CartoWeb Documentation

3.4.0 Edition

Published 2007-09-21

Table of Contents

Part I. Presentation and Architecture

This first part of the documentation is an overall presentation of both the visible and the hidden features of CartoWeb. The goals are first to give an idea of what CartoWeb can readily do if used as it is shipped, and second to explain why it is a powerful framework to build more advanced applications.

1. Project Presentation

1.1. About CartoWeb

CartoWeb3 is a comprehensive and ready-to-use Web-GIS (Geographical Information System), including many powerful features. As a modular and extensible solution, it is also a convenient framework for building advanced and customized applications.

Developed by Camptocamp SA [http://www.camptocamp.com], it is based on the UMN MapServer [http://mapserver.gis.umn.edu] engine and is released under the GNU GPL license [http://www.gnu.org/copyleft/gpl.html].

1.2. Credits

To date, the following people have been more or less involved in the development of CartoWeb :

- Yves Bolognini
- Mathieu Bornoz
- Oliver Christen
- Damien Corpataux
- Olivier Courtin
- Daniel Faivre
- Alexandre Fellay
- Marc Fournier
- Jean-Denis Giguere
- Florent Giraud
- Pierre Giraud
- David Jonglez
- Frédéric Junod
- Sylvain Pasche
- Claude Philipona
- Arnaud Saint Leger
- Alexandre Saunier

2. Cartographic Functionalities

2.1. Introduction

CartoWeb is quite a complex piece of software. This chapter is only a quick and consequently incomplete overview of the standard functionalities that are somehow visible for an end-user. The internal architecture and all the hidden features that make CartoWeb customizable and extensible are presented in the next chapter.

2.2. Overview

The first figure is an overall view of the user interface of the demo that is shipped with CartoWeb. The numbers refer to more or less visible underlying features. They are :

1. Main map
2. Dynamic keymap
3. Navigation tools : zoom-in, zoo-out, panning
4. Layers tree
5. Geographic query tool
6. Redlining tools : to draw points, lines, rectangles, polygons
7. Measuring tools : distances and surfaces
8. Language switch : internationalization support
9. Login link : users and roles support
10. Print dialog : PDF production
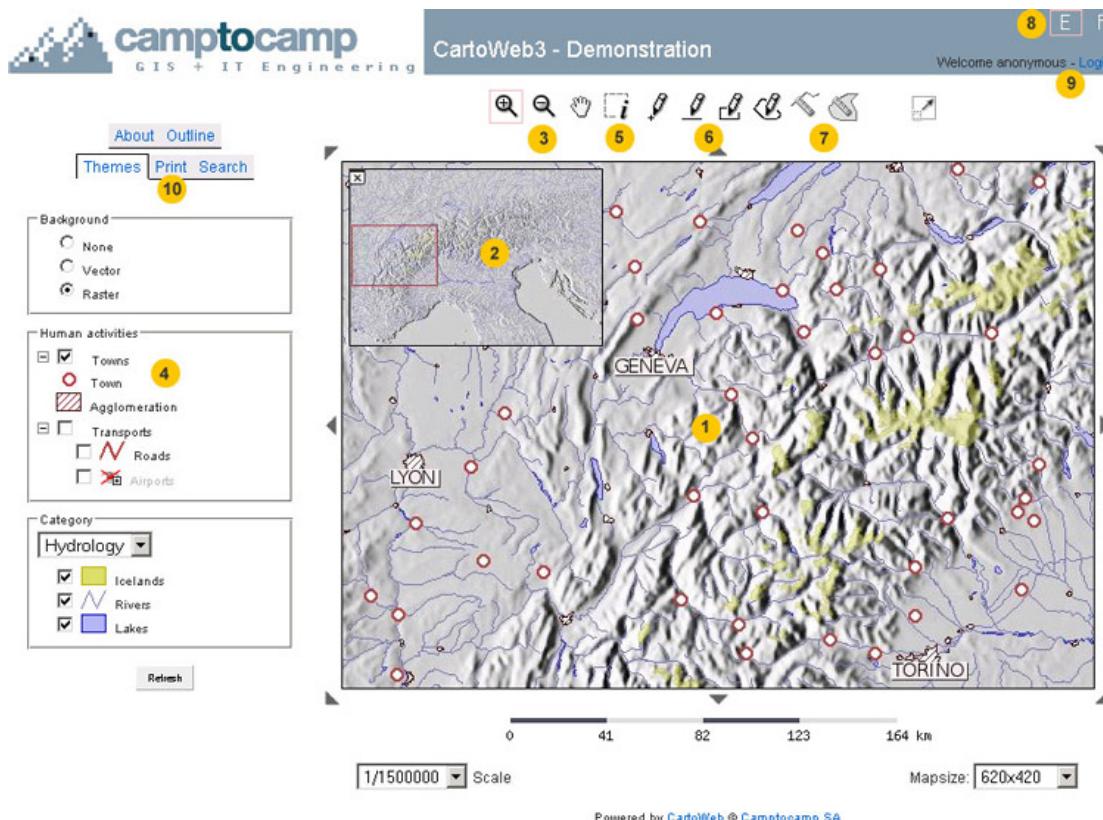
2.3. Navigation Interface

There are many possibilities to navigate on the main map, that is to change the scale and the position.

- The arrows surrounding the main map
- The dynamic (i.e. clickable) keymap
- The navigation tools (zoom and pan)
- The drop-down menu "Scale"
- The various options in the "Search" tab

The menu "Mapsize" is self-explanatory.

2.4. Arbitrarily Complex Hierarchy of Layers

Contrary to Mapserver itself, CartoWeb supports an arbitrarily complex hierarchy of layers, with infinite depth.

The elements of the layers "tree" have different rendering options :

- normal checkboxes

- blocks
- radio button (exclusive options)
- drop-down menu (exclusive options)

Examples of these rendering options are presented in the following figure.

The icons for the classes are automatically drawn, and the out-of-scale layers are grayed out.

2.5. Map Queries

Using the query tool, you can geographically search for objects. Found objects are hilighted and their attributes are displayed.

CartoWeb adds many functionalities to the raw queries supported by Mapserver. In particular, the queries may be persistent (i.e. you can add new objects to already selected objects), and the hilighting can be defined on a layer by layer basis.
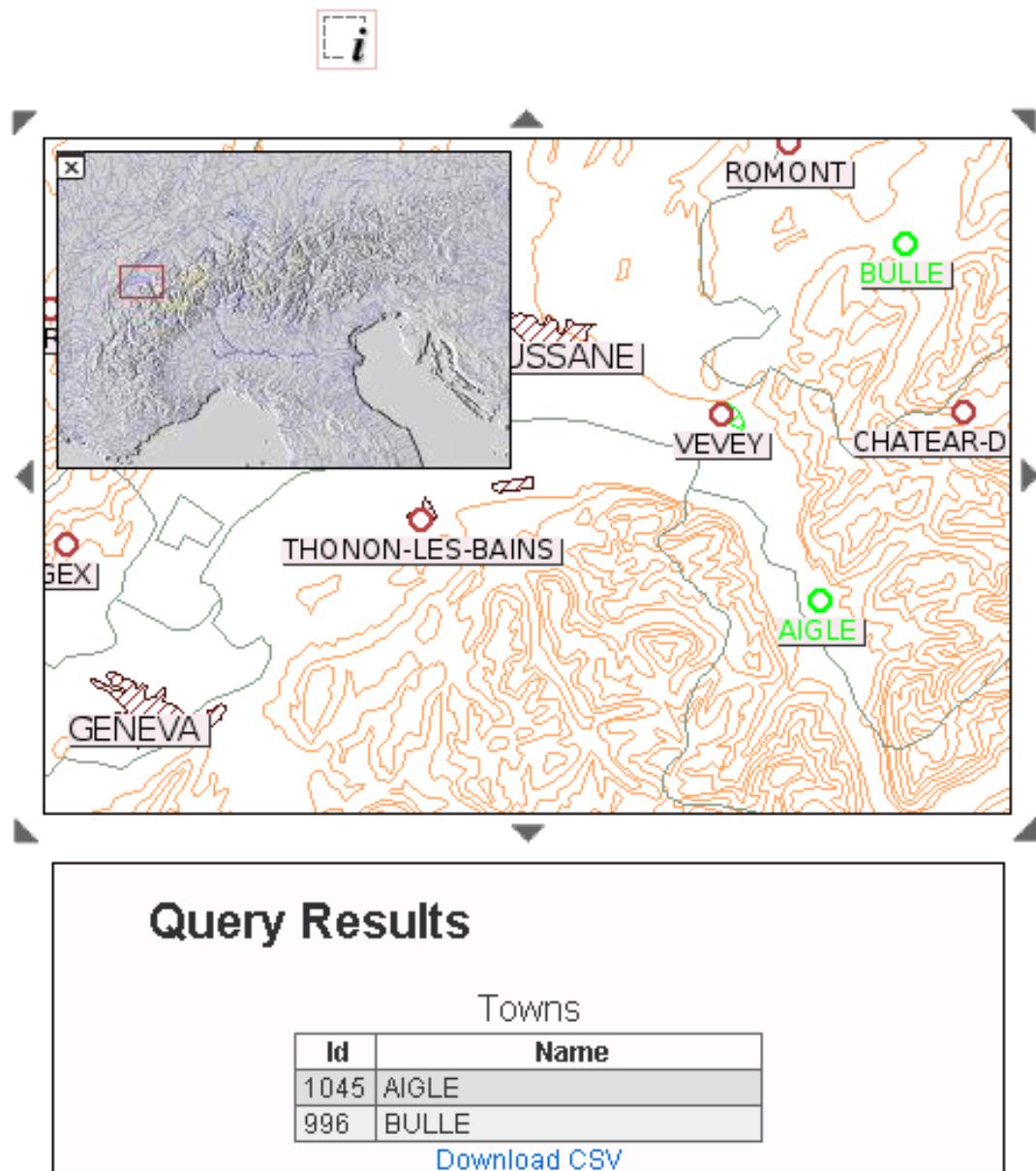
2.6. Annotation and Redlining

It is possible to freely draw points, lines, rectangles and polygons on the map, ant to attach labels to them. These features are persistent: they survive panning or zooming.

A mask mode, in which everything but the outlined polygon is masked, is provided too.
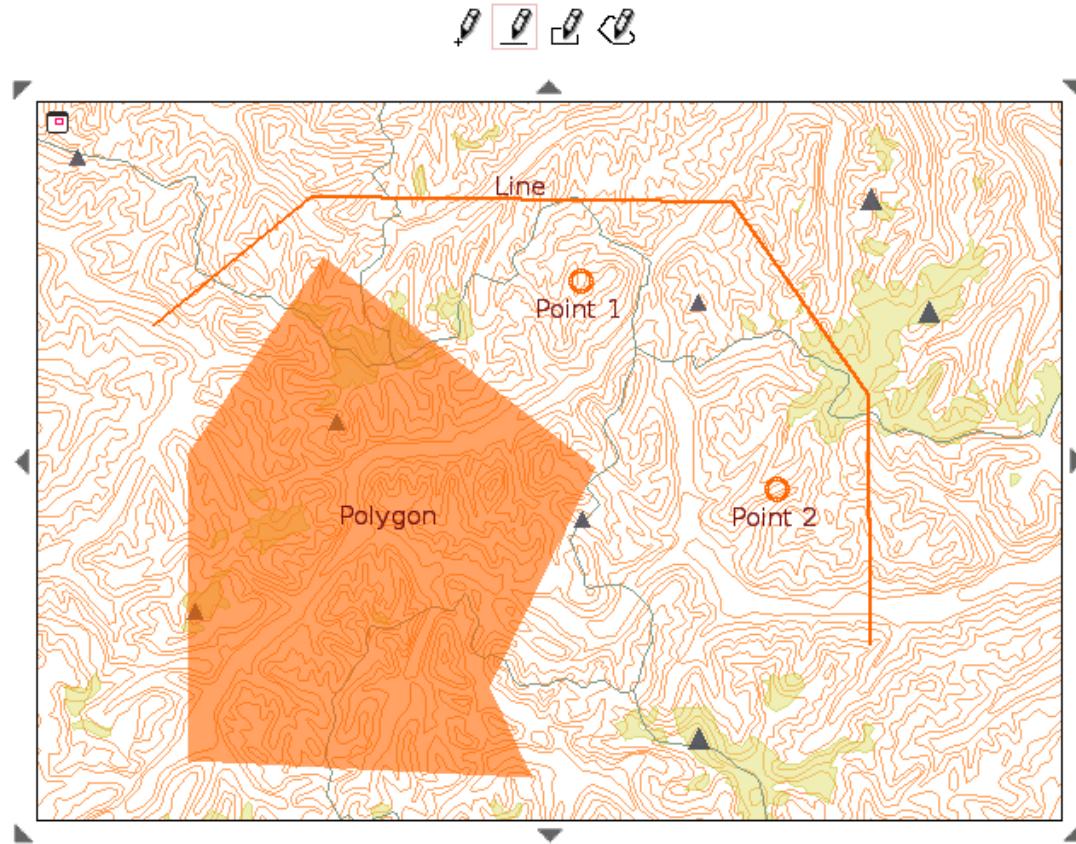
2.7. Measuring Tools

Distances and surfaces can be measured on the main map with the following tools :

2.8. Internationalization

Translation handling in CartoWeb now uses gettext [http://www.gnu.org/software/gettext/manual]. However internationalization architecture is ready for other translation systems.

Ta make life easier for translators, scripts that gather the strings to be translated in the templates and in the configuration files are available.

2.9. Access Rights

Access to different elements of CartoWeb can be allowed or denied according to who is currently using the application. Both functionalities and data may have access restrictions. For instance, PDF printing may be totally unavailable for anonymous access, limited (low resolution) for normal user and totally granted (high resolution) for superusers. Similarly, high-resolution aerial views may only be visible within an organization, while external users should be content with satellite photographs.

A basic (file-based) authentication mechanism is included, but any other mechanism that is able to authenticate an user and to link him to a role could be used as well.

2.10. PDF Output and Other Export Formats

CartoWeb is able to output a fully configurable PDF document. Some options can be chosen by the end user in the following dialog, while the CartoWeb admin defines which elements (maps, legends, tables, additional logos or watermarks...) are to be printed and sets their positions within the page.

Other output formats include the graphic formats (jpeg, png,...) of the map itself, simplified html templates and comma-separated values tables of the query results.

3. Architecture

3.1. Introduction

CartoWeb uses an innovative design and state-of-the-art technologies. The following sections briefly review the main employed approaches.

3.2. MapServer / MapScript

CartoWeb is based on the UMN MapServer [http://mapserver.gis.umn.edu/] engine. Interactions between CartoWeb and MapServer are achieved using the MapServer PHP/Mapscript [http://mapserver.gis.umn.edu/doc44/phpmapscript-class-guide.html] module.

MapServer must be installed prior to any CartoWeb setup.

MapServer resources:

- MapServer HomePage: http://mapserver.gis.umn.edu/
- MapServer Download: http://mapserver.gis.umn.edu/dload.html
- MapServer Documentation: http://mapserver.gis.umn.edu/doc.html
- MapServer PHP/Mapscript Class Reference
  http://mapserver.gis.umn.edu/doc44/phpmapscript-class-guide.html

3.3. Web-Service Architecture - SOAP

One of the CartoWeb specificities is its ability to work along a client-server model as well as a stand-alone application. Thus it is possible to host a CartoWeb client (known as CartoClient) on one machine and have it requesting a CartoWeb server (known as CartoServer), located on a separated server. A CartoServer can be called by several CartoClient simultaneously. On the other hand, a CartoClient can query several CartoServer for instance in the frame of different "projects" (Section 3.4.2, "Projects").
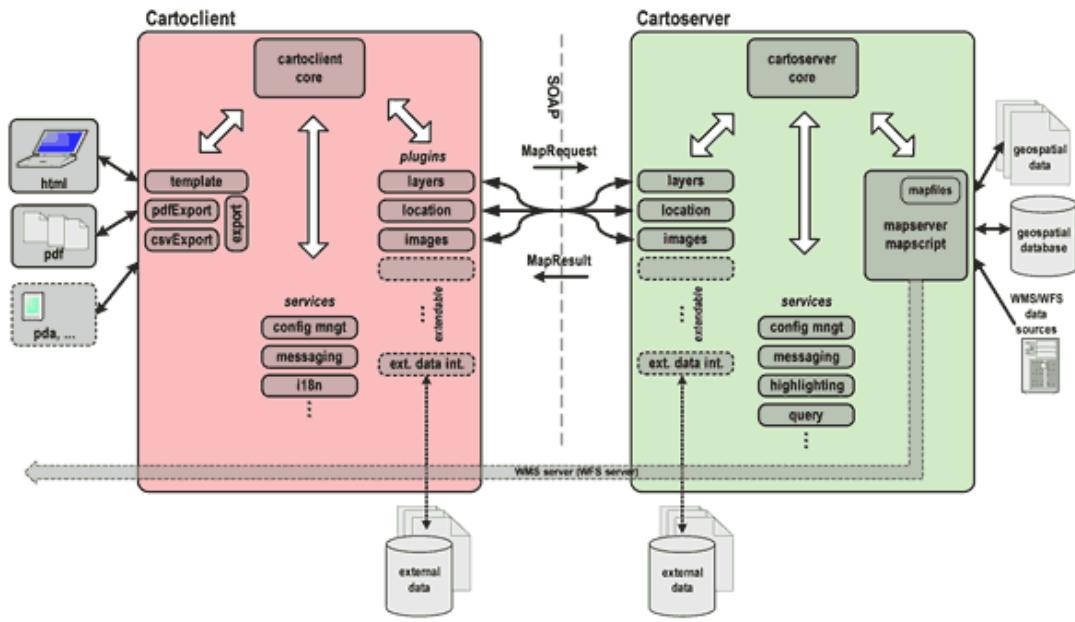
CartoWeb offers two "access" modes :

- as a web-service. CartoClient and CartoServer then interact using remote procedures based upon SOAP [http://www.w3.org/TR/soap12/].
- as a standalone application. Procedures are then performed directly between CartoWeb components, bypassing the SOAP calls.

Multi-client mode
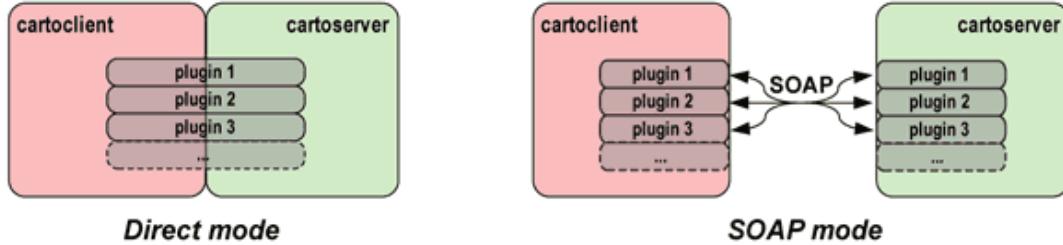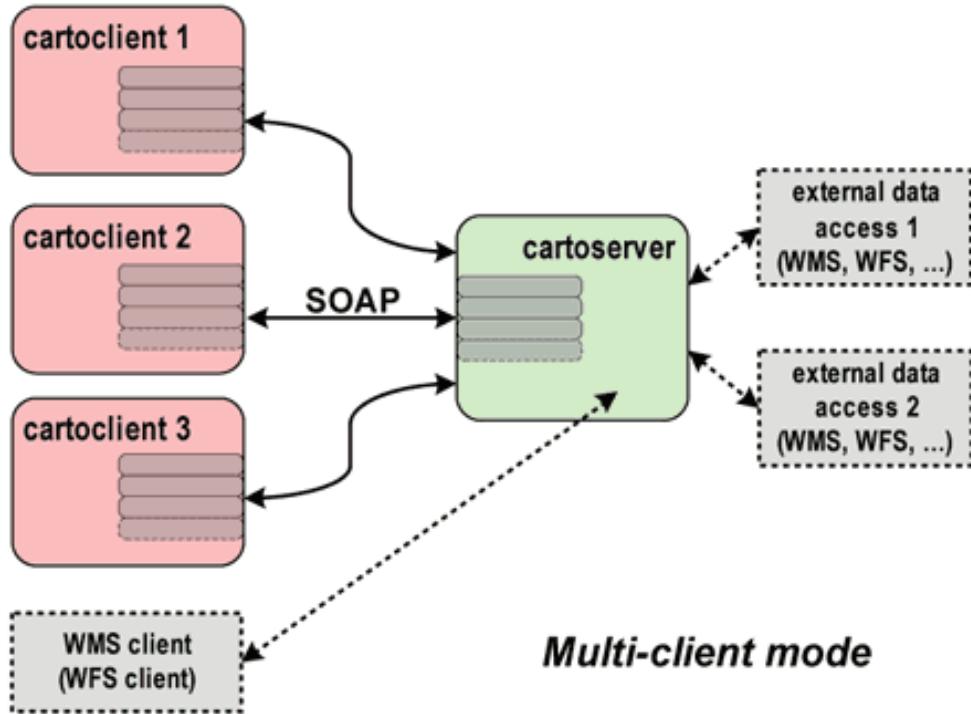
Of course it is possible to combine the above architectures as shown on the following figure:

Composition of architectures
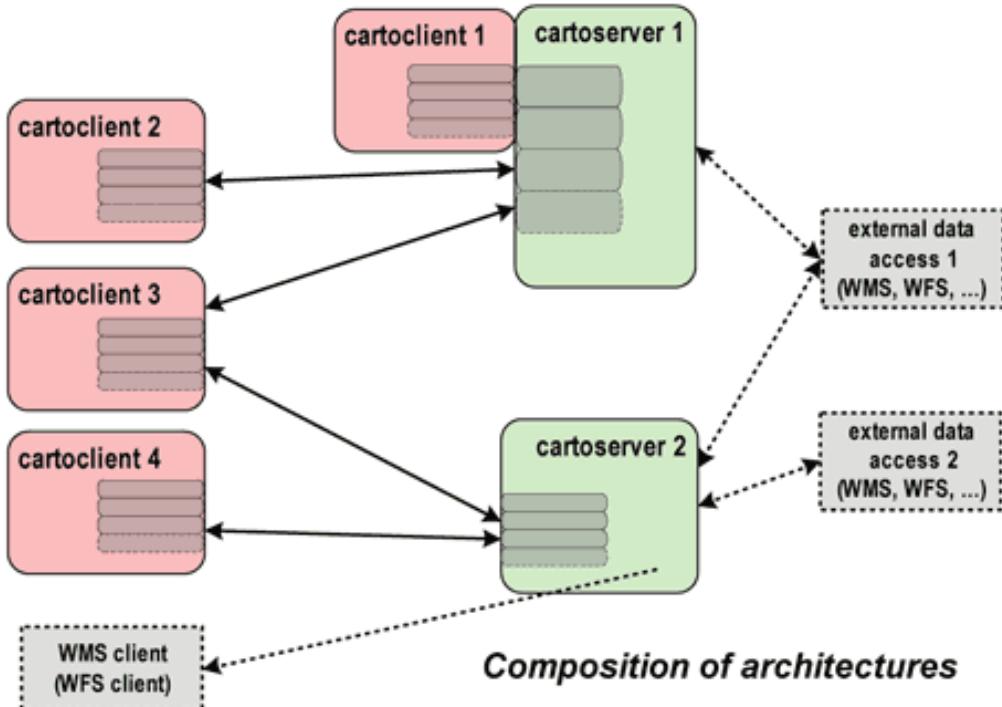
3.4. Modularity: Projects and Plugins

3.4.1. Plugins

CartoWeb buzzword is modularity. The application is built as a set of bricks that interact with each other. Those bricks are called "plugins". Each plugin performs a special group of tasks such as map browsing, layers management, images properties, users permissions, data objects querying, exportation and much more. CartoWeb is shipped with a set of standard plugins that can be separately activated or not. It is possible to write your own plugins or customize existing ones as well.
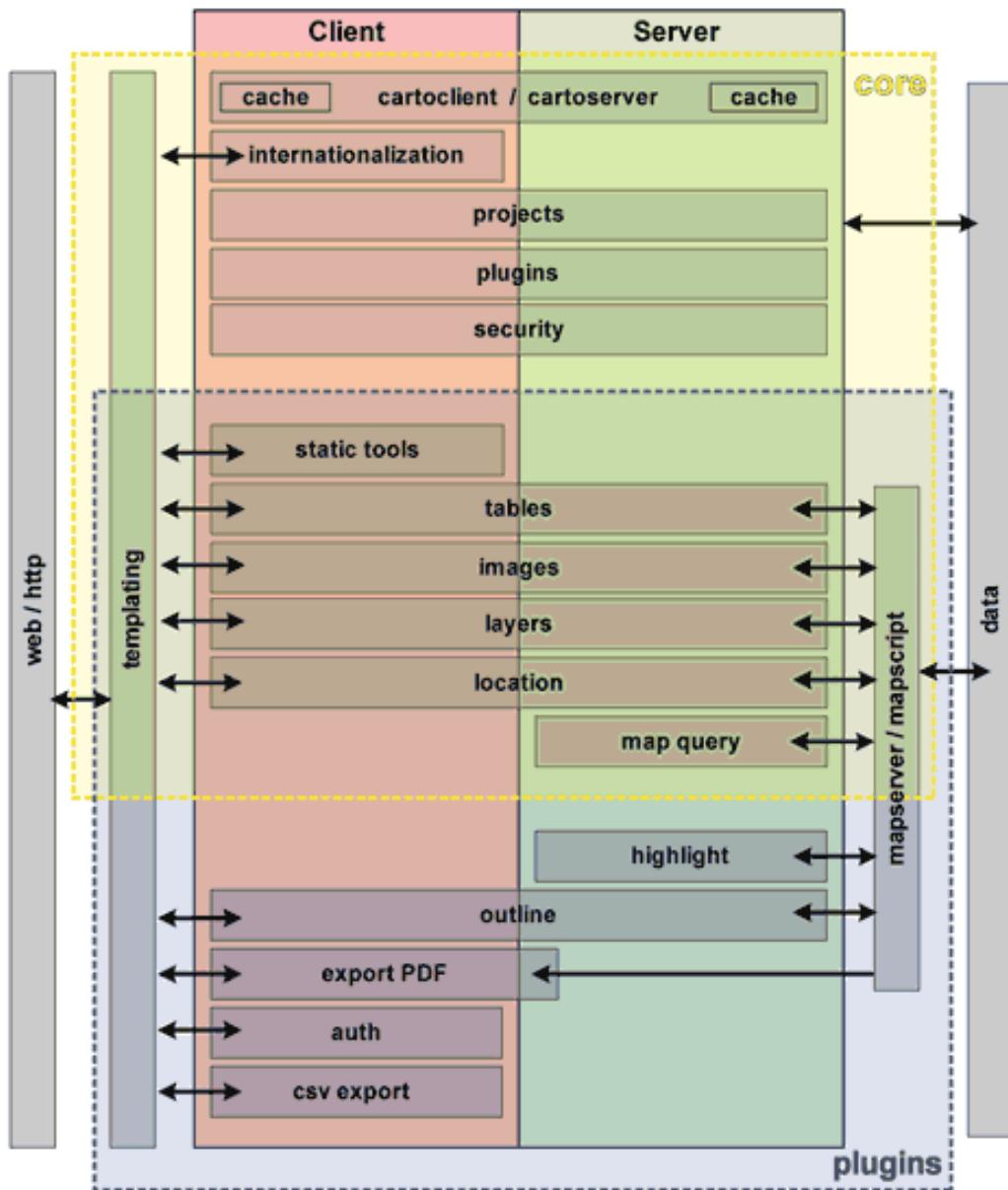
Regular plugins, including how to configure them, are precisely described in Part II, "User Manual". For more information about how to write or customize plugins, see Chapter 2, *New Plugins*.

3.4.2. Projects

Projects are used to separate the upstream application from its customizations. They contains all the modifications and adaptations you could have performed to make CartoWeb suit your needs: layout templates, pictos, special configurations, personal plugins etc.

By using projects you can:

- keep the general application clean from tricky modifications that would compromise the future upgrades
- gather your project files to easily save and copy them to another CartoWeb instance. For example when copying them from your test server to your production one.
- run different projects on the same CartoWeb instance.

For more information about projects, see Chapter 3, *Projects Handling*.

3.5. Templates

CartoWeb layout customization is achieved using the well-known and powerful Smarty [http://smarty.php.net/] template engine. For more information about templating see Chapter 17, *Templating*.

3.6. Users and Roles

CartoWeb enables to define and manage different levels of permissions. Thus it is possible to restrict some (or all) functionalities to some users or categories of users. For more information about users and roles see Chapter 15, *Security Configuration*.

3.7. Performance Enhancement

CartoWeb takes benefits of several caching systems to speed up its execution. All the possibilities are detailed in Chapter 5, *Caches Configuration*.

