Směr 2. souřadnice: Sever

Klient: S-JTSK; Jednotky: millimetre; Dostupné projekce: EPSG:102067, S-JTSK / Krovak East North; Směr 1. souřadnice: Východ; Směr 2. souřadnice: Sever

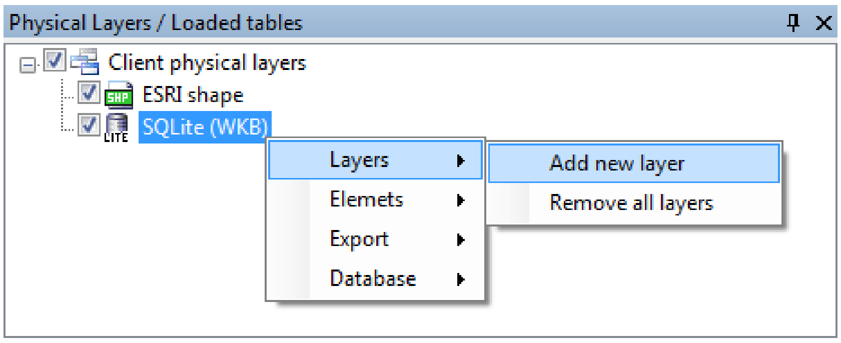
Typ transformace: [✓ Kartografická] Lineární Bez

Click OK to confirm and in Data sources you instantly see another change, a new branch appeared – this time *SQLite (WKB)* again with five nodes.

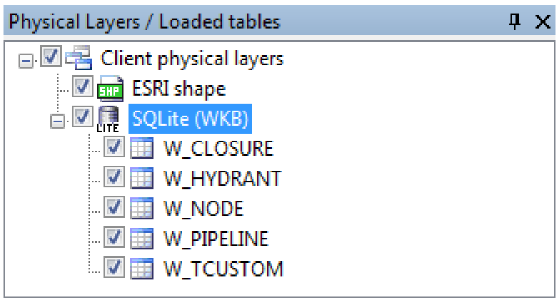
4.2 Importing Data from ESRI Shape Files into SQLite

For the actual import into any other data store, it is necessary to have a relevant physical layer, into which the source data will be directed. **Physical layers** are created in the target data store.

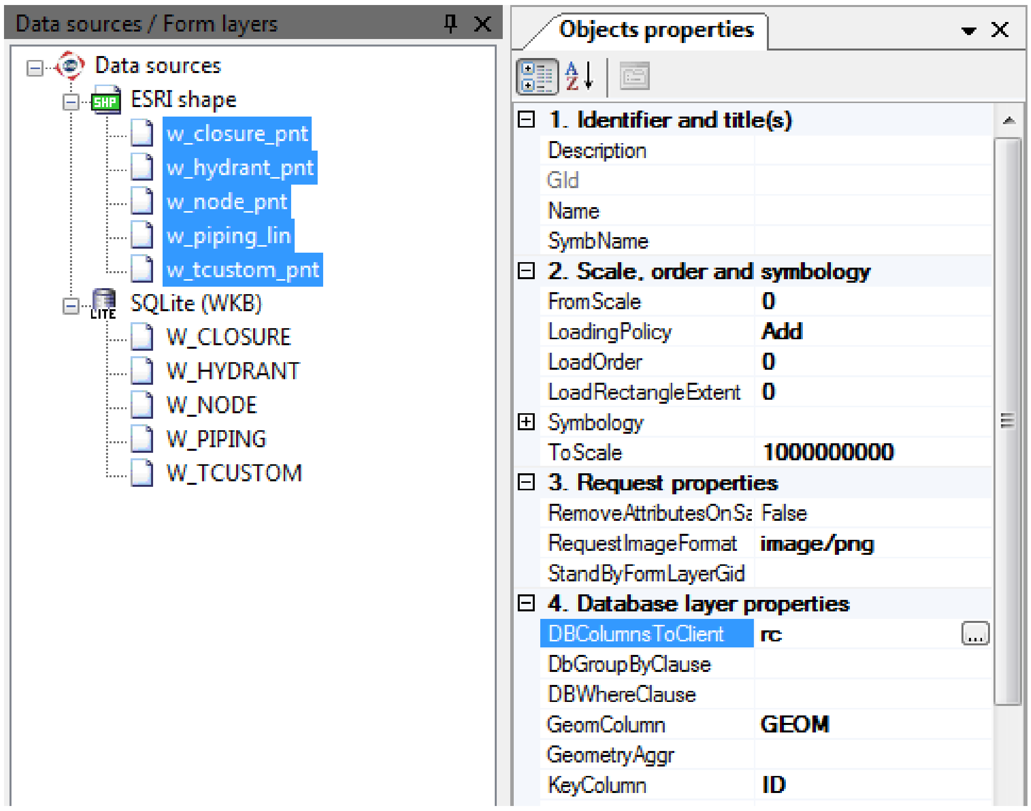
Switch to the window *Physical Layers* and in data source *SQLite (WKB)* successively create all five physical layers that you need. To invoke the context menu for creating new physical layers, right mouse click on the data source:



In the dialog box for the layer name enter the name of the first physical layer (the name of the database table into which the data will be directed). Repeat this procedure five times, for each database table separately. The list of physical layers should now look like this:

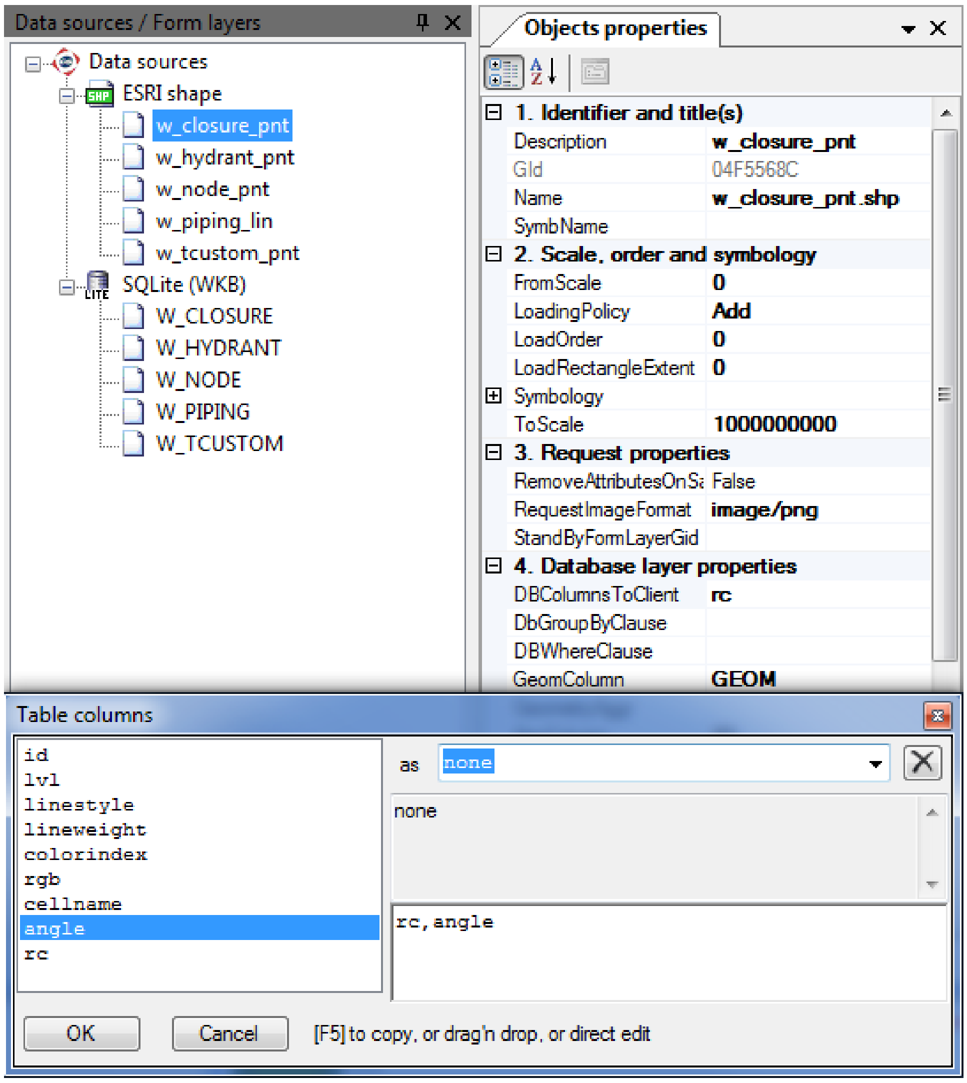


You can now direct these data into the newly created layers. Given that the data in ESRI shape files also contain *attributes* that you want to use, it is necessary to ensure, that these attributes get into the data store. To all the form layers from data store ESRI shape is needed to supplement the attributes into the properties of each layer. All the layers will require attribute *rc*, which can be filled in the properties in bulk:



1. Identifier and title(s)	
Description	
GId	
Name	
SymbName	
2. Scale, order and symbology	
FromScale	0
LoadingPolicy	Add
LoadOrder	0
LoadRectangleExtent	0
Symbology	
ToScale	1000000000
3. Request properties	
RemoveAttributesOnS	False
RequestImageFormat	image/png
StandByFormLayerGid	
4. Database layer properties	
DBColumnsToClient	rc
DbGroupByClause	
DBWhereClause	
GeomColumn	GEOM
GeometryAggr	
KeyColumn	ID

You still need other attributes from two form layers (w_hydrant_pnt and w_tcustom_pnt). Perform these changes separately for each form layer. In the form layer w_hydrant_pnt you need an attribute *angle*. Adding is done in the properties dialog box for a particular layer – *DBColumnsToClient*. Drag the desired attribute by pressing F5 key:



1. Identifier and title(s)	
Description	w_closure_pnt
GId	04F5568C
Name	w_closure_pnt.shp
SymbName	
2. Scale, order and symbology	
FromScale	0
LoadingPolicy	Add
LoadOrder	0
LoadRectangleExtent	0
Symbology	
ToScale	1000000000
3. Request properties	
RemoveAttributesOnS	False
RequestImageFormat	image/png
StandByFormLayerGid	
4. Database layer properties	
DBColumnsToClient	rc
DbGroupByClause	
DBWhereClause	
GeomColumn	GEOM

```
id
lvl
linestyle
lineweight
colorindex
rgb
cellname
angle
rc
```

as none

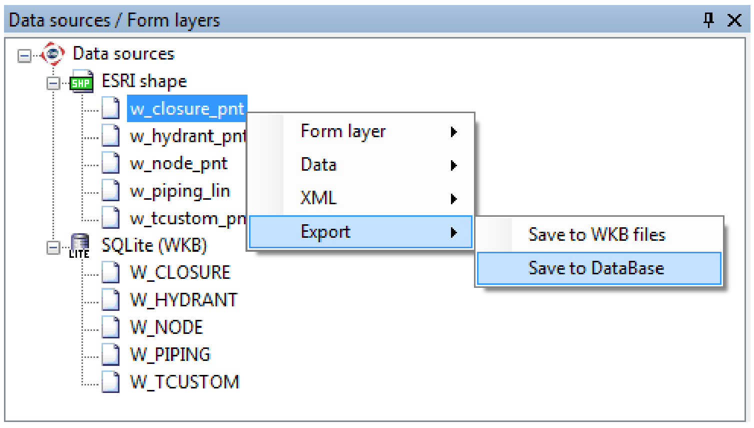
none

```
rc,angle
```

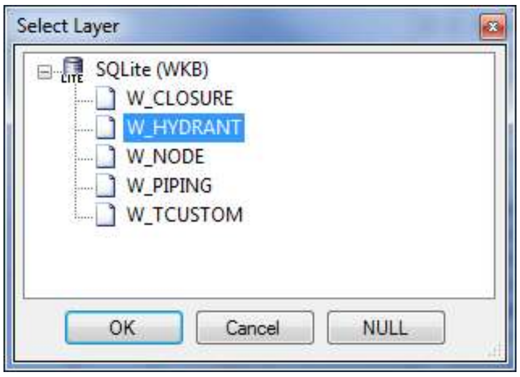
[F5] to copy, or drag'n drop, or direct edit

In a similar manner complete attributes *name*, *surname* and *city* for form layer *w_tcustom_pnt* layer.

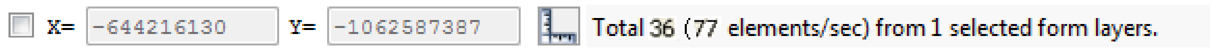
You can finally enter the data adjusted like this into database data store. You can invoke the context menu for individual layers of ESRI data store by right-clicking on it:



Then confirm the option *Save to DataBase*. In the consequently opened dialog box, select corresponding physical layer of the database data store:



After confirming with OK, you got asked, if you want to keep the original ID. Given that the elements in ESRI shape files do not come from database, you have to click on "No". After this, the import into the database will proceed; you can monitor its status in the MarushkaDesign status bar:

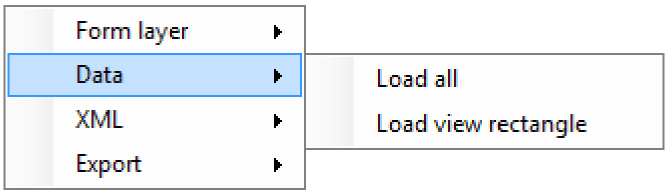

X= -644216130 Y= -1062587387 Total 36 (77 elements/sec) from 1 selected form layers.

Apply the same procedure for all the remaining form layers from ESRI shape data store and so enter all these layers into database data store SQLite (WKB). Repeat the same procedures for layers: *w_piping_lin, w_tcustom_pnt, w_closure_pnt* and *w_node_pnt*. So that's how we taught Marushka to **"write into the database"**.

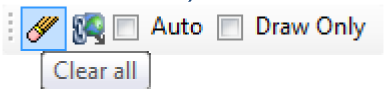
4.3 Displaying Data in MarushkaDesign from Database Data Store

In database data store SQLite you now have the data prepared for further use. We didn't take over graphical attributes from ESRI shape file source, so you will learn to change graphical symbology of data and you will **teach Marushka to draw** the way you like it. You will leave definitively ESRI data source at this moment and you will continue to work only with SQLite data source. From now on you will work primarily with following controls.

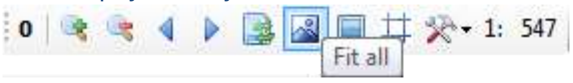
- Form layer context menu – *Data – Load all,* or *Load view rectangle* (allows us to retrieve the data layer after formal changes that we have made in the project):



- *Clear all* icon from tools panel *Load data* (removes from the drawing all physical layers – data that were loaded – it is necessary to activate this icon after each adjustment of form layer before downloading the new data, otherwise the map window won't display any change – only the elements which are not part of the drawing will be displayed in the map window):

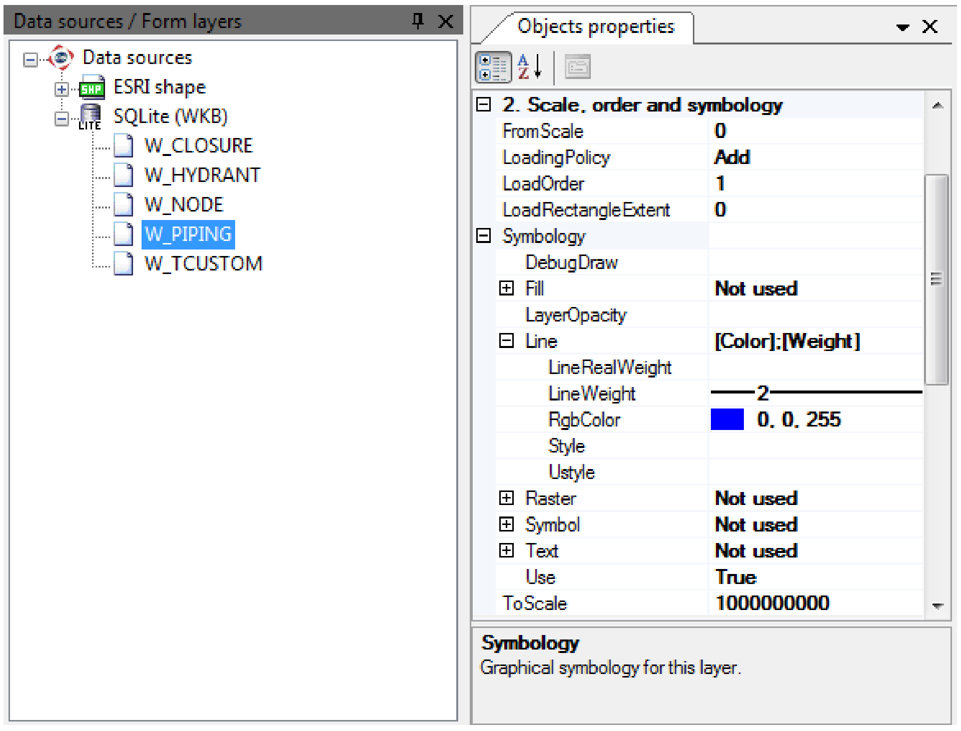


- Icon *Fit all* from map window menu is used for displaying all the elements in the available physical layers at the moment in the main map window:



4.3.1 Displaying of Line Elements

You will master the pair of commands *Clear all, Load all* or *Load view rectangle* soon and you will use them almost intuitively after every change in the form layer and for testing the functionality of new changes. Try this with an example. Clear all and Load all the data from the form layer *W_PIPING*. As a result, you see all the elements of the route in a grey design that you may not like. Instead, you want to see water distribution network in blue color, the thickness of two. So change the symbology in the form layer properties of *W_PIPING Symbology* of displayed layer:



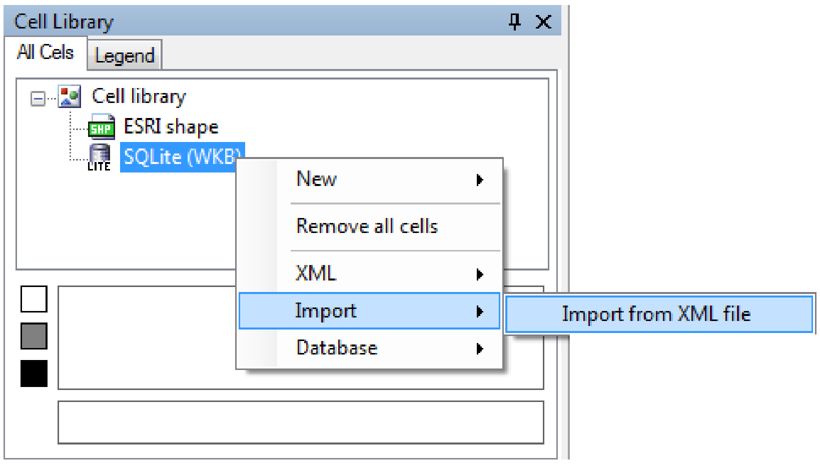
2. Scale, order and symbology	
FromScale	0
LoadingPolicy	Add
LoadOrder	1
LoadRectangleExtent	0
Symbology	
DebugDraw	
Fill	Not used
LayerOpacity	
Line	[Color]:[Weight]
LineRealWeight	
LineWeight	2
RgbColor	🟦 0, 0, 255
Style	
Ustyle	
Raster	Not used
Symbol	Not used
Text	Not used
Use	True
ToScale	1000000000

Symbology
Graphical symbology for this layer.

