Cartoweb Advanced

Edit and routing plugins, write a new plugin

Olivier Courtin

Yves Bolognini

Frédéric Junod

Advanced Cartoweb: Table of Content

Prologue: connections parameters

- SSH parameters:
  - Host: 130.223.73.235
  - Login: camptocamp
  - Password: c2c
  - Port: 2222

- PostgreSQL parameters:
  - Host: 130.223.73.235
  - Postgres user: postgres
  - Postgres password: postgres
  - Port: 5432

Prologue: Postgis Database Creation

- Create PostgreSQL/PostGIS database:

```
$ createdb -U postgres YOUR_DATABASE
$ createlang -U postgres plpgsql YOUR_DATABASE
$ cd /usr/share/postgresql/8.1/contrib
$ psql -U postgres -d YOUR_DATABASE < lwpostgis.sql
$ psql -U postgres -d YOUR_DATABASE < spatial_ref_sys.sql
```

- To check the install then connect to the database and launch following SQL statement (use pgAdmin) :

```
SELECT postgis_full_version();
```

Edit Plugin Introduction

- Allow end user to modify geographic features
- Attributes values are also handled
- Only a javascript compliant browser needed
- Geographics layers must be stored in a spatial database (PostGIS)
- Topologic snapping is also supported
- Demo online, cf: http://www.cartoweb.org/demos/demoEdit.php
- Doc, cf: http://www.cartoweb.org/doc/cw3.3/xhtml/user.edit.html

Edit Plugin: Table Of Content

Postgis Data Importation

- On the server they've been put in following directory:

```
~/data/edit
```

- So to import in PostGIS database, just type:

```
$ shp2pgsql data/edit/field.shp field | psql -U postgres -d
  YOUR_DATABASE
```

Mapfile configuration

- Copy mapfile.map to mapfile.map.in:

  - Source: cartoweb3/projects/foss4g_routing/server_conf/foss4g/foss4g.map
  - Destination: cartoweb3/projects/foss4g_routing/server_conf/foss4g/foss4g.map.in

- Add a new layer to render routing result

  - put this Layer at the end of layer's stack
  - cf next slide

Add a Layer in Mapfile.map.in

```
LAYER
  NAME "Fields"
  TYPE POLYGON
  CONNECTIONTYPE POSTGIS
  CONNECTION "user=@DB_USER@ password=@DB_PASSWD@ host=@DB_HOST@ dbname=@DB_NAME@"
  DATA "the_geom FROM (SELECT the_geom, gid, i_parcelle FROM field) as foo USING
 UNIQUE gid"
  MAXSCALE 18500
  TEMPLATE "foo"
  METADATA
    'id_attribute_string' 'i_parcelle|string'
    'query_returned_attributes' 'i_parcelle'
  END
  CLASS
    STYLE
      COLOR 220 200 30
      OUTLINECOLOR 30 30 30
    END
  END
END
```

Add Specifics Metadatas Related to Edit

```
METADATA
  'id_attribute_string' 'i_parcelle|string'
  'query_returned_attributes' 'i_parcelle'
   'edit_table' 'field'
  'edit_geometry_column' 'the_geom'
  'edit_geometry_type' 'polygon'
  'edit_srid' '-1'
  'edit_attributes' 'i_parcelle|integer'
END
```

Cartoclient Configuration

In cartoweb3/projects/foss4g_edit/client_conf/edit.ini:

```
; Cartoweb roles allowed to use edit plugin
general.allowedRoles = anonymous

; Max inserted features at once (could be 0)
insertedFeaturesMaxNumber = 10

; Edit layers names (comma separated)
editLayers = Fields

; Toolbar group used
groupPlugin = 3
```

Templating

- Templating handled by Smarty (http://smarty.php.net)
- Each project can override templates files
- Each plugin manages his own templates files
- Cartoclient.tpl is the main template container file

Templating: cartoclient.tpl (part I)

- Add CSS loader in the head tag:

```
{if $edit_allowed|default:''}<link rel="stylesheet"
  type="text/css" href="{r type=css plugin=edit}edit.css{/r}"
  />{/if}
```