Part II. User Manual

This user manual, second part of the CartoWeb documentation, is aimed at administrators who have to setup, configure and maintain a CartoWeb environment.

It is not an end-user documentation, so you won't find here instructions on how to navigate, select layers or do whatever operations the distant users are allowed to do through their browser.

On the other side, if you want to go beyond the standard, out-of-the-box possibilities of CartoWeb, you'll definitely have to look at the third part of this documentation, intended for developers.

1. Installation

1.1. CartoWeb Installation

1.1.1. Prerequisite for Installing CartoWeb

CartoWeb depends on several software components for its proper working. Some are required and others are optional, depending on what you intend to do.

Note

Windows users can ignore this chapter and go directly to Section 1.1.2, "CartoWeb Download"

The required software are:

A Web Server

, Such as Apache http://httpd.apache.org

PHP >= 5.0.3

, See http://www.php.net for more informations. You will need to have some features available in PHP:

- *Gettext* (optional): You need the Gettext module if you want to enable the Internationalization in CartoWeb. See Chapter 16, *Internationalization* for configuration.

  Note

  If you are using the demo, you need to have Gettext support installed, as it uses Gettext by default.

  If you are using Cygwin, simply install the *gettext-devel* package.

- *SOAP* (optional if using direct mode only): You need the SOAP module if you want to use CartoWeb as a Webservice or with separated CartoClient and CartoServer. This is related to the `cartoserverDirectAccess` parameter described in Section 4.2, " client.ini "

Note

If you are using Windows, simply use the Windows Installer Section 1.1.2, "CartoWeb Download". If you absolutely want to install PHP manually, see Appendix B, *Apache & Mapserver Windows Manual Installation*.

MapServer PHP/MapScript (from MapServer >= 4.4)

, See http://www.maptools.org/php_mapscript/for more information and installation instructions.

Note

If you are using Debian, and you need to install MapServer, you can have a look at Appendix A, *Mapserver Debian Installation*

Note

If you are using Windows, simply use the Windows Installer Section 1.1.2, "CartoWeb Download". If you absolutely want to install MapServer manually, see Appendix B, *Apache & Mapserver Windows Manual Installation*.

PostgreSQL with PostGIS Support (Optional)

, If you want spatial database support in CartoWeb you will need to install PostGIS of the PostgreSQL database. See http://postgis.refractions.net/for more information.

1.1.2. CartoWeb Download

There are two ways to get CartoWeb:

1. **Complete package from the official website:**
    - ° Download CartoWeb package from http://cartoweb.org/downloads.html. It is recommended that you download the version with demo for a better user experience.
    - ° Uncompress the archive somewhere in your path accessible by your web server.

2. **From CVS:** Get the current development version via CVS using the following command:

```
cvs -d :pserver:anonymous@dev.camptocamp.com:/var/lib/cvs/public co cartoweb3
```

From CVS with cw3setup.php: If you already have the `cw3setup.php` file, which comes along the CartoWeb package or CVS source, you can fetch CartoWeb from CVS and set up it at once. See Section 1.1.3.2.6, "Install or Reinstall CartoWeb from CVS and Set Up It at Once" for more details.

Once you have CartoWeb, point your web browser to the file located in `htdocs/info.php`, and check that the PHP information page displays correctly (meaning PHP is correctly setup) and that you have do not have a WARNING message at the top of the page about MapScript failing to load. If it is not the case, CartoWeb will not run correctly. You should install and set up PHP and PHP/MapScript correctly. See Section 1.1.1, "Prerequisite for Installing CartoWeb".

Having Gettext installed is recommended if you wish to use the multilingual features of CartoWeb.

1.1.3. Command Line Installation

CartoWeb installer is `cw3setup.php`, located in the root directory of the application.

You can run this file with the `--help` parameter to see the available options. For instance:

```
<PHP-INTERPRETER> cw3setup.php --help
```

where <PHP-INTERPRETER> is the location of your php interpreter. On Windows, it can be `C:\ms4w\Apache\cgi-bin\php.exe` or on Unix `/usr/lib/cgi-bin/php`.

Note

There is a deployment mechanism to automate CartoWeb installation and updates. If you are using different servers like: development, tests and production, it is highly recommended to use it: http://www.cartoweb.org/cwiki/AutomaticDeployment/

1.1.3.1. List of Available Options

```
Usage: cw3setup.php ACTION [OPTION_1] ... [OPTION_N]

Possible actions:

 --help, or -h              Display this help and exit.
```

```
--version or -v          Output version information and exit.
--install                Install CartoWeb.
--fetch-demo             Fetch the demo data from cartoweb.org, and extract
                         it in the demo project if not already there.
--clean                  Clean generated files and caches.

List of options:

--debug                  Turn on output debugging.

--writableowner OWNER    The user who should have write permissions for
                         generated files.

--cvs-root               CVS Root directory to use when fetching
                         CartoWeb/project out of CVS.
--fetch-from-cvs         Fetch CartoWeb from CVS and install it in the
                         current directory, or in the directory given by
                         the --install-location parameter.
                         NOTE: You must be located where cartoweb3 directory
                         will be created, not inside like other commands.
--cartoweb-cvs-option OPTIONS  A string which will be given to the cvs checkout
                         command of cartoweb (not projects!).
                         For instance, to fetch a specific branch,
                         use '-r MY_BRANCH'. Or for a specific date,
                         use '-D "2005-09-05 11:00"'.
--fetch-from-dir DIRECTORY Copy CartoWeb from the specified directory into the
                         current directory, or in the directory given by the
                         --install-location parameter.
                         NOTE 1: You must be located where cartoweb3
                         directory will be created, not inside like other
                         commands.
                         NOTE 2: You may either use a path relative to the
                         target cartoweb3 directory or an absolute path.
--install-location       Directory where to install CartoWeb
                         (when using --fetch-from-cvs/dir options).

--delete-existing        Overwrite existing directories if any.
--no-symlinks            Do not use symbolic links, even if your operating
                         system supports them.

--config-from-file FILE  Location of a configuration file for automatic
                         variable replacement in .in files.
                         NOTE: You may either use a path relative to the
                         target cartoweb3 directory or an absolute path.
--config-from-project PROJECT Read the configuration file containing variables
                         to replace in .in files from the specified project.

--fetch-project-cvs PROJECT Fetch the given project from CVS (see --cvs-root
                         option). To fetch several projects at a time,
                         specify this option as many times as necessary.
--fetch-project-svn PROJECT Fetch the given project from SVN (you will need to
                         give --svn-co-options to specifiy the checkout
                         command to be used).
                         To fetch several projects at a time,
                         specify this option as many times as necessary.
--svn-co-options         Checkout command to use for fetching project with
                         SVN. For instance "--username foo --no-auth-cache
                         checkout https://myproject/svn/bar/".
--fetch-project-dir DIRECTORY Fetch the given project from a directory. To
                         fetch several projects at a time, specify this
                         option as many times as necessary.
--project PROJECT        Installation is launched only for given project. To
                         install several projects at a time, specify this
                         option as many times as necessary.

--default-project PROJECT  Default project to use.
--base-url BASEURL       URL where you can find client.php.
--profile PROFILENAME    The profile to use (development/production/custom).
```

19

```
                              NOTE: default is 'development'
--clean-views                 Clean views (must be used with --clean).
--clean-accounting            Clean accounting (must be used with --clean).
--keep-directories            Do not remove the generated directories during
                              cleaning (must be used with --clean).
--keep-permissions            Do not alter the permissions of writable directories.
```

1.1.3.2. Examples of Use

1.1.3.2.1. Basic Setup

To perform a basic setup of CartoWeb, such as if you want to run the demo project, type:

```
<PHP-INTERPRETER> cw3setup.php --install --base-url
          http://www.example.com/cartoweb
```

In this example, http://www.example.com/cartoweb is the address which corresponds to the cartoweb3/htdocs directory. You should find client.php if you type this URL.

1.1.3.2.2. Installing a Project from a Directory

```
<PHP-INTERPRETER> cw3setup.php --install --fetch-project-dir
          /home/foo/my_project --base-url http://www.example.com/cartoweb
```

1.1.3.2.3. Updating CartoWeb after Modifications

When you modify or add new content/features to CartoWeb, you need to update it. This will set relative links or copy new/modified resources files (templates, images, new plugins, ...)

```
<PHP-INTERPRETER> cw3setup.php --install --base-url
          http://www.example.com/cartoweb
```

Same as Section 1.1.3.2.1, "Basic Setup". Existing files are not overwritten.

1.1.3.2.4. Cleaning Generated Files (Map, PDF, Temporary Files and Smarty Cache)

```
<PHP-INTERPRETER> cw3setup.php --clean
```

1.1.3.2.5. Fetching the Demo Data

```
<PHP-INTERPRETER> cw3setup.php --fetch-demo
```

1.1.3.2.6. Install or Reinstall CartoWeb from CVS and Set Up It at Once

1.1.3.2.6.1. Install

```
<PHP-INTERPRETER> cw3setup.php --install --cvs-root
```

```
              :pserver:anonymous@dev.camptocamp.com:/var/lib/cvs/public
              --fetch-from-cvs --base-url http://www.example.com/cartoweb
```

Note

Do no execute this command from the `cartoweb3` folder! Because this will fetch the whole cartoweb3 hierarchy from cvs, including the `cartoweb3` folder. If you executed this from the `cartoweb3` folder you would end up with something like `cartoweb3/cartoweb3/....` Instead, copy the `cw3setup.php` in the parent directory, delete (or backup) the cartoweb3 folder and execute the command.

1.1.3.2.6.2. Reinstall

```
<PHP-INTERPRETER> cw3setup.php --install --cvs-root
              :pserver:anonymous@dev.camptocamp.com:/var/lib/cvs/public
              --fetch-from-cvs --delete-existing
              --base-url http://www.example.com/cartoweb
```

Warning

This command will automatically delete the existing `cartoweb3/` folder! Be sure to backup the files and projects you wish to keep.

Note

See note on Section 1.1.3.2.6.1, "Install". Notice here the *--delete-existing* parameter. Needed here because `cartoweb3/` already exists. Without it the cw3setup script issues a warning and stops.

1.1.3.2.7. Update CartoWeb from CVS

To keep your CartoWeb up-to-date with the development version, simply type the following command in the `CartoWeb` root folder:

```
cvs -d :pserver:anonymous@dev.camptocamp.com:/var/lib/cvs/public update
```

Warning

This may have some serious effects on your existing developments, it is recommended you backup the CartoWeb root folder and all subforders before execution.

1.1.4. Windows Installation

1.1.4.1. Windows Install with Win32 Installer (Recommended)

- Download the MS4W installer at http://www.maptools.org/dl/ms4w/ ms4w-2.2.4-setup.exe. Double-click on the executable to launch it. This will install Apache with PHP5.2.3, 4.10.2 CGI and MapScript.

- Download the CartoWeb installer at http://cartoweb.org/downloads.html. Double-click on the executable to launch it. This will install CartoWeb. You can also optionaly install the cartoweb demo data and Gettext (for internationalisation).

- You need to reboot Windows after the installation if you want to use the internationalisation system with Gettext .

- That's it! Point your browser to http://localhost/cartoweb3/htdocs/client.php.

Once it is installed, you can modify CartoWeb setup with the command-line script `cw3setup.php` (Section 1.1.3, "Command Line Installation") from either a DOS or a Cygwin prompt. See Section 1.1.4.3, "CartoWeb Setup" for more info about how to use those interfaces.

1.1.4.2. Windows Manual Install

Instructions for Apache/PHP and MapServer installation are given in Appendix B, *Apache & Mapserver Windows Manual Installation*

Then download CartoWeb as described above (Section 1.1.2, "CartoWeb Download").

Eventually set CartoWeb up with the command-line script `cw3setup.php` (Section 1.1.3, "Command Line Installation") from either a DOS or a Cygwin prompt. See Section 1.1.4.3, "CartoWeb Setup" for more info about how to use those interfaces.

1.1.4.3. CartoWeb Setup

1.1.4.3.1. CartoWeb Setup with DOS

Open a command prompt (Start menu > Run > "cmd") and go to the CartoWeb root:

```
cd C:\ms4w\apps\cartoweb3
```

Then see Section 1.1.3.2.1, "Basic Setup"

Note

To enable you to execute PHP scripts easily (like *php cw3setup.php*

--*someparameters* instead of `C:\ms4w\Apache\cgi-bin\php.exe cw3setup.php` --someparameters), set the path to the PHP binary in your PATH environment variable (control panel > system > Advanced > Environment Variables. If there is no PATH variable, add a new one. If a PATH variable is already present, you can add the path to php.exe at the end of the existing path values, but add a ";" inbetween: path1;path2;path3):

```
C:\ms4w\Apache\cgi-bin;
```

The example above is true if the PHP binary are installed in `C:\ms4w\Apache\cgi-bin`.

Note

If you are using the demo, you need to have Gettext support installed, as it uses Gettext by default. If you used the win32 installer, Gettext is already installed, otherwise you must install it manually. You can get a version of Gettext for Windows there http://gnuwin32.sourceforge.net/packages/gettext.htm. Also set the path to the Gettext binary in your PATH environment variable.

```
C:\Program Files\GnuWin32\bin;
```

The example above is true if the Gettext binaries are installed in `C:\Program Files\GnuWin32\bin`. This is needed by the `po2mo.php` script to merge and compile the languages translation files.

Note

If you intend to use CVS in command line, you need to install a CVS client. Use WinCVS [http://www.wincvs.org/] or TortoiseCVS [http://www.tortoisecvs.org/download.shtml], both are free Open Source clients. You must add the path to the CVS binary in your PATH environment variable.

```
C:\Program Files\TortoiseCVS;
```

The example above is true if you installed TortoiseCVS in `C:\Program Files\TortoiseCVS`.

1.1.4.3.2. CartoWeb Setup with Cygwin

Open a Cygwin window and go to the CartoWeb root:

```
cd C:
cd ms4w/apps/cartoweb3/
```

Then see Section 1.1.3.2.1, "Basic Setup"

23

Note

You can download Cygwin here *Cygwin* [http://www.cygwin.com/]. When you install Cygwin, be sure to select the packages *tar* (or *unzip*) and *cvs*. You can also install the *gettext-devel* package, so you wont need to get an external gettext installation later. If you have already installed Cygwin, type the following command to see what package are currently installed.

```
cygcheck -c
```

If the packages mentioned above are not present, run Cygwin setup again and add the missing packages.

Note

To enable you to execute PHP scripts easily, set the path to the PHP binary in your `.bashrc` (in `C:\cygwin\home\Administrator\` by default):

```
export PATH=$PATH:/cygdrive/c/ms4w/Apache/cgi-bin
```

The example above is true if the PHP binary are installed in `C:\ms4w\Apache\cgi-bin`.

If you do not want to install the *cvs* and *gettext* Cygwin package, you need to add also the path to the external CVS and gettext binaries.

```
export PATH=$PATH:/cygdrive/c/program Files/GnuWin32/bin
export PATH=$PATH:/cygdrive/c/program Files/TortoiseCVS
```

See the note in Section 1.1.4.3.1, "CartoWeb Setup with DOS"

1.2. Demos

1.2.1. Introduction

A few demos are embedded in CartoWeb to demonstrate the range of functionalities that CartoWeb offers and give users examples on how to implement them:

- *demoCW3*: this is an overview of the standard functionalities that are somehow visible for an end-user in CartoWeb,
- *demoPlugins*: it shows the new functionalities that are available in latest CartoWeb versions,
- *demoEdit*: a simple demonstration of the geographical objects edition and persistent storage tools.
- *demoToolTips*: it demonstrates the usage of the tooltips using pseudo real data. Follow the installation notes below and also the specific database installation in

Section 1.2.5, "ToolTips demo specific installation"

Demo data are freely downloadable. Next section explains how to install them. Configuration and programming details are then described.

1.2.2. Installation

Before installing these demos, you need to have a working CartoWeb installation. You can refer to the previous chapters how to install it.

To install the demoCW3 project, you need to gather the data by launching the *cw3setup.php* with the *--fetch-demo* option.

On the other hand, the demoPlugins project uses plugins that work with databases. Consequently some databases settings and configuration are required. We describe here how to install these databases and how to use *cw3setup.php*.

Step by step guide:

1. **Install PostgreSQL with PostGIS support.** *Prerequisite*: Postgresql >= 8.0

   ### Note

   If you are on Debian, you can have a look at Appendix A, *Mapserver Debian Installation*

2. **Create a PostgreSQL database using the following command:**

   ```
   $ createdb demo_plugins
   ```

3. **Integrate PostGIS functionalities in this database.** Typically, you can type:

   ```
   $ createlang plpgsql demo_plugins
   $ psql -d demo_plugins -f lwpostgis.sql
   $ psql -d demo_plugins -f spatial_ref_sys.sql
   ```

   ### Note

   psql is a terminal-based front-end for PostgreSQL. It enables you to type in queries interactively, issue them to PostgreSQL, and see the query results. Don't forget to specify its location on your system to use it. If the *lwpostgis.sql* and *spatial_ref_sys.sql* files aren't in the current directory, you have to specify their path.

4. **Create tables used by the locate plugin to allow you to do a recenter on features by searching their names.** To do so, you should export the free

downloadable layers *airport, agglo, district* and *town* in PostgreSQL/PostGIS
tables by typing the following command:

```
$ shp2pgsql -I aerofacp.shp airport > /tmp/airport.sql
$ psql -d demo_plugins -f /tmp/airport.sql

$ shp2pgsql -I builtupa.shp agglo > /tmp/agglo.sql
$ psql -d demo_plugins -f /tmp/agglo.sql

$ shp2pgsql -I polbnda.shp district > /tmp/district.sql
$ psql -d demo_plugins -f /tmp/district.sql

$ shp2pgsql -I mispopp.shp town > /tmp/town.sql
$ psql -d demo_plugins -f /tmp/town.sql
```

Note

shp2pgsql is a command-line program that exports a shapefile into SQL
commands. Don't forget to specify its location on your system to use it. You
have to specify the path to the shapefiles if they aren't in the current directory.

5. **Execute cw3setup.php.** file, with the *--fetch-demo* option to download
geographical data.

6. **cw3setup.php Options.** To finish demoPlugins installation, you will need to
launch the `cw3setup.php` with the *--config-from-file* parameter pointing to a
property file containing database configuration informations. Such a file is
provided in the `cartoweb3/projects/demoPlugins/demo.properties` file.
You need to edit this file and change the parameter to match your environment.
In particular the *DB_HOST*, *DB_USER*, *DB_PASSWD* and *DB_PORT* options
need to match your database access configuration. *DB_LOCATE_NAME* must
be set to the name of the database for the locate plugin. This file contains
comments about the description of each variables. Here's an example how to call
the *cw3setup.php* script with the *--config-from-file* option.

```
php cw3setup.php --install --base-url http://www.example.com/cartoweb
              --config-from-file projects/demoPlugins/demo.properties
```

Note

Routing fonctionnalities are also integrated in this demo. But they need a more
advanced configuration and the PgRouting module installed in the database, so they
aren't integrated in the basic installation. The steps to integrate routing
fonctionnalities and create database tables are described in a dedicated chapter of the
User Documentation: Chapter 26, *Routing Plugin*

1.2.3. Routing specific installation

Routing installation instructions have moved to a dedicated chapter in the User Documentation: Chapter 26, *Routing Plugin*

1.2.4. Locate Plugin specific installation

This plugin allows user to search features by their names, and recenter on them.

By typing the first letters of a name, user is given a list of corresponding names. Clicking on one of them recenters the map on it.

Note

In the `locate.ini`, you also need to specify the layers on which you want to do a search by name and the sql statements to request the database.

Tip

Ensure that the tables have the correct privileges (GRANT SELECT) in the database.

1.2.5. ToolTips demo specific installation

Tip

The database used to store data is the same as for the demoPlugins

- **Create the tables for layers radio and gsm.**

  ### Note

  To do so, convert the radio and gsm data from shapefile to postGIS tables. The shapefiles should be available in `projects/demoCW3/server_conf/demoCW3/data` where all demos data are.

  ```
  $ shp2pgsql -I radio.shp radio > /tmp/radio.sql
  $ psql -d demo_plugins -f /tmp/radio.sql

  $ shp2pgsql -I gsm.shp gsm > /tmp/gsm.sql
  $ psql -d demo_plugins -f /tmp/gsm.sql
  ```

- **Create the table for channel data.**

  ```
  $ psql -d demo_plugins -f channels.sql
  ```

The `channels.sql` is located in the `projects/demoToolTips/server_conf/` `demoToolTips/sql` folder

• **Launch the install script in your CartoWeb root directory.**

```
php cw3setup.php --install --base-url <base-url> --profile development
                 --project demoToolTips --config-from-file projects/demoPlugins/demo.properties
```

1.2.6. Plugins extensions further information

The aim of this part is to give you further informations on creation of demoPlugins and demoEdit plugins extensions. If you create a new plugin or adapt an existing one, we guess it will be helpful to take these plugins extensions for example or use one of these two resources:

• User HowTo: http://www.cartoweb.org/cwiki/HowTo,

• Dev tutorial new plugins part: Chapter 2, *New Plugins*.

1.2.6.1. demoRouting Extension

Routing installation instructions have moved to a dedicated chapter in the User Documentation: Chapter 26, *Routing Plugin*

1.2.6.2. demoEdit Extension

Some useful tips are available in Section 14.3, "How To".

2. Structure

2.1. Introduction

This chapter is an overall tour of the CartoWeb code structure. It briefly explains the role of each directory. When available, links to relevant chapters of this documentation are also provided.

2.2. Global Directory Structure

After installation, CartoWeb has the following directory structure:

* `client`: Client specific code files
* `client_conf`: Client configuration files, see Chapter 4, *Configuration Files*.
* `common`: Common client and server code files
* `coreplugins`: Basic mandatory plugins
    * `images`: Image generation, see Chapter 8, *Image Format Options*
    * `layers`: Layers management, see Chapter 6, *Layers*
    * `location`: Navigation, see Chapter 7, *Navigation*
    * `mapquery`: Perform queries based on a set of selected id's, see Section 9.2.2, "MapServer Query Configuration"
    * `query`: Perform queries on layers, see Chapter 9, *Queries*
    * `statictools`: Distance and surface calculation.
    * `tables`: Table rules management, see Section 9.1.2, "Tables Configuration"
* `documentation`: documentation
    * `apidoc`: PHP source code documentation
    * `user_manual/source`: DocBook XML source of the present documentation
* `htdocs`: Web accessible directory
    * `css`: css files
    * `gfx`: icons files
    * `js`: javascript files
* `include`: libraries used by CartoWeb
* `locale`: locale files for internationalization purposes, see Chapter 16, *Internationalization*
* `log`: logs, mainly used for development and debug purposes.
* `plugins`: Standard, but not mandatory plugins, see Section 2.3, "Plugins"
    * `auth`: authentication plugin, see Chapter 15, *Security Configuration*

- o `exportCsv`: Csv export plugin, see Section 11.3, "CSV Export"
- o `exportHtml`: HTML export plugin, see Section 11.2, "HTML Export"
- o `exportPdf`: PDF export plugin, see Chapter 12, *PDF Export*
- o `hello`: test plugin
- o `hilight`: hilight plugin, see Section 9.3.3, "Hilight Configuration"
- o `outline`: redlining and annotations, see Chapter 10, *Annotation and Redlining*

- `po`: PO templates files, used for gettext translation system, see Section 16.1.2, "PO Templates"
- `projects`: CartoWeb user projects dir, see Section 2.4, "Projects"
- `scripts`: maintenance and administration scripts
- `server`: CartoWeb server code files
- `server_conf`: Cartoweb server-side configuration files, see Section 4.3, "Server Configuration Files"
- `templates`: CartoWeb Smarty templates files, see Chapter 17, *Templating*
- `templates_c`: smarty templates cached files
- `tests`: CartoWeb unit tests suite, mainly used for development and debug purposes
- `www-data`: writable and web accessible directories for generated files
  - o `icons`: Generated icons
  - o `images`: Mapserver images
  - o `mapinfo_cache`: Client-side server configuration cache, see Section 5.6, "Caches Configuration"
  - o `mapresult_cache`: Client requests and associated server results cache, see Section 5.6, "Caches Configuration"
  - o `pdf`: Pdf generated cache files
  - o `soapxml_cache`: Client SOAP XML requests and associated server results cache, see Section 5.6, "Caches Configuration"
  - o `wsdl_cache`: Client-side WSDL cache, see Section 5.6, "Caches Configuration"

2.3. Plugins

Modularity is a key feature of CartoWeb. Its functionalities are packaged in logical sets called plugins, that aim to be independent from each other, although some dependencies cannot be totally avoided. Some plugins (core plugins) cannot be disabled, while the other ones must be explicitly loaded in the server and/or client configuration files.

Nearly all plugins have configuration options set in .ini files. The full description of these options makes the bulk of this user manual.

Modifying existing plugins or writing new ones requires some acquaintance with PHP5, as it involves some coding. The related documentation is thus reported to the developer's part of this manual (see Section 2.1, " What are Plugins"), but that shouldn't deter anybody from experimenting with it.

Each plugin directory contains one or more subdirectories. Here are all the possible subdirectories:

- `client`: Client-side plugin code
- `common`: Client and server code
- `htdocs`: Web accessible directory
- `server`: Server-side plugin code
- `templates`: Smarty templates

2.4. Projects

The aim of projects in CartoWeb is to clearly separate mainstream files from project-specific files. Developers should thus only work in projects, and not modify/add/delete files in the root directory. This will ensure smooth updates.

The directory `/projects/my_project` has exactly the same structure as the root directory shown above: Section 2.2, "Global Directory Structure"

Files added in directory `/projects` override corresponding files of the root directory. For instance, if you want to change the layers template (i.e. basically the representation of the layers hierarchy), simply copy the default `/coreplugins/layers/templates/layer.tpl` to `projects/my_project/coreplugins/layers/templates/layer.tpl` and make your changes there.

For more information about projects, see Chapter 3, *Projects Handling*.

3. Projects Handling

3.1. Introduction

Projects are used to customize a CartoWeb application to your needs. By creating a new project you can override templates, resources files (pictures, style sheets, JavaScript files, etc.), configuration files and even add new plugins or modify existing ones.

It is strongly recommended to use projects when deploying a CartoWeb application with non-standard layout or plugins. The main reason is the necessity to keep upstream files unchanged in order to easily apply the application upgrades.

Projects are in fact a mirrored collection of directories and files from the original architecture. Files placed in a project are in most cases used preferentially to the original files. There is an exception with plugins PHP classes: the latter must be extended and not simply overridden. In projects you can also add brand new files (for instance new plugins) that have no original version in the upstream structure. Fore more details about how to write or customize plugins in projects, see Chapter 2, *New Plugins* in Part III, "Developer Manual".

Note that you don't need to duplicate the whole CartoWeb structure in your projects. Only directories that contain overriding files have to be created. In .ini files, only variables set in projects are overridden. Other variables keep the values set in upstream .ini files.

Following files can be "overridden":

- `client_conf/*.ini` (client.ini and plugins configuration files)
- `[core]plugins/*/client/*.php`
- `[core]plugins/*/common/*.php`
- `[core]plugins/*/server/*.php`
- `[core]plugins/*/htdocs/*.php`
- `[core]plugins/*/templates/*.tpl`
- `htdocs/css/*.css`
- `htdocs/js/*.js`
- `htdocs/gfx/layout/*.gif`
- `server_conf/server.ini`
- `server_conf/<mapId>/*.ini` (<mapId>.ini and plugins configuration files)
- `templates/*.tpl`

You can add project-specific mapfiles in directory `/projects/my_project/server_conf/my_mapfile`. To point to the new mapfile, change the mapId value in `/projects/my_project/client_conf/client.ini`.

You can add project-specific plugins in directory `/projects/my_project/plugins`. To load the new plugin, add its name in `client.ini` and/or `my_mapfile.ini` (loadPlugins variable).

3.2. Using Projects

There are several ways to tell CartoWeb what project to use:

3.2.1. Apache Environment Variable

Set environment variable CW3_PROJECT in Apache configuration.

```
<Directory /your/cartoclient/path/>
 Options FollowSymLinks
 Action php-script /cgi-bin/php5
 AddHandler php-script .php

 # [...]

 SetEnv CW3_PROJECT your_project_name
</Directory>
```

Warning: You will need Apache's Env module to use SetEnv command. To load this module, add the following line to your Apache configuration:

```
LoadModule env_module /usr/lib/apache/1.3/mod_env.so
```

3.2.2. Using current_project.txt

Add a file named `current_project.txt` in CartoWeb root directory. This file must contain a single line with project name.