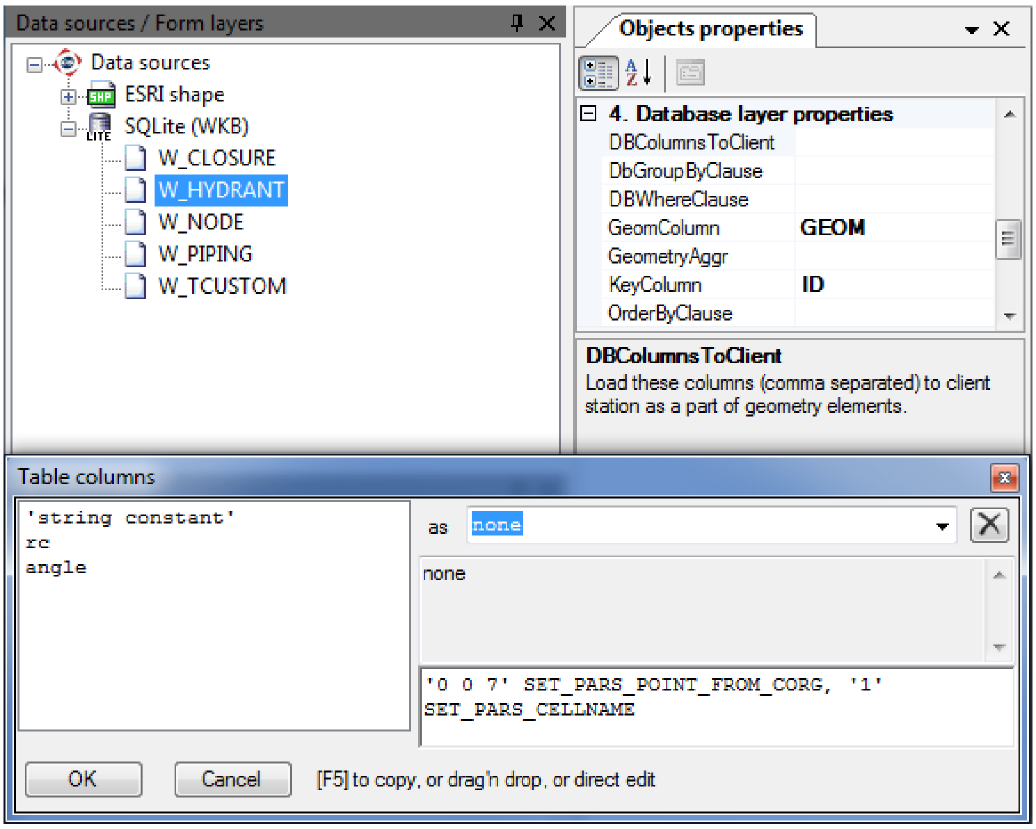
You can see immediately the results of this change using the already well-known combination of two commands *Clear all* and *Load all* from the context menu of form layer *W_PIPING*.

You have already done quite a lot of work on the project, so it would be appropriate to commemorate the interim project saving. It is advisable to get used to it and use e.g. keyboard shortcut CTRL + S for interim saving the project. The automatic save occurs only when running a local WEB server, which you will find out more about later.

You have been shown how easily is to change symbology of linear objects. As regards to the cell symbology, it is not that simple. The base is formed by an attached cell library. You will use its components to display individual elements. Open the Cell library (from menu *Tools - Libraries - Cells*). The dialog box in which you can import the cell library from `Cells_EN.xml` file displays. This file is included with this tutorial. Invoke the context menu by right mouse click and in dialog box for file selection choose the file from the tutorial folder (`c:\MarushkaExamples\Tutorial\Files\Cells.xml`).



With attached cell library can be displayed the required cells. Since the ESRI shape files are unfamiliar with element type cell, at this moment all the elements in the database data store are point type. These points will be now displayed as a specific cells. Now let's have a look at form layer properties, specifically property *DBColumnsToClient*. This is a feature, where the values from the database entity are also being loaded from specific database table columns. In addition, in MarushkaDesign environment are defined pseudo columns, which you will now utilize. You can read more about pseudo columns `SET_PARS_` in MarushkaDesign manual. For displaying the cells from a point object, you will use pseudo column `SET_PARS_POINT_FROM_CORG`. This point will be used as a reference point for corresponding cell. Writing of this column can be performed manually or by selecting pseudo column, which is a part of this dialog box. The value in this pseudo column will be '0 0 7'. It is also necessary to define the name of the cell that will be used for elements from this database. Use the second pseudo column, this time `SET_PARS_CELLNAME` and into its value write in the name of the specific cell from our cell library. For *W_HYDRANT* will transcript of these two pseudo columns look like this:



4. Database layer properties	
DBColumnsToClient	
DbGroupByClause	
DBWhereClause	
GeomColumn	GEOM
GeometryAggr	
KeyColumn	ID
OrderByClause	

DBColumnsToClient
Load these columns (comma separated) to client station as a part of geometry elements.

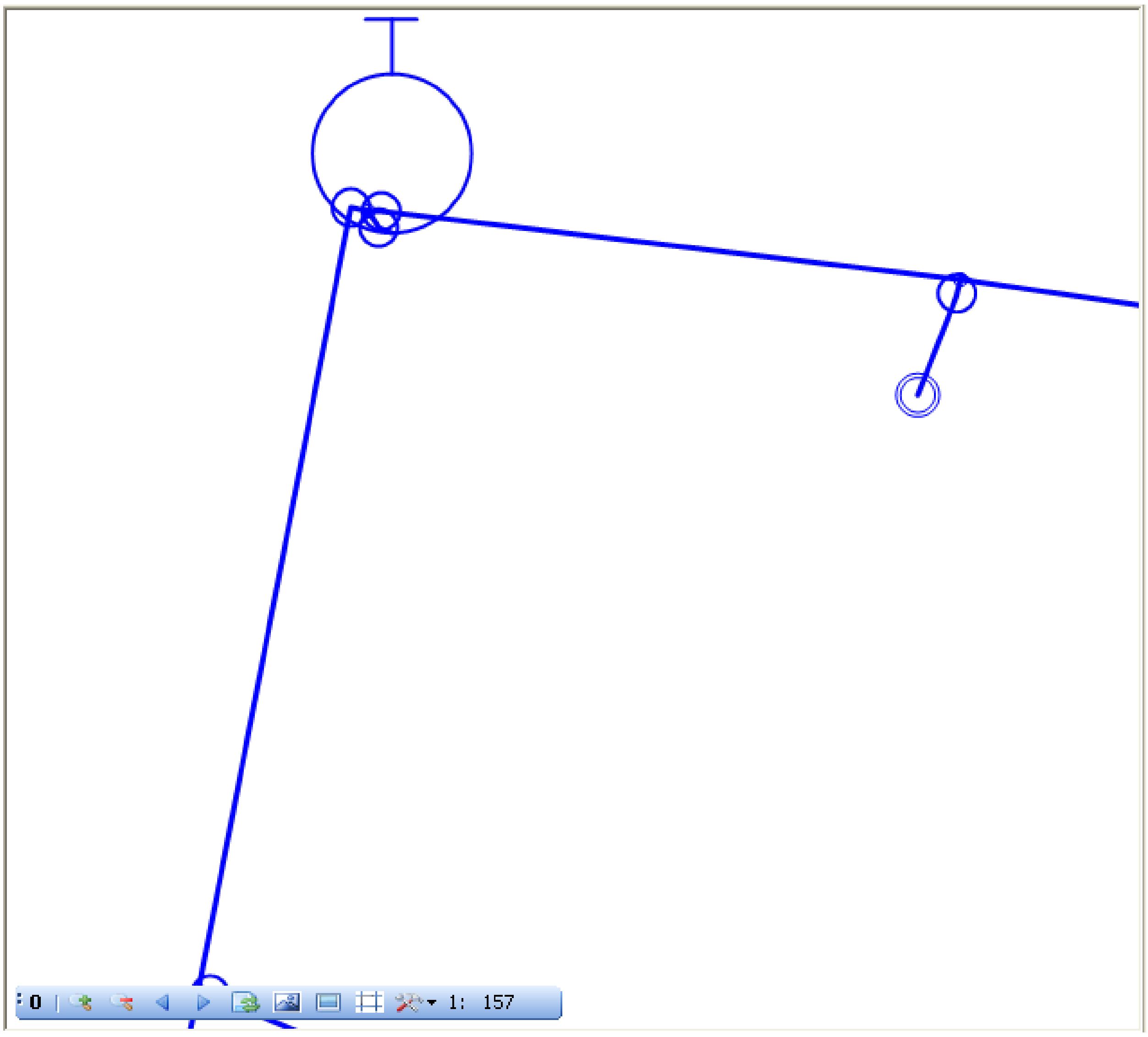
```
'string constant'
rc
angle
```

```
'0 0 7' SET_PARS_POINT_FROM_CORG, '1'
SET_PARS_CELLNAME
```

Individual exercise:

For displaying hydrant was used cell *W_HYDR* from cell library, having in its `Cellname` property value '1'. This value is the control value for the database resolution of cell names. You will now repeat the procedure independently and without detailed instructions for all the remaining form layers. To the form layer *W_TCUSTOM* assign cell *W_TCUS*, to layer *W_CLOSURE* cell *W_CLOS* and to layer *W_NODE* cell *W_NODE*.

The user can instantly verify correctness of this exercise by downloading data to MarushkaDesign map window. Perform *Clear all* and then load the data from individual form layers. Successively load form layers *W_HYDRANT*, *W_PIPELINE*, *W_TCUSTOM*, *W_CLOSURE* and *W_NODE* and you can see a result of your efforts in e.g. this form (cut-out is from the top left side of the map window):



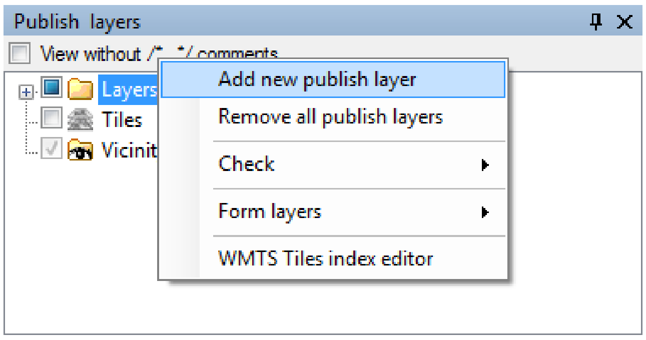
In the picture you can see all the elements of water distribution network – route pipelines, connections, fictitious nodes, closures, hydrants and connection terminations.

5 Displaying Data in Local WEB Server

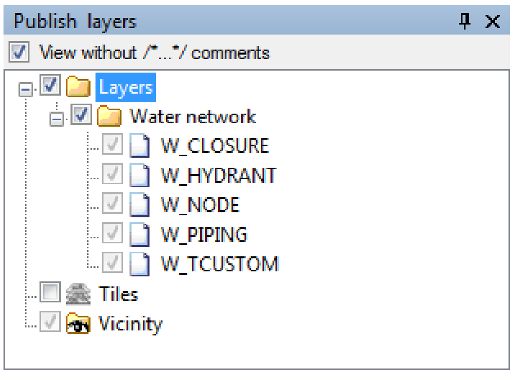
Our map composition is at the moment, although still somewhat static, and not yet completely according to our ideas, but nevertheless it is worth it to show how the result can be displayed in the **local WEB server**. The environment created like this can be transferred to any Marushka server and so will look the resulting map publication in web browser.

5.1 The Creation of Publish Layer

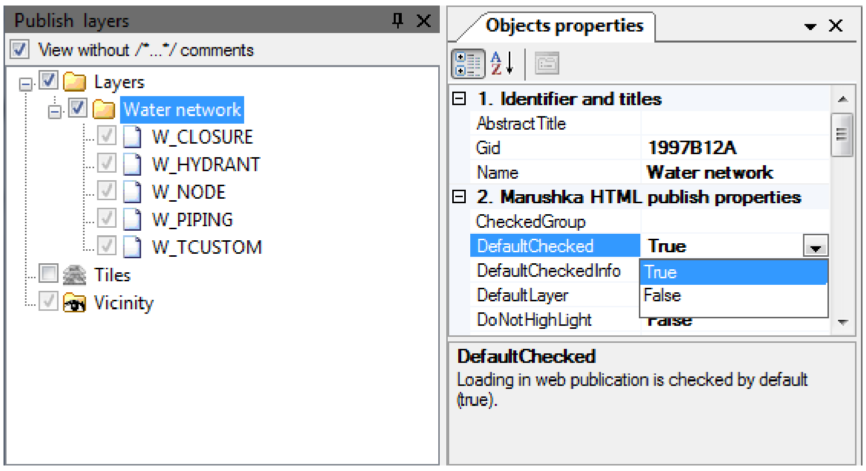
The first step to view your data in a web environment is creation of publish layer. Open the window of publish layers (if you already don't have) from tools panel *Layers and map mapsurfaces*. To create new publish layer invoke the context menu on the *Layers* item and select and choose *Add new publish layer*. For simplicity name the layer ***Water network***:



Publish layer is a layer that will be displayed in a web publication and may contain one or more form layers. For each layer you can define its scale range, allowing you within one publish layer to display different data at different scales (it is unnecessary to display e.g. fictitious cell nodes in scales lower than 1: 1 000 etc.). Move all the form layers from database data store SQLite progressively (or by using multiselect) to publish layer Water network. Therefore it will ensure, that all the elements from this category will be in one publish layer Water network. The result will look like this:



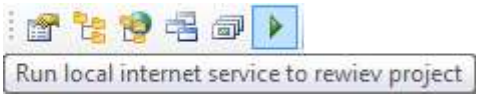
To see the result right after the local web server start, you have to change the property *DefaultChecked* of Publish layer *Water network* from value *false* to *true*. This will ensure that after start of a local web server, this layer will be displayed:



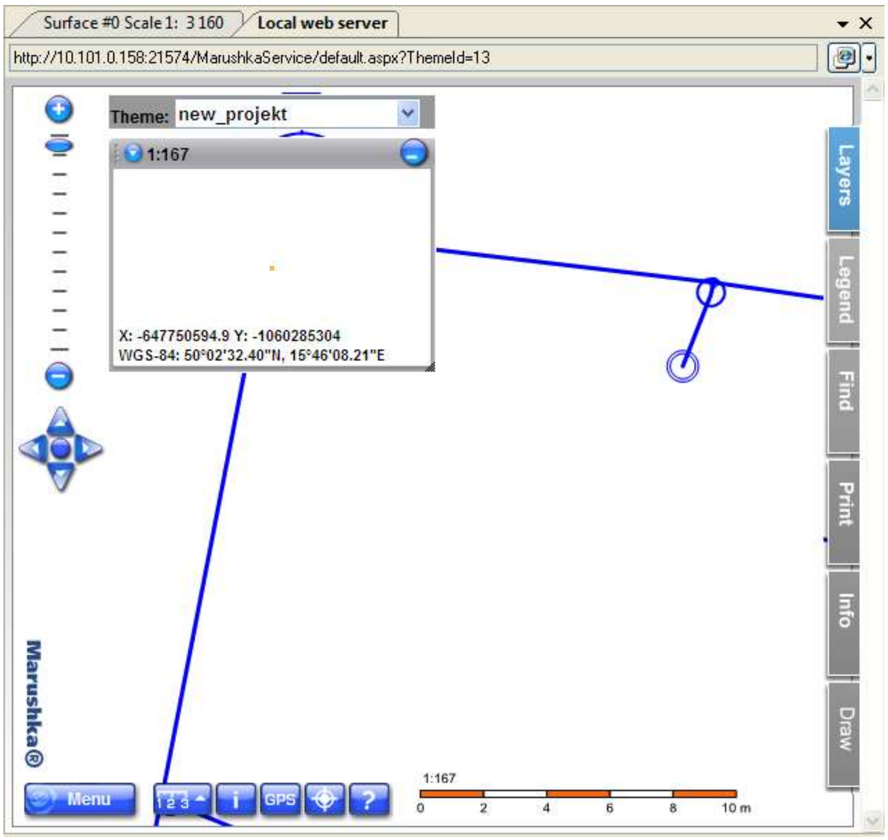
1. Identifier and titles	
AbstractTitle	
Gid	1997B12A
Name	Water network
2. Marushka HTML publish properties	
CheckedGroup	
DefaultChecked	True
DefaultCheckedInfo	True
DefaultLayer	False
DoNotHighLight	False

DefaultChecked
Loading in web publication is checked by default (true).

The project can be saved (but this time it will be saved automatically), and finally run a local WEB server from tools panel *Layers and map surfaces:*



After waiting a while in original position of the map window appears a tab to the window's internal local web browser in a position of cut-out of the original map window:


Surface #0 Scale 1: 3 160
Local web server
http://10.101.0.158:21574/MarushkaService/default.aspx?ThemeId=13
Theme: new_projekt
1:167
X: -647750594.9 Y: -1060285304
WGS-84: 50°02'32.40"N, 15°46'08.21"E
Layers
Legend
Find
Print
Info
Draw
Marushka®
Menu
GPS
1:167
0
2
4
6
8
10 m

The map composition is at the moment still quite austere and static, but at this point you can see our data in the form in which they can appear anywhere on the web, if you have properly configured web server. But configuring Web servers is just another chapter which is beyond this tutorial. Now let’s continue to focus exclusively on working in the local web environment. You can see our publish layer in tab layers, where it can be turned on or off. This is currently the only one functionality of our created project.

6 Changing the Symbology of Displayed Data - Completion

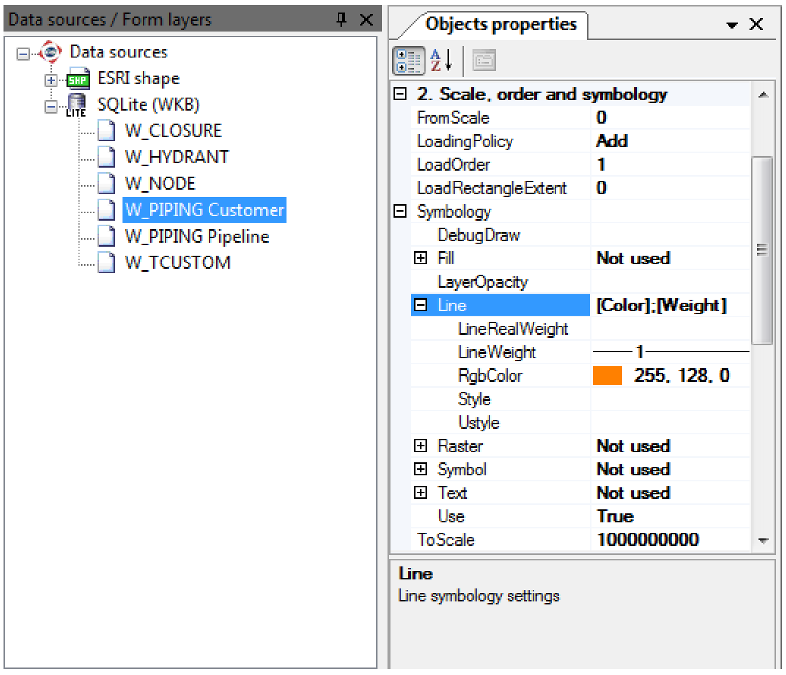
Now imagine the situation when someone comes and tells that he dislikes **cells of closures**, which confuses the WEB publication and he wants to have **orange cell connections** and **light blue connection terminations**. Even before moving on to dealing with these requirements, you could have noticed hydrant cells that you would like to see in an angle in which they were in the original data store.

Before that you might like to change the angle of rotation of hydrants. In database is stored information about the angle in column *angle*. We will use this information and we will use the information stored in it by adding it to the pseudo columns of form layer *W_HYDRANT*. But the native angular units are radians, so the type conversion is necessary. The conversion is saying that the value stored in the columns is an angle in degrees. String stored in the properties of *W_HYDRANT* layer in *DBColumnsToClient* will look like this: “`'0 0 7' SET_PARS_POINT_FROM_CORG, '1' SET_PARS_CELLNAME, '(DEG)'||angle SET_PARS_ROTANGLE.`“ You can immediately see the change by downloading new data.

Try other changes in order of complexity. Let’s start with removing cell closures. You will achieve this by removing (right mouse button) layer *W_CLOSURE* in publish layer Water network. From now on this layer will not display in the Local WEB server.

Recoloring the connection termination is also an easy step and you will again use pseudo columns with which Marushka can work, this time it will be `SET_PARS_REPLACE_COLOR` that replaces one color with another. To form layer properties *DBColumnsToClient* of *W_TCUSTOM* layer add a string: `'C 255 0 0 255 255 0 128 192' SET_PARS_REPLACE_COLOR`, which will ensure replacement of original blue color for the new light blue.