- Add folder entry in 'ul' tabnav:

```
<li id="label3"><a href="javascript:ontop(3)">{t}Edit{/t}</a></li>
```

- Add folder content:

```
<div id="folder3" class="folder">
  {if $edit_active|default:''}
     {include file="../plugins/edit/templates/edit.tpl"}
  {/if}
</div>
```

Templating: cartoclient.tpl (part II)

- Update toolbar definition:

```
<div id="toolbar">
   {include file="toolbar.tpl" group=1 header=1}
   {include file="toolbar.tpl" group=2}
  <br />
   {if $edit_allowed|default:'' && $edit_layer_selected}
     {include file="toolbar.tpl" group=3}
     <input type="checkbox" id="snapping" name="edit_snapping"
            onclick='mainmap.snap("map")'
            {if $edit_snapping|default:''}checked=checked{/if} />
     <img src="{r type=gfx plugin=edit}edit_snap.gif{/r}"
          title="{t}Allow vertex snapping{/t}"
          alt="{t}Allow vertex snapping{/t}"><br />
   {/if}
</div>
```

Templating: cartoclient.tpl (part III)

- Add attributes list display:

```
{if $edit_allowed|default:''}
   <br />
   <div align="center" id="edit_div" style="display:none"></div>
{/if}
```

Plugin Activation

- In cartoweb3/projects/foss4g_edit/client_conf/client.ini.in:

```
loadPlugins = edit
```

- In cartoweb3/projects/foss4g_edit/server_conf/foss4g/foss4g.ini:

```
mapInfo.loadPlugins = mapOverlay, edit
```

Create Install Files

- In cartoweb3/install_edit.bat:

```
php cw3setup.php -install --base-url http://127.0.0.1/cartoweb3/htdocs
  --profile development --config-from-file
  projects/foss4g_edit/foss4g_edit.properties --project foss4g_edit
```

- In cartoweb3/projects/foss4_edit/foss4g_edit.properties:

```
DB_HOST=130.223.73.235
DB_USER=postgres
DB_PASSWD=postgres
DB_PORT=5432
DB_NAME=YOUR_DATABASE
```

- Now simply launch install_edit.bat

Use Edit Plugin

- Available edit tools:

  - Select a geometry feature
    - move vertex
    - add new vertex
    - delete vertex
    - update attributes values
  - Delete a geometry feature
  - Insert a new geometry feature

Routing Plugin Introduction

- Find shortest path in a network, on the fly, between two nodes
- Directed graph supported
- Usable on significative data set (several thousand edges)
- Use PgDijkstra to compute graph

Routing Plugin: Table Of Content

Pgdijkstra Installation (FYI only)

- Requires :
  - a working PostgreSQL server
  - lib boost:

```
$ sudo apt-get install libboost-graph-dev
```

- Then install pgdijkstra itself :

```
$ cd cartoweb3/contrib/pgdijkstra
$ make
$ sudo make install
```

Create a spatial database with pgdijkstra

- Create PostgreSQL/PostGIS database

  (already done in prologue step):

- Then add pgdijkstra support:

```
$ cd /usr/share/postgresql/8.1/contrib
$ psql -U postgres -d YOUR_DATABASE < dijkstra.sql
$ psql -U postgres -d YOUR_DATABASE < dijkstra_postgis.sql
```

Retrieve spatial data

- Initials data came from:

http://ftp.intevation.de/freegis/frida/frida-1.0.1-shp.tar.gz

- On the server they've been put in following directory:

```
~/data/routing
```

- Then import in database with:

```
$ shp2pgsql strassen.shp street | psql -U postgres -d
  YOUR_DATABASE
```

Add graph data

- Connect to your PostgreSQL database:
  - use either pgAdmin
  - or psql in console mode

- Create empty graph structure:

```
ALTER TABLE street ADD COLUMN source_id int;
ALTER TABLE street ADD COLUMN target_id int;
ALTER TABLE street ADD COLUMN edge_id int;
```

- Now fill the structure (source_id and target_id) (may take a while) :

```
SELECT assign_vertex_id('street', 1);
```

Handle doublons

- Check first if doublons really occurs:

```
SELECT * FROM (SELECT source_id, target_id, count(*) AS c
FROM      street group by source_id, target_id order by c) AS
foo where     foo.c = 2;
```