3.2.3. Using a GET Parameter

You can pass a GET parameter *project=YOUR_PROJECT* to the `client.php` script, for instance:

```
http://path.to/cartoweb/client.php?project=myProject
```

3.2.4. Using the Projects Drop-down List

Have a look at the configuration of `client.ini` described in Section 4.2, " client.ini ", in particular directives *showProjectChooser* and *availableProjects*, to display the project selection drop-down menu.

If *showProjectChooser* is true, a dropdown menu will appear in your CartoClient interface, giving the list of all projects available in your `/projects/` directory. Selecting one will make it the active one. Your choice is propagated from page to page. Note that if the selected project has *showProjectChooser* set to false, the project selection dropdown will no more appear, keeping you from activating another project. To go back to the initial project, call the initial `client.php` page without posting the HTML form.

3.2.5. Using a Modified client.php

This should be avoided in production, but may be useful in development if you have to frequently switch the working project: add a new file `client_myproject.php` in the root htdocs directory. This file only sets the environment variable and then calls the normal `client.php`. Each project has so its own URL.

```php
<?php
  $_ENV['CW3_PROJECT'] = 'myproject';
  require_once('client.php');
?>
```

4. Configuration Files

When installing CartoWeb, the administrator of the application may want to adapt it to the environnement use. This can be easily done using configuration parameters.

Some are required and CartoWeb won't correctly work if they're not set. Others are optional but could hardly change the application behavior.

You will also find specific config parameters in the plugins related chapters of this documentation.

4.1. Common client.ini and server.ini Options

Common options for both client and server. These parameters are available in `client_conf/client.ini` for client and `server_conf/server.ini` for server.

- profile = development|production|custom: current settings profile. The `development` profile has no cache activated and makes SOAP work in WSDL mode. The `production` profile has all caches activated for best performances. Setting this parameter to development or production will override some parameters in the configuration. The custom profile means the parameters about caching and other are not overrided.
- useWsdl = true|false: if true, WSDL will be used for sending SOAP requests. This will add some processing time but ensures that SOAP requests are well-structured. In addition, WSDL is necessary when interfacing the Web Service with a strong-typed language built application.

Cache options. See Section 5.6, "Caches Configuration".

Developer options. See Section 4.5, "Developer Specific Configuration".

4.2. client.ini

CartoServer access configuration:

- cartoserverDirectAccess = true|false: toggles between SOAP and direct modes. Direct access gives enhanced performances, but is only available if CartoServer runs on the same server as CartoClient.
- cartoclientBaseUrl: base URL of the CartoClient
- cartoserverBaseUrl: base URL of the CartoServer (i.e. path containing the `cartoserver.wsdl.php` file)

Mapfile configuration:

- mapId = string
- initialMapStateId = string. Tells what *initialMapState* to use when creating a new CartoWeb session (see also Section 4.3.3.3, "Initial Mapstates").

Tip

initialMapStateId may be determined using several means. By order of priority:

1. URLs (using GET parameter *initialState*)
2. PHP environment variables. Place

   ```
   $_ENV['CW3_INITIAL_MAP_STATE_ID'] = 'foobar';
   ```

   in your PHP code before calling CartoWeb.
3. client.ini
4. If no *initialMapStateId* can be determined, CartoWeb use first *initialMapState* available.

Session handling:

- sessionNameSuffix = string. Optional suffix used to distinguish CartoClient sessions parts for a given user. In most cases it is not necessary but may be needed when using, for instance, "sub-projects". This string is an ulimited comma-separated list of *type*:*value* couples. Three types are available:
  - *str*: a constant string
  - *conf*: uses value of one of the current client.ini configuration parameters
  - *env*: uses value of given PHP environment variable, if set.

  For instance:

  ```
  sessionNameSuffix = "str:toto, conf:profile, env:myEnvVar"
  sessionNameSuffix = "str:foo, str:bar"
  sessionNameSuffix = "env:CW3_APP_ID"
  ```

Tools configuration:

- initialTool: indicates which tool is activated when in initial state. If not specified, the first tool in the toolbar is activated. Possible values are: zoomin, zoomout, pan, query, distance, surface.
- toolbarRendering: indicates how to render tools selectors icons within toolbar. Possible values are 'radio' (default), 'outline', 'swap'. If 'radio': an HTML radio

button appear nexts to each tool icon. If 'outline': active tool icon is outlined with a colored border. If 'swap': an alternate icon ("over") is displayed to mark the active tool.

Project handling configuration:

- showProjectChooser = true|false: Shows a drop-down list for selecting the active project.
- availableProjects = list: List of the project to show in the drop down list. If not set, all projects found will be used.

Plugins configuration:

- loadPlugins = list: list of client plugins to load in addition to the core plugins. Note that most client plugins also have a corresponding server plugin that must be loaded on the server side. See Section 4.3.3.2, " <myMap>.ini ".

Internationalization:

- I18nClass is now **deprecated** See Chapter 16, *Internationalization* for a description of the internationalization options and the corresponding configurations.
- defaultLang = string: default language, possible values are the usual ISO locale codes (en, fr, de ...)
- langList = list: list of usable languages. Used to order and restrict the available languages. In case you have many languages in directory <cartoweb_home>/locale/ and do not want to use them all in your project, you may use this parameter to specify a list of languages to use. The order in which you input the languages codes will be used to display the languages links in the interface.

Example 4.1. langList usage

```
langList = it,de,fr
```

The exemple above restricts the availablelanguages to it, de and fr. They will be displayed in that order in the interface languages list.

ToolPicker:

- toolPickerOn = true|false: enable the ToolPicker dhtml. See Section 10.5, "The ToolPicker".

4.3. Server Configuration Files

4.3.1. Introduction

This page describes the configuration options of the CartoServer. There is a global configuration file (`server.ini`) directly in the `server_conf` folder. Then all specific configurations are stored in individual folders. Each configuration contains:

- a Mapserver mapfile (`myMap.map`),
- its annexes (symbols, fonts, images, data...),
- a main configuration file (`myMap.ini`) that must have the same name as the `.map`
- smaller configuration files for the plugins.

By default, CartoWeb comes with a fully functional `test` folder, that includes the necessary geometrical datas and allows one to run an out of the box demo.

4.3.2. Main Server Configuration File (server.ini)

- imageUrl = string: Path where cartoserver generated images can be accessed.
- reverseProxyUrl = string: The url of the reverse proxy, if used.

4.3.3. Map Configuration Files

4.3.3.1. Introduction

The CartoServer has the ability to contains several different maps. These maps are represented by the mapserver mapfile, the CartoWeb configuration file for the map and each plugins configuration.

The file that contain the configuration information related to a map, is located in the same directory as the mapfile, but has a `.ini` extension. These files are in the directory `server_conf/<myMap>`.

4.3.3.2. <myMap>.ini

- mapInfo.loadPlugins = list: list of server plugins to load in addition to the core plugins. Note that most server plugins also have a corresponding client plugin that must be loaded on the client side. See Section 4.2, " client.ini ".

- mapInfo.initialMapStates.[...]: See Section 4.3.3.3, "Initial Mapstates".

4.3.3.3. Initial Mapstates

Initial map states set the initial aspects of the layers selection interface when starting using CartoClient: (un)folded nodes, selected layers... Some of these properties are not modifiable in the layers selection interface (hidden layers for instance) and thus stay unchanged throughout the session.

Several initial map states can be created in `myMap.ini` , but at least one must be present. Each one is identified by a unique initialMapStateId. The choice to activate one or another is done client-side in `client_conf/client.ini`.

Available properties and syntax for layers in "initial map states" are:

- mapInfo.initialMapStates.initialMapStateId.layers.layerId. selected = true|false: if true, layer is initially selected.
- mapInfo.initialMapStates.initialMapStateId.layers.layerId. unfolded = true|false: if true, the layerGroup is represented as an unfolded node (children layers are visible).
- mapInfo.initialMapStates.initialMapStateId.layers.layerId. hidden = true|false: if true, this layer and its children are not shown in the layers list (but are still displayed on the map if they're activated).
- mapInfo.initialMapStates.initialMapStateId.layers.layerId. frozen = true|false: if true, this layer (and its children as well) is listed in tree but without checkbox. Its selection status (defined by "selected" property) thus cannot be changed.

Example of Initial MapState:

```
mapInfo.initialMapStates.default.layers.polygon.selected = true
mapInfo.initialMapStates.default.layers.point.hidden = true
...

mapInfo.initialMapStates.map25.layers.polygon.selected = true
mapInfo.initialMapStates.map25.layers.polygon.unfolded = true
```

4.4. Ini Files for Plugins

Each plugin may have a configuration file associated with it. It may be found in `client_conf/` or `server_conf/your_project_name/` directory depending which side needs to access the parameters. They have the same name as the plugin and ends with `.ini` extension. For instance, the `layers` plugin has a configuration file named `layers.ini`.

All plugins configuration files are described in the next sections of this chapter.

Every plugin that provides tools icons can be configured for:

4.4.1. Ordering Tools Icons

To modify the toolbar icons sequence, update the tools weight values. Icons with the lowest weights are placed first in the toolbar. Parameters naming convention is to concatenate the 'weight' prefix to the capitalized-first-lettered tool name as in `weightDistance`. For two-word tool names (eg. 'exemple_tool'), each word has its first letter capitalized and the '_' (underscore) is removed. For instance:

```
exempleTool = something
```

Default weights value are :

- weightZoomin: 10
- weightZoomout: 11
- weightPan: 12
- weightFullextent: 14
- weightQueryByPoint: 40
- weightQueryByBbox: 41
- weightQueryByPolygon: 42
- weightQueryByCircle: 43
- weightOutlinePoint: 70
- weightOutlineLine: 71
- weightOutlineRectangle: 72
- weightOutlinePoly: 73
- weightDistance: 80
- weightSurface: 81

Use a negative value to deactivate a tool.

```
weightFullextent = -1
```

4.4.2. Grouping Tools Icons

CartoWeb offers the possibility to display tools icons in separated locations of its interface. For instance browsing icons may be displayed in the general toolbar whereas outline pictos may only appear in a dedicated folder. To do so, simply add an integer group index for each tool in matching plugin configuration file. For instance:

```
groupDistance = 2
```

By default, all tools icons are affected to group 1. Note that the parameters naming convention is the same than for weight parameters (see Section 4.4.1, "Ordering Tools Icons"), only substituting the 'group' prefix to the 'weight' one.

It is also possible to specify a global plugin group index that will be applied to all the given plugin tools if no individual group index has been set. To do so, use the *groupPlugin* in the matching plugin configuration file as follows:

```
groupPlugin = 3
```

To take benefit of the tools groups, update the main CartoWeb template file `templates/cartoclient.tpl` by including the `toolbar.tpl` template at the correct places with the correct group indexes as follows:

```
{include file="toolbar.tpl" group=2}
```

For more information about templates writing, see Chapter 17, *Templating*.

4.5. Developer Specific Configuration

Some configuration parameters can be activated to retrieve more display information targetted to the developers, like special timing messages, or setting the Php configuration to display notices on the page. These configuration options are described below.

Warning

These parameters will be overrided by the `profile` parameter (See Section 4.1, "Common client.ini and server.ini Options").

These parameters are available in `client_conf/client.ini` for client and `server_conf/server.ini` for server.

- showDevelMessages = true|false: Shows developer messages
- developerIniConfig = true|false: Sets ini parameters useful during development

5. Caches Configuration

Several different caching mechanisms are available in CartoWeb for maximum performances. The different types of caches are described in the next chapters.

5.1. Smarty Cache

The templating system used in CartoWeb is Smarty[1], which offers two level of caching for templates. One is compilation of templates into the `templates_c` directory, and the other is static caching of templates. In CartoWeb, because pages are very dynamic, only the first level of caching is used.

The caching feature of Smarty is totally transparent to the user. However, an option can be set in the `client_conf/client.ini` configuration file to enhance performances.

* smartyCompileCheck = true|false: Set this to false in production to improve performance. *Warning:* This parameter will be overridden by the `profile` parameter (See Section 4.1, "Common client.ini and server.ini Options").

Warning

Setting this option to `false` means that your template won't be updated any more. Pay attention to this if you need to change them on the server.

5.2. WSDL Cache

When using SOAP, and the `useWsdl` option is set to `true` in the `client_conf/client.ini` or `server_conf/server.ini`, the WSDL generated document can be cached for more performance. This is the purpose of this cache option. So it should be activated in production environment, and turned off during development, if your changes have an impact on the WSDL.

5.3. MapInfo Cache

MapInfo is a structure generated on the server and used by the cartoclient to access static server information. This cache keeps a copy of the MapInfo structure on the

---

[1]http://smarty.php.net/

client or the server, so that the client does not need ask it everytime, and the server can avoid regenerating it from scratch.

5.4. MapResult Cache

This is a server side only cache, which caches the requests made to the server. This cache works at the Php level, meaning that it can be used when CartoWeb is used in direct or SOAP mode. For the distinction between these two modes, see Section 4.2, " client.ini ". However, the XML SOAP cache is more appropriate when using SOAP mode, see details in the next chapter.

5.5. XML SOAP Cache

This one is also a server side only cache, caching requests at the lowlevel XML SOAP exchange. This means this cache is only effective when direct mode is not used, and can be used for any webservice for which the output only depends on the input arguments.

5.6. Caches Configuration

5.6.1. Rationale

All cache configuration name are in the form `noXXXName`, where `XXX` is the name of the cache. This is so, so that if the parameter is not available, its default value will be false, meaning that all caches are active by default, for maximum performance.

Warning

These parameters will be overrided by the `profile` parameter (See Section 4.1, "Common client.ini and server.ini Options").

5.6.2. Client and Server Cache Options

These parameters are available in `client_conf/client.ini` for client and `server_conf/server.ini` for server.

- noWsdlCache = true|false: disables the caching of wsdl (ignored if useWsdl is false)
- noMapInfoCache = true|false: disables the caching of MapInfo requests

5.6.3. Server Cache Options

These parameters are available in `server_conf/server.ini`.

- noMapResultCache = true|false: disables the caching of getMap requests
- noSoapXMLCache = true|false: disables the caching of SOAP XML requests

6. Layers

6.1. Introduction

Geographical data are most often apprehended as thematic layers: you may have a layer "Rivers", a layer "Aerial view", a layer "Average income", just to cite a few examples. Now the basic purpose of any viewer is to make these data available to users by allowing navigation within a layer as well as comparison between layers. A way to organize the layers is thus mandatory if an user-friendly application is to be developed. In CartoWeb, the files that contain the configuration instructions for the layers are on the server-side, in the directory `server_conf/<myMap>`. Usually this directory is part of a project.

CartoWeb is based on the geographical engine Mapserver. The Mapserver documentation http://mapserver.gis.umn.edu/doc.html [http://mapserver.gis.umn.edu/doc.html] is an integral part of the CartoWeb doc. To be concise, you have to know how to write a mapfile if you want to use CartoWeb. So it doesn't come as a surprise that a mapfile, `myMap.map`, is included in the `<myMap>` directory, together with its annexes (typically a symbol file `myMap.sym`, a directory `etc` for the required fonts, the graphic file used as keymap background, maybe also data files).

We'll see that some functionalities of CartoWeb require small changes of the mapfile content. But most of the configuration is done in the file `layers.ini`.

The file `myMap.ini` sets the required plugins and the initial state of the map. Its content is described in Chapter 4, *Configuration Files*

6.2. Hierarchy of Layers and Rendering

Contrary to Mapserver itself, CartoWeb supports an arbitrarily complex hierarchy of layers and different rendering options. The notion of LayerGroup makes it possible.

6.2.1. Layers and LayerGroups

There are two types of "layers-like objects" in `layers.ini` : Layers and LayerGroups. They play fairly different roles and consequently have different possible attributes. Layers have a 1-to-1 correspondance to Mapserver layers (as defined in the `layers.map`), while LayerGroups group together atomic Layers or other LayerGroups.

6.2.2. Layers Options

As seen before, the Layer object maps directly to a layer in the mapfile. By default, all layers in the mapfile are made available as a Layer in the layers.ini, with an identifier having the same name as the name of the mapserver layer. Thus, if you have the following option in your mapfile:

```
LAYER
  NAME "my_layer"
  TYPE line
END
```

This is equivalent as writing the following configuration in the `layers.ini`:

```
layers.my_layer.class = Layer
layers.my_layer.label = my_layer
layers.my_layer.msLayer = my_layer
```

Tip

If you don't need special parameters (see below) for your layer then you can avoid adding it in the `layers.ini`

However, if you want these layers to appear in the layer list, you still have the responsibility to affect them a parent layer, using the `children` property of the `LayerGroup` layer declarations.

Here is the syntax for the various configuration parameters of a Layer.

- layers.layerId.className = Layer : defines the object as a Layer; layerId is a string that uniquely identifies the object. The general rules of syntax for a .ini file must be respected in the choice of the layerId (e.g. no '-' are allowed).
- layers.layerId.msLayer = string : name of the corresponding Mapserver layer in the mapfile
- layers.layerId.label = string : caption of the layer in the layer tree on the client; this is a 'raw' label, before any internationalization. The i18n scripts automatically include this label in the strings that can be translated.
- layers.layerId.icon = filename : filename of the static picto that illustrates this Layer in the layer tree. The file is expected to be in `server_conf/<mapId>/icons` and a setup script launch is in most cases required. See also Section 6.4, "Layers Legends" for a description of the automatic legending process.
- layers.layerId.link = url : provides a link for the layer (e.g. to some metadata); makes the caption in the tree clickable.

6.2.3. LayerGroups Options

Here is the syntax for the various configuration parameters of LayerGroups. Note that a special LayerGroup with layerId=root must be present. Unsurprisingly, it is the root (top level) of the hierarchy. It doesn't appear in the visible tree.

- layers.layerId.className = LayerGroup : defines the object as a LayerGroup; layerId is a string that uniquely identifies the object. The general rules of syntax for a .ini file must be respected in the choice of the layerId (e.g. no '-' are allowed).
- layers.layerId.children = list of layerIds : comma-separated list of layerIds; these children may be Layers or other LayerGroups.
- layers.layerId.children.switchId = list of layerIds : comma-separated list of layerIds, when using switching (see Section 6.2.4, "Children Switching"). This option cannot be used together with simple children option.
- layers.layerId.aggregate = true|false : if true, the children objects are not listed in the tree and not individually selectable. Default is false.
- layers.layerId.label = string : caption of the layer in the layer tree on the client; this is a 'raw' label, before any internationalization. The i18n scripts automatically include this label in the strings that can be translated.
- layers.layerId.icon = filename : filename of the static picto that illustrates this Layer in the layer tree. The path is relative to `myMap.ini`. See also Section 6.4, "Layers Legends" for a description of the automatic legending process.
- layers.layerId.link = url : provides a link for the layer (e.g. to some metadata); makes the caption in the tree clickable.
- layers.layerId.rendering = tree|block|radio|dropdown : indicates how to display the LayerGroup children.
  - tree (default value): children layers are rendered below the LayerGroup with an indentation. If children are not declared as "frozen" or "hidden" they will be preceded by a checkbox input. A node folding/unfolding link is displayed before the LayerGroup.
  - radio: quite similar to the "tree" rendering with "radio" buttons replacing checkboxes. Only one child layer can be selected at a time.
  - block: children layers are separated as blocks (separation medium depends on the template layout). Note that the rendering will be applied to the children, not to the LayerGroup itself, which is not displayed at all.
  - dropdown: as for block rendering, the LayerGroup is not displayed. Its children are simply rendered as an HTML "simple select" options list. If the selected child layer cannot be determined using posted or session-saved data, first child (according to the layers.layerId.children list order) is selected. If any, only the selected child layer's own children are displayed under the dropdown list.

- layers.layerId.metadata.minScale (resp. maxScale): one may specify a numerical minimal (resp. maximal) scale for the given LayerGroup. This parameter has absolutely no effect on Mapserver behavior and is only used to display out-of-scale icons and links in the layers tree. For more info about `layers.ini` metadata parameters, see Section 6.3.2, "Metadata in layers.ini".

6.2.4. Children Switching

Children switching is useful to define several lists of children for one layer group. The children that will be displayed is choosen by a special parameter : the switch ID.

Switch IDs and labels are listed in file `layers.ini`. In default test environment, a dropdown is constructed using this list and displayed in the GUI.

```
switches.switch1.label = Switch 1
switches.switch2.label = Switch 2
switches.switch3.label = Switch 3
```

Then, each layer group that needs to change its children depending on the switch ID may add this ID as a suffix to the children option.

```
layers.layerId.className = LayerGroup
layers.layerId.children.switch1 = layer1, layer3
layers.layerId.children.switch2 = layer2, layer3
layers.layerId.children.switch3 =
```

Special suffix `default` can be used to define a default children list :

```
layers.layerId.className = LayerGroup
layers.layerId.children.switch2 = layer2, layer3
layers.layerId.children.default =
```

The following layer group definitions are equivalent :

```
layers.layerId.className = LayerGroup
layers.layerId.children.default = layer4
```

```
layers.layerId.className = LayerGroup
layers.layerId.children = layer4
```

6.3. Metadata in Mapfile and layers.ini

Metadata are (keyword, value) pairs which can be associated to a MapServer layer in the mapfile, or to a Layer or LayerGroup in the layers.ini configuration file. These metadata are used in several different contextes, such as layer specific configuration, security handling, ... The metadata related to the functionalities of CartoWeb are described in the documentation of the corresponding plugins.

6.3.1. Metadata in Mapfiles

Specifying metadata in mapfiles is based on the standard MapServer syntax. For instance:

```
LAYER
  NAME "my_layer"
  [...]
  METADATA
    "exported_values" "security_view,some_specific_parameter"
    "security_view" "admin"
    "some_specific_parameter" "value"

    "id_attribute_string" "FID|string"
    "mask_transparency" "50"
    "mask_color" "255, 255, 0"
    "area_fixed_value" "10"
  END
  [...]
END
```

The metadata key `exported_values` is a bit special: It gives the list of metadata keys (coma separated), which will be transmitted to the client. This is required for the metadata keys describing layer security for instance, as they are used on the CartoClient.

6.3.2. Metadata in layers.ini

The configuration file `layers.ini` may also contain metadata (key, value) pairs. This is needed for LayerGroup objects, as these have no counterparts in the mapfile. For simple layers, the metadata specified in the `layers.ini` take precedence over the values from the mapfile.

For each layer object in the `layers.ini` file, the following syntax is used to set metadata:

```
layers.layerId.metadata.METADATA = value
```

For instance:

```
layers.group_admin.className = LayerGroup
layers.group_admin.children = grid_defaulthilight
layers.group_admin.metadata.security_view = admin
```

Note

It is possible to specify *minScale* and *maxScale* attributes as `layers.ini` *metadata* for layerGroups items. The equivalent mapfile metadata is not available for layers. In that case, simply use mapfile MINSCALE and MAXSCALE attributes.

6.3.3. Using the metadata fields in templates (layers.tpl)

Metadata declared in mapfiles or `layers.ini` can be used in the `layers.tpl` Smarty templates using the following notation:

```
{capture name=caption}
{$element.layerLabel} <a
href="/?lang={$element.layerMeta.lang}&client_id={$element.layerMeta.client_id}">
{element.layerMeta.complex_label}</a>
{/capture}
```

As we can see in the previous example, metadata values can be used through the `layerMeta` property of the `$element` Smarty variable. Typical usage of this feature is to render for instance several HTML links for each layers.

6.4. Layers Legends

CartoWeb includes a mechanism for the automatic generation of legends. If desired, an icon is drawn for each class of the layers. There are two conditions that must be fulfilled to make it work :

1. The mechanism must be enabled by setting

```
autoClassLegend = true
```

in `layers.ini`.

2. Each class (or more precisely each class that should appear in the legend) must have a name. Here an example :

```
LAYER
  NAME "spur_spurweite"
  METADATA
  ...
  END
  TYPE line
  ...

  CLASS
    NAME "Normalspur (1435 mm)"
    EXPRESSION ([MM] = 1435)
    STYLE
      COLOR 0 0 255 # blue
    END
  END

  CLASS
#   NAME "Meterspur (1000 mm)"
    EXPRESSION ([MM] = 1000)
    STYLE
      COLOR 255 0 0 # red
    END
  END

  CLASS
    NAME "Schmalspur (< 900 mm)"
    EXPRESSION ([MM] < 900)
```

```
    STYLE
      COLOR 128 0 128 # lila
    END
  END

END
```

In this case, the second class would not appear in the legend, because the NAME is commented out.

Provided that no static icon is defined for a layer in `layers.ini`, the icon of the first visible class is used for the layer.

Legend icons are also used when exporting maps in PDF (see Chapter 12, *PDF Export*). Standard legend icons are used by default, which means that they may seem pixelized when printing a 300 dpi map. To resolve this issue, the following parameter can be set in `layers.ini`:

```
legendResolutions = 96, 150, 300
```

Each resolution must correspond to one of those set in PDF Export configuration file. This file is described in Section 12.2.1, "General Configuration".

6.5. WMS Layers Legends

Mapserver could act like a WMS client and retrieved layer from distant WMS server. In this specific case, layers legends could be retrieved directly from a distant server with SLD support (`getLegendGraphic` method).

You must also add a metadata on each WMS layer to retrieve its legend icon: *wms_legend_graphic*

Here is an example:

```
LAYER
  NAME 'prov_boundary'
  TYPE RASTER

  CONNECTION 'http://www2.dmsolutions.ca/cgi-bin/mswms_gmap?'
  CONNECTIONTYPE WMS
  METADATA
    'wms_srs'             'EPSG:4326'
    'wms_name'            'prov_bound'
    'wms_server_version'  '1.1.1'
    'wms_format'          'image/png'
    'wms_legend_graphic'  'true'
  END
END
```

6.6. Auto Layers

In some cases, it may be useful to generate several layers out of a single "template" layer. An example would be floors : each floor must be displayed with the same color, etc. but the data source is different (different shapefile or different where clause for a database layer). In CartoWeb, it is possible to add PHP code into mapfiles. True mapfiles are then generated by a batch script. The generated layers are called "Auto Layers".

6.6.1. Auto Layers in layers.ini

First, you have to declare the list of auto layers indexes. Indexes will be used as suffixes for generated layers :

```
autoLayersIndexes = index1, index2
```

Then you can use auto layers just as other layers. In this exemple, an auto layer *myAutoLayer_* was declared in the mapfile template (see Section 6.6.2, "Mapfile Templates").

```
layers.layer1.className = LayerGroup
layers.layer1.children = layer2, myAutoLayer_index1

layers.layer2.className = LayerGroup
layers.layer2.children = myAutoLayer_index2
```

6.6.2. Mapfile Templates

Mapfile templates for auto layers are PHP files. The main template must be called `mapId.map.php`. If this file exists, then CartoWeb assumes that auto layers generation is used. If the file does not exists, CartoWeb looks for a standard `mapId.map`.

The following functions are predefined and can be used in mapfile templates :

- `printLayer($name, $contents)` : adds a new auto layer. The name is the final name of the layer without the index suffix. Contents include Mapserver layer definition with PHP code for parts that will be different for each instance of auto layer
- `printIndex()` : adds the current auto layer index. Must be used in auto layer contents
- `printName()` : adds the current auto layer full name. Must be used in auto layer contents
- `getIndex()` : returns the current auto layer index. It can be useful for instance in conditional statements. Must be used in auto layer contents
- `includeFile($path)` : adds the content of another PHP mapfile template

Here is an example :

```
MAP
  NAME "MyMapfile"
  EXTENT -10 -10 10 10
# ...

<?php

printLayer('myAutoLayer_',

<<<AUTOLAYER

  # This is my auto layer with index <?php printIndex(); ?>
  LAYER
    NAME <?php printName(); ?>
    TYPE POLYGON
# ...
    CLASS
      NAME "MyClass"
      STYLE
        COLOR <?php if(getIndex() == 'index2') print "200 200 255";
                                        else print "255 153 102"; ?>
      END
    END
  END
AUTOLAYER
);

?>

END
```

6.6.3. Batch Script

Script `makemaps.php` located in directory `scripts` generates mapfiles out of mapfile templates for all projects. Mapfiles are generated in the same directory. The script must be called once during installation and then each time a change is made to the mapfile templates.

It uses file `layers.ini` to find out which auto layers are needed. For instance, with the following `layers.ini`, myAutoLayer_index3 won't be generated.

```
autoLayersIndexes = index1, index2, index3

layers.layer1.className = LayerGroup
layers.layer1.children = layer2, myAutoLayer_index1

layers.layer2.className = LayerGroup
layers.layer2.children = myAutoLayer_index2
```

At least one file is generated. Its name is `auto.mapId.all.map` and it contains all auto layers used in file `layers.ini`. When children switching is used (see Section 6.2.4, "Children Switching"), one mapfile is generated per switch. One more is generated for special switch `default`. This way, only auto layers used with the current switch are actually loaded at runtime. Mapfile `auto.mapId.all.map` is still generated.