The last requirement was to recolor connections to orange color. Connection lines and connections are located in the same form layer, but you can use the database attribute RC and display those two groups (lines and connections) separately. First **clone** the form layer (context menu, right click the layer *W_PIPING – Form Layer – Clone*). The result will be a new form layer with attribute `SymbName=copy`. For better orientation, change *SymbName* of new layer to "Customer" and in the original layer *W_PIPING* enter into *SymbName* property "Pipeline". In both layers you have to define database condition, so that you show only the data for the pipelines in one layer and data for connections in the second layer. In clone for lines fill in the *DBWhereClause* property with "`rc=Water pipeline`"'. In clone for connection fill in the same property like this: "`rc = 'Customer’s connection.`"' Now you can change the symbology of this form layer clone according to the defined requirements.



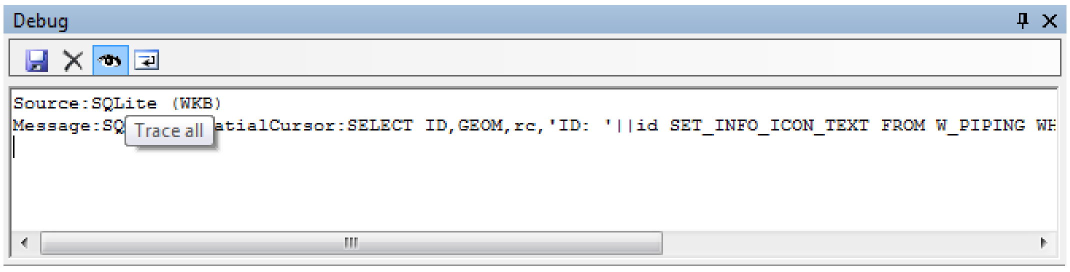
- Data sources
  - ESRI shape
  - SQLite (WKB)
    - W_CLOSURE
    - W_HYDRANT
    - W_NODE
    - W_PIPING Customer
    - W_PIPING Pipeline
    - W_TCUSTOM

2. Scale, order and symbology	
FromScale	0
LoadingPolicy	Add
LoadOrder	1
LoadRectangleExtent	0
Symbology	
DebugDraw	
Fill	Not used
LayerOpacity	
Line	[Color];[Weight]
LineRealWeight	
LineWeight	1
RgbColor	🟧 255, 128, 0
Style	
Ustyle	
Raster	Not used
Symbol	Not used
Text	Not used
Use	True
ToScale	1000000000

Line
Line symbology settings

6.1 Debug Console

In the previous step were used database conditions. To verify the correctness of database commands can be used **Debug console**, which allows "capturing" all the commands that run in the MarushkaDesign background. It can be found in the *Tools menu – Debug console*. Its base is a dialog box in which you can continuously display the results of individual queries. Turning on debug console is recommended only at the stage of query testing; otherwise it unnecessarily reduces performance and speed of MarushkaDesign. Turning on this function is showed in the picture below.



```
Source:SQLite (WKB)
Message:SQ...atialCursor:SELECT ID,GEOM,rc,'ID: '||id SET_INFO_ICON_TEXT FROM W_PIPING WH
```

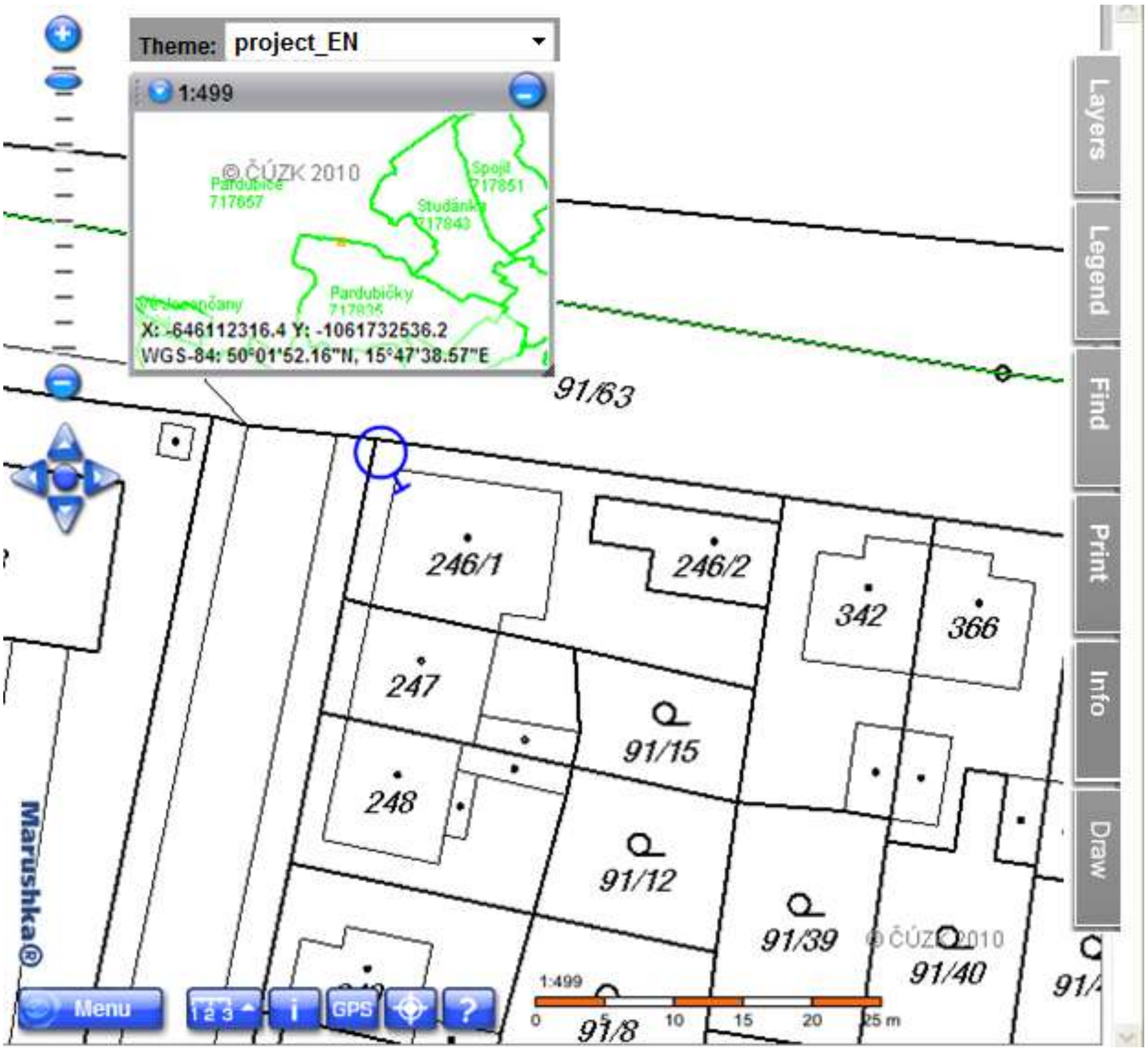
As a result, you can capture e.g. database select, the correctness of which you can then try in SQL manager, and thus you can reveal type errors like typos. Debug console can therefore be an appropriate helper when any problem with displaying data from different (not only database) data stores occurs.

6.2 Last Modifications in the Basic Display of Data

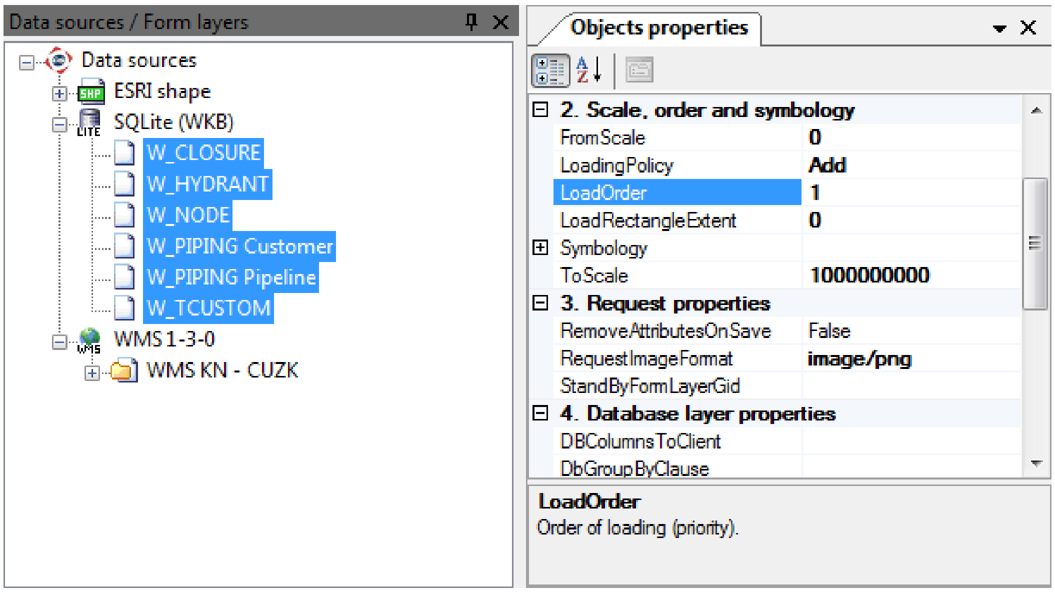
The last piece, still missing in the publication of our project is displaying the map which could clarify the course of water distribution network. For this purpose use the publicly available and free wms service of the Czech Office for Surveying, Mapping and Cadaster. Join it to the project as a partial prepared subproject.

In the data sources in the context menu choose the *Project – Add* and from folder `C:\MarushkaExamples\Tutorial\Files` \ choose file `NewWMS.xml`. You can use the guide to show an overview of data sources, form and publish layers that are available in this project. In your case do not make any changes; but in other cases you can choose which part of the subproject you want to use in your main project. After successfully importing into the project, a new WMS data source with one form layer will be added. This layer contains all the sub-layers necessary to display the cadastral map. Also in the publish layers is added one new publish layer, in properties of which is set *DefaultChecked* to *true* so that cadastral map display at startup of WEB server.

Drag the form layer *WMS KN - CUZK* from WMS to category *Vicinity* in publish layers. At this moment the data provided by the WMS service will be displayed also in the small overview appearing in the left upper corner of the WEB server. You can look at the result of your efforts again in the local WEB server. If you followed the instructions properly, you should probably get a result similar to the situation in the next picture:



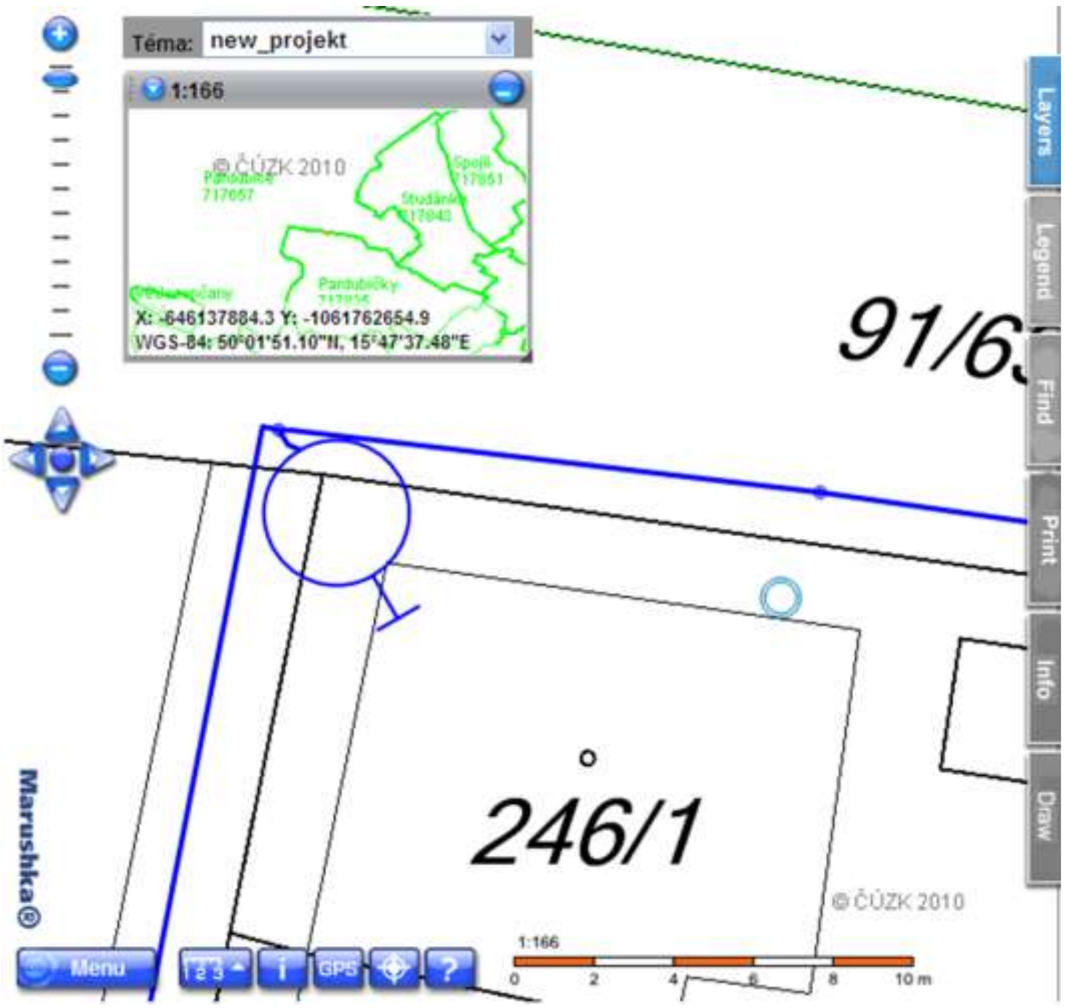
Surprisingly you found out, that our water distribution network disappeared. What is the cause of that? The only one visible item in the picture is a hydrant cell and even that is only a random phenomenon. In MarushkaDesign is defined also an order of layers in which they will be rendered. You haven't changed this value, so all the layers have this value set to "0" (in the form layer properties, it is an attribute *LoadOrder*) and all our layers are rendered over each other in a random order. Now you have to adjust this value to correctly displaying the data. In our case it will be the sufficient solution to set for all the layers from data source SQLite value of `LoadOrder=1`. In more complex map projects is available a *Map composition explorer* in *Tools* menu, that allows transparent administration of the order of the displayed layers.



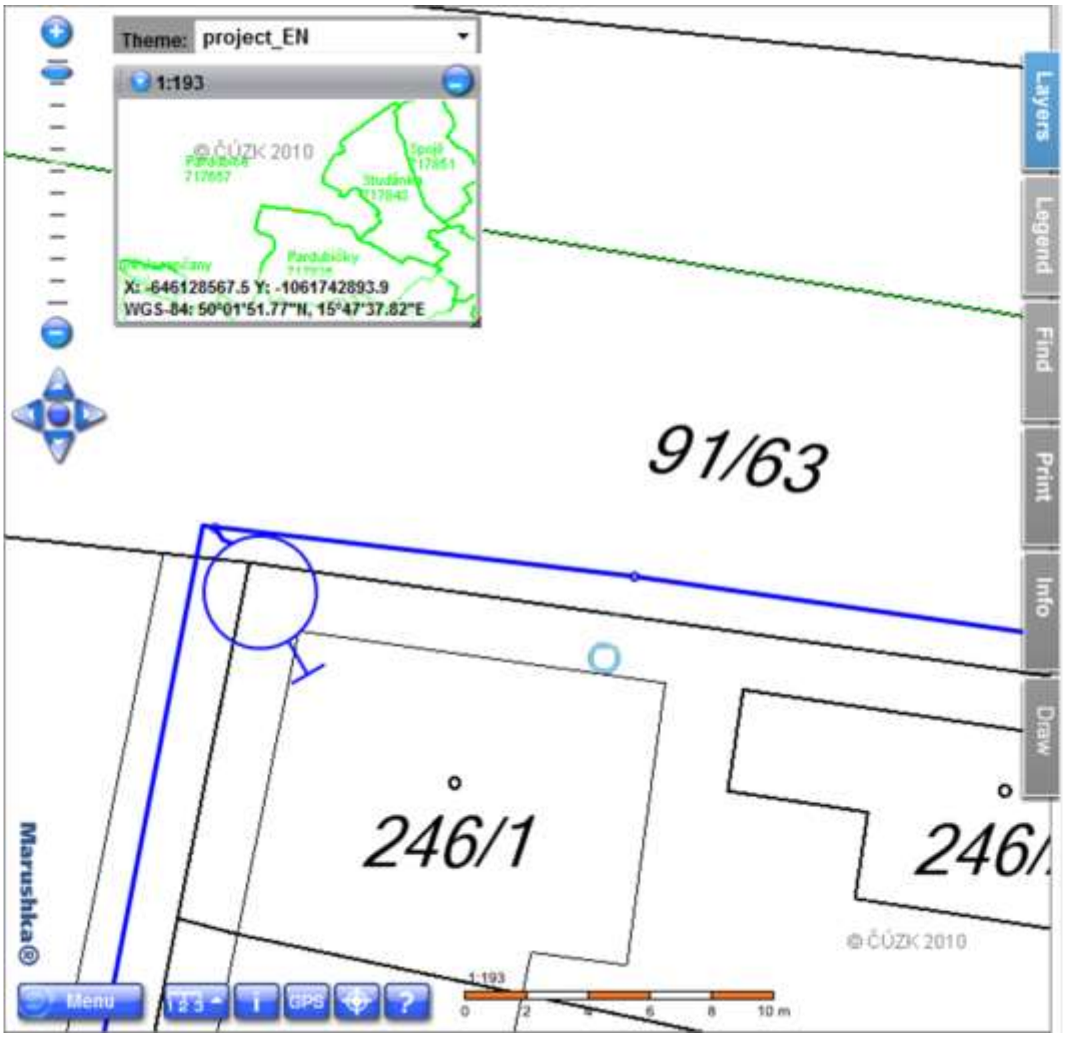
2. Scale, order and symbology	
FromScale	0
LoadingPolicy	Add
LoadOrder	1
LoadRectangleExtent	0
Symbology	
ToScale	1000000000
3. Request properties	
RemoveAttributesOnSave	False
RequestImageFormat	image/png
StandByFormLayerGid	
4. Database layer properties	
DBColumnsToClient	
DbGroupByClause	

LoadOrder
Order of loading (priority).

After this change launch the local WEB server again:



The outcome of our efforts in now clearly visible and you can see our water supply network, except for one small thing. In WEB server publication you do not see the connecting lines of connections. How did that happen? When cloning a form layers, the clone did not automatically add it also into publish layers and so a new form layer cannot be displayed. Therefore return to the project and from the list of form layers "drag" layer *W_PIPING Customer* into Publish layer *Water network*. Another restart of the local WEB server gives you finally the data in the form in which you might have wanted it to see:



By live testing of the functionality of the project were revealed some shortcomings and mistakes which were committed in the configuration of the project and you can see that the auxiliary map window and the window of local WEB server are in our work indispensable helpers. At this point you have explored the basic functions of Marushka for simple displaying your vector data in combination with raster basis from WMS source. The possibilities are practically unlimited – you can have any number of publish layers, supported formats, which you can use in your publication. For a performance of basic functions, it is sufficient at the moment and now you can start to teach Marushka more advanced activities.

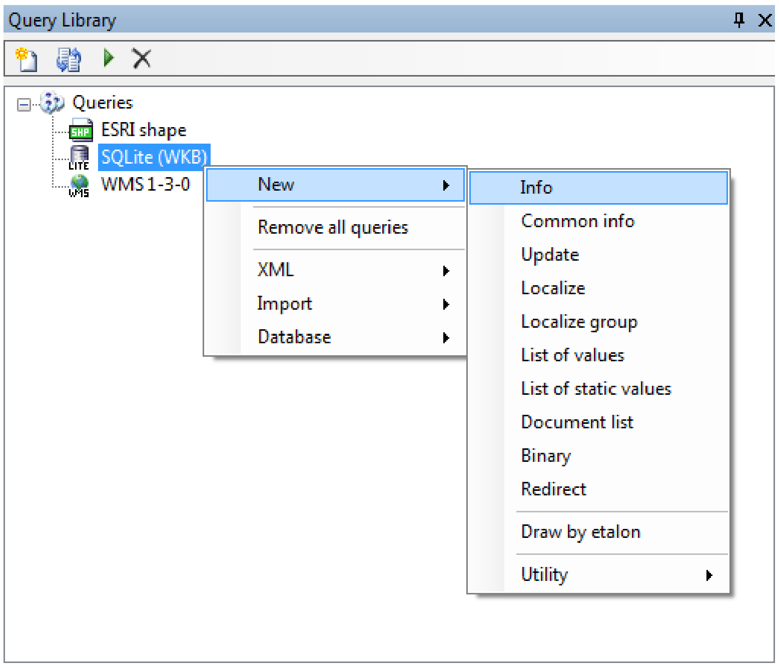
7 Interactive Marushka

Until now you have worked with a somewhat static map – you were just displaying our data. At this moment you could still have the feeling that Marushka displays only picture of our water supply network. You will try to change this feeling now. You will teach Marushka to display also database information that is not a direct part of the graphical presentation of data and is stored in the database for each element. The basic carrier category of database information consists of information queries.