- Then if needed, to remove them:

```
CREATE TABLE single AS SELECT * FROM street WHERE gid in
(SELECT gid FROM (SELECT DISTINCT on (source_id, target_id)
 source_id, gid FROM street) AS single);
```

```
DELETE FROM street;
INSERT INTO street (SELECT * FROM single);
DROP TABLE single;
```

Edges and vertices tables

- To create edges and vertices tables:

```
SELECT create_graph_tables('street', 'int4');
```

- Look then to resulting:

```
SELECT * FROM street_edges LIMIT 10;
```

Fill Cost Data on street_edges

- To fill cost data from geometry distance use following SQL statement:

```
SELECT update_cost_from_distance('street');
```

- Then to check it:

```
SELECT * FROM street_edges LIMIT 10;
```

- Nota on reverse_cost:
  - reverse_cost could be used to handle directed graph
  - -1 value for reverse_cost mean a one way edge

Check PgDijkstra

- Check shortest_path function:

```
SELECT * FROM shortest_path('SELECT id, source, target,
  cost FROM street_edges', 1, 45, false, false);
```

- Check shortest_path_as_geometry function:

```
SELECT gid, astext(the_geom) FROM
  shortest_path_as_geometry('street', 1, 45);
```

Create Routing Temporary Result Table

- The empty structure table:

```
CREATE TABLE routing_results (
      results_id integer,
      "timestamp" bigint,
       gid integer
       );
```

- Add now geometry column:

```
SELECT AddGeometryColumn('','routing_results','the_geom','-1',
  'MULTILINESTRING',2);
```

- And finally add sequence on this table:

```
CREATE SEQUENCE routing_results_seq
    INCREMENT 1
    MINVALUE 1
    MAXVALUE 9223372036854775807
    START 1
    CACHE 1;
```

Activate Routing Plugin on Cartoweb

- Routing project name is:
  - foss4g_routing

- Client-side configuration:
  - in cartoweb3/projects/foss4g_routing/client_conf/client.ini.in:

```
loadPlugins = routing
```

Server Side Override Plugin: directories

- Routing plugin need a specific Cartoserver override on project level
- A very close example could be found in demoPlugins project

- Create following directories:

```
cartoweb3/projects/foss4g_routing/plugins
cartoweb3/projects/foss4g_routing/plugins/foss4gRouting
cartoweb3/projects/foss4g_routing/plugins/foss4gRouting/server
```

- Create then in the deepest directory, this empty file:

```
ServerFoss4gRouting.php
```

Server Side Override Plugin: empty class

In ServerFoss4gRouting.php add:

```
<?php

/**
 * Server routing plugin which uses Postgres
 * @package Plugins
 */
class ServerFoss4gRouting extends ServerPostgresRouting {

    /**
     * @see PluginManager::replacePlugin()
     */
    public function replacePlugin() {
        return 'routing';
    }
}
?>
```

Server Side Override Plugin: shortestPathQuery

Add in this class shortestPathQuery method:

```
/**
 * @see ServerRouting::shortestPathQuery()
 */
protected function shortestPathQuery($node1, $node2, $parameters) {
    $db = $this->getDb();
    $table = $this->getRoutingTable();
    $prepared = $db->prepare(sprintf("
         SELECT a.edge_id  FROM shortest_path(
           'SELECT id, source, target, cost FROM %s_edges',
            ?, ?, false, false) AS a LEFT JOIN %s ON vertex_id = gid",
         $table, $table));
     Utils::checkDbError($prepared);
     return $db->execute($prepared, array($node1, $node2));
 }
```

Server Side Override Plugin: getNodes

```
protected function getNodes(DB_result $result, $resultsId, $timestamp) {
    $nodes = array();
    $table = $this->getRoutingTable();
    $routingResultsTable = $this->getRoutingResultsTable();
    $db = $this->getDb();
    while ($result->fetchInto($row, DB_FETCHMODE_ASSOC)) {
        $node = new Node();
        $node->attributes = array();
        $attribute = new Attribute();
        $attribute->set('edge_id', $row['edge_id']);
        $routingResultsTable = $this->getRoutingResultsTable();
        $edgeId = $row['edge_id'];
        $r = $db->query("INSERT INTO $routingResultsTable SELECT $resultsId, " .
                "$timestamp, gid, the_geom FROM $table WHERE edge_id =
 $edgeId");
        Utils::checkDbError($r, 'Error quering routing database');
        $nodes[] = $node;
    }
    return $nodes;
}
```