In the following example, 4 mapfiles are generated :

```
autoLayersIndexes = index1, index2, index3, index4

switches.switch1.label = Switch 1
switches.switch2.label = Switch 2

layers.layer1.className = LayerGroup
layers.layer1.children.switch1 = layer2, myAutoLayer_index1
layers.layer1.children.default = layer2, myAutoLayer_index2

layers.layer2.className = LayerGroup
layers.layer2.children = myAutoLayer_index4
```

- `auto.mapId.all.map` : contains all layers used in `layers.ini`, ie. layers myAutoLayer_index1, myAutoLayer_index2 and myAutoLayer_index4
- `auto.mapId.switch1.map` : contains layers used when Switch 1 is selected, ie. layers myAutoLayer_index1 and myAutoLayer_index4
- `auto.mapId.switch2.map` : contains layers used when Switch 2 is selected, ie. layers myAutoLayer_index2 and myAutoLayer_index4
- `auto.mapId.default.map` : contains layers used when no switch is selected, ie. layers myAutoLayer_index2 and myAutoLayer_index4

6.7. Layer Reorder Plugin

Ability to reorder displayed layers could be appreciated by end user. And so, even if your map is well designed with largest features layers on bottom and smallest on top, and/or even with well used transparent properties. The plugin layerReorder , is designed to do so. Layers could be reorder on the displayed stack to improve visibility of one layer.

Each layer defined in the mapfile and currently selected in legend menu (and only them) will be displayed in the reorder menu list.

6.7.1. Plugin activation

To activate the plugin, add it to *loadPlugins* from Both CartoClient and CartoServer configuration files: `client_conf/client.ini` and `server_conf/<mapId>/<mapId>.ini`. For instance:

```
loadPlugins = layerReorder
```

6.7.2. Layer exclusion

If you want to exclude some layers and then not allow them to be reordered, have a look on `client_conf/layerReorder.ini`. You will be able to exclude some layers

and to choose to keep them on bottom (or on top). To do so, use the following configuration parameters:

- *topLayers*: list of all layerIds to keep on top of the genererated map (LayerIds must be comma separated).
- *bottomLayers*: same as previous one, but obviously keep these layerIds on bottom. Useful for example with a raster background layer.

6.7.3. Transparency selector

You could also add a transparency selector on each layer displayed with this plugin. End user will so be able to choose a different transparency level for each displayed layer.

To activate it, set *enableTransparency* to `true` in `client_conf/layerReorder.ini`.

Default transparency selector values are : 10%, 20%, 50%, 75% and 100%. You could also customize transparency values available in this selector, with a comma separated value in *transparencyLevels*. A transparency value is a integer between 1 to 100 ; with 100 mean an opaque layer as 1 mean a nearly invisible one.

7. Navigation

The plugin location is the core plugin that deals with geographic navigation on the map. It handles bboxes (visible areas) and scales; it drives the tools zoom-in, zoom-out and panning, the directional arrows around the main map and the overview map.

The corresponding configuration files are `location.ini` (client-side) and `location.ini` (server-side).

7.1. Client-side Configuration

Here are the options that can be set:

- scalesActive: boolean, if true, the scales dropdown list is displayed (default: false). If no visible scales are defined on the server, a simple input text will be displayed.
- freeScaleActive: boolean, if true, the free scales input is displayed (default: false). This allow the user to freely set a scale value that is different from the predefined scales. There are a few conditions for this to work:
    - scalesActive must be activated (see above).
    - scaleModeDiscrete must be deactivated (false) (see Server-side Configuration).
    - location.js must be loaded.
    - prototype.js must be loaded (by default if ajax is on).
- recenterActive: boolean, if true, the coords recentering form is displayed (default: false)
- idRecenterActive: boolean, if true, the id recentering fields will be displayed (default: false)
- idRecenterLayers the comma separated list of layers which will appear in the id recentering selection form. If this list is absent, all msLayers appear in the form.
- shortcutsActive: boolean, if true, the shortcuts (direct access dropdown list) are displayed (default: false)
- scaleUnitLimit: scale above which DHTML measures use km ; below, they use m.
- panRatio: ratio for panning by clicking directional arrows. Default is 1 (no overlap, no gap). Values below 1 result in an overlap of the old and the new maps; values above 1 in a gap between these two maps.
- weightZoomin: integer defining display order of the zoomin tool icon in toolbar

(if not specified, default to 10). A negative weight disables the tool.

Note

Zoomin is the default tool and it will be activated automaticaly. If you want to deactivate it, you need to set another default tool. See *initialTool* in Section 4.2, " client.ini ".

- weightZoomout: see weightZoomin (default to 11)
- weightPan: see weightZoomin (default to 12)
- weightFullextent: see weightZoomin (default to 14)
- crosshairSymbol: (integer) numerical id of symbol to use to materialize recentering point (see your *symbolset* file)
- crosshairSymbolSize: integer defining the size used to display the recentering crosshair symbol.
- crosshairSymbolColor: RGB code (comma-separated list of integers) of the crosshair color.
- showRefMarks: if true, will add reference marks to the main map. Reference marks configuration is set on server.

7.2. Server-side Configuration

Here are the options that can be set:

- minScale: if set, minimal scale allowed.
- maxScale: if set, maximal scale allowed
- scaleModeDiscrete: boolean, if true, only specified scales (see below) can be set.
- zoomFactor: the zoom factor to use when scaleModeDiscrete is set to false.
- noBboxAdjusting: If set to true, all data contained in the initial extent (mapfile) is visible on the full extent (first) map. This case is the default mapserver behavior. If set to false (default), the user won't be able to see anything outside the initial extent, even on full extent map.
- scales.#.value (# = 0, 1, 2, ...): available value of the scale in discrete mode.
- scales.#.label (# = 0, 1, 2, ...): label of the scale, to be displayed in the dropdown list on the client.
- scales.#.visible (# = 0, 1, 2, ...): boolean, if true, the scale is displayed in the dropdown list. If false, this scale can only be selected by zoom-in/zoom-out. Default is true.
- shortcuts.#.label (# = 0, 1, 2, ...): label of the shortcut; appears in the direct access dropdown list on the client.

- shortcuts.#.bbox (# = 0, 1, 2, ...): geographic bbox of the shortcut.
- recenterMargin: margin to add around the centered-on object (valid for lines and polygons). Expressed in percent of the width/height of the object. If not set, recenterDefaultScale is used.
- recenterDefaultScale: fixed scale to use when recentering. Mandatory when centered-on object is a point.
- refMarksSymbol: symbol used to draw lines for reference marks (typically an ellipse).
- refMarksSymbolSize: size in pixel of symbol used to draw lines for reference marks (typically 1 or 2).
- refMarksSize: size of reference marks. Size is fixed and won't change with scale.
- refMarksColor: color of reference marks (comma-separated R, G and B values).
- refMarksTransparency: transparency of reference marks.
- refMarksOrigin: origin for reference marks (comma-separated X and Y values, typically "0, 0").
- refMarksInterval.#.maxScale: maximum scale for corresponding interval.
- refMarksInterval.#.interval: real interval between two reference marks (comma-separated X and Y values).
- refLinesActive: if true, will display reference marks on page borders.
- refLinesSize: size in pixel of reference marks on page borders.
- refLinesFontSize: size of font for reference marks on page borders.
- scaleUnitLimit: scale above which the scalebar use km ; below, m is used.

7.3. Related Elements Elsewhere

The maximal extent of the geographical zone is set by the EXTENT command in the mapfile.

The initial bbox is set by the active initialMapState. It is to be configured in the `myMap.ini` file, using the following syntax:

mapInfo.initialMapStates.initialMapStateId.location.bbox = "xmin,ymin,xmax,ymax"

8. Image Format Options

The plugin images is the core plugin that deals with the formatting options for the main map. It handles the size of the image and its filetype.

8.1. Client-side Configuration

8.1.1. Main Map Options

Here are the options that can be set:

- mapSizesActive: boolean; if true, displays a drop-down list of available mapsizes ; if false a fixed size is used (defined by the two following parameters).
- mapWidth: mainmap width in pixels if mapSizesActive = false
- mapHeight: mainmap height in pixels if mapSizesActive = false
- mapSizes.#.width : (#=0,1,2...) available mainmap width in pixels for mapsize # (when mapSizesActive = true)
- mapSizes.#.height : (#=0,1,2...) available mainmap height in pixels for mapsize # (when mapSizesActive = true)
- mapSizes.#.label : (#=0,1,2...) label that describes mapsize #; appears in drop-down list. If not specified, a <width>x<height> pattern is used as the label.
- mapSizesDefault : integer indicates the default mapsize to be used (among the above #). Only when mapSizesActive = true.
- maxMapWidth : maximum allowed width in pixels of the generated mainmap when setting manually the mapsize using GET or POST requests. If no value is specified, default max width is 1500 px.
- maxMapHeight : maximum allowed height in pixels of the generated mainmap when setting manually the mapsize using GET or POST requests. If no value is specified, default max height is 1000 px.

If no mapsize settings are found, the default mapsize is 400x200 pixels.

8.1.2. Keymap and Scalebar Options

The plugin images can also deal with few keymap and scalebar options:

- collapsibleKeymap: if true, keymap is shown on map, and possibly hidden.
- noDrawKeymap: if true, keymap will not be drawn at all.
- noDrawScalebar: if true, scalebar will not be drawn.

8.2. Server-side Configuration

Here are the options that can be set:

- maxMapWidth: maximum allowed width in pixels of the generated mainmap. If the value requested by the client is greater, maxMapWidth takes precedence over it.
- maxMapHeight: maximum allowed height in pixels of the generated mainmap. If the value requested by the client is greater, maxMapHeight takes precedence over it.

8.3. Related Elements in Mapfile

8.3.1. General Image Type

The general output fileformat is handled by Mapserver. The basic command in the mapfile is

```
IMAGETYPE png|jpeg|gif...
```

Then the OUTPUTFORMAT objects may set properties for each possible fileformat. Example (for jpeg) :

```
OUTPUTFORMAT
  NAME jpeg
  DRIVER "GD/JPEG"
  MIMETYPE "image/jpeg"
  IMAGEMODE RGB
  FORMATOPTION QUALITY=85
  EXTENSION "jpg"
END
```

See the available options for each format in the Mapserver doc.

Important note if you intend to use pdf printing : interlaced png images are not supported by the fpdf library that is used in this module. Consequently, you must have the option

```
FORMATOPTION "INTERLACE=OFF"
```

in the definition of the png OUTPUTFORMAT. Here is the complete object :

```
OUTPUTFORMAT
  NAME png
  DRIVER "GD/PNG"
  MIMETYPE "image/png"
  IMAGEMODE PC256
```

```
  EXTENSION "png"
  FORMATOPTION "INTERLACE=OFF"
END
```

8.3.2. Automatic Image Type Switch

It is often desirable to adapt the imagetype of the map to the represented data.
Typically, vector data is well rendered in png, while the colors of a raster background
require either jpeg or png24 format. CartoWeb includes a mechanism that
automatically switches the format when specific layers are selected. A special
metadata of the layers triggers this behavior :

```
METADATA
  ...
  "force_imagetype" "jpeg|png..."
  ...
END
```

This metadata overrides the general fileformat of the mainmap (but not of the legend
icons; these stay in the initial fileformat).

9. Queries

Core plugin Query allows to search for geographical objects. Found objects are hilighted and if requested related data are returned to client.

Depending on configuration and user choices, queries are executed on one layer, several layers or all layers currently displayed on map.

Queries can be executed on a geographic selection or using a list of object IDs. Geographic selection can be a point, a rectangle, a polygon or a circle but is a rectangle by default. The second way to execute a query is used in particular to maintain selection persistence: object IDs are stored client-side and sent to server each time the page is reloaded.

Hilighting objects can be done using standard Mapserver queries or using special Hilight plugin. See Section 9.3.3, "Hilight Configuration" for more information.

Results are returned and displayed using Tables plugin. See Section 9.1.2, "Tables Configuration" for more information.

9.1. Client-side Configuration

9.1.1. query.ini

Here are the options that can be set in client's query.ini:

- persistentQueries: if true, queries will be persistent. If false, selection is lost after next page reload. Note that persistency will work only with layers with id_attribute_string set (see Section 9.3, "Related Elements in Mapfile")
- displayExtendedSelection: if true, shows form for selection extended functions. This form is mainly used by developers (see TODO: link to developer's doc)
- queryLayers: the comma separated list of layers which will appear in the extended selection form. If this list is absent, all msLayers appear in the form.
- returnAttributesActive: if true, the layers attributes can be requested. If false, only object IDs will be returned (default: false)
- defaultPolicy: can be either POLICY_XOR, POLICY_UNION, POLICY_INTERSECTION or POLICY_REPLACE. Defines how new selected objects are mixed with previous ones (default: POLICY_XOR)
- defaultMaskmode: if true, selected objects are highlighted in mask mode (default: false). See also Section 9.3.3.2, "Mask Mode"
- defaultHilight: if true, objects are highlited (default: true)

- defaultAttributes: if true, the layers attributes are returned (default: true)
- defaultTable: if true, the results table is displayed (default: true)
- weightQueryByPoint: integer which defines display order of the query by point tool icon in toolbar (default: 40). Negative weighted tools are disabled
- weightQueryByBbox: see weightQueryByPoint (default: 41).
- weightQueryByPolygon: see weightQueryByPoint (default: 42).
- weightQueryByCircle: see weightQueryByPoint (default: 43).

9.1.2. Tables Configuration

Tables plugin can be used by any plugin to manage, transfer and display tables structure. In basic CartoWeb installation, only Query plugin uses this functionality.

To configure table appearance, use tables client-side rules described in Developer's Documentation (Section 2.4.3, "Tables").

9.2. Server-side Configuration

9.2.1. query.ini

Here are the options that can be set in server's query.ini:

- drawQueryUsingHilight: if true, query hilighting will use Hilight plugin. In this case, Hilight plugin must be loaded on server. If false, objects will be hilighed using MapServer query functionality. See also Section 9.3.3, "Hilight Configuration" (default: false)
- noRowId: if true, row id will not be included in the table. In this case the row id will not be displayed in the output table (html, pdf, etc.). (default: false)

9.2.2. MapServer Query Configuration

MapQuery plugin can be used by any plugin to retrieve objects information from MapServer.

Following options can be set in server's mapquery.ini:

- maxResults: Maximum number of results to handle in the query plugin. This limit is to avoid high load on the server. It should be the client responsibility not to ask too many objects to avoid reaching this limit. Ignoring big queries can be done with the ignoreQueryThreshold parameter, which give a better behaviour for the user

- ignoreQueryThreshold: Do not take into account the elements selected by a shape (rectangle, polygon) in a query, if this shape intersects more than ignoreQueryThreshold objects. It should be less than maxResults to have informative messages to the user

9.3. Related Elements in Mapfile

9.3.1. Making a Layer Queriable

To make a MapServer layer queriable, one have to add a `TEMPLATE` parameter in the mapfile layer definition. In our case, any value fits. For instance:

```
LAYER
  NAME "foobar"
  ...
  TEMPLATE "ttt"
  ...
END
```

9.3.2. Meta Data

Here are the meta data that can be set to mapfile's layers:

- "id_attribute_string" "name|type": describes the attribute used for the id, and the type of the id. Type can be either "int" or "string". Caution: case sensitive
- "query_returned_attributes" "attribute1 attribute2": the names (space separated) of the fields returned by a query. If not set, all fields are returned. Caution: case sensitive
- "hilight_use_logical_expressions" "true"/"false": Set this to true if you are using a layer for hilight (using the convention `yourlayer_hilight`) and you want to preserve your expressions in the hilight layer for hilighted objects. IMPORTANT: you need to set this metadata on the hilight layer, not the original one.

Example:

```
METADATA
  "id_attribute_string" "FID|string"
  "query_returned_attributes" "FID FNAME"
END
```

9.3.3. Hilight Configuration

Hilight plugin can be used by any plugin to hilight objects on the map. In basic CartoWeb installation, only Query plugin uses this functionality. As Hilight plugin is not a core plugin, it must be loaded in order to use it with queries.

9.3.3.1. Normal Mode

Hilight on a specific layer can be generated by several means: special layer activation, special class activation, dynamic layer/class generation. Decision is made in the following order:

1. looks for a layer named "<layer_name>_hilight"
2. if not found, looks for a class named "hilight" in the current layer
3. if not found, dynamically creates a layer if meta data "hilight_createlayer" is set to "true"
4. if meta data "hilight_createlayer" is not set or set to "false", dynamically creates a class

Here are the meta data that can be set to mapfile's layers:

• "hilight_color" "0-255 0-255 0-255": the hilight color of a dynamically generated class
• "hilight_createlayer" "true": if true, a new layer will be dynamically generated for the hilight
• "hilight_transparency" "1-100": the transparency, for dynamically generated layers

Examples:

Hilight using "<layer_name>_hilight" layer

```
LAYER
  NAME "foo"
  ...
END

LAYER
  NAME "foo_hilight"
  ...
END
```

Hilight using class "hilight"

```
LAYER
  NAME "foo"
  ...
  CLASS
    EXPRESSION /_always_false_/
    NAME "hilight"
    STYLE
      ...
    END
```

```
  END
  # other layer classes
  CLASS
    ...
  END
END
```

Hilight using dynamically generated layer

```
LAYER
  NAME "foo"
  ...
  METADATA
    "hilight_createlayer" "true"
    "hilight_color" "255 255 0"
    "hilight_transparency" "50"
  END
END
```

9.3.3.2. Mask Mode

When mask mode is requested, decision is made in the following order:

1. looks for a layer named "<layer_name>_mask"
2. if not found, dynamically creates a mask layer by copying current layer

Masking process also tries to find a layer which would hide the area outside all possible selections:

1. tries to activate a layer with name set in meta data "outside_mask"
2. if meta data "outside_mask" is not set, looks for a layer named "default_outside_mask"
3. if not found, no outside mask will be displayed

Here are the meta data that can be set to mapfile's layers:

- "mask_color" "0-255 0-255 0-255": color of the mask when mask layer (<layer_name>_mask) is not defined
- "mask_transparency" "true": transparency of the mask when mask layer (<layer_name>_mask) is not defined
- "outside_mask" "layer_name": name of layer which will mask the outside (aka "complement"). If not set, will try to find a layer named "default_outside_mask"

Examples:

Mask using "<layer_name>_mask" layer

```
LAYER
  NAME "foo"
  ...
END

LAYER
  NAME "foo_mask"
  ...
END
```

Mask using "<layer_name>_mask" layer and an outside mask

```
LAYER
  NAME "foo"
  METADATA
    "outside_mask" "bar"
  END
  ...
END

LAYER
  NAME "foo_mask"
  ...
END

LAYER
  NAME "bar"
  ...
END
```

Mask using dynamically generated layer and default outside mask

```
LAYER
  NAME "foo"
  METADATA
    "mask_color" "255 255 255"
    "mask_transparency" "60"
  END
  ...
END

LAYER
  NAME "default_outside_mask"
  ...
END
```

10. Annotation and Redlining

Imagine you want to draw some features (points, lines, polygons ...) in your map to show specific data and/or print it. The outline plugin is what you need.

It allows you to draw features and to annotate them with label texts.

It's also possible to use polygons shapes as a mask layer see "Mask mode". User can choose between mask or draw mode with radio buttons. See Section 10.2.1, "outline.ini" for more information.

Shapes colors and styles should be set with specific configuration in the mapfile (See Section 10.3, "Related Elements in Mapfile").

The total area of the drawn polygons is displayed on the interface.

10.1. Client-side Configuration

10.1.1. outline.ini

Here are the options that can be set on the client:

- multipleShapes: boolean, if true, the user is allowed to draw multiple shapes. That means that drawn features remain until browser session is closed. If false, will outline only one shape at a time, new feature simply erases / overrides currently drawn one. When he wants, the user can erase all the drawn feature by clicking on the "outline_clear" form button (default true).
- labelMode: boolean, if true, user is asked to input a label text that is drawn to annotate drawn shape.
- weightOutlinePoint: integer defining display order of the outline_point tool icon in toolbar (if not specified, default to 70). A negative weight disables the tool.
- weightOutlineLine: see weightOutlinePoint (default to 71)
- weightOutlineRectangle: see weightOutlinePoint (default to 72)
- weightOutlinePoly: see weightOutlinePoint (default to 73)

10.2. Server-side Configuration

10.2.1. outline.ini

Here are the options that can be set on the server:

- pointLayer: defines the name of the mapserver LAYER set in the mapfile to display points.

- lineLayer: defines the name of the mapserver LAYER set in the mapfile to display lines.

- polygonLayer: defines the name of the mapserver LAYER set in the mapfile to display polygons. Note that rectangles are displayed as polygons.

- maskColor: defines the RGB color of the mask. This color is used to fill the polygons outside's area. Drawn polys will appear as holes in a colored sheet, showing the map under. This parameter is optional and set by default to 255 255 255 (white).

- areaFactor: defines an optional value for ratio by which the area is multiplied before it is returned to the client. For example, people can use it to convert square meters to square kilometers.

- symbolPickerHilight = inversed|borderhilight: set the hilight mode to use in the SymbolPicker interface. See Section 10.5.3.2, "SymbolPicker".

10.3. Related Elements in Mapfile

10.3.1. Layers

Specific layers must be set in the mapfile in order to draw the shapes:

```
LAYER
  NAME "cartoweb_point_outline"
  TYPE POINT
  ...
END

LAYER
  NAME "cartoweb_line_outline"
  TYPE LINE
  ...
END

LAYER
  NAME "cartoweb_polygon_outline"
  TYPE POLYGON
  ...
END
```

Don't forget that layers names must be set in the server-side outline.ini file (See Section 10.2.1, "outline.ini")

10.3.2. Labels

In case of using labels (i.e. labelMode set to true, see Section 10.1.1, "outline.ini"), a LABEL object is needed in the layers' class.

Example:

```
LAYER
  NAME "cartoweb_point_outline"
  TYPE POINT
  ...
  CLASS
    STYLE
      ...
    END
    LABEL
      FONT Vera
      TYPE truetype
      COLOR 51 51 51
      SIZE 10
    END
  END
END
```

You will find more examples looking in the test.map file in server_conf directory.

10.4. GET Parameters

Creates outline drawing on-the-fly by passing a geometry type and one or more point coordinates via the URL.

10.4.1. Accepted Parameters

Parameters list

• outline_point

```
http://www.example.com/cartoweb/myproject.php?outline_point=611748,5232242
```

• outline_line

```
http://www.example.com/cartoweb/myproject.php?outline_line=611748,5232242;748204,5232242;748204,505030
```

• outline_poly

```
http://www.example.com/cartoweb/myproject.php?outline_poly=611748,5232242;748204,5232242;748204,505030
```

Note

If you do not want to have persistent outline, you can add the *prevent_save_session* parameter in the URL. For example:

```
http://www.example.com/cartoweb/myproject.php?outline_point=611748,5232242&prevent_save_session
```

This may be useful if you use the *mode=image* output:

```
http://www.example.com/cartoweb/myproject.php?mode=image&outline_point=611748,5232242&prevent_save_session
```

10.5. The ToolPicker

10.5.1. What is the ToolPicker

The ToolPicker is a mini javascript framework used to display dynamically on screen several DHTML tools.

Some tools come pre-included:

- ColorPicker : The ColorPicker allows the user to pick a color. He/she can either pick it from a gradient map, use RGB or HSL slider, input RGB or HSL values , input an hexadecial value or pick it from a preselected list of colors. The color value is then returned to the main page.
- SymbolPicker : The SymbolPicker allows the user to pick a symbol in a predefined list of symbols. The symbols are generated off the .sym file on the serverside of CartoWeb. The symbol is then returned to the main page.

10.5.2. Installation

To use the ToolPicker in a project, simply enable it in your client.ini configuration file, by setting toolPickerOn = true.

The ToolPicker requires the corresponding script, css and template in the cartoclient.tpl file.

In the header :

```
<link rel="stylesheet" type="text/css" href="css/toolPicker.css" />
```

In the body, preferably at the end :

```
{include file="toolPicker.tpl"}
```

10.5.3. ToolPicker Usage

The main ToolPicker function accepts 3 parameters, 2 are mandatory, the last is optional.

```
toolPicker( string tool_id_list, string tool_input_list [, string tool_output_list])
```

tool_id_list

You can initialize one or more tools at once. The *tool_id_list* parameter is composed of one or more numerical values, separated by comma and enclosed by quotes (or simplequotes). Example: '2' or '1,4' or '3,2,1'

71

The order of the id influes on the order of the tool panels when displayed on screen.

tool_input_list and **tool_output_list**

The script needs an input value and will return an output value. The input value is retrieved from an html input element. In the same way the output value is returned to an html input element.

The script needs as many input element id as initialised tools. If two tools are initialised (parameter *tool_id_list*), *tool_input_list* must also contain two input id. Ids are separated by comma and enclosed by quotes (or simplequotes). Example: 'return_id_tool_4,return_id_tool_2'

The first input id corresponds to the first tool id in *tool_id_list*.

A different id can be specified if the element storing the output value must be different from the element storing the input value. The same rule applies to *tool_input_list* and *tool_output_list*.

See Section 10.5.5, "Examples of Usage"

Each tool also have some particularities:

10.5.3.1. ColorPicker

When a color is selected, the script will set the css property *background-color* of a div in the template accordingly to the color picked.

The id of the div is based on the output element id (if it specified, if not, the input element id is used), to which a *_d* is added.

Example, if the input element id is *return_color_12*, the id of the div used to display the color will be *return_color_12_d*.

10.5.3.2. SymbolPicker

When a symbol is selected, the script will set the css property *background-image* of a div in the template accordingly to the symbol picked.

The id of the div is based on the output element id (if it specified, if not, the input element id is used), to which a *_d* is added.

Example, if the input element id is *return_symbol_12*, the id of the div used to display the symbol will be *return_symbol_12_d*.

The SymbolPicker is used with CartoWeb outline plugin usually, but it also can be

used without the plugin.

10.5.3.2.1. Within the Outline Plugin

In CartoWeb *server_conf/yourprojectname/outline.ini* file defines a list of symbols to use. For each kind of symbols you want to use:

```
pointSymbols = "circle, square, star [, anyothersymbol, ... ]"
```

All these symbol names **MUST BE** defined in the .sym file in *server_conf/yourprojectname/*.

If you want to add language support for the symbol name, add the following parameter in *server_conf/yourprojectname/outline.ini*:

```
pointSymbols.labels = "symbolname1, symbolname2 [, anyothername, ... ]"
```

These names will be used to translate into the chosen language using the gettext functionnality. Add the corresponding translations in the server .po files.

10.5.3.2.2. Without the Outline Plugin (Standalone Use)

If you do not use the outline plugin, you will need to set the following javaScript variable somewhere in your page:

```
<script type="text/javascript">
  /*<![CDATA[*/
    var imgPath = 'path/to/symbol/images/';
    var symbolType = 'type/of/image/files/'; //jpeg, gif, png, whatever

    var symbolNamesArray = new Array("symbolfilename1","symbolfilename2" [,"othersymbolfilename", ...] );

    var symbolLabelArray = new Array("symbollabel1","symbollabel2" [,"othersymbollabel", ...] );
  /*]]>*/
</script>
```

symbolNamesArray contains the list of filename to use for your images.

symbolLabelArray contains the list of label used on screen for the symbols.

Create your symbol images and put them in *path/to/symbol/images/*. You also need to create hilighted versions of your images, give them the same filename but add the suffix '_over'. Exemple: symbol_1.gif -> symbol_1_over.gif

Put them in the same folder.

10.5.3.2.3. Symbol Hilight Mode

There are two modes to hilight the selected symbol in the *SymbolPicker* interface. You can select between *border hilight* and *inverted image*. *border hilight* simply use

a css property to hilight the border of the selected symbol. *inverted image* need the php library GD to create an inverted version of the symbol, which is then used for the selected symbol.

The hiligh mode is set in the *server_conf/outline.ini* file:

- symbolPickerHilight = inversed|borderhilight

10.5.4. ToolPicker Default Values

If the toolPicker is used with the Outline plugin, the default values for Color, OutlineColor, Transparency and Symbol are set in the mapfile.

The default layer name are cartoweb_point_outline, cartoweb_line_outline and cartoweb_polygon_outline

See Section 10.2.1, "outline.ini"

10.5.5. Examples of Usage

```
toolPicker('4','outline_point_symbol')
```

this will call the symbol picker (id 4), the input id is *outline_point_symbol*, and the output id will be *outline_point_symbol* too

```
toolPicker('1','outline_point_color', 'color_return')
```

this will call the color picker, the input id is *outline_point_color*, and the output id will be *color_return*

```
toolPicker('4,1','symbol_input,color_input', 'return_symbol,return_color')
```

this will call the symbol picker and the color picker, the input id is *symbol_input* for the first tool (here the symbol picker) and *color_input* for the second tool (color picker), and the output id will be *return_symbol* for the first tool and *return_color* for the second.