Let's begin with a linear route object information query. Let's display the following information about the element: *ID* (database element identifier), *rc* (closer description specifying the element), *date_of_building* (date type attribute – not filled for elements – only for demonstration work with date), *dim_cust* (pipeline dimension – you will fill in the values later), *commission* (order number – only for our testing).

7.1 Creating First Information Query

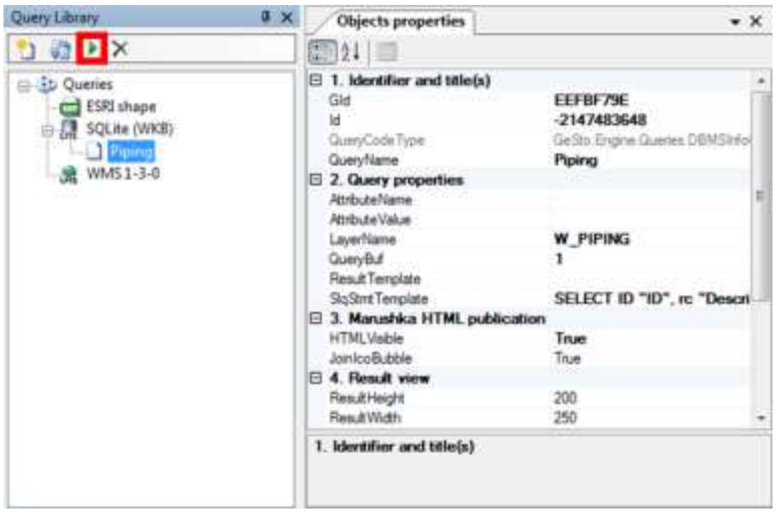
Open the query library (from menu *Tools – Libraries – Queries).* Each generated query is bound to a specific data source, so create a new **information query** in the data source SQLite:



You can give the query any name you want. Into the property *QueryName* of this query enter e.g. "Piping" – this name of the query will be visible even in the web publication. Into property *LayerName* enter the name of the physical layer, in which will be the query feasible (in our case database table *W_PIPING*). For database data sources, you must be careful that the value contained in this field does not have to match with the name of form layer (particularly when using database views that combine several database tables together). The final perquisite for the proper functioning of our first information query is *SqlStmtTemplate* property, into which you type the actual database select, which will return specific information to a given element. In our case the select will look like this: `SELECT ID "ID", rc "Description", date_of_building "Date of building", dim_cust "Dimension of pipeline", commission "Number of commission" FROM W_PIPING WHERE ID=~(long)ID~`

This simple select will return values defined in the columns, using the tilde syntax, which appears in the where clause and it interconnects condition with element that was selected in the graphic. Type casting **(long)** is used for security reasons as prevention against possible "outside attacks" in the form of so-called **SQL injection.**

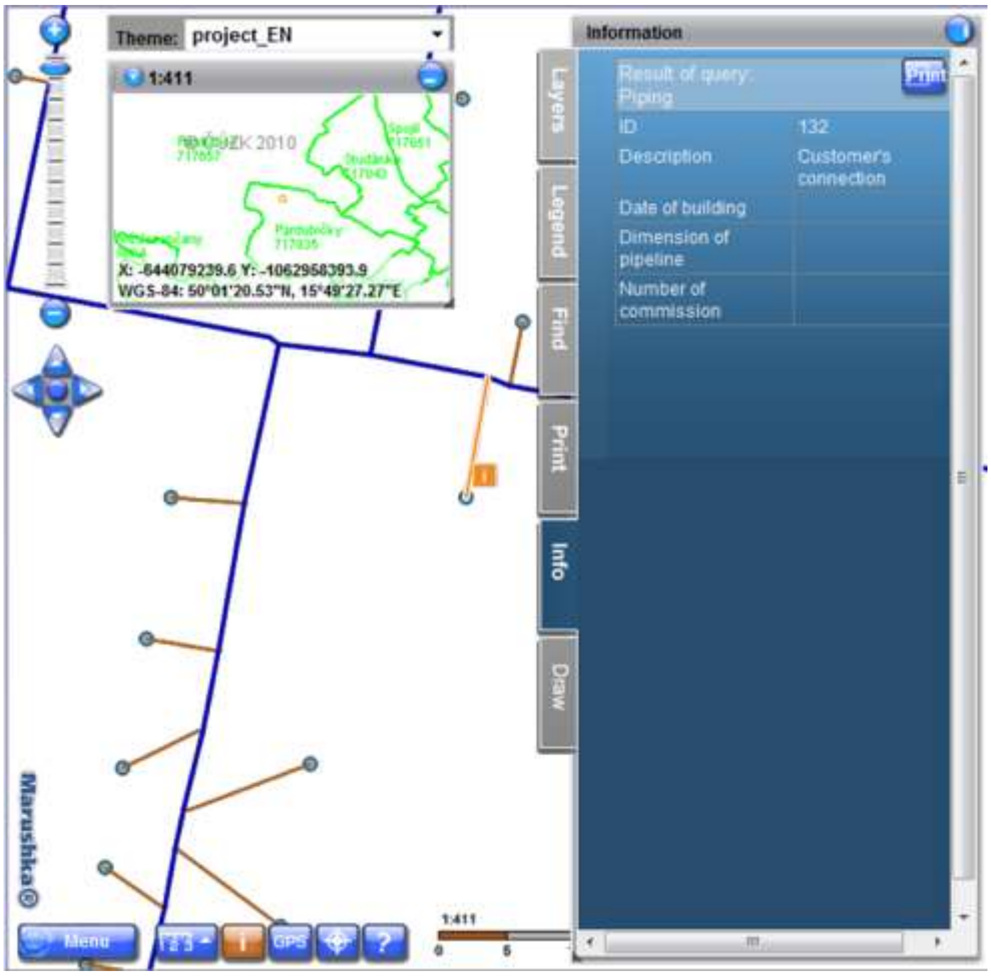
You can test the functionality of a particular query by clicking the icon for testing queries (highlighted in the picture). In the right part of the picture you can see a part of the dialog box that you have just changed.



1. Identifier and title(s)	
GId	EEFBF79E
Id	-2147483648
QueryCodeType	GeSto.Engine.Queries.DBMSInfo
QueryName	Piping
2. Query properties	
AttributeName	
AttributeValue	
LayerName	W_PIPING
QueryBuf	1
ResultTemplate	
SqStmtTemplate	SELECT ID "ID", rc "Descri
3. Marushka HTML publication	
HTMLVisible	True
JoinIcoBubble	True
4. Result view	
ResultHeight	200
ResultWidth	250

1. Identifier and title(s)

Now let’s try your first information query also in the local WEB server. In the local WEB server switch into Info mode by the button at the bottom of the server window and by clicking the left mouse button on any element of the route activate the "*preselect*" function, which will highlight the closest element to the path. By right click you can switch to other elements. While the *Preselect* function is active, next to the highlighted picture occurs an Info icon. Its content can vary and instead of it any raster picture may be displayed. But in this tutorial we do not need to bother with such a detail. After clicking on information icon will in the right side of the local WEB server appear result of an information query, which is linked with a highlighted element. In case one of the physical layers is defined by multiple queries, after first click on the information icon it will display the content, from which you select the required query.



X: -644079239.6 Y: -1062958393.9
WGS-84: 50°01'20.53"N, 15°49'27.27"E

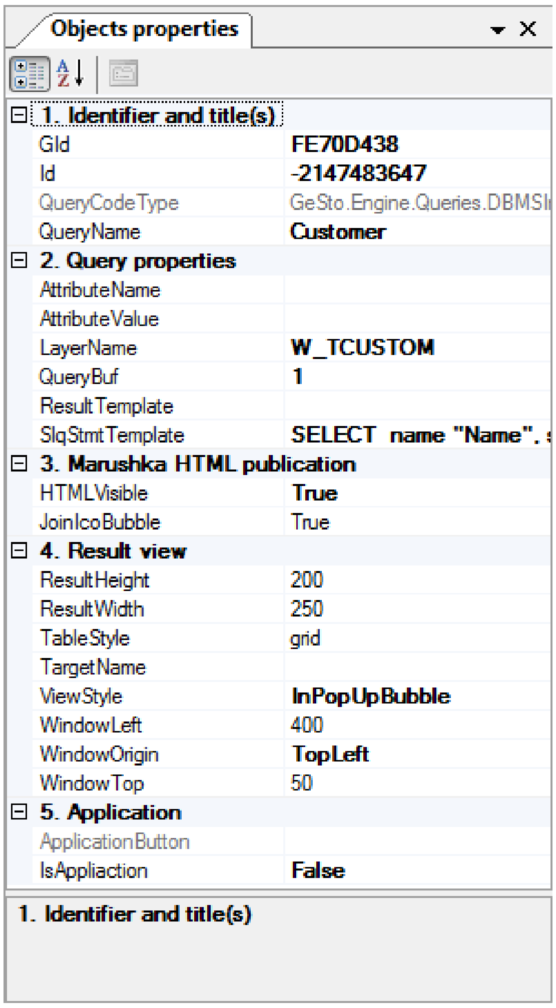
Result of query: Piping

ID	132
Description	Customer's connection
Date of building	
Dimension of pipeline	
Number of commission	

To display the text that appears in the information icon (instead of the standard text "i") you may display different text. Do it using another pseudo column, the value of which is defined in the corresponding form layer. Into *DBColumnsToClient* property of layer *W_PIPING Pipeline* and *W_PIPING Customer* enter a string: `'ID: '||id SET_INFO_ICON_TEXT`. That will ensure that instead of description *W_PIPING*, will be displayed ID of the corresponding element in the floating help.

7.2 Information Query at a Cell

Let’s use a bit different approach for non-linear elements. Create an information query for connection terminations. Let's start again by creating an information query using the same procedure, you chose in the previous case. The result will be the query *Customer* with the following properties:



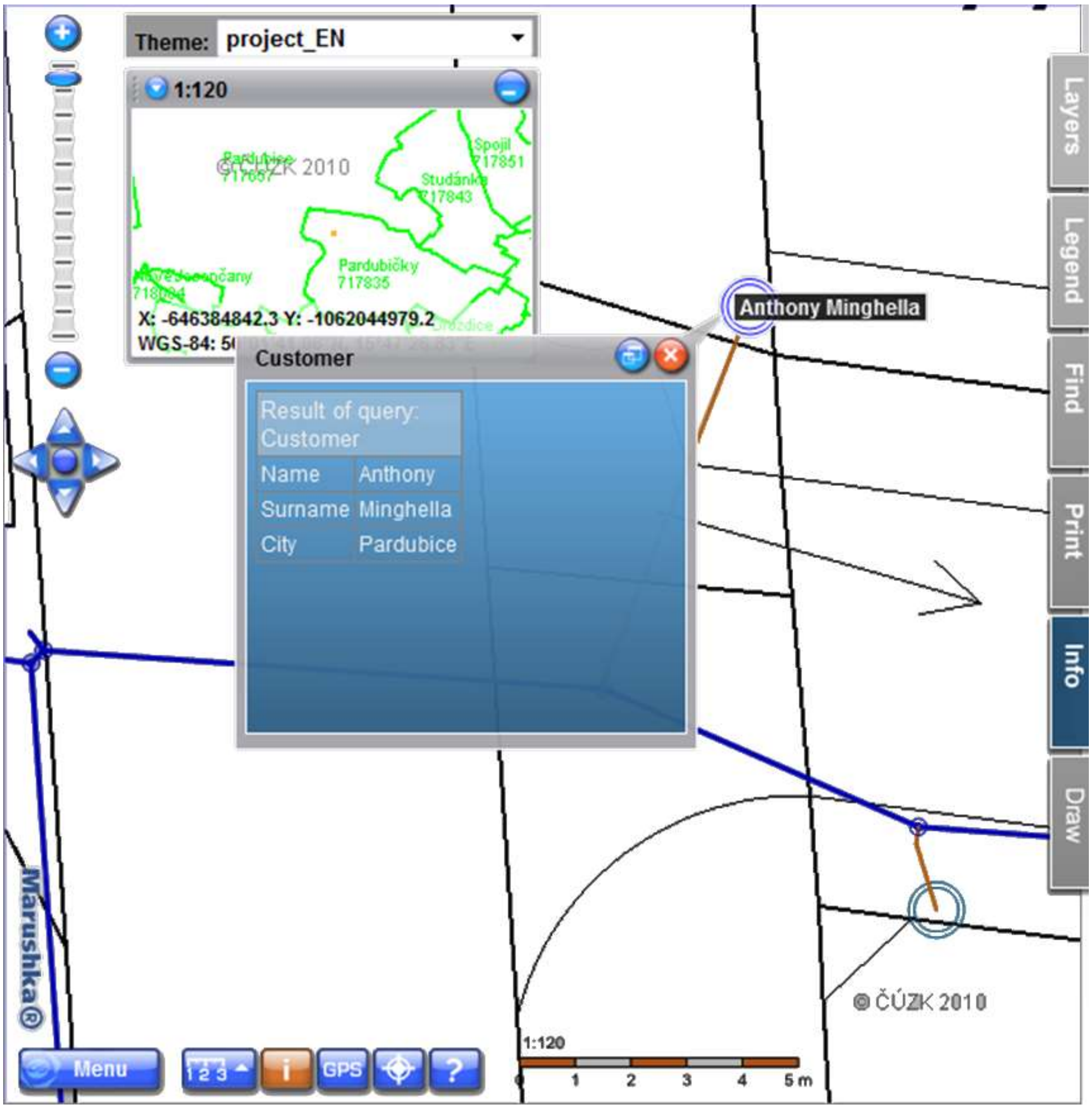
1. Identifier and title(s)	
GId	FE70D438
Id	-2147483647
QueryCodeType	GeSto.Engine.Queries.DBMSI
QueryName	Customer
2. Query properties	
AttributeName	
AttributeValue	
LayerName	W_TCUSTOM
QueryBuf	1
ResultTemplate	
SlqStmtTemplate	SELECT name "Name",
3. Marushka HTML publication	
HTMLVisible	True
JoinIcoBubble	True
4. Result view	
ResultHeight	200
ResultWidth	250
TableStyle	grid
TargetName	
ViewStyle	InPopUpBubble
WindowLeft	400
WindowOrigin	TopLeft
WindowTop	50
5. Application	
ApplicationButton	
IsAppliaction	False

1. Identifier and title(s)

In this example we will have a look at options of displaying the information query results. You can choose to keep the attribute in the query properties `ViewStyle=InPanel` (the result will be displayed on the information tab to the right of the WEB server window), *InPopUpBubble* (the result will be displayed in a bubble above the corresponding element) or *InNewWindow* (the result will be displayed in a new browser window). In our case we chose the middle option *InPopUpBubble* and the information query result will be displayed in "pop-up bubble" above the element.

In addition change the text displayed in the icon using the well-known pseudo column *SET_INFO_ICON_TEXT*. In the information icon should be displayed the name and surname of the customer. Do this by concatenating of two columns and the definition of the pseudo column will look like: `ifnull(name,'')||' '||ifnull(surname,'') SET_INFO_ICON_TEXT`. The function "*ifnull*" is in this case in the database environment almost a necessity, because the chain cannot handle empty values. Further you can almost in the same way add the pseudo column *SET_INFO_ICON_LABEL*, which displays the floating help above the corresponding element. The last important pseudo column suitable for cells is ***SET_INFO_ICON_COVER***. This pseudo column displays transparent icons above the vector icon and thus can create an active element from a cell. In the property *DBColumnsToClient* in form layer *W_TCUSTOM* enter the following string (the order of columns doesn't matter): `'true' SET_INFO_ICON_COVER, 'Customer: '|| ifnull(name,'')||' '||ifnull(surname,'') SET_INFO_ICON_LABEL, ifnull(name,'')||' '||ifnull(surname,'') SET_INFO_ICON_TEXT, '0 0 7' SET_PARS_POINT_FROM_CORG, '2' SET_PARS_CELLNAME, 'C 255 0 0 255 255 0 128 192' SET_PARS_REPLACE_COLOR.`

Given that the value 'true' in pseudo column *SET_INFO_ICON_COVER* display information icons (although invisible), so you will display them. Displaying information icons is the property of the form layer. So stay in the properties of the layer *W_TCUSTOM* and set property *GenerateInfo* to *true*. In the publish layer properties change the property *DefaultCheckedInfo* to *true*, so that these information icons display just after starting the local WEB server. The result can be seen in the local web server by the left mouse click on connection termination, which will display the following information:



Customer

Result of query: Customer	
Name	Anthony
Surname	Minghella
City	Pardubice

8 Marushka Adjusting the Data

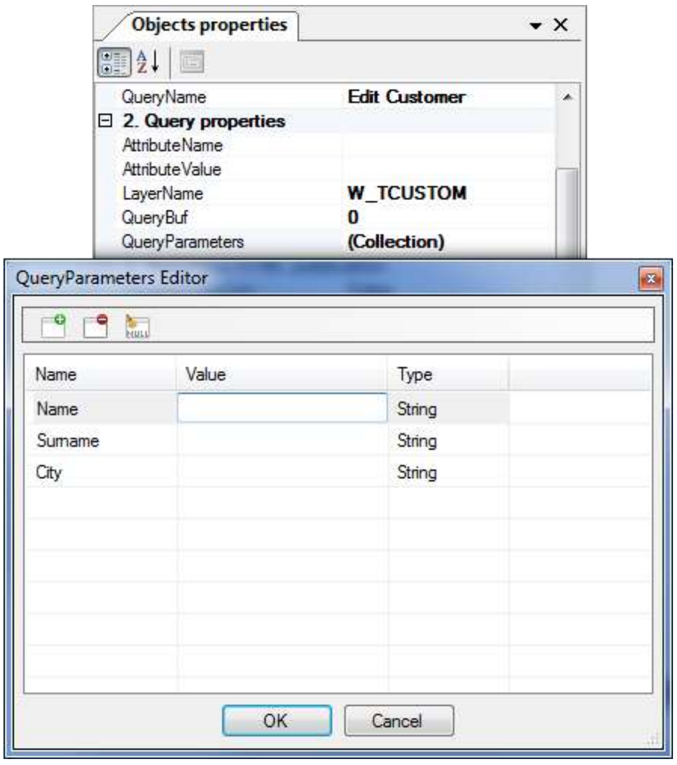
You taught Marushka to display graphical data, database information to graphical data, and now you will teach her to edit database data. Let's create a pair of editing queries. In the first case, it will be a simple text editing of customer data.