Activate Server Side Plugin

- in cartoweb3/projects/foss4g_routing/server_conf/foss4g/foss4g.ini :

```
mapInfo.loadPlugins = routing, foss4gRouting
```

Mapfile configuration

- Copy mapfile.map to mapfile.map.in:
  - Source: cartoweb3/projects/foss4g_routing/server_conf/foss4g/foss4g.map
  - Destination: cartoweb3/projects/foss4g_routing/server_conf/foss4g/foss4g.map.in

- Add a new layer to render routing result
  - put this Layer at the end of mapfile layer stack
  - cf next slide

Mapfile configuration: Layer add

```
LAYER
  NAME "street_routing"
  CONNECTIONTYPE postgis
  CONNECTION "user=@DB_USER@ password=@DB_PASSWD@
 host=@DB_HOST@ dbname=@DB_NAME@"
  TYPE LINE
  CLASS
    STYLE
      SYMBOL 'line'
      SIZE 2
      COLOR 230 20 20
    END
  END
END
```

Server side configuration: routing.ini.in

In cartoweb3/projects/foss4g_routing/server_conf/foss4g/routing.ini.in:

```
routingDsn =
 pgsql://@DB_USER@:@DB_PASSWD@@@DB_HOST@:@DB_PORT@/@DB_NAME@

; Routing table name
postgresRoutingTable = street

; Postgres routing result table name
postgresRoutingResultsTable = routing_results

; MapServer layer name to render routing
postgresRoutingResultsLayer = street_routing
```

Template Configuration: cartoclient.tpl

In cartoweb3/projects/foss4g_routing/templates/cartoclient.tpl:

```
<div>
   <ul id="tabnav1" class="tabnav">
       <li id="label1"><a href="javascript:ontop(1)">{t}Themes{/t}</a></li>
       <li id="label2"><a href="javascript:ontop(2)">{t}Routing{/t}</a></li>
   </ul>
</div>
<div id="container">
  <div id="folder1" class="folder">
     {$layers}
     <input type="submit" name="refresh" value="refresh" class="form_button" />
  </div>
  <div id="folder2" class="folder">
     {$routing}
  </div>
</div>
```

Template Configuration: routing.tpl

- Create directories:
  - cartoweb3/projects/foss4g_routing/plugins/routing
  - cartoweb3/projects/foss4g_routing/plugins/routing/templates

- FYI original routing.tpl file from cartoweb core:
  - cartoweb3/plugins/routing/templates/routing.tpl

Template configuration: routing.tpl

```
<p><b>{t}Find path{/t}</b></p>
<p>
  {t}from{/t} <input type="text" id="routing_from" name="routing_from"
                  value="{$routing_from}" size="8" maxlength="10" />
  <br />
  {t}to{/t} <input type="text" id="routing_to" name="routing_to"
                   value="{$routing_to}" size="8" maxlength="10" />
</p>

<p><input type="submit" name="routing_submit" value="{t}Routing Compute{/t}"
      class="form_button"/>
   <input type="submit" name="routing_reset" value="{t}Reset{/t}"
      class="form_button"/>
</p>
```

Create Install Files

- In cartoweb3/install_routing.bat:

```
php cw3setup.php --install --base-url http://127.0.0.1/cartoweb3/htdocs --
  profile development --config-from-file
  projects/foss4g_routing/foss4g_routing.properties --project
  foss4g_routing
```

- In cartoweb3/projects/foss4_routing/foss4g_routing.properties:

```
DB_HOST=130.223.73.235
DB_USER=postgres
DB_PASSWD=postgres
DB_PORT=5432
DB_NAME=YOUR_DATABASE
```