10.5.6. Creating New Tools

You can add new tools.

Add the new tool name in the *toolArrayRef* array at the begining of the script.

```
var toolArrayRef =  new Array( 'colorPicker',
                               'hashArray',
                               'pencilArray',
                               'symbolArray',
                               'yourTool');
```

Create the following four functions, these functions can be empty.

```
function yourToolInit(inputValue) { ... }
function yourToolSetup() { ... }
function yourToolReturn() {
    ...
    return someValue;
}
function yourToolDisplay(targetElm) { ... }
```

The Init function is used to set the incoming values. It is always called when the ToolPicker starts. It receives the input value recovered from the main page.

The Setup function can be used to do all the actions you want.

The Return function is used to return a value to the main page.

The Display function is used to display something in the main page. It receives the id of the target element to use (*something_d*).

Add the following four function definitions:

```
functionList["yourToolInit"]=yourToolInit;
functionList["yourToolSetup"]=yourToolSetup;
functionList["yourToolReturn"]=yourToolReturn;
functionList["yourToolDisplay"]=yourToolDisplay;
```

These are needed because of the dynamic function call inside the framework, which do not use the eval() function to increase performances.

The width of the tool area available on screen is 245 pixel. Height is variable.

11. Export Plugins

11.1. Introduction

It is possible to export maps and data from the viewer (ie. the CartoWeb user interface) in order to print them or to save them on the user's computer. Six formats are available: PDF, CSV, HTML, DXF, RTF and raw image.

The PDF export plugin is precisely described (usage, configuration) in next chapter: Chapter 12, *PDF Export*.

11.2. HTML Export

The `exportHtml` plugin outputs an HTML simplified version of the viewer: main map, keymap and layers list. It is often launched by clicking on a "HTML print" link or button and opens a new browser window. It is specially useful to quickly (basic HTML layout, same maps used than in viewer) display raw maps in order to print them.

No configuration is required. To enable or deactivate this plugin, simply add or remove `exportHtml` from the `loadPlugins` parameter in `client_conf/` `client.ini`. For instance:

```
[...]

; exportHtml is listed as activated:
loadPlugins = auth, outline, exportHtml

[...]
```

To customized the exported page layout, simply edit the plugin template file `exportHtml/templates/export.tpl`. It is recommended not to modify the regular template file but to override it in a project version of the `exportHtml` plugin. For more info about projects, see Section 2.4, "Projects".

To get advanced printing capabilities, rather use the CartoWeb PDF export plugin, `exportPdf`, described in next chapter.

11.3. CSV Export

The `exportCsv` plugin enables to export tabular data from various queries results in a comma-separated (CSV) or assimilated text format. The returned CSV file can then be opened and edited in any text editor or rendered as a table document in OpenOffice or MS Excel. For instance:

```
"Id";"Object Description"
```

```
"1";"A Line"
```

"CSV export" links are generally displayed in the viewer at the bottom of each queries results tables. Each link can only export a single layer table at a time.

A few configuration parameters are available in `client_conf/exportCsv.ini`:

- *separator* indicates what character should be used to distinguish each tabular cell value. Default is ";" (semi-colon).
- *textDelimiter* tells what character should be used to delimit the text in each cell. It is specially useful when the character used as the *separator* may be found within the cell content. Default parameter value is *double-quote* ie. `"`. The alias is defined to overcome INI syntax issues.
- *filename* specifies the filename naming convention to use. It can include the table name (using the *[table]* keyword) and the generation date under various formats. Date formating is performed by indicating between a couple of brackets the keyword *date*, followed by a comma and PHP date()-like date format. (see http://php.net/date). Default filename convention is *[date,Ymd-Hi]_[table].csv* which gives for instance *20050220-2021_myLayer.csv*.
- If the *charsetUtf8* boolean is set to *true* the result file will be UTF-8 encoded. Else ISO-8859-1 encoding is used, which is the default behavior.

After initial CartoWeb installation, `client_conf/exportCsv.ini` is set as follows:

```
; separator between each value, default is ";"
separator = ";"

; delimiter before and after each value, default is double-quote
; special characters:
;     double-quote = "
textDelimiter = double-quote

; file name format for exported CSV file, default is "[table]-[date,dMY].csv"
fileName = "[date,Ymd-Hi]_[table].csv"

; if true, exported CSV file will be UTF-8 encoded
; if false, it will be ISO-8859-1 encoded, default
charsetUtf8 = false
```

11.4. DXF Export

DXF is a drawing format supported by AutoCAD and assimilated software. With CartoWeb, it is possible to export a DXF description of the shapes drawn with the *outline* plugin. The latter plugin must be activated before using *exportDxf* plugin.

A few configuration parameters are available in `client_conf/exportDxf.ini`:

- *roundLevel* (integer) indicates how many digits must appear after the decimal point in points coordinates. Default is "2".

- *fileName* (string) specifies the filename naming convention to use. It may be static or contain athe generation date under various formats. Date formating is performed by indicating between a couple of brackets the keyword *date*, followed by a comma and PHP date()-like date format. (see http://php.net/date). Default filename convention is *export_[date,Ymd-Hi].dxf* which gives for instance export_20060725-2021.dxf.

- *exportDxfContainerName* (string) specifies the ID of the block containing the *{$exportDxf}* variable in the template when working with Ajax mode enabled. For exemple:

```
exportDxfContainerName = "exportDxfContainer"
```

in exportDxf.ini and

```
<div id="exportDxfContainer">{$exportDxf}</div>
```

in your cartoclient.tpl.

11.5. RTF Export

The Rich Text Format (RTF) is a document file format used for cross-platform document interchange. Most word processors are able to read and write RTF documents. RTF should not be confused with enriched text format which is a completely different specification.

The main goal of The ExportRtf plugin is to allow users to create templates directly in a word editor. The plugin compatibility has been tested on :

- Office Word 2004 and later (Mac, Windows),
- OpenOffice 2.0 and later,
- NeoOffice (Mac)

Known incompatibilities :

- Pages (Mac),
- TextEdit (Mac) images are not supported,

Supported image formats (set in the mapFile):

- JPEG,
- GIF,
- PNG,

11.5.1. Setting-up the template

Open your editor and do your layout setup by opening the `exportRtf.rtf` file in the plugin template folder.

Note

Please take into consideration that the RTF template will be generated by your editor. You may encounter some compatibility issues. For example: an RTF template done with Word can be open with OpenOffice without too many difficulties while an RTF generated with OpenOffice may not open correctly with Word.

You can insert different fields in the template by typing keywords inside triple brackets. For Example : [[[$TITLE]]]

- $TITLE stands for a specific title of the form.
- $MAP stands for the map. The map dimensions are defined in the exportRtf.ini.
- $QUERYRESULTS stands for the query results. If you want to customize the table output, edit the results.tpl file.
- $SCALEBAR stands for the scale bar.
- $KEYMAP stands for the minimap.

You can apply word formatting and alignment to your tags. The formatting will then be reported to their contents.

Warning

When you write your tags in the file, please write them in a continuous way and without spaces and/or formatting. If you want to apply any formatting, you need to select all the tags with the "[[[ ]]]" included. If by mistake an RTF tag is inserted inside the brackets, you will have to erase the tag and rewrite it or edit the RTF sources and remove the RTF keywords.

11.5.2. Configuration parameters

You will find some configuration parameters in the `exportRtf.ini` file

```
;fields shown in the form
activatedBlocks = mainmap,title,scalebar,overview,queryResult
;dimention of the map in pixels X*Y
mapDimention ="400*400"
;allowed roles for the Auth plugin
allowedRoles = all
;rtf optional inputs
```

```
optionalValues = ref,num,other
```

11.5.3. Optional Inputs

You can add free dynamic content inside your RTF. For instance, to insert a reference number in you RTF file:

First, you have to add the name of your optional fields in the `exportRtf.ini` file. Please do not use special characters there. Please do not use special characters here.

```
optionalValues = ref,num,other
```

Then, open the exportRtf.rtf file with you favorite word editor and just add tags the same way you do it for standard components.

```
 [[[$REF]]] [[[$NUM]]] [[[$OTHER]]]
```

This tags are case sensitive and have to be in UPPER CASE. The optionalValues parameters that you have set in the ini file have generated some hidden inputs in the HTML output. These inputs are named with the prefix "optRtf" and the tag name.

```
<input id="optRtfref" type="hidden" name="optRtfref" value=""/>
```

To have a value shown in the RTF, you just have to set a value to the corresponding HTML input.

```
//JavaScript
     $('optRtfref').value ='123';
```

Note

If you want to have an optional input available in a form (like a text input), you can simply remove its name from the ini file. Then open an HTML template and add a form field name, which must contain the prefix "optRtf" and the tag name.

11.6. Image Export

CartoWeb offers 2 way to get an image.

11.6.1. exportImg

To display only the map image without the other elements of the main template, you can simply add *?exportImg* to the url.

```
http://yourservername/yourproject.php?exportImg
```

The template used is *imgoutput.tpl* in the main *templates* folder. It can be customised to your needs.

This plugin must be activated in your `client.ini` list of plugins.

11.6.2. mode=image

CartoWeb allows to return a raw image map with no HTML. It is not an export plugin and thus has not to be activated anywhere. CartoWeb simply needs to receive a `mode=image` REQUEST parameter (GET or POST). For instance:

```
http://your/cartoweb3/?recenter_x=123456&recenter_y=654321&show_crosshair=1&mode=image
```

See Section 2.4.2.2, "Available Parameters" for more details about the possible extra parameters allowed by this feature.

12. PDF Export

12.1. Introduction

This chapter describes how to configure PDF export, using `client_conf/exportPdf.ini` parameters.

First part (Section 12.2, "Configuration Reference") is a comprehensive reference of user-available configuration parameters. Second part (Section 12.3, "Tutorial") is intended to be a small tutorial giving explanations and examples about how to configure the PDF exportation tool.

12.2. Configuration Reference

12.2.1. General Configuration

Parameters are named using a <object>.<property> convention. These parameters are grouped within dedicated objects that deal with separated aspects of the export. For instance:

```
general.horizontalMargin = 10
general.verticalMargin = 10

formats.A4.label = A4
formats.A4.bigDimension = 297
```

Some objects handle the general description of the PDF document as well as user form generation: "general" and "formats".

Other objects are "blocks"-typed. They deal with block presentation (color, size, content, etc.) and positioning. Blocks are basic entities of the PDF document: text, images. For instance overview, mainmap, scalebar, title are described with block objects.

To factorize blocks description, it is possible to describe a "template" block object that defines the default configuration of blocks. Those default settings are then overriden by each block specific description.

- *general.guiMode* (string): type of GUI used. Must be either 'classic' (default) or 'rotate'. 'rotate' mode adds the possibility to rotate the map to be printed directly on screen.
- *general.formats* (comma-separated list of strings): ids of available sheet sizes. Must match format objects names (see Formats configuration below).

- `general.defaultFormat` (string): default selected/used format. Must be one of the above list items.

- `general.resolutions` (comma-separated list of integers): list of available resolutions in dot-per-inch (dpi). To have legend icons generated in same resolutions, you must configure resolutions in plugin Layers (see Section 6.4, "Layers Legends").

- `general.defaultResolution` (integer): default selected/used resolution. Must be one of the above list items.

- `general.scales` (comma-separated list of integers): list of available printing scales (1000 should be read 1:1000). Used only when guiMode is 'rotate'.

- `general.defaultScale` (integer): default printing scale. Must be one of the above list items. Used only when guiMode is 'rotate'.

- `general.defaultOrientation` (portrait|landscape): sheet orientation. The same orientation is used for every pages of the PDF document.

- `general.activatedBlocks` (comma-separated list of strings): names of blocks that will be used in the document. Names must match block objects names. See Blocks configuration section.

- `general.overviewScaleFactor` (float): sets the extension of the overview map. The bigger this parameter, the wider the extension. Default: 10.

- `general.overviewColor` (string): sets the inner color of the current map outline in the overview. Default: 'none'.

- `general.overviewOutlineColor` (string): sets the border color of the current map outline in the overview. Default: 'red'.

- `general.showRefMarks` (boolean): if true, reference marks are added to the exported main map. They are never displayed on the overview.

- `general.pdfEngine` (CwFpdf|PdfLibLite): name of the PDF engine to use. For now only FPDF is available and must be called using "CwFpdf" pdfEngine value. A PDFLib Lite version is (very) partially implemented and as a result should not be used for now (pdfEngine value: "PdfLibLite").

- `general.pdfVersion` (string): PDF version that must be used (only available with "PdfLibLite" pdfEngine).

- `general.output` (inline|attachment|link|redirection): indicates how generated PDF file must be output (default: redirection). "inline": file is generated and displayed on the fly (might not work with buggy IE). "attachment": generated on the fly but a dialog box asks the user weither the file must be opened or saved. "link": PDF file is actually written on the server and a link pointing towards it is displayed. "redirection": PDF file is written on the server and than user's browser is automatically redirected towards it.

- `general.filename` (string): indicates how generated PDF files must be named. It is possible to specify a date-formatting using PHP date format (see

http://php.net/date). Default is `map-[date,dMY-His].pdf` which gives for instance `map-03Jan2005-193256.pdf`.

- *general.distUnit* (mm|cm|pt|in): unit used to measure blocks distance properties. Only one unit can be used in the whole configuration.

- *general.horizontalMargin* (float): horizontal distance between the sheet border and the useful area (in distUnit).

- *general.verticalMargin* (float): vertical distance between the sheet border and the useful area (in distUnit).

- *general.allowedRoles* (comma-separated list of strings): list of roles allowed to print PDF documents.

- *general.importRemotePng* (boolean): if true, remote PNG files (gathered using a URL) are copied in a local directory before being included in the PDF document. This option is recommended when using physically separated CartoClient and CartoServer (See Chapter 4, *Configuration Files*).

- *general.extraFonts* (comma-separated list of strings): list of additional fonts to import in FPDF (not available for other pdfEngines). Each font contains the font family name, the font style (optional) and the font PHP definition file (optional), separated by "/". PHP definition files must be uploaded in FPDF fonts directory. Parameters are described in FPDF AddFont() documentation [http://fpdf.org/en/doc/addfont.htm]. Definition files generation is explained here [http://fpdf.org/en/tutorial/tuto7.htm]. Example:

```
general.extraFonts = Comic/I/comici.php, MyFont/BI/myfontbi.php, MyOtherFont
```

PHP built-in default values:

- pdfEngine: CwFpdf
- pdfVersion: 1.3
- guiMode: classic
- distUnit: mm
- horizontalMargin: 10
- verticalMargin: 10
- formats: NULL
- defaultFormat: NULL
- resolutions: 96
- defaultResolution: 96
- activatedBlocks: NULL
- filename: map-[date,dMY-His].pdf
- overviewScaleFactor: 10

- overviewColor: NULL
- overviewOutlineColor: red
- output: redirection
- allowedRoles: (empty ie. nobody)
- importRemotePng: false
- extraFonts: NULL

Tip

Should you use the `rotate guiMode`, do not forget to add a general PROJECTION tag in your mapfile. For instance:

```
PROJECTION
  "init=epsg:21781"
END
```

12.2.2. Formats Configuration

Formats are described as a set of "formats" sub-objects. For instance:

```
formats.A4.bigDimension = 297
formats.A4.smallDimension = 210

formats.A3.label = A3
formats.A3.bigDimension = 420
```

Format ids (A3, A4, etc.) must match those listed in general.formats.

- `formats.`<format_name>`.label` (string): user-friendly name. Is translated using Cw3I18n device Chapter 16, *Internationalization*.
- `formats.`<format_name>`.bigDimension` (float): largest sheet side (in distUnit).
- `formats.`<format_name>`.smallDimension` (float): smallest sheet side (in distUnit).
- `formats.`<format_name>`.horizontalMargin` (float): optional, if set, overrides general.horizontalMargin for the given format.
- `formats.`<format_name>`.verticalMargin` (float): optional, if set, overrides general.verticalMargin for the given format.
- `formats.`<format_name>`.maxResolution` (integer): optional, maximum allowed resolution for the given format. If selected resolution is above maxResolution, it will be set to maxResolution value.
- `formats.`<format_name>`.allowedRoles` (comma-separated list of strings): list of roles allowed to use the format.

PHP built-in default values: all format parameters have NULL default values except `allowedRoles` ("all").

12.2.3. Blocks Configuration

All blocks parameters are optional since default values are hard-coded in the PDF export plugin PHP code (see code documentation for PdfBlock class). Those default properties can be partially or totally overriden using a template object whose syntax is similar to this:

```
template.type = text
template.fontFamily = times
template.fontSize = 12
template.fontItalic = false
```

whereas real blocks are described this way:

```
blocks.title.weight = 10
blocks.title.verticalBasis = top
blocks.title.verticalMargin = 15
```

"Template"-described properties have a global scope whereas "blocks" ones are specific to the given block. Properties names are identical for both "template" and "blocks" objects. Properties not related with the given block type are generally ignored (for instance "fontSize" is not taken into account if block is an image). A lots of properties use CSS-like denominations and effects. All following parameters names must be precede by either "template." or "blocks.<block_name>." prefixes.

- `type` (text|image|table|legend): block type, indicates what kind of processing must be applied.
- `content` (string): optional, for "text" blocks: the textual content ; for "image" blocks, the image filename ; for "table" blocks, a comma-separated list of textual content (limited to one row, commas delimit cells).
- `fontFamily` (string): name of font family to use. Naming depends on used pdfEngine.
- `fontSize` (float) : font size in points (pt)
- `fontItalic` (boolean): if true, text will be emphasized.
- `fontBold` (boolean): if true, text will be bold-weighted.
- `fontUnderline` (boolean): if true, text will be underlined.
- `color` (string): color of text in hexadecimal code #rrggbb. A few color aliases ("white", "black") are available as well.
- `backgroundColor` (string): background color in hexadecimal code or color aliases.
- `borderWidth` (float): border line width in distUnit.

- `borderColor` (string): border line color in hexadecimal code or color aliases.
- `borderStyle` (solid|dashed|dotted): border line style. Only available with "PdfLibLite" pdfEngine. "solid" value is used for other pdfEngines.
- `padding` (float): vertical and horizontal distance between block content and its borders in distUnit.
- `horizontalMargin` (float): horizontal margin in distUnit around the block (outside borders).
- `verticalMargin` (float): vertical margin in distUnit around the block (outside borders).
- `horizontalBasis` (left|right): indicates what document side must be used for horizontal positioning.
- `verticalBasis` (top|bottom): indicates what document side must be used for vertical positioning.
- `hCentered` (boolean): if true, block is horizontally centered (overloads horizontalBasis property).
- `vCentered` (boolean): if true, block is vertically centered (overloads verticalBasis property).
- `textAlign` (center|left|right): indicates how content must be horizontally aligned within the block extent.
- `verticalAlign` (center|top|bottom): indicates how content must be vertically aligned within the block extent.
- `orientation` (horizontal|vertical): indicates if content will be displayed form left to right or from bottom to top (only for text blocks).
- `zIndex` (integer): indicates how overlapping blocks must be stacked. The higher the zIndex, the higher the block in the stack (foreground).
- `weight` (integer): indicates in what order, blocks with the same zIndex must be processed. Low-weighted blocks are handled first.
- `inNewPage` (boolean): indicates if a new page must be added before printing the block.
- `inLastPages` (boolean): indicates if block must be added in a new page at the end of the document.
- `width` (float): block width in distUnit.
- `height` (float): block height in distUnit.
- `multiPage` (boolean): if true, given block will be displayed on every document page.
- `parent` (string): if specified, current block will be printed inside given parent block.
- `inFlow` (boolean): if true, block will be printed right after previous block, preserving the vertical alignment.

- *caption* (string): for "table" blocks only: name of separated block used to describe table caption formatting.
- *headers* (string): for "table" blocks only: name of separated block used to describe table headers (colunms titles) formatting.
- *allowedRoles* (comma-separated list of strings): list of roles allowed to draw the current block.
- *i18n* (boolean): if true, block content is translated using CartoWeb Gettext system ("i18n" stands for InternationalizatioN). Translated contents may be standard texts or even URLs (useful for instance to display a logo adapted to the used language).

Some additional block-properties are available but are automatically set depending on other parameter values and thus are not listed above ("private" access).

PHP built-in default values:

- type: NULL
- content: (empty string)
- fontFamily: times
- fontSize: 12
- fontItalic: false
- fontBold: false
- fontUnderline: false
- color: black
- backgroundColor: white
- borderWidth: 1
- borderColor: black
- borderStyle: solid
- padding: 0
- horizontalMargin: 0
- verticalMargin: 0
- horizontalBasis: left
- verticalBasis: top
- hCentered: false
- vCentered: false
- textAlign: center
- verticalAlign: center
- orientation: horizontal

- zIndex: 1
- weight: 50
- inNewPage: false
- inLastPages: false
- width: NULL
- height: NULL
- multiPage: false
- parent: false
- inFlow: true
- caption: (empty string)
- headers: (empty string)
- allowedRoles: all
- i18n: false

12.2.4. Colors definition

Colors definitions in `exportPdf.ini` configuration file are done using hexadecimal color codes or a few textual aliases such as *white*, *black*, *red*, *green*, *blue*. For instance:

```
general.overviewOutlineColor = red
template.backgroundColor     = white
template.borderColor         = #000000   ; hexadecimal code for black
blocks.title.color           = #0000ff   ; hexadecimal code for blue
```

12.3. Tutorial

12.3.1. General Principle

The PDF document to generate is described using elementary objects blocks:

- texts: titles, copyrights, dates...
- images: maps, logos...
- tables: queries results...
- legend: list of icons+labels for each printed layers

Those blocks are drawn and positioned on the document pages according to properties detailed in each block.

In addition, some data must be specified to describe the general document

characteristics such as formats (pages size), orientation, available resolutions, PDF engine and more.

Parameters containing several values are represented using comma-separated lists of strings/integers. If some values contain special characters such as accentuated letters, quotes or other exclamation marks, it may be wiser to encapsulate the whole parameter value between double quotes (").

Warning

Note3: don't forget to activate the plugin by adding it in the `loadPlugins` from BOTH CartoClient and CartoServer configuration files: `client_conf/client.ini` and `server_conf/<mapId>/<mapId>.ini`. For instance:

```
; in client_conf/client.ini:
loadPlugins = hilight, outline, exportPdf

; in server_conf/<mapId>/<mapId>.ini:
mapInfo.loadPlugins = hilight, outline, exportPdf
```

12.3.2. Overall Configuration

Two objects are available for overall configuration:

12.3.2.1. general

This is where you can set the list of available resolutions and formats (made available to the user through dropdown menus). Use commas to separate the available values. `defaultResolution` and `defaultFormat` are used to preselect options in the matching dropdowns. If for some reason the system failed to determine what resolution and format the user desires, it will use those default values.

Specify the default orientation to select in user interface as well. You don't need to set a list of available orientations since it is already fixed to "portrait, landscape".

The `activatedBlocks` parameter lists the blocks that may be drawn in the document. It does not mean that they will. For instance the title block will not appear if no title input has been submitted by the user.

Choose the PDF generator you want to use. For now only FPDF is fully available. PDFLib is only partially implemented and should not be used.

Set the `output` parameter in order to choose how the generated file will be served: download box, link, inline...

The `distUnit` parameter is very important since it sets the unit used to measure the

length specified in every blocks. However it does not affect font sizes, that are always indicated in points (pt).

horizontalMargin and *verticalMargin* indicate how much space will be kept blank around each page. No block will be printed below these distances from the sheets borders. Note that these values can be overriden in the formats description.

For instance:

```
general settings
general.guiMode = classic
general.formats = A4, A3
general.defaultFormat = A4
general.resolutions = 96, 150, 300
general.defaultResolution = 96
general.overviewScaleFactor = 10
general.defaultOrientation = portrait
general.activatedBlocks = mainmap, title, note, scalebar, overview,
                          copyright, queryResult, page, legend,
                          logo, scaleval, tlcoords, brcoords

general.pdfEngine = CwFpdf
general.pdfVersion = 1.3
general.output = inline
general.filename = "map-[date,dMY-Hi].pdf"
general.distUnit = mm
general.horizontalMargin = 10
general.verticalMargin = 10
general.importRemotePng = false
```

Some overview map settings are available as well. The *overviewScaleFactor* indicates how wide the overview will be compared to the mainmap actual extent. *overviewColor* and *overviewOutlineColor* respectively set the inner and border colors of the mainmap outline in the overview map. Note that the *outline* plugin must be activated (ie. added in the *client.ini* *loadPlugins* parameter) for those settings to be taken into account.

Tip

To use the classical MapServer keymap in place of the overview, set the overview block *content* parameter to *static* (see Section 12.3.4, "Blocks Configuration") as follows:

```
blocks.overview.content = static
```

In that case parameters *overviewScaleFactor*, *overviewColor*, *overviewOutlineColor* are ignored.

12.3.2.2. formats

Formats objects describe the PDF pages sizes (*bigDimension*, *smallDimension*). One can describe as many formats as desired. Moreover there is no theoritical page

size limit except that the bigger the maps, the longer the document generation.

A format is determined by its biggest dimension (`bigDimension`) and its smallest dimension (`smallDimension`).

The `label` parameter will be translated using Cw3 I18n internationalization system (Chapter 16, *Internationalization*). It is not required to be identical to the format object id. On the other hand, the latter id must be textually listed in `general.formats` parameter or else it will not be taken into account.

In addition, a couple of format parameters can be specified: `horizontalMargin` and `verticalMargin` override the corresponding general parameters if different margins must be applied for the given format. `maxResolution` indicates the highest allowed resolution for the given format: it enables to limit the applied resolution when, for instance, printing big-dimensioned maps.

For instance:

```
; formats settings
formats.A4.label = A4
formats.A4.bigDimension = 297
formats.A4.smallDimension = 210

formats.A3.label = A3
formats.A3.bigDimension = 420
formats.A3.smallDimension = 297
formats.A3.horizontalMargin = 30
formats.A3.verticalMargin = 30
formats.A3.maxResolution = 150
```

12.3.3. Form Settings

The HTML interface of the PDF tool is fully customisable by:

- editing the `general` attributes for formats, resolutions and orientation lists (see Section 12.3.2.1, "general" above).
- adding or editing blocks `content` parameters to specify default values for text inputs (such as title or note) and other options (legend, scalebar, overview, query results,...). See Section 12.3.4, "Blocks Configuration". For instance:

```
blocks.title.content = "My default title"
blocks.scalebar.content = 1   ; scalebar is checked by default
blocks.legend.content = in    ; legend is checked as included in map by default
```

12.3.4. Blocks Configuration

Whatever their types (image, text, table, legend, north), blocks use the same PHP object model and, as a result, the same object properties. Some parameters can be used in several ways depending on the block type. Some others are simply ignored.

Blocks naming is quite free but some names are reserved to system-defined blocks such as title, mainmap, overview, scalebar, note, copyright, queryResult, legend, page, scaleval, tlcoords, brcoords, north. System blocks should not be renamed.

Note that blocks that are not mentioned in `general.activatedBlocks` won't be displayed in any case.

```
general.activatedBlocks = mainmap, title, note, scalebar, overview,
                          copyright, queryResult, page, legend,
                          logo, scaleval, tlcoords, brcoords, north
```

12.3.4.1. Block Template and Overriding

Since all blocks descriptions are based on the same PdfBlock object, a template block has been defined to factorize blocks common configuration (for instance font-family, background color, borders, padding, etc.). It is also a way to specify default blocks parameters values. It is then possible to personalize blocks by overriding those properties within the block own description. Blocks configuration overriding can be schemed as follows:

PHP PdfBlock (hard coded) >>> block template (user configured) >>> final block (user configured)

Note that it is not necessary to redefine properties in blocks or in their template if their current values (defined in "parent" structures) are already OK.

For instance:

```
; blocks default settings
template.type = text
template.fontFamily = times
template.fontSize = 12
template.fontItalic = false
template.fontBold = false
template.color = black
template.backgroundColor = white
template.borderWidth = 0.25
template.borderColor = black
template.borderStyle = solid
template.padding = 3
template.horizontalMargin = 0
template.verticalMargin = 0
template.horizontalBasis = left
template.verticalBasis = top
template.hCentered = false
template.zIndex = 1
template.textAlign = center
template.verticalAlign = center
template.orientation = horizontal
```

12.3.4.2. Blocks Positioning

CartoWeb uses a CSS-like syntax for blocks description and positioning. Description

depends on the block type and is detailed in following sections.