8.1 Customer Editing

In the query library using the context menu on the data source SQLite create a new query *Update* type. Name it for example *Edit customer*. In *LayerName* property must be again given a physical layer name (table) of data source – in our case it is *W_TCUSTOM*. Leave other properties unchanged. Stop at the pair of properties *InitSqlStmtTemplate* and *UpdSqlStmtTemplate*. The first of them is the select of the columns, which you want to display in the edit query. Display and consequently edit these attributes: *Name, Surname* and *City*. For this select are applied specific rules of notation. Columns that will be subsequently edited by editing query must be named by number and this number must be in an ascending order. The quantity of editable columns must match the number of parameters defined in the edit query (property *QueryParameters*). Columns, that won't have indexed numerical naming, will be considered read-only columns and their values cannot be changed. Our select will look as follows: `SELECT id "ID", name "1", surname "2", city "3" from W_TCUSTOM where id=~(long)ID~.`

The second SQL query is a phrase for the update of database data. Individual data are updated according to order of the parameter, which must be the same as order in the previous select. In this case, it is again important type casting of the parameters as a protection against SQL injections. The construction of our update will look like this: `UPDATE W_TCUSTOM set name=~(string)1~, surname=~(string)2~, city=~(string)3~ where id=~(long)ID~.`

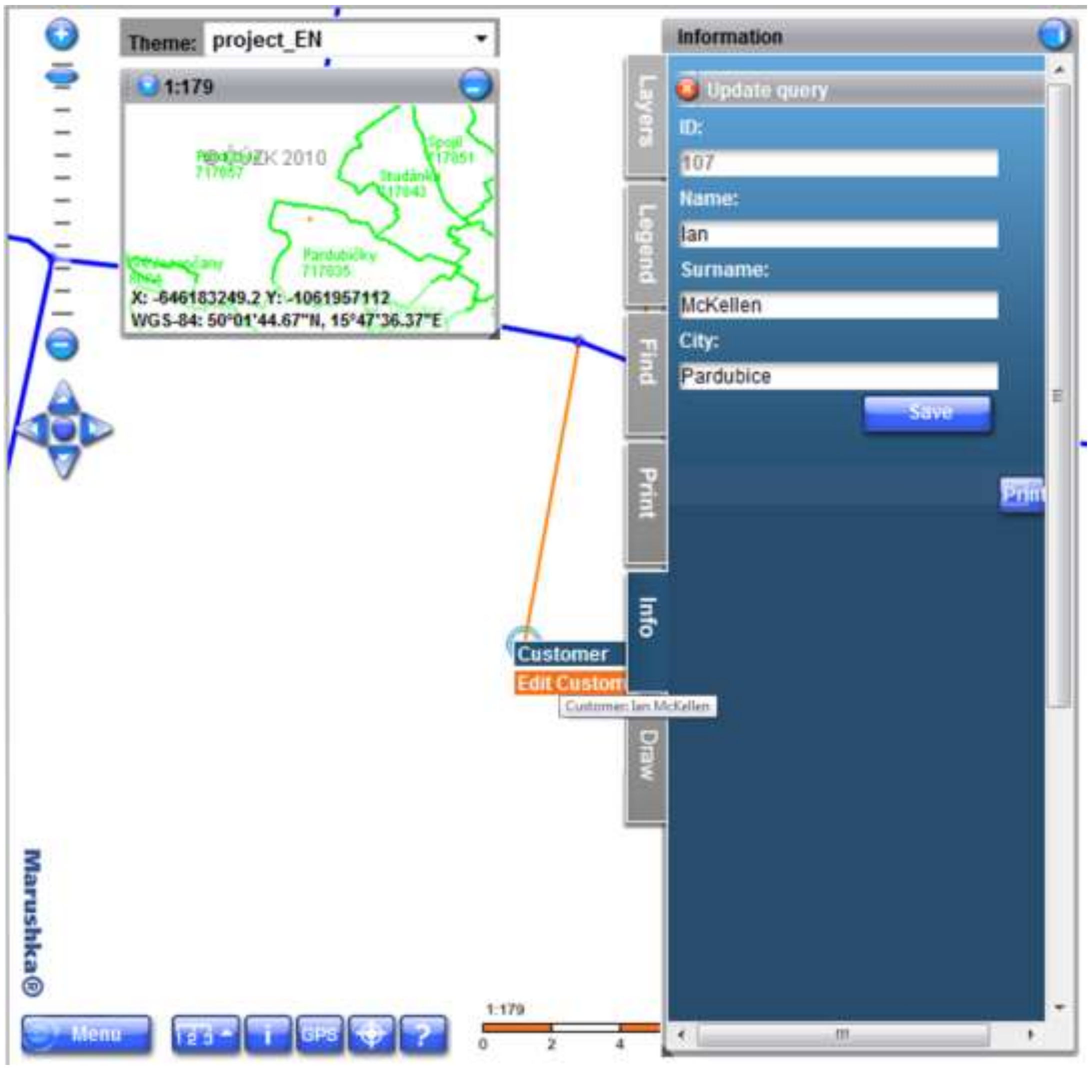
You must not forget these mentioned parameters, which you have yet to define in the *QueryParameters* properties. Order of the parameters must match the order of columns (parameters) defined in the relevant select. Their names will be displayed in editable boxes in a WEB publication. Editing of parameters is intuitive in an opened dialog box and there is no need for long descriptions. Adding a new parameter is performed by the first icon, editing the name of parameters by mouse double click. Definition of parameters will in this case look like this:



QueryName	Edit Customer
2. Query properties	
AttributeName	
AttributeValue	
LayerName	W_TCUSTOM
QueryBuf	0
QueryParameters	(Collection)

Name	Value	Type
Name		String
Surname		String
City		String

In the WEB publication (in our local WEB server) you can see the result of your effort. At the connection termination you have a possibility of two queries, which are sorted alphabetically. Also you can see the floating help, which you created in the last information query. After activation of edit query *Edit customer*, will in the right panel in the information tab appear the outcome of your created edit query. In the appropriate fields, you retrieve information about the selected element, of which only the **ID cannot be edited**. The other information can be edited and updated data can be stored in the database.



Update query
ID:
107
Name:
Ian
Surname:
McKellen
City:
Pardubice
Save

8.2 Editing Connection (line)

On the route of connection will be demonstrated another option of editing queries. You will learn to edit data using static code list and you will be shown how to work with date type data. You will get to know how to use this query only for connections and not the pipelines, when all the elements of the route are in the same physical layer.

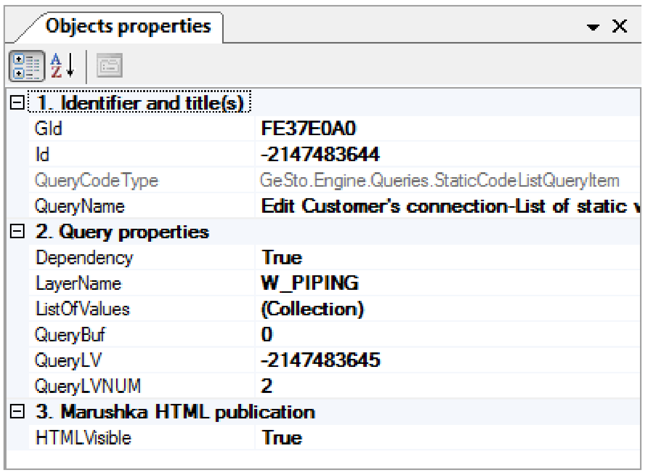
Let's start with creating another editing query as in the previous case. Name the query "*Edit Customer's connection*" and it will be connected to the physical layer *W_PIPING*. We want to edit *Date of building, Dimension of connection* and *Commission number.* Moreover we will display the *ID* and *rc* attributes. The resulting basic select for our editing query will have following form: `SELECT id "ID", rc "Description", date_of_building "1", dim_cust "2", commission "3" from W_PIPING where id=~(long)ID~`. Enter all the editable parameters directly into *QueryParameters*, *Date of building* will be the *DateTime* type, *Dimension of connection* and *Commission number* wil be *Int* type.

Return to the update phase and enter the following command here: `UPDATE W_PIPING set date_of_building=~(datetime)1~, dim_cust=~(int)2~, commission=~(int)3~`

`where id=~(long)ID~`. In this example you can see that you used other data types then you have used so far. The query prepared like this would be fully functional and editing would have been successful. That, moreover, you can try out either by testing the query in Query library or in the local WEB server. But in this moment, let's finish the editing query.

8.2.1 Creating a Static Code List

We wanted to make our editing query work with static code list. In case of dimension you must take into account, that it can have dimensions 0 (in case of unknown dimension), 32 or 40. Create an auxiliary query that will work with our editing query. In the context menu choose a *New query – List of static values*. For better orientation in the library of queries name this query e.g. *Edit Customer's connection – List of static values.* In the properties of the query enter the name of the physical layer to which the query relates *(W_PIPING)*, then define a list of static values (*ListOfValues)*, which is filled with values: 0, 32 and 40 (as a separator of individual values use ENTER key). Finally, we are interested in property *QueryLV* – here fill the parent query ID, which you copy from *Edit customer's connection* query properties and the last is the order of the parameter in parent query (in our case it is 2, because the parameter for connection dimension is in the parent query second in the query sequence. The resulting form of properties of static values would look like this (only ID in QueryLV will vary depending on the parent query ID):



1. Identifier and title(s)	
GId	FE37E0A0
Id	-2147483644
QueryCodeType	GeSto.Engine.Queries.StaticCodeListQueryItem
QueryName	Edit Customer's connection-List of static v
2. Query properties	
Dependency	True
LayerName	W_PIPING
ListOfValues	(Collection)
QueryBuf	0
QueryLV	-2147483645
QueryLVNUM	2
3. Marushka HTML publication	
HTMLVisible	True

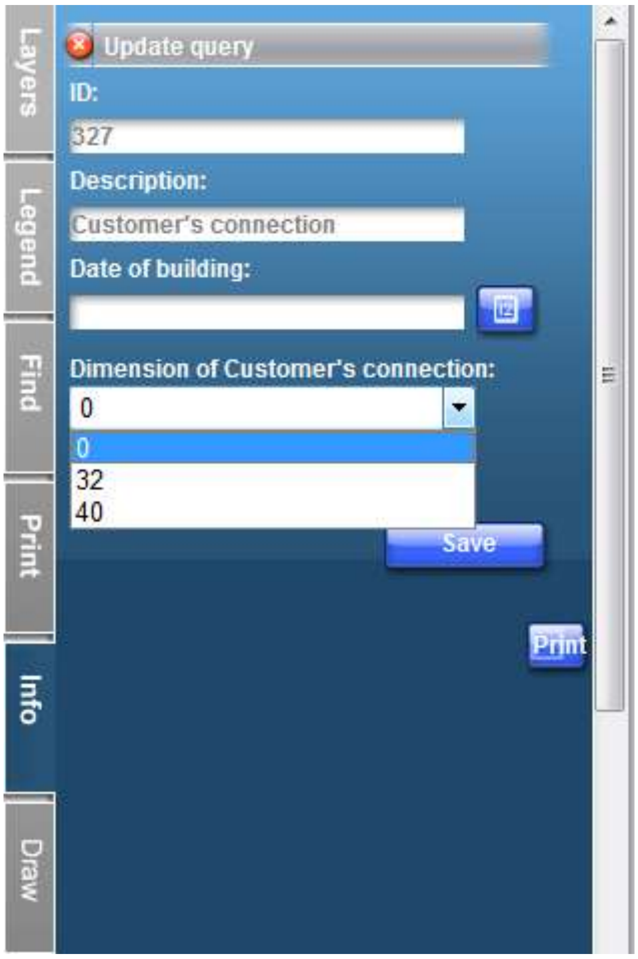
8.2.2 Limitations of the Editing Query to a Subset of Elements

The query is now displaying, according to our expectations (after trying it in the local WEB server), but we still want to limit the query only to connections. Use the rc attribute, which further specifies the element. Enter it into property *DBColumnsToClient* in form layer *W_PIPING Customer*, so that you can continue working with it.

Go back to Query library to our query "*Edit Customer's connection*". Into property *AttributeName* enter the string*: rc.* Into property *AttributeValue* enter value "*Customer's connection* ". This condition will ensure starting the query only for the physical layer, which will also satisfy the attribute condition. The condition is not defined directly in the database environment, so it is necessary to use attribute, with which the condition works and mention it also in the attributes of the downloaded layer.

8.2.3 Testing the Resulting Editing Query

If we worked properly, so the editing query won't be at this moment available for the line routes and will be available only for connections. The picture shows the result of our efforts. In the preview you can see the possibility to edit the element with ID=327. ID is as well as the description "*Customer's connection* " non-editable. You also see a new active element to the right of the date text box. It is an icon of a calendar, by which user can select a particular date. In the *Dimension of Customer's connection* we see the possibilities that we have defined as static values for this particular query. **Try to fill in this value for several connections** (so that the values vary for different connections). You will need this data in some other steps of this project. In the preview, there is not visible the attribute *Number of commission*, which is not significant and it is a part of the project just for testing different data types.



ID:
327
Description:
Customer's connection
Date of building:

Dimension of Customer's connection:
0
0
32
40

9 Teaching Marushka to Draw

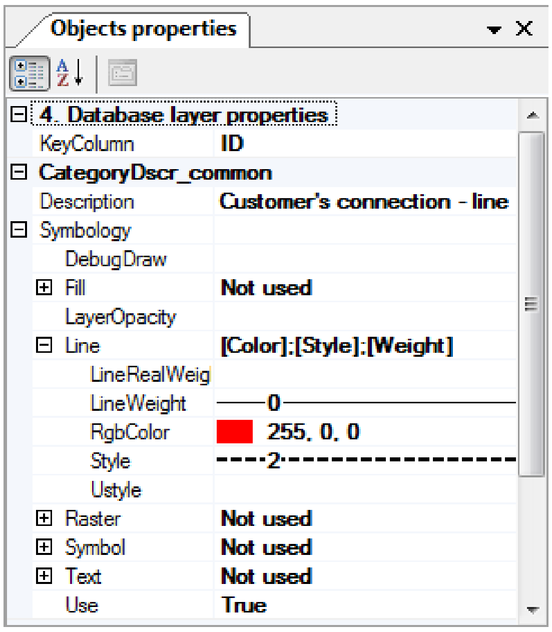
After you taught Marushka to update the original data, it is now time to teach Marushka to draw the new data. You will create the pair of drawing queries, by which you will be able to draw new data in a web environment, which will be then entered into our SQLite database.

9.1 Creating Draw Query – Line Object

The basis for the drawing, the outcome of which will be then stored in the database, is again in the Query library. You will use again the context menu in the query library on data source SQLite, where you choose an option *Draw by etalon.* Name the query: *Draw Customer's connection*. In the properties of the query change only two things. Basic select, allowing to set the attributes, which will enter the database together with a new element (these can also be user-defined attributes, which will be demonstrated in the second example). There is the attribute *"userdraw"* in some database tables, which will provide the differentiation of elements that were created in a web environment. In the corresponding select therefore set the value to 1 and we also fill in RC attribute that defines us it's a connection. The select will then look like this: `SELECT '1' "userdraw", 'Customer's connection' "rc".` The second thing you need to add in the query is an etalon item, which will be needed by the drawing query. Let's create it right now.

9.2 Creating an Etalon Item

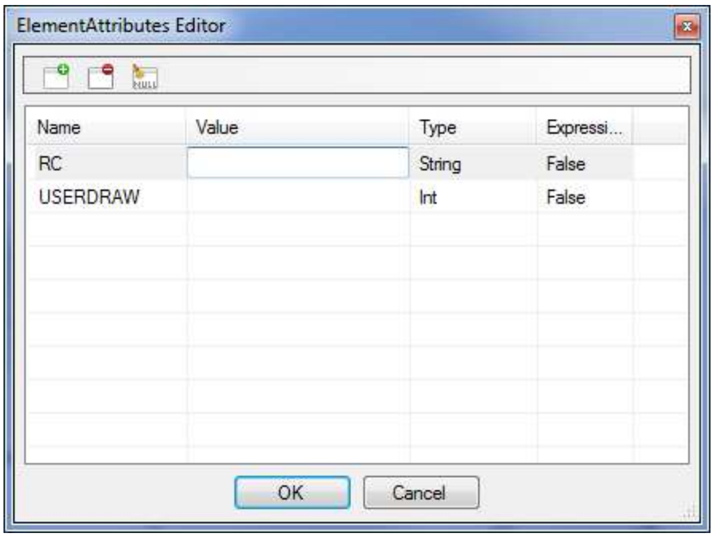
Etalon will in this case serve as a definition of symbology for a newly drawn element, including the definition of the database into which the element should be saved. Open the Etalon library (from menu *Tools – Libraries – Etalon*). In the context menu of the data source SQLite choose *Add Etalon item*. In the property *Description* name it *Customer's connection – line*; *TableName* will in this case be *W_PIPING*. We want to distinguish the connection depicted on the WEB server from the lines that are already in the data store. Choose a suitable user symbology in property *Symbology* as in the picture below:



⊟ 4. Database layer properties	
KeyColumn	ID
⊟ CategoryDscr_common	
Description	Customer's connection - line
⊟ Symbology	
DebugDraw	
⊞ Fill	Not used
LayerOpacity	
⊟ Line	[Color]:[Style]:[Weight]
LineRealWeig	
LineWeight	0
RgbColor	🟥 255, 0, 0
Style	2
Ustyle	
⊞ Raster	Not used
⊞ Symbol	Not used
⊞ Text	Not used
Use	True

Then we know that we want to draw the line element so into the property *GeomType* set *WKBLineString* item.

Finally, we still have to define the parameters in this etalon item that will be used by our drawing query. Define these attributes in the property *ElmAttribs* (see picture below):



Name	Value	Type	Expressi...
RC		String	False
USERDRAW		Int	False

The etalon item created like this can be written also into property *EtalonItem* of our drawing query.

9.3 Testing Draw Query and a Weakness Identification

We can see the result of our drawing query in the local WEB server environment. It can be found in the ***Draw*** card. After drawing the linear object and after clicking the *Save* icon we can see the result straight away. If you followed the procedure correctly, you will find out, that there is no difference between the connection that is already in the database and the connection you have just drawn – all the connections are orange. This is caused because of all the connections are in one form layer that is displayed in the user defined (orange) symbology.

9.3.1 Error Correction

To see the result we wanted, you will achieve by **cloning** the form layer W_PIPING *Connections*. Name the clone *W_PIPING Customer*. Add this new layer also into publish layer *Water network*. These two layers would at this moment return the same data, so we have to modify the condition to achieve the desired result. The form layer *W_PIPING Customer* will have *DBWhereClause* looking like this: *rc='Customer's connection' and userdraw is null*. We used attribute userdraw, which we found out in the database about, that it is not filled in the elements which were created outside Marushka environment. In the cloned layer *W_PIPING Customer userdraw* change *DBWhereClause to: rc='Customer's connection' and userdraw=1*, so that we ensure, that in this layer will be displayed only connections, that we draw in Marushka environment. Change the property Symbology so that this clone has a desired sense. In this property, switch off using user symbology. After further testing in the local WEB environment we will get the correct outcome and our connection will be displayed as it was stored in a data store (in our case a red dashed line).

9.4 Creating Draw Query with Attributes

By the exactly same way create a drawing query for *Connection termination*. The only difference, which is necessary to point out, is the type of the element in the Etalon item. Here choose the value *WKBPoint*. Otherwise, the procedure is the same as the previous example.

After creating a pair of *drawing query* and *etalon item* we go on with what needs to be developed and how is the new one drawing query different from the previous one.

9.4.1 Drawing query

The property *SqlStmtTemplate* will have included attributes, which user can fill in while acquisitioning the record – database item so just gets its properties. In our case, the select will include three attributes: *Name*, *Surname* and *City,* (all the string type). Therefore these must be defined in the *QueryParameters* properties. Given that it’s a cell, we must also specify the Cellname. The resulting select will look like this: `SELECT '1' "userdraw", 'Connection termination' "rc",`

```
~(string)1~ name, ~(string)2~ surname, ~(string)3~ city, 'W_TCUS2' CELLNAME.
```

The attribute *userdraw* again distinguishes data created in Marushka environment.