- Now simply launch install_routing.bat

Write a New Plugin: Introduction

- To extend native cartoweb features
- Several interfaces available to interact with cartoweb core
- PHP 5 Object mainly used
- Plugin board schema cf:
    - http://www.cartoweb.org/doc/misc/plugins_diagram.pdf

Main Cartoweb Interfaces

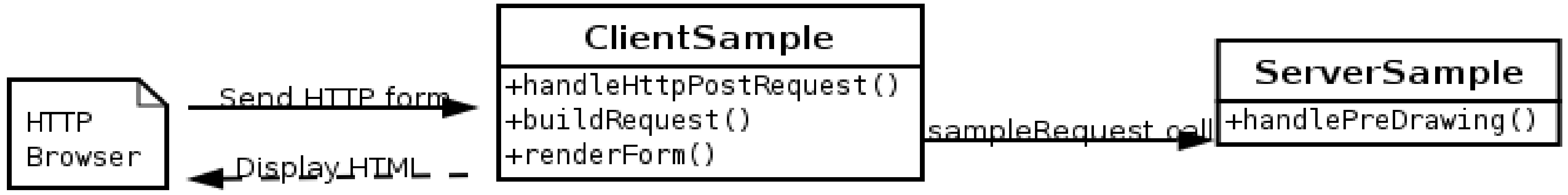
- Cartoclient:
  - GuiProvider: to handle GUI stuff
  - ServerCaller: to call cartoserver and get back results
  - Sessionable: to play with sessions values
  - ToolProvider: to manage toolbar entries
  - InitUser: to get cartoserver's initials parameters

- Cartoserver
  - ClientResponder: to handle cartoclient queries (need ServerCaller)
  - InitProvider: to provide initials parameters to cartoclient (need InitUser)

Write a New Plugin: Table of Content

Plugin Sample

- Plugin schema:



Create Directories structure

- Create following directories:
  - cartoweb3/projects/foss4g_routing/plugins/sample
  - cartoweb3/projects/foss4g_routing/plugins/sample/client
  - cartoweb3/projects/foss4g_routing/plugins/sample/server
  - cartoweb3/projects/foss4g_routing/plugins/sample/common
  - cartoweb3/projects/foss4g_routing/plugins/sample/templates

ClientSample.php: class structure

```
<?php
class ClientSample extends ClientPlugin
      implements GuiProvider {
    /**
     * @var Logger
     */
    private $log;

    /**
     * Constructor
     */
    public function __construct() {
        $this->log =& LoggerManager::getLogger(__CLASS__);
        parent::__construct();
    }
}
?>
```

ClientSample.php: GuiProvider Interface

```
/**
 * @see GuiProvider::renderForm()
 */
public function renderForm(Smarty $template) {}

/**
 * @see GuiProvider::handleHttpPostRequest()
 */
public function handleHttpPostRequest($request) {}

/**
 * @see GuiProvider::handleHttpGetRequest()
 */
public function handleHttpGetRequest($request) {}
```

ClientSample.php: renderForm

```
public function renderForm(Smarty $template) {

    $smarty = new Smarty_Plugin(
        $this->getCartoclient(), $this);

    $sample = $smarty->fetch('sample.tpl');
    $template->assign('sample', $sample);
}
```

ClientSample.php: handleHttpPostRequest

```
/**
 * @var sample text
 */
protected $sample;

/**
  * @see GuiProvider::handleHttpPostRequest()
  */
public function handleHttpPostRequest($request) {
   if(!empty($request['sample']))
       $this->sample = $request['sample'];
}
```

Templating: sample.tpl

```
<div id="sample">
  <label>{t}Sample Text{/t}</label>
  <input type="text" name="sample" />
  <br />
  <input type="submit" class="form_button"
         value="{t}Submit{/t}" />
</div>
```

Templating: cartoclient.tpl

```
<li id="label3"><a
href="javascript:ontop(3)">{t}Sample{/t}</a></li>

<div id="folder3" class="folder">
    {$sample}
</div>
```

Client Side Plugin Activation

- In cartoweb3/projects/foss4g_routing/client_conf/client.ini.in:

```
loadPlugins =  routing, sample
```