Blocks are positioned one after the other, beginning by the ones with the lowest `zIndex` (vertical position: the low-zIndex'ed blocks are placed under the high-zIndexed ones - ie. closer to the background). When blocks have equal zIndexes, those with lowest `weight` are processed first. Eventually, if blocks have identical zIndexes and weights, the system will use `general.activatedBlocks` order to make its choice.

Blocks with different zIndexes wil not interact except if one is marked has the other's `parent`: in that situation the child block will be located inside the parent block instead of using the general referencial. All following siblings will share the same parent block except if they have a `inFlow = false` property. Parent blocks must have lower zIndexes than their children.

For instance:

```
blocks.mainmap.zIndex = 0
blocks.mainmap.weight = 10

[...]

blocks.overview.parent = mainmap
blocks.overview.zIndex = 1
```

Block margins are used to position a given block. Horizontal positioning is achieved by specifying the `horizontalBasis` (the side of the document - left|right - used as a reference) and the `horizontalMargin` (the latter value tells how far in distUnit the block will be spaced from the reference line). For instance to position a block at 10mm from the document right side, use the following configuration :

```
general.distUnit = mm

[...]

blocks.myBlock.horizontalBasis = right
blocks.myBlock.horizontalMargin = 10
```

If you want to horizontally center a block, it is simply done using

```
blocks.myBlock.hCentered = true
```

Block centering overloads any other kind of positioning (margin...).

Vertical positioning is achieved in the same way (substitute the dedicated keywords).

The `inFlow` parameter defaults to true (except if set differently in the template block). As a result, blocks with the same `zIndex` will be positioned right below the first block preserving the left-side alignment. To cancel this behavior, set `inFlow` to false.

In that case, the given block will be positioned related to the previous block using margin or centering ways.

12.3.4.3. Text Blocks

Text blocks are boxes with a textual content. The text is set in the `content` parameter in case of a static block. `content` has to be left blank for visitor-set blocks such as title or note. Text formatting is achieved using usual CSS-like parameter : `fontFamily`, `fontSize` (expressed in points!!), `color`. Small exception: `fontItalic`, `fontBold`, `fontUnderline` are booleans.

Box properties (background color, border size, color and style, padding) are set in the same way whatever the block type is.

Width is generally detected automatically according to the text length. On the other hand, the system has poor means to evaluate its height, so it is recommended to set it by hand. Note that padding (space between border and content) is taken into account only in the horizontal direction for text blocks. To remove a block border, one can simply set its width (`borderWidth`) to 0. Note that some border styling parameters are not available with all pdfEngines.

For instance:

```
blocks.title.zIndex = 2
blocks.title.weight = 10
blocks.title.verticalBasis = top
blocks.title.verticalMargin = 15
blocks.title.hCentered = true
blocks.title.fontSize = 20
blocks.title.fontItalic = true
blocks.title.fontBold = true
blocks.title.fontUnderline = true
blocks.title.backgroundColor = #eeeeee
blocks.title.height = 15
```

It is possible to specify a text content in the configuration file by using the `content` parameter. Note that `\n` strings in the `content` value will result in line feeds in the final block.

```
blocks.myText.content = "This is myText block content!"

blocks.myText2.content = "myText2 block\nis multilined."
```

Text blocks content may also be retrieved from external data source such as text files or databases using following syntaxes. "Connection" parts are separated by a "~". First part indicates what medium (`file`|`db`) is used:

- `file`: file source

```
blocks.myText.content = "file~my/source/file/path"
```

The source file path can either be a absolute URL or a filesystem path. Relative filesystem paths are allowed and are based on the CartoWeb root path. For instance:

```
blocks.myText.content = "file~http://myserver/mysourcefile.txt"
blocks.myText.content = "file~C:\mydir\mysourcefile.txt"
blocks.myText.content = "file~/home/johndoe/mysourcefile.txt"
blocks.myText.content = "file~www-data/mysourcefile.txt"
```

If your source file contains several lines or `\n` strings, the generated block will also be multilined.

- `db`: database source

```
blocks.myText.content = "db~DSN~SQL query"
```

The DSN (Data Source Name) gives the database connection parameters. Any DSN format supported by PEAR::DB can be used. See PEAR::DB manual for more info about DSNs at http://pear.php.net/manual/en/package.database.db.intro-dsn.php. Note that you can call any database server type as long as it is supported by PEAR::DB and that your PHP distribution has the matching extension enabled.

For security reasons, only SELECT SQL queries are allowed but there is no other limitation. On the other hand, the query results are basically rendered using a new line for each result line and simply separating result elements with blank spaces.

For instance:

```
blocks.myText.content = "db~pgsql://user:password@localhost/dbname~select name,
                         phone_number from phone_book where area_id = 12"
```

Tip

It is possible to make some text blocks have their contents translated by setting their `i18n` parameter to `true`. For instance:

```
blocks.myTextBlock.content = "This text will be translated."
blocks.myTextBlock.i18n = true
```

Translatable blocks are CartoWeb built-in blocks `copyright` and `date` as well as

your own text blocks. Some other CartoWeb built-in blocks such as `legend`, `scaleval`, `page` or query results table blocks are translated anyway.

Note that you will have to explicitly add those entries to your Gettext PO files since there is no way for script `client2pot.php` to automatically detect them!

Tip

The `date` text block is fully customizable. You may specify the date format using PHP date formatting (http://php.net/date). Both configurations are correct:

```
; use [] to add the date string in complete sentence:
blocks.date.content = "Generated on [Y/m/d H:i] by CartoWeb3"

; or if the date string is alone:
blocks.date.content = "Y/m/d H:i"
```

If you want to have the date block translated, set its `i18n` parameter to `true` and add the translated version (including date format) to your PO files.

12.3.4.4. Image Blocks

Image blocks are used to display the maps (main + overview), the scalebar and other user-defined pictures such as logos or diagrams. Except for system image blocks (maps + scalebar), the image file location must be specified in content using either an absolute URL or an absolute path within the server file system. It is also possible to use a relative file path based on the CartoClient root directory (eg. `htdocs/gfx/layout/c2c.png`).

`width` and `height` have to be set by the user.

For instance:

```
blocks.logo.type = image
blocks.logo.content = http://server/gfx/layout/c2c.png
blocks.logo.width = 40
blocks.logo.height = 7
blocks.logo.padding = 2
blocks.logo.parent = mainmap
blocks.logo.horizontalBasis = right
blocks.logo.inFlow = false
blocks.logo.horizontalMargin = 5
blocks.logo.verticalMargin = 5
```

Tip

If you want to switch the image file depending on the used language, set the block `i18n` parameter to `true` and add languages matching URLs/filepathes to your PO

files.

12.3.4.5. Table Blocks

Tables blocks are used to represent tabular data such as query results. A table is composed of a caption row (title), a headers row (columns/fields titles) and one or more data rows. All cells can only contain textual content.

The table block describes how data cells will be rendered (font, text color, background color...) using the same parameters than for text blocks. The table width will be computed according to the cells content with a upper limit materialized by the page or the parent block available extent. If the max extent is reached, line feeds will be added automatically to the cells contents. The `height` parameter gives the height of a single-lined row (and not the total table height!). Note that if a table is too high to fit a page, page breaks will automatically be added.

Caption and headers are described using separated blocks. They are specified by filling the table block `caption` and `headers` fields with the matching blocks ids. Since they are not standalone, there is no need to activate the blocks in the `general.activatedBlocks` list. That way it is possible to set the caption and headers layouts separately from the general table one (which is their default layout). Note that these subblocks are optional and will not be displayed if no content can be determined for them. The same headers and caption subblocks can be shared amongst several table blocks.

CartoWeb output table content (and its caption and headers subblocks) is automatically determined. But it is also possible to user-define independent tables by filling the table and its caption/headers `content`: cells content are represented as comma-separated string lists (only one row available for table data).

For instance:

```
blocks.queryResult.type = table
blocks.queryResult.inNewPage = true
blocks.queryResult.caption = tableCaption
blocks.queryResult.headers = tableHeaders
blocks.queryResult.height = 10
blocks.queryResult.verticalMargin = 10
blocks.queryResult.hCentered = true

blocks.tableCaption.backgroundColor = #ff0000
blocks.tableCaption.color = white
blocks.tableCaption.content = "Table test"
blocks.tableCaption.height = 15

blocks.tableHeaders.backgroundColor = #ffaaaa
blocks.tableHeaders.fontBold = true
blocks.tableHeaders.color = #eeeeff
```

Tip

A special table block exists to specify which TableGroup (see Section 2.4.3.1, "Tables Structures") you want to display in the PDF.

```
blocks.queryResult.content = query
```

In that case only the TableGroup *query* is visible but you can specify more than one, separate by comma. If this block is not defined in the configuration file, all TableGroup are visible in the PDF file (backward compatibility).

The query results data come from the plugin Table which offers the mechanism of table rules (see Section 2.4.3.2, "Setting Rules") which describes how tables must be displayed. You can also add table rules specificaly for the PDF output like this example :

```
$outputType = $this->cartoclient->getOutputType();

if ($outputType == Cartoclient::OUTPUT_HTML_VIEWER) {
    /* rules for HTML output */
}

if ($outputType == 'pdf') {
    /* rules for PDF output */
}
```

outputType is generally the export plugin name without the "export" prefix.

12.3.4.6. Legend Block

There is usually only one instance of legend block in a CartoWeb PDF document. It gives the list of the {selected layer/class + icon} couples. Content is automatically retrieved according to selected layers. You only need to set the text formatting and row height (as for a table block). Legend blocks does not provide any table-like headers or caption.

You can avoid that a given set of layers be displayed in the legend blocks (no change on the maps) by listing the "forbidden" layers ids in the legend block *content* parameter, prefixing each layer id with an exclamation point ("!"). For instance:

```
blocks.legend.content = "!background, !layer3"
```

Note the double quotes around the content value that contains special characters such as "!".

Two behavior are offered to the visitor: embedding the legend block in the map or printing it in a separated page. First case is OK when there is little legend items to

print (block is limited to one column) whereas the second possibility enables to display lots of legend items, using several columns if needed. In the map-embedded case, if the overview map is printed, the legend block `width` will be automatically set to the overview `width` value.

For instance:

```
blocks.legend.type = legend
blocks.legend.zIndex = 1
blocks.legend.weight = 40
blocks.legend.fontSize = 8
blocks.legend.height = 5
blocks.legend.width = 50
blocks.legend.padding = 1.5
blocks.legend.verticalMargin = 10
blocks.legend.horizontalMargin = 10
blocks.legend.content = "!nix"
```

12.3.4.7. North Arrow Block

There is usually only one instance of north arrow block in a CartoWeb PDF document. It displays an arrow (or any image) that will automatically show the north direction. Path set in arrow block `content` parameter is relative to CartoWeb root directory.

For instance:

```
blocks.north.type = north
blocks.north.content = "plugins/exportPdf/htdocs/gfx/arrow.png"
blocks.north.width = 10
blocks.north.height = 36
blocks.north.padding = 2
blocks.north.parent = mainmap
blocks.north.verticalBasis = bottom
blocks.north.horizontalBasis = left
blocks.north.inFlow = false
blocks.north.horizontalMargin = 15
blocks.north.verticalMargin = 5
```

12.3.5. Roles Management

CartoWeb enables to perform roles restrictions on most of the PDF document parts: general access, formats, resolutions, blocks. All these elements can be restricted to some authorized users. For a more detailed discussion of the concept of security restriction and roles, see Chapter 15, *Security Configuration*.

To do so, simply add to considered objects/blocks an `allowedRoles` property set to the comma-separated list of roles you want to restrict them to.

For instance:

- to enable PDF printing only for `loggedIn` users, use:

```
general.allowedRoles = loggedIn
```

- to reserve A3 format to editors or admins, use:

```
formats.A4.allowedRoles = editor, admin
```

- to set a resolution limit for *anonymous* users when printing in A3, you can define 2 similar formats with different *maxResolution* and *allowedRoles* properties:

```
general.formats = A4, A3, A3full

[...]

formats.A4.label = A4
formats.A4.bigDimension = 297
formats.A4.smallDimension = 210

formats.A3.label = A3
formats.A3.bigDimension = 420
formats.A3.smallDimension = 297
formats.A3.maxResolution = 150
formats.A3.allowedRoles = anonymous

formats.A3full.label = A3full
formats.A3full.bigDimension = 420
formats.A3full.smallDimension = 297
formats.A3full.allowedRoles = loggedIn
```

Note that all formats that may be used must any way be specified in *general.formats*.

- to hide a block for non-admin users, use:

```
blocks.overview.allowedRoles = admin
```

13. Views

13.1. Introduction

Views are recordings of CartoWeb maps states at a given moment. One may see them as a kind of bookmarks: while browsing maps in CartoWeb interface, users can save the current state (map extent, selected layers, queries, annotations, etc.) under a label of their choice. It is then possible to access later those saved states.

Views are shared among users and are - in the current version - available to every user. Access to views edition functionalities (creation, modification, deletion) can be restricted to some categories of users. The default configuration only allows "admin"-roled users to perform view edition.

13.2. Views Configuration

The views device consists of two parts:

- a main controlling system that handles views detection, loading, recording and storage.
- a "views" plugin that provides a graphical interface for performing views loading and edition. This plugin is not mandatory to have views working but greatly enhances the device usability.

13.2.1. Views Plugin

No configuration is required for that plugin. However it still needs to be activated using the `client_conf/client.ini` *loadPlugins* parameter. For instance:

```
loadPlugins = auth, exportPdf, views
```

13.2.2. Main Views Controller

Main views controller configuration is done in the `client_conf/client.ini` file. The dedicated parameters are:

- *viewOn* (boolean: 'true'|'false'): activates/deactivates the main view controller. Views plugin will not work if the controller is off.
- *viewStorage* (string: 'file'|'db'): tells what storage container must be used: file or database.
- *viewablePlugins* (comma-separated list of strings): list of plugins that must

have their data saved in views. Note that "viewable" plugins must implement the Sessionable interface. If you wish to save data from extended plugins, you must add the extended plugin to the list, not the standard one (eg. `myLocation` instead of `location`).

- *viewMetas* (comma-separated list of strings): list of metadata fields that must be saved in addition to the viewable plugins data (for instance author, email, description...). Common metadata fields such as "viewTitle" are anyway saved. Note that metadata fields must have homonym form fields in `plugins/views/templates/views.tpl` to be taken into account.

- *viewDsn* (string): DSN (Data Source Name), gives the database connection parameters. Only used in database storage mode. Any DSN format supported by PEAR::DB can be used. See PEAR::DB manual for more info about DSNs at http://pear.php.net/manual/en/package.database.db.intro-dsn.php. Note that you can call any database server type as long as it is supported by PEAR::DB and that your PHP distribution has the matching extension enabled.

- *viewAuth* (comma-separated list of strings): list of roles allowed to perform views edition actions. Given roles must match `client_conf/auth.ini` defined ones. See also Chapter 15, *Security Configuration*.

- *viewSavedRequest* (comma-separated list of strings): list of REQUEST variables (GET, POST, COOKIE) that must be preserved when loading a view. Some are already built-in listed such as *tool*, *js_folder_idx*, *project*, *collapse_keymap* and views related parameters. Adding other CartoWeb reserved REQUEST variables names to the list may prevent the view system from working correctly.

- *viewLogErrors* (boolean): if true, outdated views loadings will be logged in `log/viewErrors.log`.

- *viewUpgradeOutdated* (boolean): if true, CartoWeb will try to upgrade outdated views on the fly. Else view outdated parts will be discarded and current plugin session data will be used instead.

- *viewHooksClassName* (string): name of the PHP class used to customize some views processing. See Section 11.3, "Customizing Views Processing Using Project Hooks" for details on how to write customized views hooks.

For instance:

```
; Views management
viewOn = true
viewStorage = db
viewablePlugins = location, layers, query, outline
viewMetas = author
viewDsn = pgsql://www-data@localhost/cw3
viewAuth = admin,
viewSavedRequest = personalVar1, personalVar2
viewLogErrors = true
viewUpgradeOutdated = true
viewHooksClassName = FoobarViewHooks
```

Should you decide to use a database storage, you would need first to create an appropriate table called `views`. Here is an example of SQL statements for PostgreSQL:

```
CREATE TABLE views (
        views_id serial NOT NULL PRIMARY KEY,
        views_ts timestamp without time zone,
        viewtitle character varying(50),
        author character varying(50),
        sessiondata text,
        viewshow boolean,
        weight integer
);
```

Tip

Field "views_ts" is just present in table "views" but not visible in the graphical interface of the "views" plugin. This value is usefull to know the time of the creation/modification of a view (only available with option "viewStorage = db").

13.3. Views Usage

13.3.1. Loading Views

Any defined view may be loaded by any users. Loading a view may be achieved selecting it among a dropdown views list. Two dropdown lists are available:

- a public one that only displays views labels that have been marked as "visible" by their creators.
- a selector within the Views plugin edition interface. This view selector contains all views including the ones that have not been marked as "visible" by their creators. Note that this selector provides a view ID (integer) input as well.

Views may also be loaded using GET parameters in URLs that look like:

```
http://<cartoweb/client/base/url>?handleView=1&viewLoad=1&viewLoadId=<viewId>
```

or simply

```
http://<cartoweb/client/base/url>?view=<viewId>
```

Tip

Views not marked as "visible" may anyway be loaded by anyone. "Visible" only means that their labels appear in the public view dropdown selector whereas "invisible" ones don't.

13.3.2. Editing Views

Views edition is restricted to users whom roles match at least one of those specified in the `client_conf/client.ini` *viewAuth* parameter (see Section 13.2.2, "Main Views Controller").

Views editors can create, update or delete views.

13.3.2.1. Saving a View

To save the current map state as a view, simply fill in the form fields with matching metadata. To make the view "visible", check the "Show view" option. Finally push the "save" button.

You may also save a new view using an existing view. To do so, load the desired view as described in Section 13.3.1, "Loading Views", modify it as explained in Section 13.3.2.3, "Updating a View" and push the "Save as new view" button.

13.3.2.2. Deleting a View

Load the view you want to delete as shown above and push the "Delete" button. A JavaScript confirmation pops before irreversible deletion.

13.3.2.3. Updating a View

To modify a view map properties (selected layers, annotations, map extent, etc.) as well as its metadata (title, "visibility", etc.), load it as seen above.

If your changes only concern metadata, simply update the dedicated form fields and push the "Update" button.

To update the view map properties, check the "Memorize form" option. Metadata and selected view ID are then memorized while you perform your changes (pans, zooms, annotations, layers selections, etc.). Eventually push the "Update" button.

14. Editing

The `edit` plugin allows users to edit geographical data from their web browser.

14.1. Client-side Configuration

14.1.1. edit.ini

Here are the options that can be set on the client:

- general.allowedRoles: comma separated list of the roles that are allowed to use the edit plugin
- insertedFeaturesMaxNumber: integer, number of new features that user is allowed to draw for insertion in the database

  Set it to 0 if user is only allowed to update or delete features.
- editLayers: comma separated list of layers that are editable
- editResultNbCol: number of columns to use to display the attributes table
- editDisplayAction: this string parameter is used to tell if the "validate" and "cancel" buttons have to be displayed also under the attributes table. If empty or `both`, buttons will be displayed both in the folder and under the attributes table. If `folder`, buttons will be displayed only in the folder.

14.2. Related Elements in Mapfile

14.2.1. Metadatas

Specific metadatas must be set in the mapfile for the editable layers

```
METADATA
    ...
    'edit_table' 'edit_poly' # PostGIS table
    'edit_geometry_column' 'the_geom' # PostGIS geometry column
    'edit_geometry_type' 'polygon' # PostGIS geometry type
    'edit_srid' '-1'
    'edit_attributes' 'parc_id,name|string,culture|string,surf,parc_type|integer' # list of the editable
END
```

- edit_table: string, name of the postGIS table to edit
- edit_geometry_column: string, name of the geometry field in the table
- edit_geometry_type: string, type of the geometry of the features for the layer

  possible values are point, line, polygon

May differ from the msLayer type (rendering)

- edit_srid: integer (optional), id of the SRID
- edit_attributes: string, comma separated list of attributes used for edition.

  Each value represents the attribute name and type separated by a pipe symbol.
  Only attributes with type set are editable.

Warning

Attribute types available values are *'string'* and *'integer'*.

14.3. How To

In this section are described the steps to get the edit plugin working on a cartoweb
project.

- **First, you need a PostgreSQL database with postGIS enabled.** Let's say, it is
  named "edit_db" for the following explanations.
- **At that point, you may launch** *demoEdit_schema.sql* **and** *demoEdit_data.sql*.
  located in the

  `<CARTOWEB_HOME>/projects/demoEdit/`directory.

  This will create 3 sample geometry tables, one for each type of geometry.
- **In CartoWeb, you'll have to load the edit plugin in your project on both
  client and server sides.**

  ```
  loadPlugins = [...], edit
  ```

- **Then, you'll need to configure the plugin by setting the layer_ids in the
  editLayers parameter in the edit.ini file.**

  ```
  editLayers = edit_poly, edit_line, edit_point
  ```

- **In your mapfile, in the corresponding LAYER, you'll have to set the specific
  metadatas as following :**

  ```
  LAYER
    NAME "edit_poly"
    STATUS ON
    TYPE POLYGON
    CONNECTIONTYPE POSTGIS
    CONNECTION 'dbname=edit_db user=www-data password=www-data host=localhost'
    DATA 'the_geom from (select the_geom, oid, parc_id, name, culture, surf, parc_type  from edit_poly)
  ```

```
  TEMPLATE 'ttt'
[...]
  LABELITEM "name"
  CLASS
    NAME "class"
    STYLE
      COLOR 50 50 255
      OUTLINECOLOR 255 50 50
    END
    LABEL
[...]
    END
  END
  METADATA
    'id_attribute_string' 'parc_id' # query
    'query_returned_attributes' 'parc_id name culture surf parc_type'
    'edit_table' 'edit_poly' # PostGIS table
    'edit_geometry_column' 'the_geom' # PostGIS geometry column
    'edit_geometry_type' 'polygon' # PostGIS geometry type
    'edit_srid' '-1'
    'edit_attributes' 'parc_id,name|string,culture|string,surf,parc_type|integer' # list of the editab
    'edit_filter' '' # mapserver filter
  END
END
```

Warning

Make sure that your `edit_attributes` list does not include spaces between attributes.

15. Security Configuration

15.1. Introduction

Access to different parts of the CartoWeb can be allowed or denied according to who is currently using the application.

The following concepts are used in this chapter.

Security Mechanisms Concepts

User   Representation of a user accessing CartoWeb. If the user is not logged in, she is rerefenced as the `anonymous` user.

Role   A user can have zero or more roles associated to her. These roles are used to allow or deny a permission to a resource of feature.

Permissions   Permissions describe parts of the application which can be allowed or denied access. A permission can have roles for which access is allowed, and roles for which it is denied.

15.2. Auth Plugin

The security system in CartoWeb was developped to be modular and to allow different authentications systems to be easily plugged-in. This section describes one implementation of authentication, user password and roles management, which is the auth plugin shipped with CartoWeb.
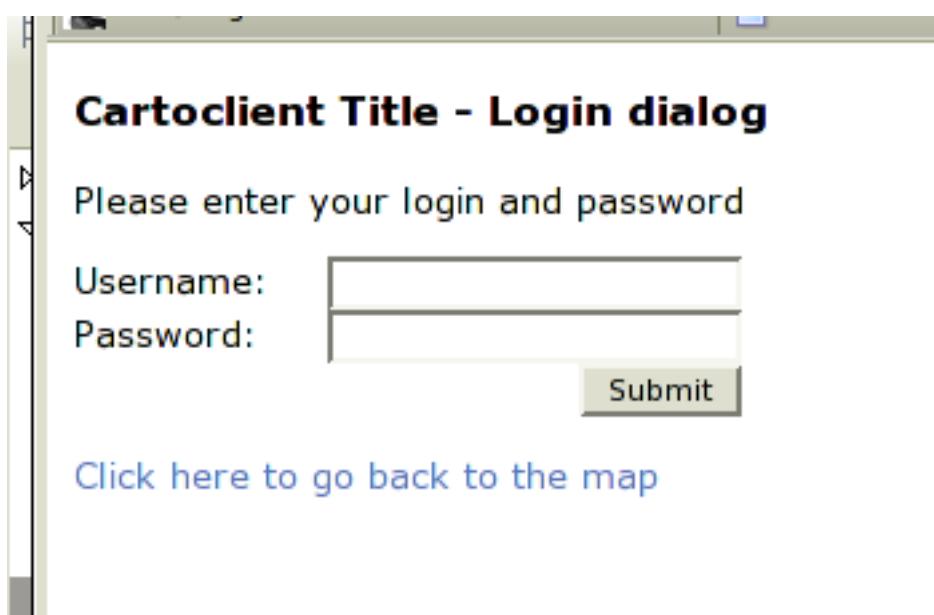
The auth plugin is not a core plugin. That's why you need to enable it if you want to enable users to log-in. See Chapter 4, *Configuration Files* how to enable it in the list of plugins. If it is not activated the login dialog won't be available, so users will remain anonymous.

The next section describes the configuration file of the auth plugin. It is basically the management of usernames, passwords and roles.

15.2.1. auth.ini (Client-side)

The `auth.ini` configuration is located in the client. This file contains the list of usernames, their password and the roles they belong to:

- authActive: boolean Whether to show the Authentication login/logout buttons. (note: this is not related to the fact the authentication system will be active or not)
- users.USERNAME List of users and their passwords. `USERNAME` is the name of the user for whom the password is set. Passwords values are md5sums of the password. To get this value, you can type in a shell:

  ```
  echo -n 'mypassword' | md5sum
  ```

  Example:

  ```
  users.alice = e3e4faee0cc7fc55ad402f517cdaf40
  ```

- roles.USERNAME List of roles for each user. `USERNAME` is replaced by the user for whom the roles are set. Some roles have a special meaning, see Section 15.2.2, "Special Role Names"

It is possible to use three different locations where user and password are stored. There are three mechanisms built-in in CartoWeb:

1. .ini file
2. database
3. ldap

The parameter `securityContainer` controls which backend is used.

- securityContainer: can be `ini` for the file storage, `db` for the dabase storage, or `ldap` for the ldap storage.

Parameters for database and ldap storage are described below

15.2.1.1. database storage

- dbSecurityDsn: The database connection string to be used. For instance
  `pgsql://user:password@localhost/test`.

- dbSecurityQueryUser: The SQL query to perform to check a username/password
  pair is valid. It should return a row if the pair is correct. For instance: `"SELECT *
  FROM users WHERE username='%s' AND password='%s'"`.

- dbSecurityQueryRoles: The SQL query used to retrieve roles for a given user. It
  should return one column with all roles for the user each row. For instance:
  `"SELECT roles FROM users WHERE username='%s'"`

15.2.1.2. ldap storage

The following parameters are inspired from Pear::Auth_Container_LDAP

- host: hostname of the server that hosts the ldap server `localhost` (default),
  `ldap.netsols.de`.

- port: Port number on which the ldap server listens `389` (default).

- basedn: the base dn (Distinguished Name) of the ldap server For instance:
  `dc=ldap,dc=netsols,dc=de`

- userattr: the user attribute to search for. `uid` (default)

- groupdn: gets prepended to basedn when searching for group. For instance:
  `ou=Groups`

- groupattr: the group attribute to search for. `cn` (default)

- groupfilter: filter that will be added to the search filter when searching for a
  group: `(&(groupattr=group)(memberattr=username)(groupfilter))`
  `(objectClass=groupOfUniqueNames)` (default). For instance:
  `(objectClass=posixGroup)` or `(cn=cw*)(objectClass=posixGroup)` if you
  have groups starting with "cw" specifically set for Cartoweb.

- memberattr: the attribute of the group object where the user dn may be found
  `uniqueMember` (default)

- groupscope: Scope for group searching: `one`, `sub` (default), or `base`

15.2.2. Special Role Names

The auth plugin configuration described in the previous section references the notion
of users and roles. Basically a role can be any string, the application will only use
them as a way to check if a feature is allowed or not. However, a set of role name
have a special meaning. They are described below.

Special Roles

all	This role belongs to any user. It is useful in case a permission should not be restricted access.
loggedIn	This role is given to all users who have logged in the application. It means they are identified to the system with a username.
anonymous	This role is given to all users not known to the application. This role is attached to anyone who has not yet entered her username and password.

Example 15.1. Special Role Name Usage

```
general.allowedRoles = loggedIn
```

15.3. Global CartoWeb Permissions

The whole application can be denied access to anonymous users: only authenticated (or a set) of users can access the application. Anonymous users arriving on the main page will see a login dialog page if they have no rights to view the page (if the `auth` plugin is not loaded, they will simply see a denied page).