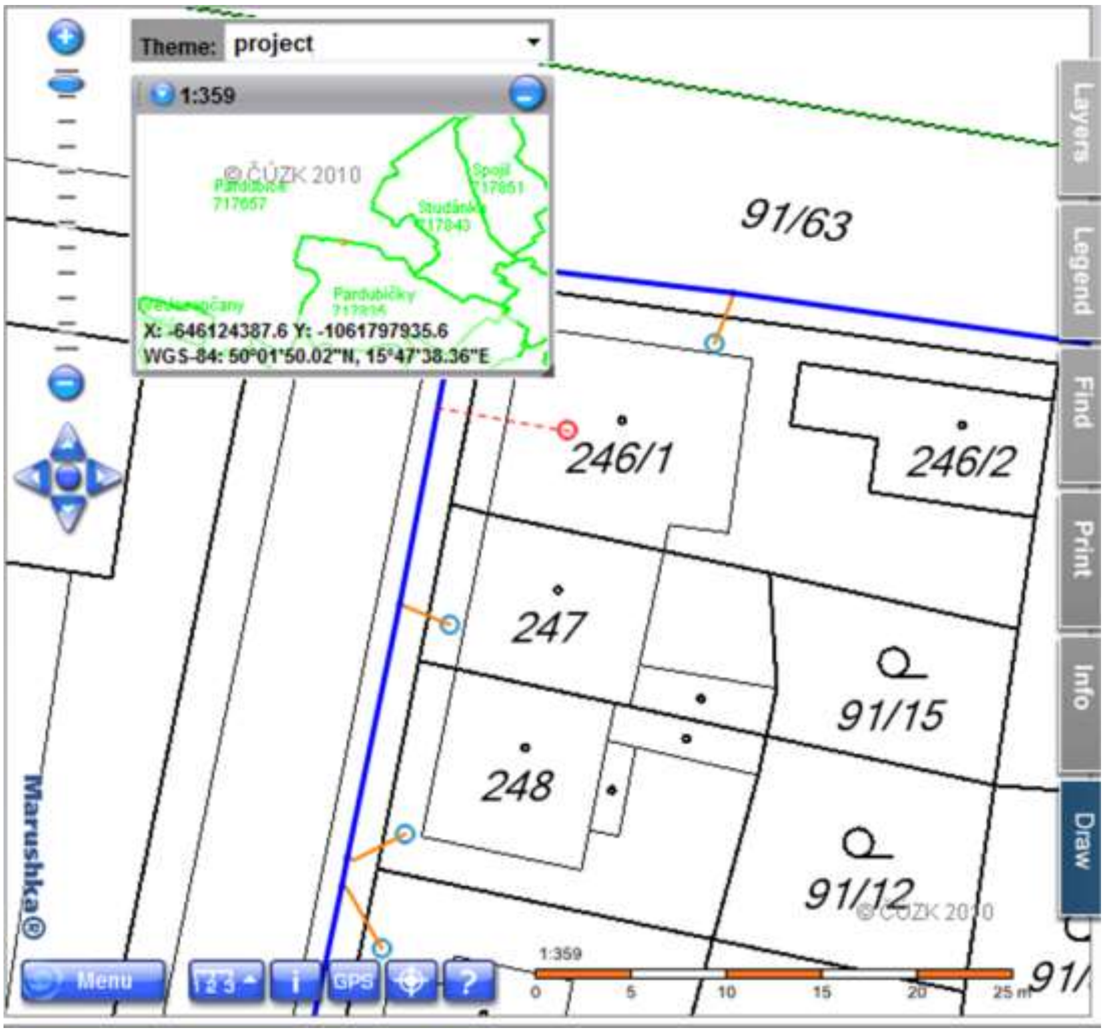
9.4.2 Etalon Item

In etalon item we must define all the parameters –enter both those in a drawing query, as well as those that will be filled in automatically when you insert the element into the database. In this case, the parameter names are named after the columns in the corresponding table.

9.4.3 Form Layer

In the form layer clone this time we won't modify the *Symbology* properties, instead in *DBColumnsToClient* remove two pseudo columns: *SET_PARS_CELLNAME* and *SET_PARS_REPLACE_COLOR,* because of this time we used the cell that is already displayed red in the cell library and there is no need to display it differently. Parts of the condition in the *DBWhereClause* property remains the same, as in the previous drawing query.

If you worked properly, the result of our efforts should look like this:



The picture shows orange database connections with light blue connection terminations – these are the connections that already were in the database and red dashed connection with red connection termination are the ones we entered into database in Marushka environment. And this was aim of this chapter.

10 Cunning Marushka

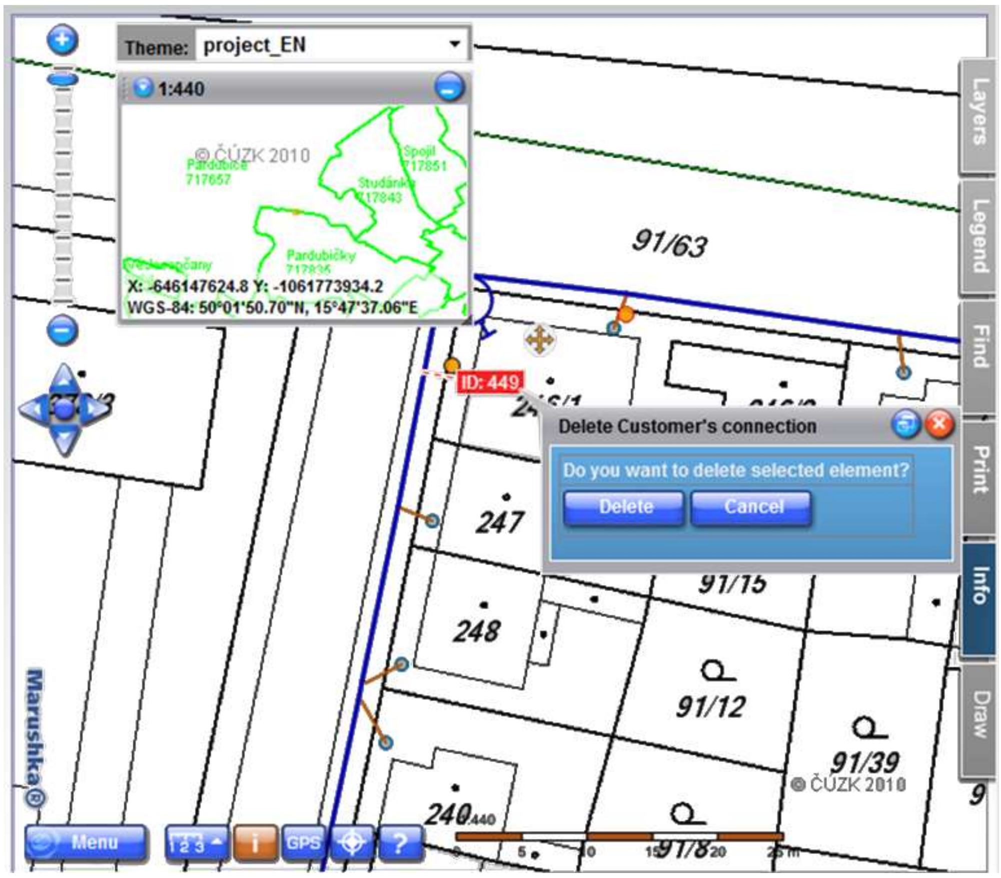
The objective of this chapter is to teach you how Marushka can delete elements that we have drawn ourselves before, from the database. We do not want to delete the data already stored in the database and so the original data and the user data will be again accessed in two ways. Again use attribute: *userdraw*.

Primarily put this attribute into the *DBColumnsToClient* property in form layers, where you need them. This means, you will adjust this property in form layers *W_PIPING Customer userdraw* and *W_TCUSTOM userdraw*.

10.1 Creating a Query for Deleting the Element

The query for deleting of element is very simple. We will create the pair of queries – for deleting the connection and connection termination that we have drawn. Both queries will look the same – they differ only in the physical layer name (database tables). Therefore we will describe only one of the queries, the other one you will create by yourself without a detailed description.

In the database source SQLite select *New – Utility – Delete element* in the Query library context menu. Rename the query to: *Delete Customer's connection*. Enter the name of physical layer *W_PIPING* into the *LayerName* property. The *SqlStmtTemplate* will contain a simple phrase for deleting an actual element: `DELETE FROM W_PIPING WHERE ID = ~(long)ID~`. To achieve, that deleting query is offered to us **only when dealing with elements created in Marushka environment**, it is necessary to add another query properties *AttributeName* and *AttributeValue*. Enter the text *userdraw* into *AttributeName* property and value 1 into *AttributeValue* property. The corresponding attribute gains value 1 just in the case it was created in the Marushka environment. The last property of this query we will change is *ViewStyle*. In this case it is considerably more efficient to display question if we really want to delete the element into "bubble". We set the value *ViewStyle* to *InPopUpBubble*. Similarly create the query for deleting connection termination. If you worked properly, then while in the info mode in Local WEB server after hover the mouse cursor on the elements created in Marushka environment will display a query for deleting an element. After selecting it, a dialog box (bubble) will display with a question for confirmation to delete the element.


Delete Customer's connection

Do you want to delete selected element?

11 Marushka Seeks and Finds

Often the user needs to find an element based on any specific properties, resp. he needs to find geographical coordinates. To locate using GPS coordinates Marushka has available predefined query, in the Local WEB server window under GPS icon. There is no need to describe this query, but you can try it out.

Now we will pay attention to creation of custom localization query. We will use the database properties of the elements that are already in the data store, and we want to find a connection belonging to a particular customer (by his name). We want the localization query uses the criteria from the dynamic selection list. We will create a pair of queries – the main localization query and the auxiliary list of values.

11.1 Creating a Localization Query

In the query library using context menu of SQLite data store we select *New – Localize*. Name the query *Localize customer.* The customer information is stored in the *W_PIPING* table, which we type in *LayerName* property. We may keep the *QueryBuf* property without changes. It depends on how we will use this query. If we will search only by surname, it is recommended to increase the value and the localization query will return more possibilities for the final localization. In the *QueryParameters* property we create one record "surname" string type. The basic *SqlStmtTemplate* will have the following form:

```
select xmin-5000 XMIN, ymin-5000 YMIN, xmax+5000 XMAX, ymax+5000 YMAX,
id  ID,  ifnull(name,'')||'  '||ifnull(surname,'')  LABEL  FROM  W_TCUSTOM
WHERE ifnull(name,'')||' '||ifnull(surname,'')=~(string)1~
```

We can see, that the result will be a connection termination in the map window cut-out, which is defined by corners *XMIN*, *YMIN*, *XMAX*, *YMAX* – these coordinates are taken directly from the selected database element and are adjusted so that the displayed cell doesn't cover the whole map window. In an attribute LABEL, which will be a description of corresponding element is combination of name and surname of the customer, separated by a space (for better lucidity). Defining **ifnull** is necessary for the cases when one of the fields (first name or surname) would be empty and in database SQLite environment is this transcription required for the proper conduct select.

Next, change the outcome symbology of localization. In the property *Symbology* set a color interpretation according to our own discretion. Predefined color is light blue with a light blue fill, so even without manual intervention in the symbology should in most cases return a visible localization result.

The last property which we will change is *DynamicCodeList*. By changing the value of this property to true we turn on the Search suggestion. According to the entered characters, we will receive an interactive menu with existing records defined in a dynamic list of values.

11.2 Creating a List of Values

In the query library using the context menu on SQLite database store choose *New – List of values*. Rename the created query to: *Localize Customer-List of values*.

In the property *LayerName* enter the corresponding physical layer name (*W_TCUSTOM*), enter the *ID of parent query* (*QueryLV*), which can be founded in properties of created localization query. Localization of the customer and the order of the parameter in the parent query (*QueryLVNUM*). In our case we have just one parameter, so we enter the value 1.

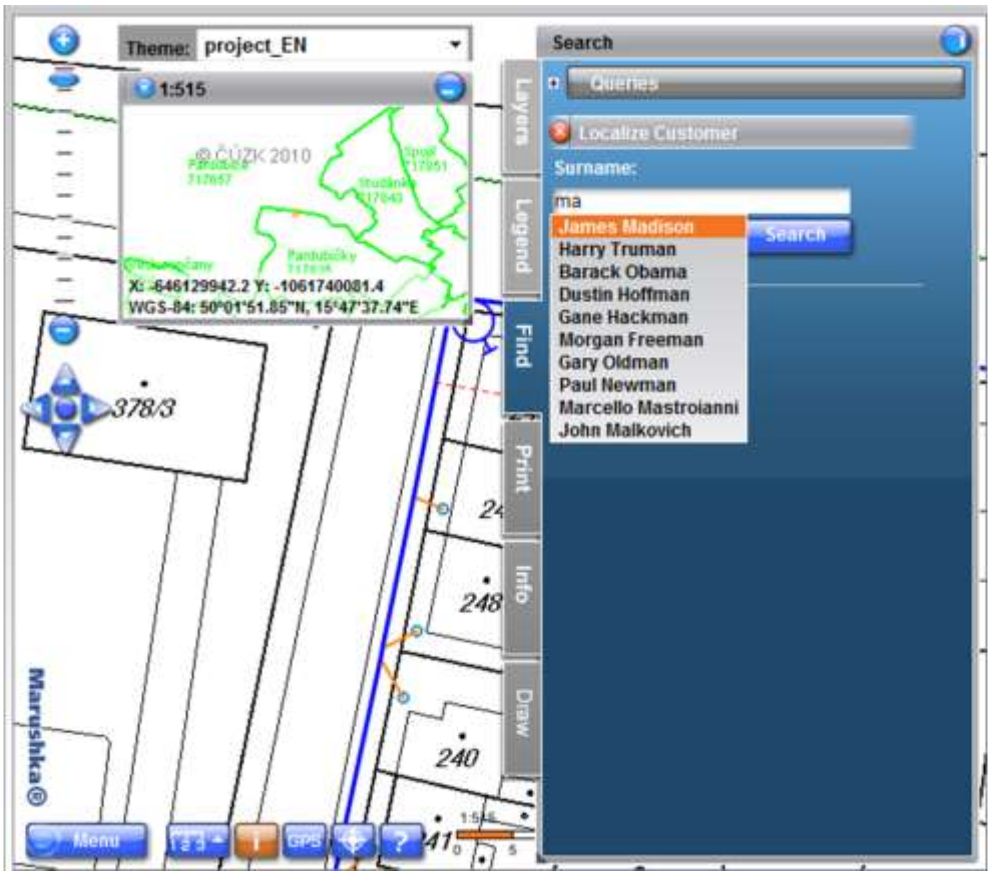
Enter the following phrase into *SqlStmtTepmlate* in the following form:

```
SELECT ifnull(name,'')||' '||ifnull(surname,'')   FROM   W_TCUSTOM WHERE
surname like '%~1~%'
```

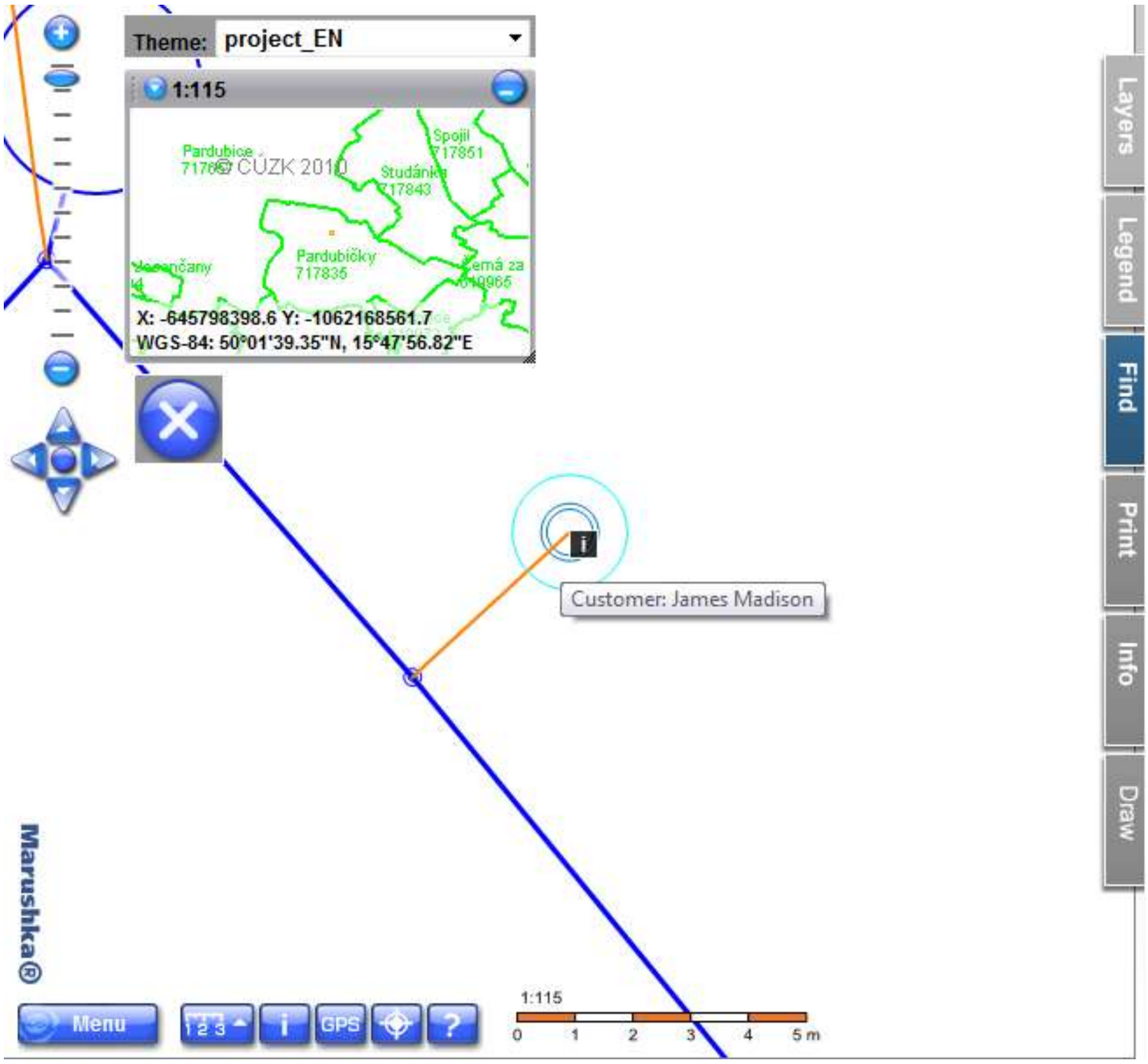
The result of the select will be the list of values satisfying the defined condition. Such a definition will achieve that in the list of values; will be offered items, where existing entry in the text field of localization query will contain any part of the value in the column "surname". If we remove the "%" symbol before the parameter, then the result shows only the surname, which started this string.

11.3 Testing Localization Query

In the local WEB environment we move on to Search Tab, where we can see a single localization query that we just created. Select this localization query and it will display a dialog box for entering the name (or surname) of customer, we want to find. At the moment, when we gradually enter the text, the list of values satisfying the criterion will display as described above. An example of one such attempt you can see in the following picture:



You can see that after entering the string "Ma" ten of possibilities appears. All these Customers names contain the string "Ma" (regardless on the font size). After selecting the item, you really wanted to find and after confirming by the Search button, you will finally get the result of localization:



The picture shows the connection that is centered and a limiting rectangle of the map window is adjusted to a greater edge, which we have defined in select phrase of our localization query. In the example was modified the result symbology, which is without filling to see the connection terminations. After hovering the mouse on the information icon, detailed information about customer will appear. Now we see that the result of our search is correct.

12 Marushka – The Documentarian

Marushka can work comprehensively with all the documents that are associated with single database elements. Documents can be displayed, inserted or removed from the data store. In this chapter, we will create a pair of document queries that will allow a complete document management. The document query will apply to the connection termination – it includes any documentation concerning the connections.

12.1 Creating a Binary Query

To work with the document itself, we have to create a binary query, through which we open the document. This query is not yet linked with a graphic database table. In the query library using the data store SQLite context menu choose *New – Binary*. Rename the created query to: *Documentary of Customer's connection - binary*. In its properties we will edited only the database phrase select – *SqlStmtTemplate*. It will have the following form:

```
select DOC DOCUMENT, BLOB_TYPE EXTENSION, BIRTH_DATE BIRTH_DATE FROM
demo_doc WHERE id=~(long)ID~
```

The SQL phrase includes a list of three required columns that are contained in our database table that we have created in the introductory part of this project.

12.2 Creating a Query for Document Management

In the query library using the context menu of SQLite data store select *New – Utility – File Browser*. For Clarity rename the created query to *Documentary of Customer's connection.* This query will already be tied to the graphic database layer and SQL phrases defined in the properties will be linked directly to the graphic elements, to which will be the documents associated with. In the *LayerName* property enter the name of the physical layer for connection termination (*W_TCUSTOM*). For better lucidity change the property *ViewStyle* to *NewWindow*. Keep other properties unchanged, move to the last three properties – the SQL phrases allowing upload, view and delete document. Let's start with a list of documents to the appropriate database element. The *SqlDocListTemplate* will look like this:

```
SELECT id ID, BLOB_TYPE EXTENSION, LABEL LABEL, BIRTH_DATE BIRTH_DATE,
'-2147483637'        BINARYQUERY        FROM        demo_doc        WHERE
ID_GRAPHICS_ELEMENT=~(long)ID~
```

In this select we need to **adjust the ID of binary query** – put there a real query ID in apostrophes that we created in the previous step.