- Then launch again:

```
install_routing.bat
```

ClientSample.php: ServerCaller Interface

```
/**
 * @see ServerCaller::buildRequest()
 */
public function buildRequest() {}

/**
 * @see ServerCaller::initializeResult()
 */
public function initializeResult($result) {}

/**
 * @see ServerCaller::handleResult()
 */
public function handleResult($layerReorderResult) {}
```

ClientSample.php: buildRequest

- Add ServerCaller interface:

```
class ClientSample extends ClientPlugin implements
  GuiProvider, ServerCaller
```

- buildRequest code:

```
public function buildRequest() {
    $sampleRequest = new sampleRequest();
    $sampleRequest->sample = $this->sample;

    return $sampleRequest;
}
```

Sample.php

```
<?php
require_once(CARTOWEB_HOME . 'common/CwSerializable.php');

class SampleRequest extends CwSerializable {
    /**
     * @var string Sample Text
     */
    public $sample;

    /**
     * @see CwSerializable::unserialize()
     */
    public function unserialize($struct) {
        $this->sample = self::unserializeValue(
            $struct, 'sample', 'string');
    }
}
?>
```

WSDL: sample.wsld.inc

```
<!-- sample -->

<complexType name="SampleRequest">
  <all>
    <element name="className" type="xsd:string"/>
    <element name="sample" type="xsd:string"/>
  </all>
</complexType>
```

ServerSample.php

```php
<?php
class ServerSample extends ClientResponderAdapter {
    /**
     * @var Logger
     */
    private $log;

    /**
     * Constructor
     */
    public function __construct() {
        parent::__construct();
        $this->log =& LoggerManager::getLogger(__CLASS__);
    }
}
?>
```

ServerSample.php: handlePreDrawing

```
/**
 * @see ClientResponderAdapter::handleDrawing()
 */
public function handlePreDrawing($requ) {

    $msMapObj = $this->serverContext->getMapObj();
    $layer= $msMapObj()->getLayerByName('sample');
    $layer->set('status', MS_ON);

    $feature = ms_shapeObjFromWkt('POINT(10 10)');
    $feature->set('text', $requ->sample);

    $layer->addFeature($feature);
}
```

Add Sample Layer in Mapfile.in

```
LAYER
  NAME 'sample'
  TYPE point
  TRANSFORM OFF
  POSTLABELCACHE true
  CLASS
    LABEL
      TYPE BITMAP
      SIZE GIANT
      FORCE ON
      COLOR 0 0 0
      POSITION cr
    END
  END
END
```

Server Side Plugin Activation

- In cartoweb3/projects/foss4g_routing/server_conf/foss4g/foss4g.ini:

  mapInfo.loadPlugins = routing, foss4gRouting, sample

- Then launch once again:

  ```
  install_routing.bat
  ```

ClientSample.php Sessionable Interface

```
/**
 * @see Sessionable::createSession()
 */
 public function createSession(MapInfo $mapInfo,
                    InitialMapState $initialMapState) {}

/**
 * @see Sessionable::loadSession()
 */
public function loadSession($sessionObject) {}

/**
 * @see Sessionable::saveSession()
 */
public function saveSession() {
```

ClientSample.php Sessionable Interface (II)

- Add sessionable interface:

```
class ClientSample extends ClientPlugin
      implements GuiProvider, ServerCaller, Sessionable
```

- Add sampleState propertie:

```
 /**
  * SampleState Object (session object)
  */
 protected $sampleState
```

- Create SampleState class:

```
class SampleState {
     /**
      * @var sample text
      */
     public $sample;
}
```

ClientSample.php Sessionable Interface (III)

```
public function createSession(MapInfo $mapInfo,
                InitialMapState $initialMapState) {

     $this->sampleState = new SampleState;
}

public function loadSession($sessionObject) {
     $this->sample = $sessionObject->sample;
}

public function saveSession() {
     $this->sampleState->sample = $this->sample;
     return $this->sampleState;
}
```

Contacts

Camptocamp SA

PSE A – Parc Scientifique EPFL

CH-1015 Lausanne

Tel. +41 21 619 10 10

www.camptocamp.com / www.cartoweb.org

olivier.courtin@camptocamp.com

yves.bolognini@camptocamp.com

frederic.junod@camptocamp.com