This feature can be parametrized on the CartoClient in the `client.ini` configuration file:

- securityAllowedRoles = list : List of roles which are allowed to access the cartoweb. Set to 'all' if no if no restriction is given. (see Section 15.2.2, "Special Role Names" for typical predefined roles)

Example 15.2. Global Permissions Usage

```
securityAllowedRoles = loggedIn
```

15.4. Plugin Specific Permissions

The main part of permissions is set in the different plugins of CartoWeb. For instance, permissions related to which layer can be viewed are set in the layer plugin

configuration files, and pdf printing permissions in the pdf plugin. These sections either describes the plugin permissions which can be used, or make references the the corresponding chapters.

15.4.1. Layers Related Permissions

It is possible to restrict which layer can be viewed by a set of users. For instance, only logged in admin users can be allowed to view security sensitive layers.

These layer permissions are inherited by children. It means that if a parent LayerGroup is not allowed to be viewed, then all the children won't be visible.

The permissions for layer must be activated in the `layers.ini` CartoClient configuration file, and are set in the metadata of the mapfile or the `<layers>.ini` configuration file. The concept of metadata in mapfiles and .ini file is described in Section 6.3, "Metadata in Mapfile and layers.ini". The metadata key name which can be used is called `security_view` and contains a coma separated list of roles which are allowed to view this layer. If no such metadata key is associated to a layer or layerGroup, anyone can see the layer.

Heres the description of the `layers.ini` configuration file:

• applySecurity: True if the security check of layer access is activated. It might have a minor impact on performances, if a large number of layer is used.

Here's an example of a security metadata key used in layer of the mapfile:

```
METADATA
    "exported_values" "security_view,some_specific_parameter"
    "security_view" "admin"
    "some_specific_parameter" "value"

  "id_attribute_string" "FID|string"
  "mask_transparency" "50"
  "mask_color" "255, 255, 0"
  "area_fixed_value" "10"
END
```

In this example, the layer containing this metadata description will only be visible for users having the `admin` role.

Note

Notice the usage of the `exported_values` metadata key which lists the security related metadata key. It is explained in Section 6.3, "Metadata in Mapfile and layers.ini"

Now let's look the case where the metadata key is set on a layer group in the `<layers>.ini` configuration file:

```
layers.group_admin.className = LayerGroup
layers.group_admin.children = grid_defaulthilight
layers.group_admin.metadata.security_view = admin
```

In this example, the `admin` role is set for the layerGroup called `group_admin`. You can notice the very similar syntax as used in the mapfile.

Warning

Don't forget to set `applySecurity` to True in the `layers.ini` CartoClient configuration file, otherwise security metadata keys won't be taken into account.

15.4.2. PDF Printing Permissions

The roles management in Pdf printing is explained in detail in Chapter 12, *PDF Export*. In particular, see Section 12.3.5, "Roles Management".

16. Internationalization

16.1. Translations

Translation handling in CartoWeb was designed to use gettext [http://www.gnu.org/software/gettext/manual] . However internationalization architecture is ready for other translation systems.

Note

Scripts detailed below are only available in command line prompts ("DOS", shell). They cannot be launched using a web browser.

16.1.1. Configuration

For now only gettext translation system is implemented. If gettext is not installed, you can use a dummy translation system which translates nothing. To use gettext, you will need to have PHP gettext module installed.

Translation system is now automatically set to gettext if gettext is installed otherwise it's disabled. Parameter *I18nClass* is now deprecated.

16.1.1.1. Unix-like

In Unix-like environments, file `/etc/locale.alias` contains aliases to installed locales. For each language used, a line must be present in this file. The alias ('fr' in the example below) must point to a locale installed on the system.

```
...
fr    fr_CH.ISO-8859-1
...
```

You will need to run **locale-gen** after editing `/etc/locale.alias` to regenerate system's locales.

To install a locale on a Debian installation, use following command with root priviledges:

```
dpkg-reconfigure locales
```

If package locales has never been installed, you have to install it before:

```
apt-get install locales
```

Two scripts have been prepared to check if gettext is correctly installed on the

system:

- `scripts/testgettext.sh`: This will compile and run a small C program that will test **xgettext** and **msgfmt**.
- `scripts/testgettext.php`: Given that gettext was correctly installed on the system, this will test that gettext works in PHP.

Note

These scripts need that locales have been correctly configured. They were tested on GNU/Linux with a Debian/Sarge install.

16.1.1.2. Win32

In order to have the language translation working correctly in Cartoweb with Gettext, you need to have the languages you want to use installed in the *Input Locales* of Windows (windows 2000: Control Panel > Regional Options > Input Locales. windows XP: Control Panel > Regional and Language Options > Details > Settings)

16.1.2. PO Templates

Texts to be translated can be found in:

- Smarty templates using SmartyGettext [http://sourceforge.net/projects/smarty-gettext/] (see Section 17.2, "Internationalization")
- Client plugins .ini files (for instance map sizes)
- Server plugins .ini files (for instance scales labels)
- Mapfile's .ini and .map (layers labels)
- Client and server PHP code (see Section 5.1, "Translations")

To generate PO templates, you will need to launch scripts on server and on client. Templates are generated in directory `<cartoweb_home>/projects/ <project_name>/po`. If translation files (see Section 16.1.3, "Translating") already exist, a merge is done using **msgmerge** [http://www.gnu.org/software/gettext/manual/html_chapter/gettext_6.html#SEC37] command. Follow these steps:

- generate project and mapfile templates on server:

```
cd <cartoweb_home>/scripts
php server2pot.php
```

For each mapfile, two templates will be generated: `server.po` and

```
server.<mapfile_name>.po
```

• generate project template on client:

```
cd <cartoweb_home>/scripts
php client2pot.php
```

For each project, one template will be generated: `client.po`

It is possible to specify for which project you want to generate the .po file. Simply add the project name as a parameter when launching the scripts client2pot and server2pot:

```
php server2pot.php myprojectname
```

If you do not specify a project name, the .po files will be generated for all projects.

16.1.3. Translating

As for any gettext system, translating PO files can be done in Emacs, in Poedit [http://poedit.sourceforge.net/] or in any text editor.

Translated PO files must be saved under name `<template_name>.<lang>.po` ; where <lang> is the 2-letters ISO language: en, fr, de, etc.. For instance, the mapfile test of test_main project will have three PO files in its PO directory for a complete french translation:

• `server.fr.po`

• `server.test.fr.po`

• `client.fr.po`

16.1.4. Compiling PO to MO

To compile all PO files to MO files (gettext's binary format), use the following commands on client side. This should be done each time configuration (client or server) is updated, and after each system update. All languages are compiled at the same time.

```
cd <cartoweb_home>/scripts
php po2mo.php
```

Warning: When CartoWeb is installed in SOAP mode, the script uses PHP curl functions to retrieve PO files from server to client. PHP curl module must be installed.

16.1.5. Example

To translate texts in french for project test_project and map file projectmap, follow

these steps:

- On server:

```
cd <cartoweb_home>/scripts
php server2pot.php
```

Copy `<cartoweb_home>/projects/test_project/po/server.po` to `<cartoweb_home>/projects/test_project/po/server.fr.po` and `<cartoweb_home>/projects/test_project/po/server.projectmap.po` to `<cartoweb_home>/projects/test_project/po/server.projectmap.fr.po`. Edit french files with Poedit (or any editor).

- On client:

```
cd <cartoweb_home>/scripts
php client2pot.php
```

Copy `<cartoweb_home>/projects/test_project/po/server.po` to `<cartoweb_home>/projects/test_project/po/server.fr.po`. Edit french file with Poedit (or any editor).

Merge and compile files with the following commands:

```
cd <cartoweb_home>/scripts
php po2mo.php
```

Now you should have the file `test_project.projectmap.mo` in directory `<cartoweb_home>/locale/fr/LC_MESSAGES`. The directory `fr/LC_MESSAGES` will be created if it does not exist.

16.1.6. Debuging translation problems

There can be several reasons why the translation is not working.

First you need to check if gettext is correctly installed and configured. For so run the following scripts:

-
```
cd <cartoweb_home>/scripts
./testgettext.sh
```

This will test if Gettext is correctly installed in your system

Note

Unix-like system only, this script is not intended for Windows users

- ```
  cd <cartoweb_home>/scripts
  php testgettext.php
  ```

  This will test if Gettext works correctly with php

Once you have verified Gettext is installed and run correctly, check if there are no errors in your *po* files.

- Verify you do no have duplicate *msgid* in your po files. Each *msgid* must be unique in a po file.

  A typical error message would be:

  ```
  Error message: Failure while launching "msgcat --to-code=iso-8859-1 --use-first --output=/www/cartowe
  /www/cartoweb3/projects/demo/po/demo.en.po /www/cartoweb3/projects/demo/po/client.en.po"
  (output is /www/cartoweb3/projects/demo/po/client.en.po:392: duplicate message definition
  /www/cartoweb3/projects/demo/po/client.en.po:390: ...this is the location of the first definition
  msgcat: found 1 fatal error)
  ```

  duplicate message definition on line 392 in *client.en.po*

- Check if you have several *fuzzy* entries in your **merged** po file (the file that is automatically created by gettext, merging the client and server po files). This happens when you have used the same *msgid* in different po files.

Note

Each po file contain a single *fuzzy* entry at the beginning, this should not be removed

```
...
#, fuzzy
msgid ""
msgstr ""
"POT-Creation-Date: 2005-09-13 09:36+0200\n"
...
```

- Gettext automatically comments *msgid* and *msgstr* that are not found anymore in your templates. Commented lines start with **#**. Sometimes this is unwelcome, simply uncomment the lines.
- If your CartoWeb profile is set to *production* (Section 4.1, "Common client.ini and server.ini Options"), do not forget to clean the temporary templates files by typing:

  ```
  php cw3setup-php --clean
  ```

  See Section 1.1.3.2.4, "Cleaning Generated Files (Map, PDF, Temporary Files and Smarty Cache)" for more details.

Note

It can also be usefull sometimes in *development* mode.

- If your are running PHP in a webserver as a module, the compiled .mo files are cached. If no change happens after compiling new .mo files, it could be useful to restart the webserver.

16.2. Character Set Encoding Configuration

Character set configuration is needed when CartoWeb strings may include international characters with accents, or other special characters. Two types of encodings must be set:

- how files (map files, configuration files, etc.) are encoded on server and on client. To set this encoding, add the following line in `server.ini` and in `client.ini`:

```
EncoderClass.config = <encoder_class>
```

- how CartoWeb exports (including HTML output) must be encoded. To set this encoding, add the following line in `client.ini`:

```
EncoderClass.output = <encoder_class>
```

Where <encoder_class> is the class used for encoding. Currently, following encoder classes are implemented:

- `EncoderISO`: handles strings coded in ISO-8859-1
- `EncoderUTF`: handles strings coded in UTF-8

17. Templating

17.1. Introduction

Smarty Template Engine is a convenient way to write HTML templates for dynamic pages. It enables to delocalize a great amount of layout processing. Thus it is pretty easy to customize a CartoWeb application layout without affecting the application core.

CartoWeb templates are located in `templates/` root directory and in plugins `templates/` directories (see Section 2.3, "Plugins").

More info about Smarty templates can be found here: http://smarty.php.net. A comprehensive online documentation including a reference and examples is available in various languages here: http://smarty.php.net/docs.php.

17.2. Internationalization

It is possible - and recommended! - to use the SmartyGettext [http://smarty.incutio.com/?page=SmartyGettext] tool in order to translate template-embedded textual strings. Texts to be translated are identified using the *{t}* tag:

```
<p>{t}Default text, please translate me{/t}</p>
<p>{t name="John Doe"}Hello my name is %1{/t}</p>
<p>{t 1='one' 2='two' 3='three'}The 1st parameter is %1, the 2nd is %2 and the 3nd %3.{/t}</p>
```

It is also possible to use the "|tr" smarty modifier if you want to apply some Smarty transformations on a translated string:

```
{assign var="label" value="Themes"}
<div {if $label|tr|count_characters:true < 10}style="width:72px;"{/if}>{$label|tr}</div>
```

See also Chapter 16, *Internationalization*

17.3. Resources

Resources are identified using the *{r}* tag. *{r}* tags have a mandatory *type* attribute and an optional *plugin* one. First attribute is used to indicate the relative file location (files are grouped by types) in the file system whereas the second one tells what plugin uses the resource. Filename is placed between opening and closing tags.

For instance to get the `logo.png` file located in `htdocs/gfx/layout/`, type in your template:

```
<img src="{r type=gfx/layout}logo.png{/r}" alt="" />
```

To get the zoom-in icons from the `location` plugin, type:

```
<img src="{r type=gfx plugin=location}zoom_in.gif{/r}" alt="" />
```

Following list shows all CartoWeb resource types.

- Htdocs root directory
    - path: `/htdocs/css/toto.css`
    - Smarty: `{r type=css}toto.css{/r}`
    - generated URL: `css/toto.css`
- Coreplugins
    - path: `/coreplugins/layers/htdocs/css/toto.css`
    - Smarty: `{r type=css plugin=layers}toto.css{/r}`
    - generated URL: `layers/css/toto.css`
- Plugins
    - path: `/plugins/hello/htdocs/css/toto.css`
    - Smarty: `{r type=css plugin=hello}toto.css{/r}`
    - generated URL: `hello/css/toto.css`
- Projects
    - path: `/projects/myproject/htdocs/css/toto.css`
    - Smarty: `{r type=css}toto.css{/r}`
    - generated URL: `myproject/css/toto.css`
- Projects Coreplugins (override)
    - path: `/projects/myproject/coreplugins/layers/htdocs/css/toto.css`
    - Smarty: `{r type=css plugin=layers}toto.css{/r}`
    - generated URL: `myproject/layers/css/toto.css`
- Projects Plugins (override)
    - path: `/projects/myproject/plugins/hello/htdocs/css/toto.css`
    - Smarty: `{r type=css plugin=hello}toto.css{/r}`
    - generated URL: `myproject/hello/css/toto.css`
- Projects specific plugins
    - path: `/projects/myproject/plugins/myplugin/htdocs/css/toto.css`
    - Smarty: `{r type=css plugin=myplugin}toto.css{/r}`
    - generated URL: `myproject/myplugin/css/toto.css`

18. Accounting

CartoWeb provides a mechanism to store accounting informations. These informations can be used to generate statistics of the application usage.

18.1. Accounting Configuration

18.1.1. Introduction

Accounting configuration specifies if the accounting is enabled or not, and how to store the accounting information. The configuration is the same for the client or server. Notice however that if you are using CartoWeb in direct mode, the server accounting configuration is not taken into account. See Section 4.2, " client.ini " for more information about direct mode.

When you enable accounting, you also need to activate the accounting plugin in your configuration. This is done through the *loadPlugins* parameter of `client.ini` and *mapInfo.loadPlugins* of your `server_conf/<mapId>/<mapId>.ini` file.

18.1.2. client.ini or server.ini

Here are the options that can be set on the client or server:

- *accountingOn* (boolean: 'true'|'false'): sets whether accounting is enable or not.
- *accountingStorage* (string: 'file'|'db'|'db_direct'): controls how the accounting information is stored.

  file means that accounting information will be stored in files named `cartoweb3/www-data/accounting/<MAP_ID>/` `{client|server}_accounting.log` where *<MAP_ID>* is the mapId currently used.

  db means that accounting information is stored inside a database table. This table can be created using the following command:

  ```
  CREATE TABLE cw_accounting (date timestamp, info text);
  ```

  The database to be used can be configured using the *accountingDsn* parameter. See next item.

  db_direct the information is stored inside a database table. Unlike *db* the data are not inserted in an unique columns but are directly splited.

Warning

This mode can only be used if this client and the server are in direct mode. See Section 4.2, " client.ini " for more information about direct mode.

- *accountingBasePath* (string): Optional base path where to store the accounting log files. This is only used when *accountingStorage* is set to `file`.
- *accountingDsn* (string): DSN (Data Source Name), used to specify which database to use for storing accounting information. This is only used when *accountingStorage* is set to `db` or `db_direct`.

Note

Do not forget to enable the accounting plugin if you want to use this feature.

18.2. Managing Accounting Information

18.2.1. Accounting Log Files Administration

Accounting log file are used to store one line of accounting information each time a request is made on CartoWeb. The log files will grow with time, as more and more data are stored inside. This means that special care has to be taken by the CartoWeb administrator to avoid that the log file eats all the available free disk space.

It exists several tools which can be used to manage the log files and to avoid it takes too much place. For instance the Logrotate software can be used to rotate the log files regularly.

Warning

Activating accounting and not taking care of log file management can slow down or interrupt CartoWeb good working if the log file becomes too large.

19. Locate Plugin

19.1. Introduction

The locate plugin deals with autocompletion of input fields used to recenter on a feature on the map. It is based on AJAX requests.

When a few characters have been typed in the dedicated input field, an AJAX request is processed on the database defined by its DSN. The first results beginning with these characters are displayed in a drop down list.

When a result is clicked, the map is recentered on it.

The JavaScript part of this plugin is based on the script.aculo.us [http://script.aculo.us/] Autocompleter.

19.2. Activation

The plugin needs obviously to be activated using the `client_conf/client.ini` *loadPlugins* parameter. For instance:

```
loadPlugins = auth, exportPdf, views, locate
```

To enable the plugin, you also need to add required parts in the `cartoclient.tpl` template. In the <head> element:

```
{if $locate_active|default:''}<link rel="stylesheet" type="text/css" href="{r type=css plugin=locate}loca
    ...
{if $locate_active|default:''}<script type="text/javascript" src="{r type=js}prototype.js{/r}"></script>{
```

And in the <body> of the document (for example in the navigation folder):

```
    {if $locate_active|default:''}
        {$locate_form}
    {/if}
```

Tip

Warning: make sure that the recentering in enabled (*recenterActive = true* in `client_conf/location.ini`)! The hidden inputs *id_recenter_layer* and *id_recenter_ids* are needed so the recentering functionnality works correctly.

19.3. Configuration

- `locate.#.id` (# = 0, 1, 2, ...): id of the layer in layers.ini to recenter on.
- `locate.#.label` (# = 0, 1, 2, ...): label of the locate input; Note that if you use the same label as in the layers.ini translations will operate.
- `locate.#.sql` (# = 0, 1, 2, ...): SQL statement for the AJAX request to the database. It must return two fields:
    - an identifier : the value of the `id_attribute_string` field of the layer, in order to be able to recenter on the feature,
    - a label : a name for the feature to recenter on.

20. AJAX

20.1. Introduction

CartoWeb implements an AJAX layer, enabling asynchronous update of the HTML GUI.

20.1.1. Browser Compatibility

Browser compatibility:

- Mozilla 1.7+
- Firefox 1.0.7+
- Internet Explorer 6+
- Safari 1.3.2+

Known browser incompatibility:

- Opera
- Internet Explorer 5 for Mac
- Konqueror

20.2. Make Your Project AJAX Enabled

To enable your project with AJAX, it is recommended that you build your project based on demoCW3 or test_main projects (CartoWeb version 3.3.0 or higher). A project based on demoCW3 won't need any tuning. A project based on test_main will require the following:

- Enable AJAX in your client.ini (or client.ini.in):

If you have a project running since version 3.2.0 or before, and want to enable AJAX, you'll need to:

- Enable AJAX in your `client.ini` (or `client.ini.in`)
- Adapt your `cartoclient.tpl`, `mainmap.tpl` and all redefined plugin templates (if they are based on test_main or demoCW3 project templates from CartoWeb version <= 3.2.0).

20.2.1. Client.ini Configuration

Add the ajaxOn directive and set it to true in your `/project/[yourProjectName]/client_conf/client.ini` or `/project/[yourProjectName]/client_conf/client.ini.in`

```
ajaxOn = true
```

20.2.2. Templates Adaptation

Warning

Adapting your templates is a tricky bit. Unless you customized your templates thoroughly, we recommend that you start over again your templates customization using demoCW3 or test_main as a basis, as these projects templates are AJAX ready.

We recommend that you diff your cartoclient.tpl, mainmap.tpl and all redefined plugin templates with the upstream. This is the best way to be up to date, especially if you use the latest CVS version of CartoWeb.

Tip

Also, you can have a look at the templates patch: http://bugzilla.maptools.org/attachment.cgi?id=126&action=view

21. Geostatistics

21.1. Introduction

This chapter describes how to configure geostatistics functionnalities.

21.2. General Behavior

The Geostat plugin allows advanced cartographic representation based on attributes of geospatial data. Currently, there is only support for polygonal entities. There will be support for ponctual symbol very soon. Every vectorial formats supported by Mapserver may be use.

The client side of the plugin allow the specification of the mandatory and optional paramaters. These parameters will be passed to the server side which will create the map layer and return some informations about the dataset.

21.3. Server-side Configuration

21.3.1. Mapfile Configuration

On the server side, the first configuration step is made within the mapfile. The only requirement is a least on polygonal layer which expose at least one attribute. You may consider an attribute as exposed if you can use it as CLASSITEM for this layer. You must add one (and only one) class to this layer. It will be use as template for the generated classes. You may use this class to define border color, label or other default parameter. You should not add this layer to the `layers.ini` configuration file.

21.3.2. geostat.ini Configuration

Choropleth layers are managed differently then ordinary layer within cartoweb. `server_conf/mapId/geostat.ini` is used to define layers that may be used for geostatistics. Take a look to a sample of this file.

```
geostat.0.mslayer = region_geostat_fill
geostat.0.label = Region
geostat.0.choropleth = true
geostat.0.choropleth_attribs = POPULATION,HOUSINCOME
geostat.0.choropleth_attribs_label = "Population","Average house income"
```

These example represents a single layer definition. The general for of parameter is geostat.*n.key* = *value*. *n* is an incremental number (first layer of the config file is 0.

key is a parameter for this layer. *value* is the value associated to the following parameter. The following parameters defined choropleth generation :

- *msLayer* : name of the layer in the mapfile
- *label* : Label for the layer in the interface
- *choropleth* : Should choropleth be activated. This is present for future development
- *choropleth_attribs* : Comma separated list of attributes that may be used for choropleth
- *choropleth_attribs_label* : Comma separated list of labels to display as description of the preceding attributes

21.4. Client-side Configuration

Client-side configuration allow to specify which algorithms should be display within the user interface. Currently, there are 2 algoriths families that may be configured. The first is about classification algorithms. The second is about generation of color ramp. The signification of the parameter is pretty straightforward.

The *choroplethDataFromCurrentBoundingBox* parameter allows to choose whether classes should change when users zoom or if they should remain constant for every scales.

```
;The following classification method are currently supported
;   Custom : 0
;   Equal intervals : 1
;   Quantils : 2
;   Modes : 3
choroplethClassifMethodDefault = 2
choroplethClassifMethodsList = 0,2,3
;The following color ramp generation method are currently supported
;   Custom : 0
;   RGB interpolation : 1
;   HSV interpolation : 2
;   Max diffrence : 3
choroplethColorRampMethodDefault = 1
choroplethColorRampMethodList = 1,3
choroplethDataFromCurrentBoundingBox = false
```

22. ToolTips

22.1. Introduction

The tooltips plugin aim is to show features attributive information on top of the map without refreshing the whole page.

22.2. Basic Usage

22.2.1. Introduction

When user moves the mouse over the map and stops during several milliseconds, an *AJAX* request is sent to the CartoClient using the geographical coordinates of the cursor.

If the webserver finds features corresponding to the defined *timeout_async* layers, a response is sent back to the browser with some generated HTML code showing some attributive information.

22.2.2. Pros / Cons

Pros

*   fully customizable layout,
*   several results can be displayed even for stacked features,
*   isn't dependant to the feature counts, performances should not fail even for huge quantity of feature

Cons

*   database (postGIS) required

22.2.3. Configuration

Here are defined parameters available for *timeout_async layers*

*   `label` : (optional) label for the layer. If not set, layerId is diplayed
*   `dsn` : connection string to database (Data Source Name)
*   `dbTableName` : name of the table in database
*   `attributes` : comma separated list of attributes to be displayed

- *template* : (optional) custom template, must be in overriden `toolTips` plugin templates directory. If not set, generic `layerResult.tpl` is used. The template to use can also be set in `renderResult()` method in PHP code if `QueryableLayer` is extended (See Section 22.3.2, "Extending classes")

```
tooltips.region.label = "Régions"
tooltips.region.dsn = "pgsql://www-data:www-data@localhost:5432/france"
tooltips.region.dbTableName = region
tooltips.region.attributes = "nom, code"
tooltips.region.template = "layerResult_region.tpl"
```

22.3. Custom Tooltips

22.3.1. Templates

Tip

Make sure that the custom templates are in the `templates` folder in the `toolTips` plugin directory in your project.

22.3.1.1. Main Template

As for the other plugins, templates can be overriden in the projects. Then, user can define a new `layerResult.tpl` template for all *tooltips* layers.

22.3.1.2. Layer Specific Template

One can also define a specific template for each layer. It can be defined using the template parameter (See Section 22.2.3, "Configuration") or in `renderResult()` method in PHP code if `QueryableLayer` is extended (see above).

22.3.2. Extending classes

If a class extending `ByXyQueryableLayer` with a name like `LayerIdQueryableLayer` exists, it will be taken into account.

To do so, you should extend (replace) the `toolTips` plugin (See Section 2.3.3, "Extending a Plugin"). You can name it `ProjectToolTips` for example.

So create a `ClientProjectToolTips.php` file containing something like :

```
require_once 'CustomLayers.php';

/**
 * Client part of ClientToolTips plugin
 * @package Plugins
 */
class ClientProjectToolTips extends ClientToolTips {
```

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

Then create a new PHP file in the same directory named `CustomLayers.php`.

Important

The name of the class that extends `ByXyQueryableLayer` should match the layerId.

It should look like :

```
class DepQueryableLayer extends ByXyQueryableLayer {

    public function __construct() {
        parent::__construct();

        $this->addReturnedAttribute('nom_chf_l');
    }

    /**
     * Sets the type of ResultLayer returned by ResultLayer::queryLayer()
     * @see QueryableLayer::newLayerResult()
     */
    protected function newLayerResult() {
        return new DepLayerResult();
    }
}

class DepLayerResult extends LayerResult {

    /**
     * @see LayerResult::renderResult()
     */
    public function renderResult($smarty) {
        $smarty->assign('layerId', $this->getId());
        $smarty->assign('layerLabel', Encoder::encode($this->getLabel(), 'config'));
        $smarty->assign('depName', $this->getAttribute('nom_dept'));
        $smarty->assign('depCode', $this->getAttribute('code_dept'));
        $smarty->assign('depChefLieu', ucfirst(strtolower($this->getAttribute('nom_chf_l'))));
        return $smarty->fetch('layerResult_depLayer.tpl');

    }
}
```

This method allows people to build tooltips with results coming from several sources (joined tables for example).

Tip

Don't forget to load the plugin on client-side (`client_conf/client.ini`)

22.3.3. Styling

133

Here are some considerations on how to customize the tooltip appearance.

TODO

22.4. Incompatibilities

Please note that some features were removed from the tooltips plugin to enhance its stability. The *area_async* and the *area_direct* modes are not available anymore as they complicated the javascript code a lot and weren't used at all. And the *timeout_async* was replaced by *tooltips* in the configuration files.

23. Search

23.1. Introduction

The search plugin aim is to offer standards search features (like recentering, result offset, guided search, ajax features...) in a generic way. Take note that in his current state the plugin only offers a server framework. There is not yet any generic JavaScript framework.

Warning

The following examples only work with AJAX mode on !

23.2. Basic Usage

Introduction

23.2.1. General Considerations

The search plugin must be activated on both CartoClient and CartoServer side ie. the `loadPlugins` parameter must contain `search` in your project `client_conf/client.ini` and `server_conf/<mapId>/<mapId>.ini`.

Search use is mainly based on three files. `client_conf/search.ini`, `plugins/search/htdocs/js/Search.ajax.js` and `plugins/search/templates/search.tpl`.

The search plugin uses the PEAR DB abstraction class and should be able to handle all databases supported by PEAR DB.

23.3. Basic Sample

23.3.1. Basic Mechanism and Setup

The data set and the query specification are set in the `search.ini` file. The design and the fields of the search form is set in the `search.tpl` file. The search behavior mainly reside in the `Search.ajax.js`.

Let's imagine that we want to do a search with a recenter on data that are represented on a layer Addresses. The source of the data is a table in a PostGIS database.

The data are stored in a table "addresses" with the followings rows: num, street, zipcode, town, gid and the_geom. Primary key is gid.