Furthermore create a query that will allow us to insert the document, which creates a link to the selected graphic element. The phrase in property will look like this:

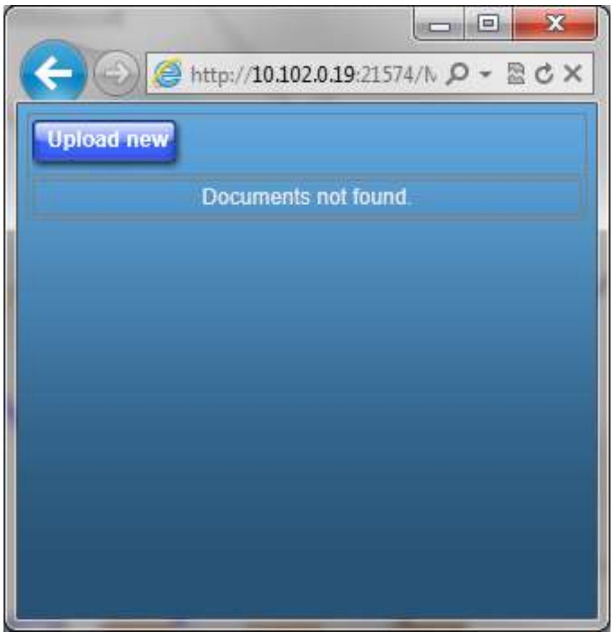
```
INSERT  INTO  demo_doc  (BLOB_TYPE,LABEL,DOC,ID_GRAPHICS_ELEMENT)  VALUES
(~BLOB_TYPE~,~FILE_NAME~,~DOCUMENT~,~ELEMENTID~)
```

These parameters are mandatory in `this` phrase and indicate the type of the document, its title, the document itself and the link to the graphic element.

The last step to create a documentation query for complete documentation file management is a phrase used to delete the document. *SqlDeleteDocumentTempate*, which is the simplest one of these three queries:

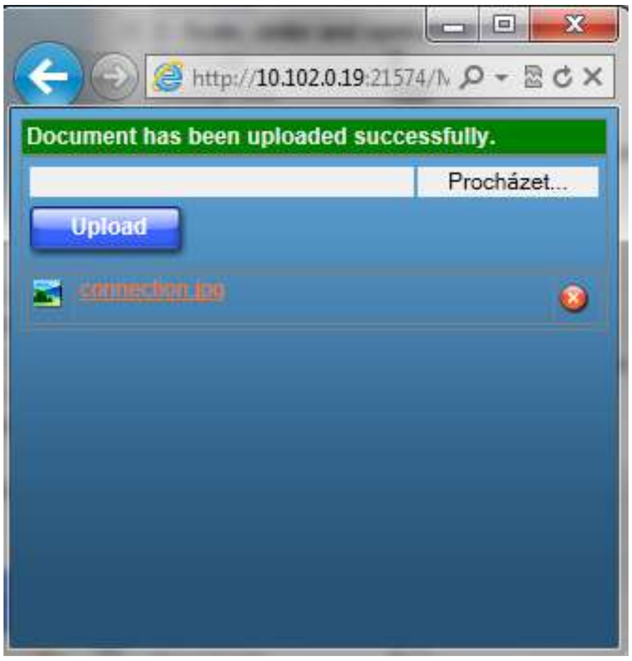
```
DELETE FROM demo_doc WHERE ID=~(long)ID~
```

In the previous steps we fulfilled all the requirements for functional documentation query and we can directly test its functionality. In the local WEB server choose any connection termination cell, on which we will test the query. From the queries available for this cell, we will select the item connection documentation. Now a new window of document browser opens:



At this moment, the document list is empty. Now we can try the work with documents. Click on *Upload* button. A row for a name and path of document appears. After clicking on *Browse* button a dialog box for opening a file will appear. In the folder `C:\MarushkaExamples\Tutorial\Files` find an image file "`connection`". Open this file and a in the text box will appear the text with path to our file. After clicking on *Upload* button the database insert of relevant document will occur.

Dialog box with a list of files changes immediately and we will get the temporary information about successful file upload.


http://10.102.0.19:21574/M
Document has been uploaded successfully.
Procházet...
Upload
connection.jpg

The uploaded image can then be viewed by simply by clicking on a line with its name. The image will open in a new Web browser window. The document can be also deleted by icon with a cross on the right side of the dialog box with a list of documents. This will of course remove only the documents in the database, not the files from the hard drive, where we uploaded the files from.

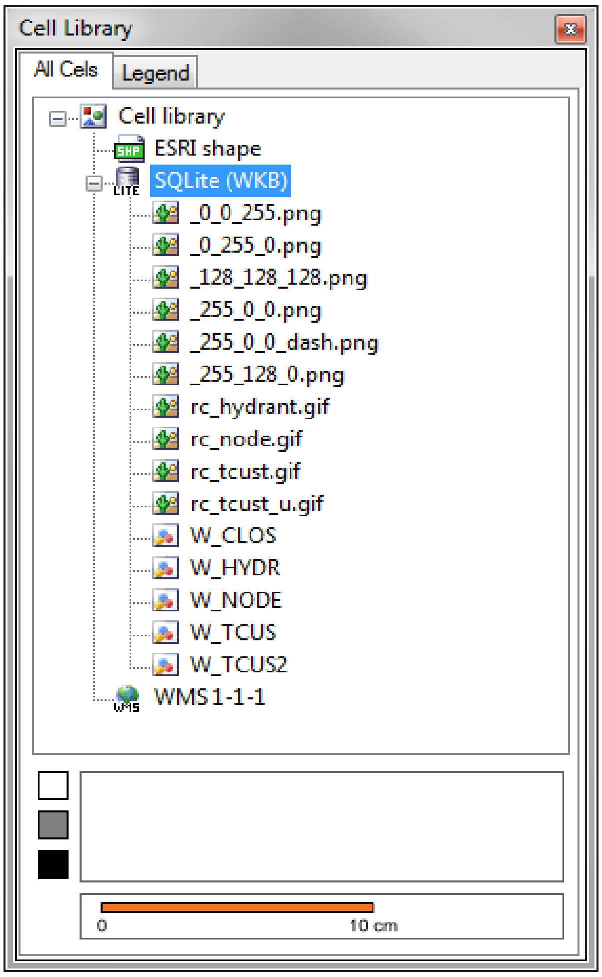
13 The Legend in Marushka

For better orientation in individual elements that are displayed in the map publication serves a legend, which we have to create. The basic premise is the existence of raster cells, which will then be displayed as individual legend items. In Marushka we have two kinds of a legend. The **Static legend**, where each item will be displayed always when in the cut-out of the map window will be displayed at least one element of the form layer, which is legend item related to. The second option is to create a **dynamic legend**, which will only display the relevant items of the legend. This means that the legend item is displayed only when in the window cut-out will occur at least one element meeting the criteria.

13.1 Preparation of Cell Libraries

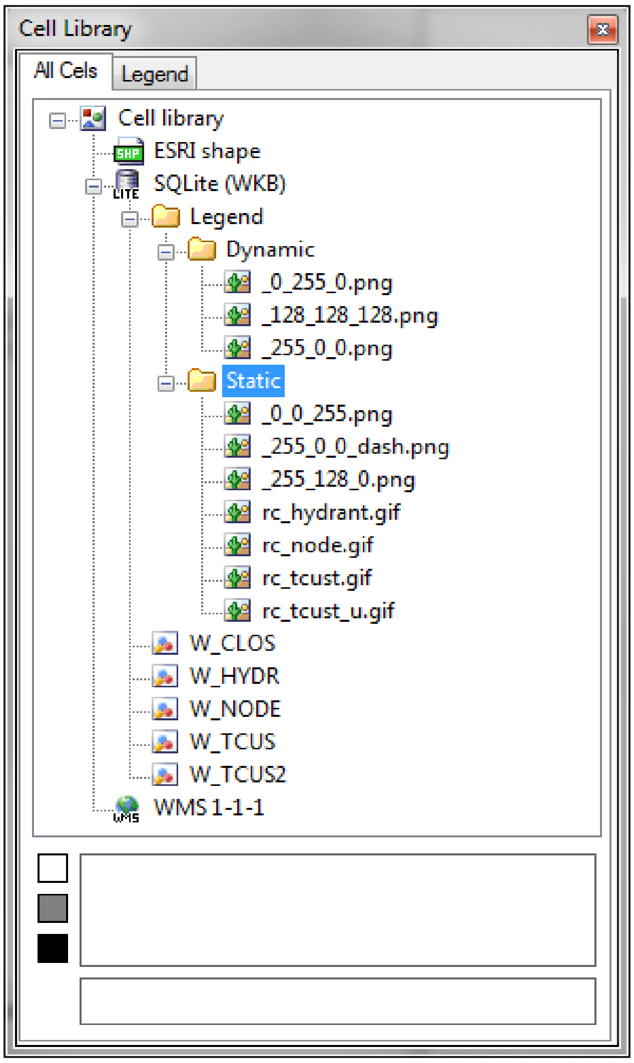
Both variants of legend displaying have one thing in common. That is the necessity of raster cells, which will serve as a single legend item. In this tutorial we have prepared small images, which will be created from single cells. In the Cell library prepare cells for the static and also dynamic legend.

In the cell library in the context menu of the database source SQLite open *New cell – Raster – New cell from image*. In the dialog box for opening a file, select the first cell, which can be found in the files of our tutorial (C:\MarushkaExamplex\Tutorial\Files\rb_hydrant.gif). Repeat the same procedure for other raster cells for static legend (rc_tcust.gif, rb_tcust_u.gif, rb_node.gif, _0_0_255.png, _255_0_0_cark.png and _255_128_0.png) and for three cells that will be used for creating dynamic legend (_128_128_128.png, _255_0_0.png and _0_255_0.png). We will not demonstrate creating such pictures in our tutorial, because it is not directly related to work in MarushkaDesign and these images can be created in any graphic editor. After creating the last raster cell, our library cell should look like this:



As a picture below shows, the newly created raster cells are not very neatly arranged, so in the first phase do an adjustment in their arrangement. Change the property *description* at cells

_128_128_128.png, _255_0_0.png and *_0_255_0.png,* so that you get the string in the following form: "*Legend~Dynamic~original name*". In the remaining newly created raster cells put into the *description* property text: "*Legend~Static~original name*". After this modification will our dialog box with a library cell look as following:



```
Cell library
  ESRI shape
  SQLite (WKB)
    Legend
      Dynamic
        _0_255_0.png
        _128_128_128.png
        _255_0_0.png
      Static
        _0_0_255.png
        _255_0_0_dash.png
        _255_128_0.png
        rc_hydrant.gif
        rc_node.gif
        rc_tcust.gif
        rc_tcust_u.gif
    W_CLOS
    W_HYDR
    W_NODE
    W_TCUS
    W_TCUS2
  WMS 1-1-1
```

The organization of cells looks already more arranged, the tilde notation is an effective way how to make order in the data. In the next step you may change the caption property, which is the text that will appear next to the legend. At the bottom there are the image cells, by which the user can estimate (for static legend), what particular cell was it created for. The items that pertain to the dynamic legend will be discussed in the next section of this chapter, where we will try to understand their deeper meaning. When you will edit the item by the order in which they appear in the picture above, enter to the *Caption* properties gradually these descriptions:

Dynamic legend:

_0_255_0.png – Connection dimension 40

_128_128_128.png – Unspecified connection

_255_0_0.png – Connection dimension 32

Static legend:

_0_0_255.png – Route of pipeline

_255_0_0_cark_png – Route of water connection – drawn by user

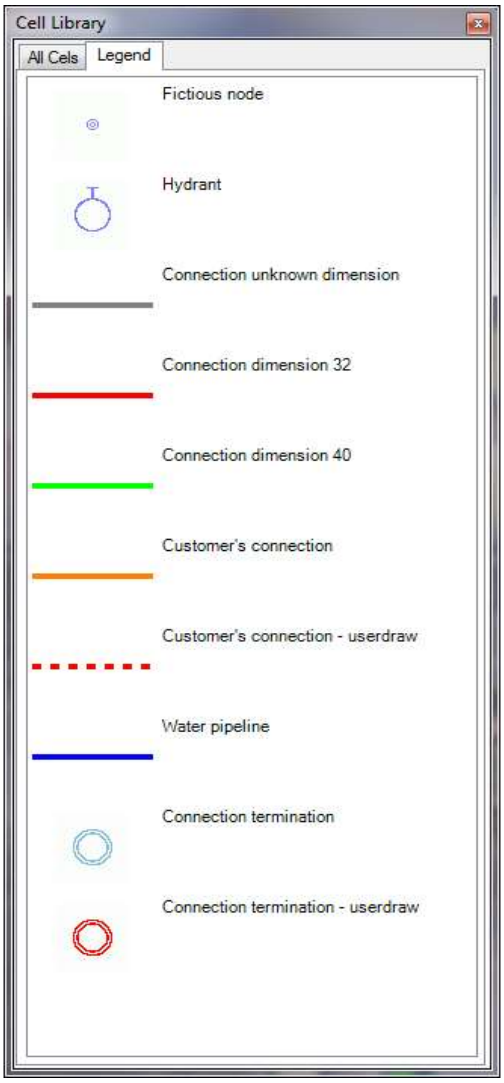
_255_128_0.png – Route of water connection rb_hydrant.gif – Hydrant

rb_tcustom.gif – Connection termination

rb_tcustom_u.gif – Connection termination – drawn by user

rb_node.gif – Fictitious node

In *CellName* property for simplification remove the file extension ".png" in all the cells. When you look at the *Legend tab* in the cell library, you will see an empty dialog box. This is due to the fact that in this tab, there are displayed only the cells that have in the *CellType* property entered: *CaptionItem*. This property can be changed in bulk (through MultiSelect in the cell library). After this change you will see in the legend tab the following status:



Fictious node

Hydrant

Connection unknown dimension

Connection dimension 32

Connection dimension 40

Customer's connection

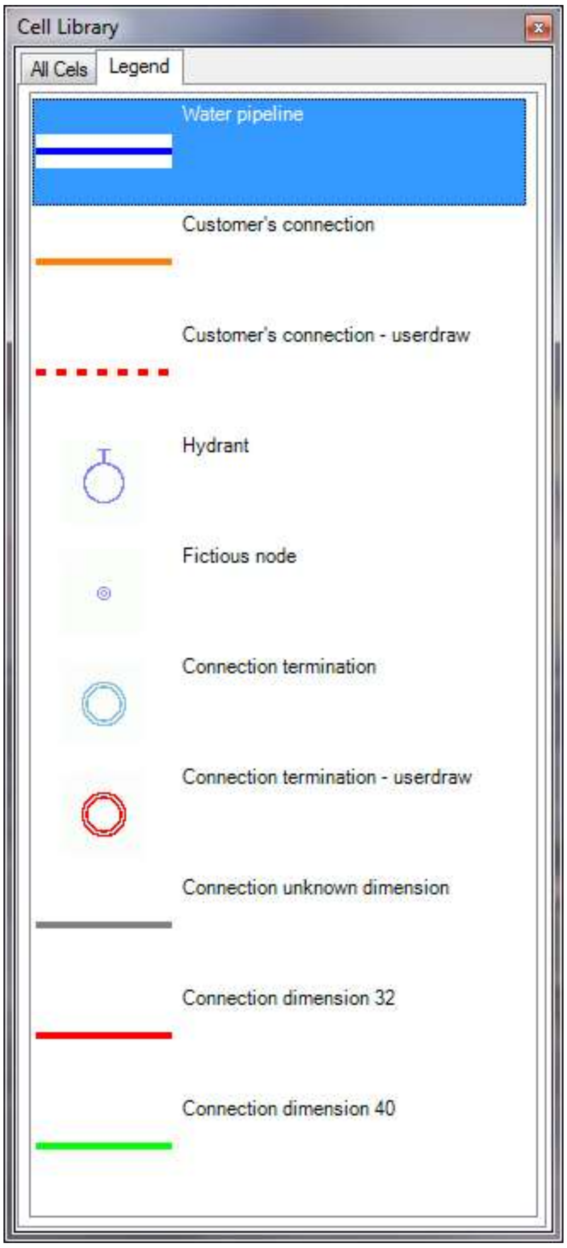
Customer's connection - userdraw

Water pipeline

Connection termination

Connection termination - userdraw

See from the diagram, that sorting of the legend may not suit us. Property classification of legend is a property classification of the whole project. This feature can be found in the properties tab for the *Data source*. The initial setup is sorted by the *Caption* parameter. It is not necessary to change this property. You can sort the legend using drag and drop, according to your requirements. The first time you try to drag a legend item, a dialog box will appear, informing that you are currently not allowed to change the sorting of legend. If you confirm this by pressing yes, you can continue in manual sorting. When manually sorting, it is possible to use the MultiSelect again (using CTRL or SHIF key and left mouse button) and sort the entire group of cells. Let’s try to sort our legend this way:

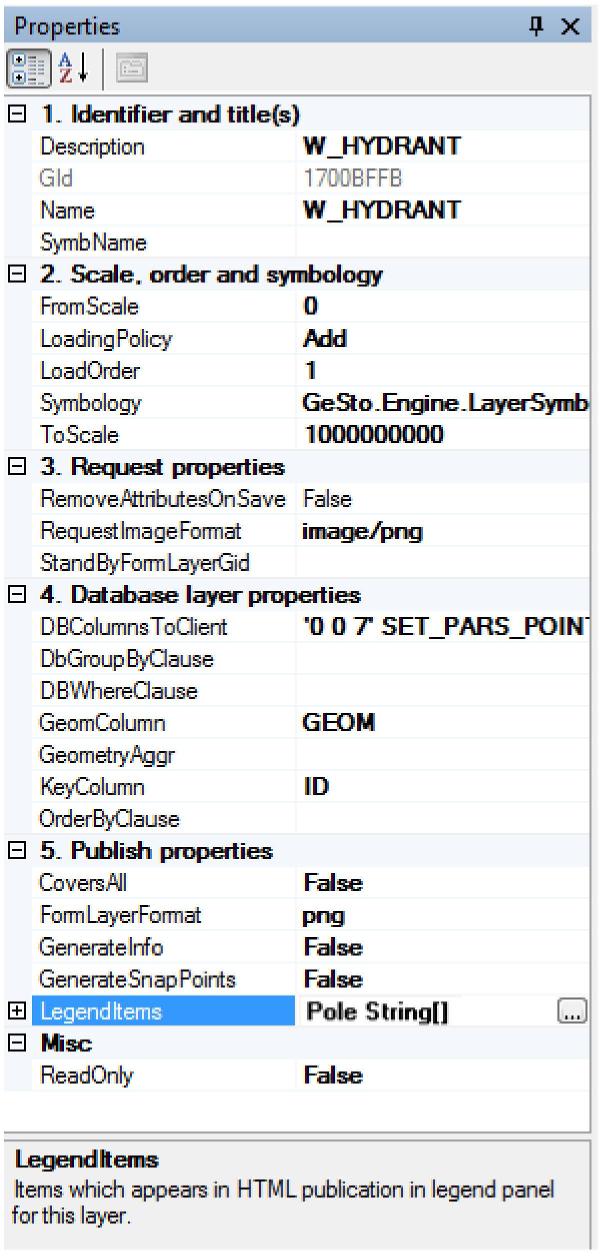


Water pipeline
Customer's connection
Customer's connection - userdraw
Hydrant
Fictious node
Connection termination
Connection termination - userdraw
Connection unknown dimension
Connection dimension 32
Connection dimension 40

In the top part are the items of the static legend with priority to display line objects. In the bottom part we see the items of the dynamic legend, which will be explained in the next section of this chapter.

13.2 Creating a Static Legend

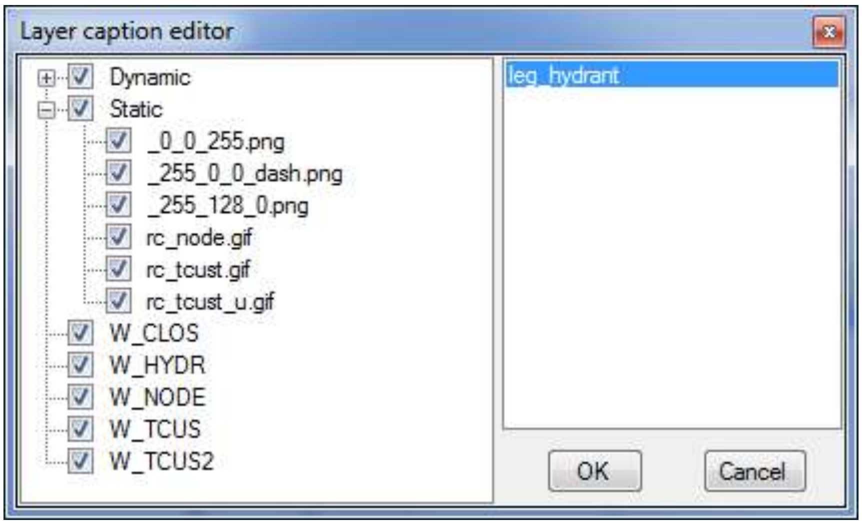
In the first stage create a static legend. Let's leave our Cell library and move on to the properties of form layers, which are individual legend items bound to. In the form layer properties *W_HYDRANT* find property *LegendItems*:



1. Identifier and title(s)	
Description	W_HYDRANT
GId	1700BFFB
Name	W_HYDRANT
SymbName	
2. Scale, order and symbology	
FromScale	0
LoadingPolicy	Add
LoadOrder	1
Symbology	GeSto.Engine.LayerSymb
ToScale	1000000000
3. Request properties	
RemoveAttributesOnSave	False
RequestImageFormat	image/png
StandByFormLayerGid	
4. Database layer properties	
DBColumnsToClient	'0 0 7' SET_PARS_POIN
DbGroupByClause	
DBWhereClause	
GeomColumn	GEOM
GeometryAggr	
KeyColumn	ID
OrderByClause	
5. Publish properties	
CoversAll	False
FormLayerFormat	png
GenerateInfo	False
GenerateSnapPoints	False
LegendItems	Pole String[]
Misc	
ReadOnly	False

LegendItems
Items which appears in HTML publication in legend panel for this layer.

After clicking on the button with three dots, the dialog box with form layer editor pops up. Find an item for hydrant and either by F5 key or by dragging the mouse in combination with the left mouse button drag the item to the right part of the dialog box.



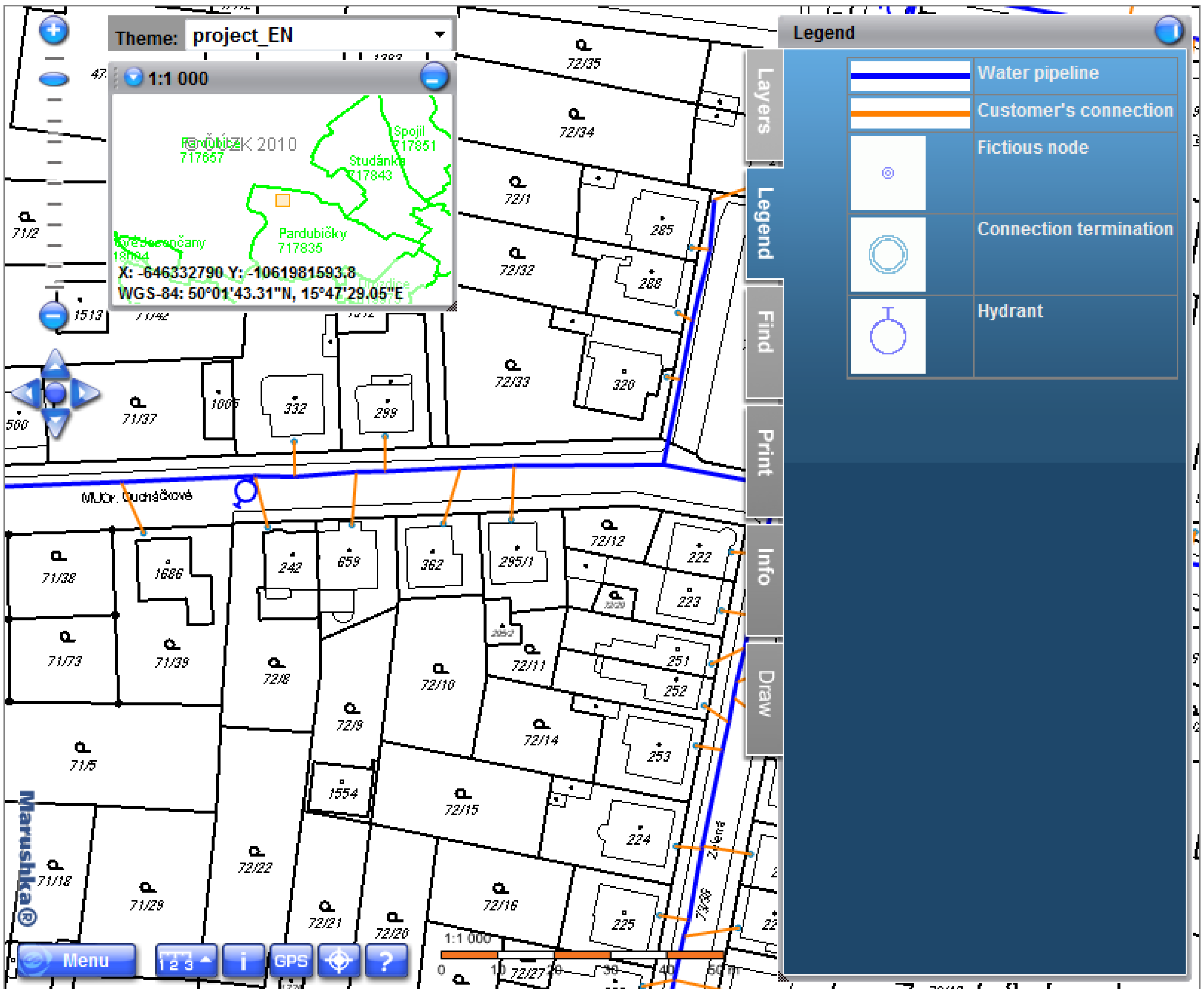
Layer caption editor

- Dynamic
- Static
  - _0_0_255.png
  - _255_0_0_dash.png
  - _255_128_0.png
  - rc_node.gif
  - rc_tcust.gif
  - rc_tcust_u.gif
- W_CLOS
- W_HYDR
- W_NODE
- W_TCUS
- W_TCUS2

leg_hydrant

OK Cancel

So we created this item for the corresponding form layer and this item will be displayed in the local WEB server anytime, whenever the current cut out will contain at least one element belonging to this layer. The whole procedure is applied successively to all our form layers (except for the layer *W_CLOSURE*) and assigns each of them just one legend item. In general, one form layer can have an unlimited number of legend entries. You can try out displaying of the created legend in the local WEB server environment. If you will have all the legend items displayed in the map window, we will see a complete legend. Otherwise you will see only a part of the legend:



Legend

▬	Water pipeline
▬	Customer's connection
◎	Fictious node
◯	Connection termination
♁	Hydrant

13.3 Dynamic Legend and Thematization

Thematization is offered to be used directly in dynamic legends. By thematization is meant displaying the map composition by other graphic symbology based on the properties of individual graphic elements. In our case, we want to display different color connections, according to their dimensions. For this thematization we will need a legend that will allow us to more easily "read" different graphical views.

13.3.1 Thematization Creation

To create a new thematization we need a new formal layer. To do this, we will use existing form layer "*W_PIPING Customer* " from which we will create a clone as we have already explained in a previous chapters. Rename the *SymbName* property for newly created clone of form layer to "*Customer theme*". In this clone we want to display both original database connections and connections newly created in MarushkaDesign environment. So we modify the property *DBWhereClause* from condition: *rc="Customer's connection' and userdraw is null"* to *rc="Customer's connection*".

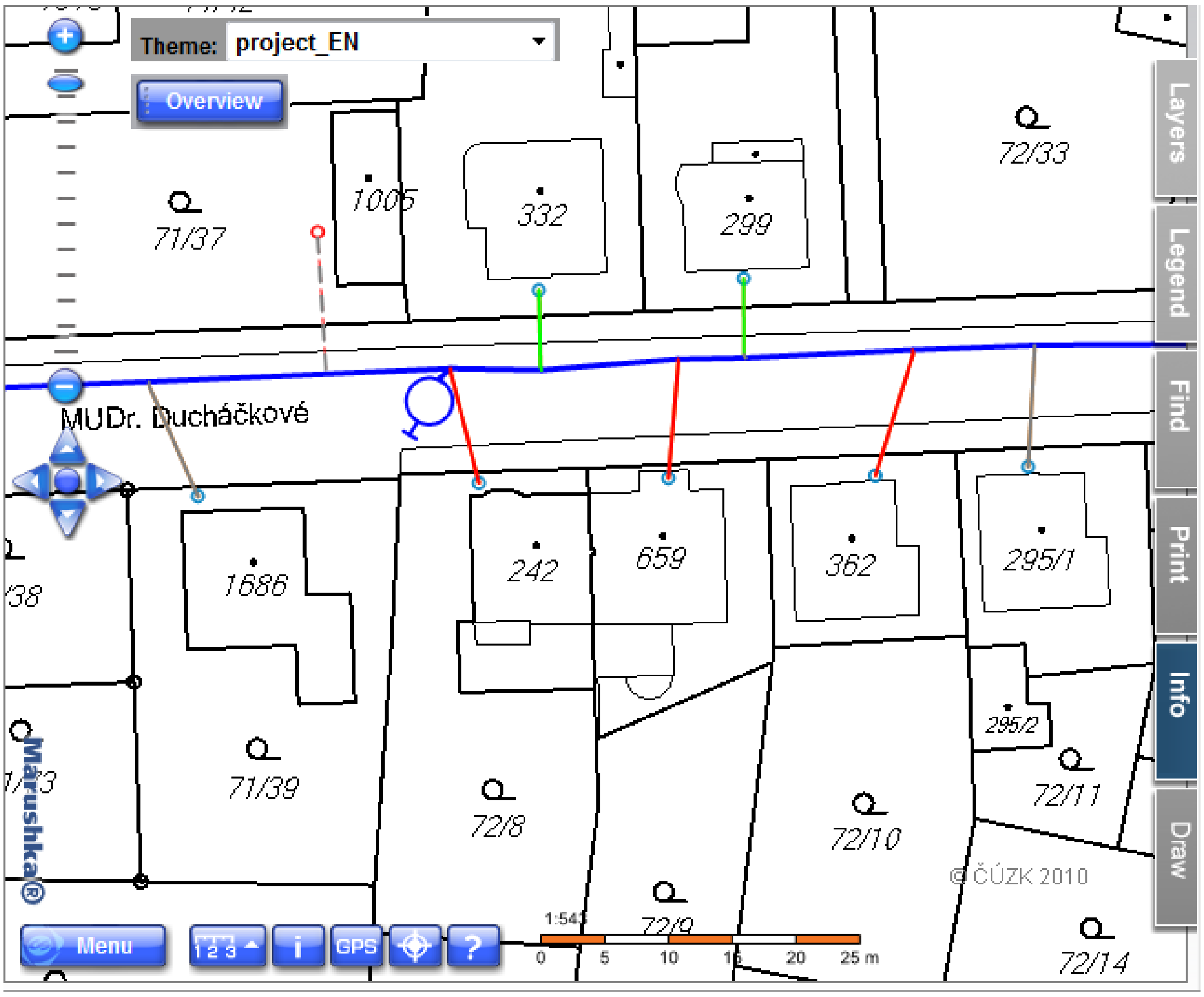
To change the color presentation of database graphic elements we use pseudo columns in property *DBColumnsToClient* in combination with slightly advanced database structure phrase, in which we use conditional branching "case". The whole property will look like this:

```
rc,userdraw,date_of_building,dim_cust,commission,case  when  dim_cust=32
then '255 255 0 0' when dim_cust=40 then '255 0 255 0' else '255 128 128
128' end SET_PARS_RGBCOLOR.
```

To ensure that the thematized connections are displayed always above the database elements, we must change the property of form layer *LoadOrder* to "2". This will ensure that the "recolored data" are shown above the original database data symbolization.

Create a new publish layer named "*Connections according dimensions*", to which drag the newly created clone of form layer.

If you worked honestly when passing through this tutorial and fulfilled the task in the chapter Editing Query - Testing the Resulting Editing Query and really filled in a few lines, then you can test the functionality of your thematization. Result in a local WEB server (after manually switching on the new publish layer for thematization) will be a window with such colorfulness:



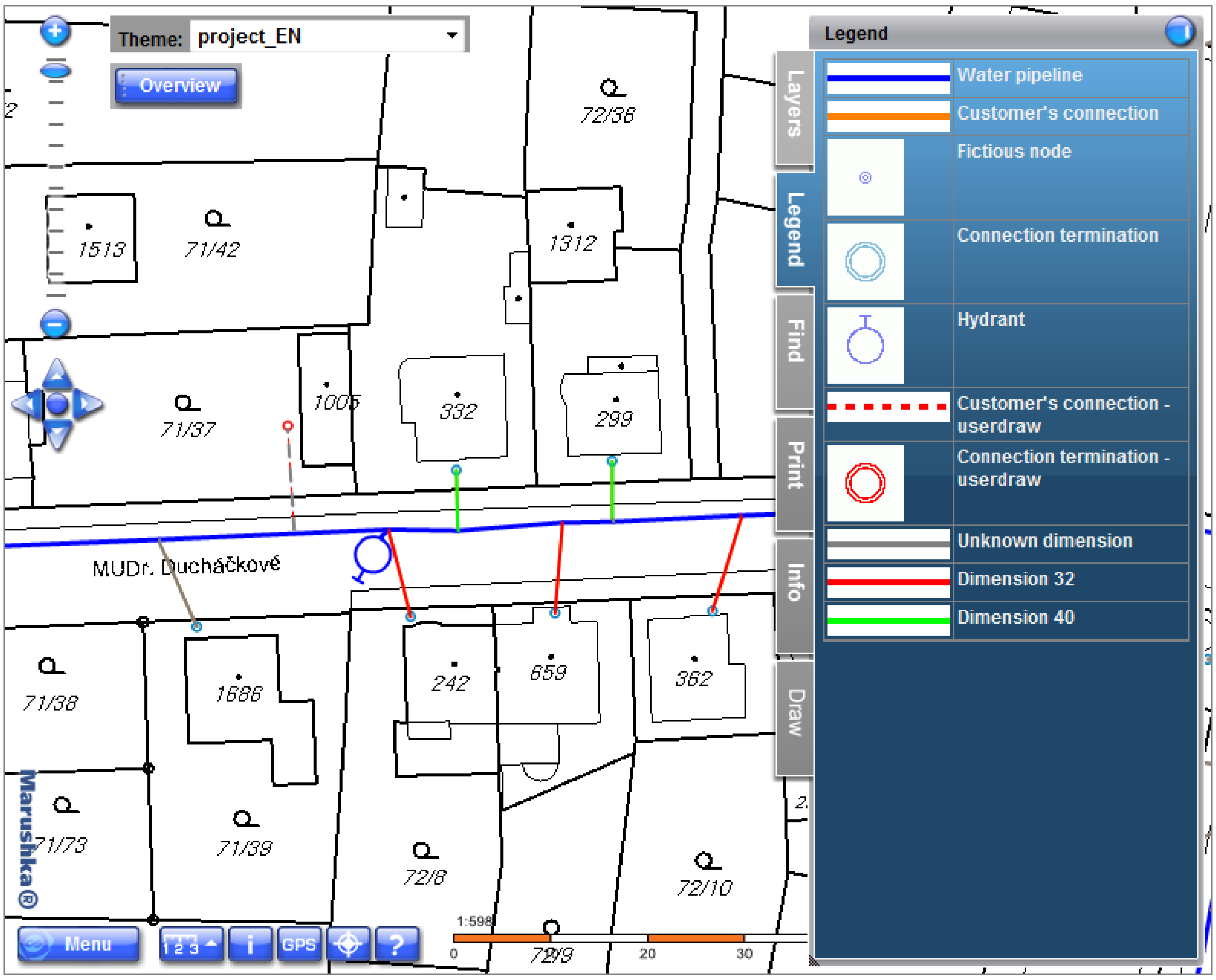
In the picture you can see red, green and gray connections, from which the line style of the connections drawn in Marushka environment remains and these can be visually distinguished. But at this moment, there is not available legend for thematized connections. So now it's time to create it.

13.3.2 Creating Dynamic Legend

This step is already relatively simple and is analogous to recoloring connections using pseudo column *SET_PARS_RGBCOLOR*. In this case use pseudo column *SET_LEG_ITEM.* Extend the original property *DBColumnsToClient* to the following form:

```
rc,userdraw,date_of_building,dim_cust,commission,case  when  dim_cust=32
then '255 255 0 0' when dim_cust=40 then '255 0 255 0' else '255 128 128
128' end SET_PARS_RGBCOLOR, case when dim_cust=32 then '_255_0_0' when
dim_cust=40 then '_0_255_0' else '_128_128_128' end SET_LEG_ITEM
```

In the pseudo column *SET_PARS_RGBCOLOR* are individual parameters defined by ARGB code of the corresponding colors. In the pseudo column *SET_LEG_ITEM,* it is a property of individual cells which is defined as legend items. Therefore for the property *CellName* was used this code, which can be read as a RGB code for the color and its writing is in compliance with this structure quite intuitive. Once again, we will test the legend displaying in the local WEB server – again after manually switching on publish layer Connections according dimensions:



During movement in the map window, you can verify that even when a form layer for thematized connections is displayed, so the legend will be variable depending on which types of attributes we have at the moment displayed in the map window.

14 Teaching Marushka to Sing and Dance

In this manual we have tried all the basic functionality for creating web map compositions in MarushkaDesign environment. With Marushka, we went through her first steps, we have taught her to read, write, draw and otherwise manipulate with data. You really won't teach her how to sing and dance, but it is up to you how you will work with the available data base and how much you will be able to use the data. The possibilities are really wide and the entire document is only a guide to the basic functions of Marushka. We didn't show everything, the entire tutorial is a simplified overview of the functionality of each component. Each project can be continuously developed and improved, if we continue to think about the further expansion of each function. So we have successfully managed to complete a fundamental work with MarushkaDesign.

The results of our efforts can be then sent to the official WEB server and we can publish our map composition (either internally or publicly). But the principles of this publication goes far beyond the scope of this tutorial, more about this issue can be found in the official manual, which is a part of MarushkaDesign.

And where to go next? Marushka is still living and her possibilities are endless, so therefore in conclusion a few more ideas about what we could in our test project yourself.

14.1 Suggestions for the Individual Exercises

- Individual adjustment of drawing query for drawing connections, while you are drawing you can also add a dimension of connection.
- Let's return to the cell closures, which we have omitted from the publication. Try to go back to the fact that we want to highlight the closed closures. Modify the database table W_CLOSURE by adding column STATE. You will be able to edit the column in the local WEB server environment (creating editing query with static list of values). Then create a simple thematization or form layer that will have a special graphic symbology for closed closures. The result will display turned off closures.
- In cooperation with the official manual, you can create a set (a common query), which will display with for example a list of customers in a particular city. In this case, you can only outline the path along which you should proceed, because such a query is already quite user demanding. Apart from the knowledge of database environment it requires also knowledge of HTML code to display query correctly. In the query library create a *New query – New – Common info*. In the property *SqlStmtTemplate* create a query, which will return a required list (this query is not connected with any graphical element) and in property *ResultTemplate* you would create a HTML template for displaying the result of this query. HTML template can be used in other cases, when the result is displayed in a new window.

At this moment we can congratulate ourselves that we have reached on the go with Marushka up to this stage and we were able to create the whole project. More possibilities depend just on our own imagination.