First we have to set the configuration in the `client_conf/client.ini/search.ini` like this:

```
dsn = SEARCHDSN
encodingContext = myencoding_context
; Do not forget to add  EncoderClass.myencoding_context = EncoderISO (or any other context)
; in your client_conf/client.ini

config.Adresses.provider.type = fulltextTable
config.Adresses.provider.table = addresses
config.Adresses.provider.aliases = adr_num, num, adr_street, street, adr_zip, zipcode, adr_town, town
config.Adresses.provider.id = gid
config.Adresses.provider.columns = adr_num, adr_street, adr_zip, adr_town,
config.Adresses.provider.fulltextColumns = adr_num, adr_street, adr_zip, adr_town
config.Adresses.provider.sortColumn = adr_num
config.Adresses.provider.sortDirection = asc
config.Adresses.provider.sortPriorities = adr_num, adr_street, adr_zip, adr_town
config.Adresses.provider.labels =  adr_num, adr_street, adr_zip, adr_town
config.Adresses.formatter.type = smarty
config.Adresses.formatter.template = search_results
```

- The dsn correspond to your dsn dataBase connexion string. For more information please see the php PEAR corresponding page [http://pear.php.net/manual/en/package.database.db.intro-dsn.php]
- The encodingContext correspond to the context in your `client_conf/client.ini` see below code
- The config.XXXX is used for making a differentiation on tables.
- The config.Adresses.provider.type defines the type of query that should be done on the table. Value can be :
  - fulltextTable: That means the search is a textual search (ilike clause) that will be done on the fields listed in the config.Adresses.provider.fulltextColumns.
  - table: In this case a where clause should be set. (see Section 23.3.1.1, "Guided Search")
  - user-defined (see Section 23.3.1.4, "Writing a Formatter").
- The config.Adresses.provider.table specifies the name of the table to query.
- The config.Adresses.provider.aliases defines column names aliases. This could be useful if there are two tables with same column name, or if you want to migrate from one set of tables to another with different column names.
- The config.Adresses.provider.id designate the unique value used for recentering. For example, if you want to do a recenter by id (gid in our case) the id value should be set to gid. Do not forget the id-attribute-srting in the mapFile see Map

file references [http://mapserver.gis.umn.edu/docs/reference/mapfile/layer]

- The config.Adresses.provider.columns specifies the columns to be returned by the query. It contains the columns names separated by a coma.

- The config.Adresses.provider.fulltextColumns specifies the columns to be use in the ilike clause when the config.Adresses.provider.type is fulltextColumns. It contains the columns names separated by a coma.

Caution

This parameter should not be use if the type is Table.

- The config.Adresses.provider.sortColumn specifies the columns to by used for ordering the rows.

- The config.Adresses.provider.sortDirection give the ordre type. It value can be asc or desc for ascendent or descendent.

- The config.Adresses.formatter.type specifies the method for presenting the result(s). Actually only smarty is implemented. But it is possible to write your own Formatter (see chapter Writing a Formatter).

- The config.Adresses.formatter.template specifies the name of the template for formatting query results. The file must be in the template folder of the plugin search in your project directory.

- The config.Adresses.provider.labels specifies the label of the columns that should be shown (the label will be used in the translation files).

Now we must set up the form that will be used for doing the search. Edit or create the `plugins/search/templates/search.tpl` file and write:

```
<div id="search_div">
    <fieldset><legend>{t}My Title{/t}</legend> <br/>
        <table width="100%">
            <tr>
                <td>
                    {t}Street: {/t}
                </td>
                <td>
                    <input type="text" id="search_adr_street"
                       name="search_adr_street" size="13"/>
                </td>
            </tr>
            <tr>
                <td>
                    {t}adr_town: {/t}
                </td>
                <td>
                    <input type="text" id="search_adr_town"
                       name="search_adr_town" size="13"/>
                </td>
            </tr>
```

```
        ...

      </table>
      <p>
        <input type="submit" value="{t}Search{/t}" class="your_form_button_class"
         onclick="JavaScript: CartoWeb.trigger('Search.DoIt'); return false;"/>

        <input type="hidden" id="search_config" name="search_config" value="Adresses" />
        <input type="hidden" id="search_sort_column" name="search_sort_column" value="adr_num"/>
        <input type="hidden" id="search_sort_direction" name="search_sort_direction" value="asc" />
        <input type="hidden" id="search_number" name="search_number" />


        <div id="search_results_div"></div>
      </p>
    </fieldset>
</div>
```

Each field present in the fulltextColumns parameter of the configuration file must
have a corresponding input in the template. Each input should have an id composed
of the name of the column prefixed with the key-word "search_". For example, the
the column adr_num should give:

```
<input type="text" id="search_adr_num" name="search_adr_num" size="13"/>
```

The template contains a submit button that calls the CartoWeb.trigger('Search.DoIt')
JavaScript method. This method defined in the `Search.ajax.js` (see below)
launches the action.

You may notice the hidden inputs in the template. They provide parameters that can
be dynamically changed by JavaScript:

- The search_config input determines the table to be queried. Its value should
  correspond to the name of a table defines in the `search.ini`
  (config.here_is_the_value). This value allows to set different table in the
  configuration file and to dynamically choose which table should be queried !!!!
  This input is not optional and the search plugin will generate an error !!!!
- The search_sort_column input defines the column to be used by the sort clause. If
  this input is not set or has no value, it's overridden by the sortColumn parameter
  defined in `search.ini`. If this parameter is also not set in the configuration file it
  will be ignored.
- The search_sort_direction input defines the sort order. It should be asc or desc. If
  this input is not set or has no value, it's overridden by the sortColumnDirection
  parameter defined in `search.ini`. If this parameter is also not set in the
  configuration file it will be ignored.

138

- The search_number input define the maximum number of rows to be returned by the query. If this input is not set or has no value, it will be ignored.

There is also two other inputs parameters available:

- offset
- page

They allow the creation of a navigation of the query result. The offset input gives the number of rows a "page" of result should contains. And page gives the current page number.

Note: The form is injected in the cartoclient.tpl inside the Smarty tag search:

```
{if $search_active|default:''}
    {$search}
{/if}
```

Warning

Do not forget to include the Search.ajax.js file in your cartoclient.tpl.

```
{if $search_active|default:''}<script type="text/javascript"
                                     src="{r type=js plugin=search}Search.ajax.js{/r}"></script>{/
```

Now we must set up the result template file. Create or edit the plugins/search/templates/search_results.tpl that has been defined in the configuration file and write:

```
{if $table->numRows > 0}
<table class="yourCssClass">
    <tr>
      <th>{t}Id{/t}</th>
      {foreach from=$table->columnIds item=column}
      <th> <a href="JavaScript: order('{$column}');">{t}{$column}{/t}</a></th>
      {/foreach}
    </tr>
    {foreach from=$table->rows item=row}
    <tr>
      <td>{$row->rowId}</td>
      {foreach from=$row->cells item=value}
      <td><a href="JavaScript: recenter('{$row->rowId}');">{$value}</a></td>
      {/foreach}
    </tr>
    {/foreach}
</table>
{else}
{t}No results{/t}
```

```
{/if}
```

This will return a table with the columns defined in the configuration file. By clicking on a row you will call the recenter method. By clicking on a row header you will order the result.

Note that the place of the result table is defined by a div that will be modified by the JavaScript.

```
<div id="search_results_div"></div>
```

The div id must be the same as the target id in function "handleResponse" ! (see below)

Finally we have to set up the java script file. Open or create the `plugins/search/htdocs/js/Search.ajax.js` and write:

```
AjaxPlugins.Search = {

  handleResponse: function(pluginOutput) {
    //should be the same value as the input search_config
    if (pluginOutput.htmlCode.myconfig)
      $('search_results_div').innerHTML = pluginOutput.htmlCode.myconfig;
  }
};


/*
* Search plugin's Actions
*/

AjaxPlugins.Search.Actions = {};

AjaxPlugins.Search.Actions.DoIt = {

  buildPostRequest: function(argObject) {
    return AjaxHandler.buildPostRequest();
  }
};

function order(column) {

  if (column != $('search_sort_column').value) {
    $('search_sort_column').value = column;
    $('search_sort_direction').value = 'asc';
  } else {
    if ($('search_sort_direction').value == 'asc') {
    $('search_sort_direction').value = 'desc';
    } else {
    $('search_sort_direction').value = 'asc';
    }
  }

  CartoWeb.trigger('Search.DoIt')
}
```

```
    function recenter(id){
      if ($('search_config').value == 'Adresses') {
        //id to recenter, be sure there is a hidden input with that name and
        //id in your template! if not, add it
        $('id_recenter_ids').value = id;

        /* if the search database table name is the same as the related layer
        to recenter on, you can set it as below
        be sure there is a hidden input with that name and id in your template!
        if not, add it */
        $('id_recenter_layer').value = $('search_config').value;

        /* do not use the recenter_scale parameter if you recenter on ids,
        only use it with recenter on x,y.
        you can set the recenter scale in location.ini on server side */

        /* do not use the recenter_doit parameter if you recenter on ids,
        only use it with recenter on x,y. */

        CartoWeb.trigger('Location.Recenter');
      }
    }
```

23.3.1.1. Guided Search

Note

The Basic Usage chapter is a prerequisite to the read of this one

A often meet case is to have a incremental search. Each time a search parameter is selected the following parameter is completed.

Let's now take a concrete case: We keep the same database as in the above sample. Imagine we have a select field that is initialised with the towns names. When you choose a city, a select fields containing all the streets of the town appears. When you choose a street the final search is launched.

As mentioned above we have to set the `search.ini` file

```
dsn = SEARCHDSN
; Do not forget to add  EncoderClass.myencoding_context = EncoderISO
;(or any other context)
; in your client_conf/client.ini

config.Adresses_init_town.provider.type = table
config.Adresses_init_town.provider.table = addresses
config.Adresses_init_town.provider.aliases = adr_town, town
config.Adresses_init_town.provider.id = adr_town
config.Adresses_init_town.provider.columns = adr_town
config.Adresses_init_town.provider.sortColumn = adr_town
config.Adresses_init_town.provider.sortDirection = asc
config.Adresses_init_town.formatter.type = smarty
config.Adresses_init_town.formatter.template = init_adresse_town_select

config.Adresses_init_street.provider.type = table
config.Adresses_init_street.provider.table = addresses
```

```
config.Adresses_init_street.provider.aliases = adr_street, street, adr_town, town
config.Adresses_init_street.provider.id = adr_street
config.Adresses_init_street.provider.columns = adr_street
config.Adresses_init_street.provider.sortColumn = adr_street
config.Adresses_init_street.provider.sortDirection = asc
config.Adresses_init_street.provider.where = "adr_town like '@adr_town_init@'"
config.Adresses_init_street.formatter.type = smarty
config.Adresses_init_street.formatter.template = init_adresse_adr_street_select


config.Adresses.provider.type = table
config.Adresses.provider.table = addresses
config.Adresses.provider.aliases = adr_num, num, adr_street, street, adr_zip, zipcode, adr_town, town
config.Adresses.provider.id = gid
config.Adresses.provider.columns = adr_num, adr_street, adr_zip, adr_town
config.Adresses.provider.sortColumn = adr_num
config.Adresses.provider.sortDirection = asc
config.Adresses.provider.sortPriorities = adr_num, adr_street, adr_zip, adr_town
config.Adresses.formatter.type = smarty
config.Adresses.formatter.template = search_results
config.Adresses.provider.where = "adr_street like '@adr_street_init@' and adr_town like '@adr_town_in
```

As you can see, we have have declared 3 search configurations that are pointing on
the same database entity. The first two tables are used for initializing the select. They
are not of type FullTextTable as we do not want to use the default search
comportement. So the provider.label and provider.fulltextColumns are not used.
There is also a new configuration parameter that is :

```
config.Adresses.provider.where
```

This parameter allows to specifiy a where clause. The where clause is written in
traditional sql but the data corresponding to the value of the form have to be replaced
by @the_field_name@.

Warning

Do not forget to remove the "search_" prefix that is in the select field when defining
the @the_field_name@ parameter in the ini file.

For the first two tables, the template file have been modified in order not to give back
a result table but a select containing all the data required

```
config.Adresses_init_street.formatter.type = smarty
config.Adresses_init_street.formatter.template = init_adresse_adr_street_select
```

Here is the content of the `init_adresse_town_select.tpl` file :

```
{t}Town:{/t}<br />
<select name="search_adr_town_init" id="search_adr_town_init"
  onchange="javascript: initializeStreet()">
```

```
    <option value="">&nbsp </option>
        {foreach from=$table->rows item=row}
        <option value="{$row->rowId}">
            {foreach from=$row->cells item=value}
            {$value}
            {/foreach}
        </option>
        {/foreach}
</select>
```

Here is the content of the `init_adresse_adr_street_select.tpl` file :

```
{t}Street:{/t} <br />
<select name="search_adr_street_init"
  id="search_adr_street_init" onchange="search()" >
    <option value="">&nbsp </option>
        {foreach from=$table->rows item=row}
        <option value="{$row->rowId}">
            {foreach from=$row->cells item=value}
            {$value}
            {/foreach}
        </option>
        {/foreach}
</select>
```

Now we have to setup the `search.tpl` file.

```
<div id="search_div">
<fieldset><legend>{t}My Title{/t}</legend> <br/>
  here we inser the placeholder for the select(s)
  <br />
  <div id="town_select_div"> Here comes the list containing the towns </div>
  <br />
  <div id ="street_select_div"> Here comes the list containing the streets  </div>
  <br />
  <input type="hidden" id="search_config" name="search_config" value="" />
  <input type="hidden" id="search_sort_column" name="search_sort_column" value=""/>
  <input type="hidden"
    id="search_sort_direction" name="search_sort_direction" value="asc" />
  <input type="hidden" id="search_number" name="search_number" />
<div id="search_results_div"></div>
</fieldset>
</div>
```

Finally we have to set up the java script file. Open or create the `plugins/search/`
`htdocs/js/Search.ajax.js` file and write:

```
AjaxPlugins.Search = {

  handleResponse: function(pluginOutput) {
    //we inject the select input in the HTML
    if (pluginOutput.htmlCode.Adresses_init_town) {
      //should be the same value as the input search_config
      $('town_select_div').innerHTML = pluginOutput.htmlCode.Adresses_init_town;
```

```
    }

    if (pluginOutput.htmlCode.Adresses_init_street) {
      //should be the same value as the input search_config
      $('street_select_div').innerHTML = pluginOutput.htmlCode.Adresses_init_street;
    }

    if (pluginOutput.htmlCode.Adresses) {
      //should be the same value as the input search_config
      $('search_results_div').innerHTML = pluginOutput.htmlCode.Adresses;
    }
  }
};


/*
 * Search plugin's Actions
 */

AjaxPlugins.Search.Actions = {};

AjaxPlugins.Search.Actions.DoIt = {

  buildPostRequest: function(argObject) {
    return AjaxHandler.buildPostRequest();
  }
};

function order(column) {

  if (column != $('search_sort_column').value) {
    $('search_sort_column').value = column;
    $('search_sort_direction').value = 'asc';
  } else {
    if ($('search_sort_direction').value == 'asc') {
      $('search_sort_direction').value = 'desc';
    } else {
      $('search_sort_direction').value = 'asc';
    }
  }

  CartoWeb.trigger('Search.DoIt')
}

//we empty the old selects and reinsert the new ones instead
//Please do not forget to modify the search_config input value
function initializeTown() {
$('town_select_div').innerHTML = '';
    var myinput = $('search_config');
    myinput.value = 'Adresses_init_town';
    CartoWeb.trigger('Search.DoIt');
}

function initializeStreet() {
    $('street_select_div').innerHTML = '';
    var myinput = $('search_config');
    myinput.value = 'Adresses_init_street';
    CartoWeb.trigger('Search.DoIt');
}

function search() {
    var myinput = $('search_config');
    myinput.value = 'Adresses';
    CartoWeb.trigger('Search.DoIt');
}

function recenter(id){
    if ($('search_config').value == 'Adresses') {
```

```
        //id to recenter, be sure there is a hidden input
        //with that name and id in your template! if not, add it
        $('id_recenter_ids').value = id;

        /* if the search database table name is the same
        as the related layer to recenter on, you can set it as below
           be sure there is a hidden input with that name and id in your template! if not, add it */
        $('id_recenter_layer').value = $('search_config').value;

        /* do not use the recenter_scale parameter
        if you recenter on ids, only use it with recenter on x,y.
           you can set the recenter scale in location.ini on server side */

        /* do not use the recenter_doit parameter
        if you recenter on ids, only use it with recenter on x,y. */

        CartoWeb.trigger('Location.Recenter');
    }
}
Event.observe(window, 'load', initializeTown, true);
```

For each initialisation, we have to set the correct value inside the search_config input
and add a "onload" event in order to initialize the town select when the page is
loaded.

```
Event.observe(window, 'load', initializeTown, true);
```

23.3.1.2. Multi table Search

Sometimes, you may want to have a search that is done on multiple tables. It can be
done easily. You have just to use the mechanisms presented in the basic sample and
create in the search.ini a table that looks like this:

```
config.parcels.provider.type = table
config.parcels.provider.table = parcelles, addresses
config.parcels.provider.aliases = adr_num, num, adr_street, street, adr_zip, zipcode, adr_town, town
;//be sure not to have twice the same id column name in different tables (or use aliases)
config.parcels.provider.id = parcel_id
config.parcels.provider.columns = parcel_ref, adr_street
config.parcels.provider.fulltextColumns = parcel_ref, adr_num, adr_street, adr_zip, adr_town,
config.parcels.provider.sortColumn = parcel_ref, adr_street
config.parcels.provider.sortDirection = asc
config.parcels.formatter.type = smarty
config.parcels.formatter.template = search_results
config.parcels.provider.labels = parcel_ref, street
config.parcels.provider.where = "adr_street like '%@adr_street@%' AND addresses.parcel_ref = parcels.
```

In this sample we have a search that is done on a table adresses and a table parcels
where a parcel contains many adresses. The search return all the streets where the
name matches the name given in the form and their corresponding parcel reference.

23.3.1.3. Writing a Provider

Warning

Documentation below is still in draft state

A provider is an interface implementation that allows you to acess a type of data that is not supported by PEAR DB.

You have to modify the `Search.php`. In this file there is an abstract class that is named :

```
abstract class ResultProvider
```

You have to extend this class :

```
class MyResultProvider extends ResultProvider {
    public function getResult(SearchRequest $request) {
        $table = New Table();
        $table->tableId = 'search';
        $table->columnIds = array('col1');
        $table->noRowId = false;
        $table->rows = array('hello world');
        $table->numRows = $dbResult->numRows();
        // Generates pages information
        $table->rowsPage = 1;
        $table->totalPages = 1;
        $table->page = 1; //curent page number
        $result = new SearchResult();
        $result->table = $table;
        return $result;
    }
}
```

Warning

Do not forget to modify the provider in the ini file.

You have to implements the getResult fonction. It should return a Table() instance.

23.3.1.4. Writing a Formatter

Warning

Documentation below is still in draft state

A formatter allows you to format the result of the query. Actually it returns code generated by Smarty. But if you want to use an other template engine or a more specifique formatter (JSON for example or in our case a smarty with differents delimiters), you have to modify the `ClientSearch.php`

In this file there is an abstract calss :

```
abstract class ResponseFormatter
```

you have to extend this class

```
/**
 * Formats a response using a Smarty template with different separators
 * @see ResponseFormatter
 */
class MYSmartyResponseFormatter extends ResponseFormatter {

    /**
     * @see ResponseFormatter::getResponse()
     */
    public function getResponse(SearchResult $result) {

    $smarty = new Smarty_Plugin($this->plugin->getCartoclient(),
    $this->plugin);
    $smarty->right_delimiter = '[EOF';
    $smarty->left_delimiter  = 'EOF]';
    $smarty->assign('table', $result->table);
    return $smarty->fetch($this->template . '.tpl');
    }
}
```

Warning

Do not forget to modify the formatter in the ini file.

You have to implements the getResponse function. It should return a string.The $result variable correspond to the value return by the provider.

24. OGC Layer Loader

24.1. Introduction

The *ogcLayerLoader* plugin allows user to add OGC layers such as WMS and WFS through a call by URL.

For exemple, using a simple URL, you could call your cartoweb project and add an OGC layer at startup.

Once the layer added, the coreplugin Layer will be called to update user layers. Added layer could then be used by other plugin in the project (legend, layer reorder, ...). This plugin is used by wmsBrowserLight and catalog to add OGC layers to the map.

24.2. Plugin Activation

To activate the plugin, load from both CartoClient and CartoServer configuration files: `client_conf/client.ini` and `server_conf/"mapId"/"mapId".ini`. For instance:

```
loadPlugins = mapOverlay, ogcLayerLoader
```

24.3. Define where to insert OGC layers

To do so, in `server_conf/"mapId"/ogcLayerLoader.ini`, set the following parameters in the ini file of the plugin.

```
ogcInsertLayerAfter = raster ; raster is the name of the layer
```

24.4. Specify in which layergroup you want to insert OGC layers

To do so, in `client_conf/layers.ini`, set the following parameters in the ini file of the layers coreplugin.

```
userLayerGroup = root ; root is the name of the layerGroup in which user layers will be added
```

24.5. Define where to link a geonetwork catalog

To do so, in `client_conf/ogcLayerLoader.ini`, set the following parameters in the ini file of the plugin.

```
urlCatalog = http://my.linked.geonetwork.catalog ;
```

25. WMS Browser Light

25.1. Introduction

wmsBrowserLight plugin allows you to add WMS layers to your map using the ogcLayerLoader plugin. Main difference between the wmsBrowerLight and the wmsBroser plugin is the managment of the server list. The plugins manage WMS servers compatible with version 1.1.1 of the OGC WMS spec.. Using the wmsBrowerLight plugin you have three ways to add specify the server end point (url).

- Simply type in the text box the server you heard about like http://mywmsserverurl.iheard.about (Do not set the version or any other parameters).
- Define a list of WMS servers using the ini file of the plugin. The servers will be presented to the end user in a list box.
- Connect to a GeoNetwork node having metadata on WMS services and load server urls from this catalog.

25.2. Plugin Activation

To activate the plugin, load ogcLayerLoader from both CartoClient and CartoServer configuration files and wmsBrowserLight plugins from CartoClient configuration files: `client_conf/client.ini` and `server_conf/"mapId"/"mapId".ini`. For instance:

```
loadPlugins = mapOverlay, ogcLayerLoader, wmsBrowserLight
```

25.3. Define a list of WMS servers using the ini file

To do so, in `client_conf/wmsBrowserLight.ini`, set the following parameters in the ini file of the plugin.

```
catalogtype = ini

servers.0.label = Sandre / Ouvrage
servers.0.url = http://services.sandre.eaufrance.fr/geo/ouvrage

servers.1.label = Données administratives et routières en France (Geosignal)
servers.1.url = http://www.geosignal.org/cgi-bin/wmsmap
```

25.4. Connect to a GeoNetwork node having metadata on WMS services

In order to use this type of connection the name of the GeoNetwork node has to be known. Optionnaly, the plugin will first logged into the node if user and password are specified. To do so, in `client_conf/wmsBrowserLight.ini`, set the following parameters in the ini file of the plugin.

```
catalogtype = gn
gn = http://sandre.eaufrance.fr/geonetwork
; unset gnuser and gnpassword if no login has to be made
gnuser = cartoweb
gnpassword = cartoweb
gnlang = en
; gnQuery is how CartoWeb could get the list of services in the node
gnQuery = "category=Services"
```

For the time being, there is no way to query GeoNetwork for services in a consistent manner using the ISO 19115 information. It should be possible to get this information from the tag ScopeCd of the DataQuality section. But this is not a searchable criteria in version 2.0.2. On tip is to create on the GeoNetwork node a category set to "services" and then used this categrory to get the list of server to propose to the end user.

26. Routing Plugin

26.1. Introduction

The routing plugins deals with the pgRouting [http://script.aculo.us/] Postgresql extension allowing to perform, among several algorithms, computation of the shortest path between two nodes of a graph. This plugin provides a graphical user interface to query a Postgresql/PostGiS database with pgRouting functions installed on it.

Extended documentation on pgRouting functions can be found at the pgRouting website [http://script.aculo.us/].

The demo_plugins demonstration shows an example of the routing plugin.

This documentation details how to install pgRouting on a PostGIS-enabled database and how to load demo data to be used in the demo_plugins project.

26.2. Installation

Before installing the pgRouting extension, a sample PostGIS database have to be created, as explained in the installation manual: Chapter 1, *Installation*.

26.2.1. Quick Install

- **Install the PgRouting PostgreSQL module.**

  Note

  To do so,
  - ○ Download pgRouting from http://pgrouting.postlbs.org/ and follow install instruction. Mac users can find a version here: http://www.kyngchaos.com/ software/unixport/postgres or find some basic compilation instructions (in French), here: http://www.davidgis.fr/documentation/win32/html/apa.html

- **Execute the *routing.sql* file to install the functions in your database by typing:**

  ```
  $ psql -d demo_plugins -f routing.sql
  ```

- **If you have PostGIS installed, you should launch *routing_postgis.sql* :**

  ```
  $ psql -d demo_plugins -f routing_postgis.sql
  ```

- **Import Europe road geodata in PostGIS, create its graph structure and**

configure plugin routing database. To do so, simply execute the *demo_routing.sql* file, located in the `<CARTOWEB_HOME>/projects/demoPlugins/server_conf/sql` directory.

Note

These steps are detailed in the next section.

- **Edit the cartoweb3/projects/demoPlugins/demo.properties file.** and uncomment the line beginning with *;ROUTING_PLUGINS*
- **Execute cw3setup.php.** file, with the *--config-from-file* option as described in the previous chapter.

26.2.2. Detailled installation steps

The routing module is a set of functions that compute a shortest path from a set of edges and vertices. Some functions are provided for importing data from geometric tables, and for generating results as geometries.

Note

For more information on these functions, you can have a look at the pgRouting documentation: http://pgrouting.postlbs.org/wiki/pgRoutingDocs.

This section explains the main steps to integrate the routing fonctionnalities in a custom application. We describe the steps followed to install the routing demo. To make short, we used an Europe roads shapefile, imported it in PostGIS, generated the graph tables and configured files to suggest a search of the shortest path between two European towns.

Note

The following chapters describe steps necessary to build a graph structure from shapefiles. It is not necessary to execute these commands if the demo_routing.sql file (`<CARTOWEB_HOME>/projects/demoPlugins/server_conf/sql`) was loaded into the demo_plugins database. This file already installs all the routing structure.

26.2.2.1. Europe Roads Geodata Importation in PostGIS

```
$ shp2pgsql -I roadl.shp roads_europe > /tmp/roadl.sql
$ psql -d demo_plugins -f /tmp/roadl.sql
# Only lauch the following commmands if you did not import the town table already
$ shp2pgsql -I mispopp.shp town > /tmp/town.sql
$ psql -d demo_plugins -f /tmp/town.sql
```

26.2.2.2. Graph Importation

The first step is to add needed columns to the table roads_europe. To do so, you can type:

```
$ ALTER TABLE roads_europe ADD COLUMN source_id int;
$ ALTER TABLE roads_europe ADD COLUMN target_id int;
$ ALTER TABLE roads_europe ADD COLUMN edge_id int;
-- next line is to work around a pgRouting bug in update_cost_from_distance (fixed in latest CVS)
$ ALTER TABLE roads_europe RENAME id TO id_old;
```

You can then fill the columns *source_id* and *target_id* with the "assign_vertex_id" function.

```
$ SELECT assign_vertex_id('roads_europe', 1);
```

Here is the content of the roads_europe table:

```
$ SELECT gid, source_id, target_id, edge_id, AStext(the_geom) FROM roads_europe limit 3;

  gid  | source_id | target_id | edge_id |                    AStext
-------+-----------+-----------+---------+---------------------------------------------
 13115 |     11051 |     11099 |      14 | MULTILINESTRING((1062096.06 4861316.234,...))
 12869 |     10918 |     10916 |     267 | MULTILINESTRING((250681.597 4779596.532,...))
 12868 |     10918 |     10913 |     268 | MULTILINESTRING((250681.597 4779596.532,...))
(3 lignes)
```

But if the data quality is poor, you need to delete the duplicates edges (they have the same source-target pairs of vertices). For example, to check that you have duplicated edges, you can type:

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

If there is duplicated edges, to delete one of two rows, you can type:

```
$ CREATE TABLE doublons AS SELECT * FROM roads_europe WHERE gid  in
(SELECT gid FROM (SELECT DISTINCT on (source_id, target_id) source_id, gid
FROM roads_europe) AS doublon);
$ DELETE FROM roads_europe;
$ INSERT INTO roads_europe (SELECT * FROM doublons);
$ DROP TABLE doublons;
```

The following step is to create and fill the edges and vertices tables of the resulting graph. To do so, you can use "create_graph_tables" function.

```
$ SELECT create_graph_tables('roads_europe', 'int4');
```

```
SELECT * FROM roads_europe_edges LIMIT 3;
 id | source | target | cost | reverse_cost
----+--------+--------+------+--------------
  1 |      1 |      2 |      |
  2 |      3 |      3 |      |
  4 |      2 |      2 |      |
```

```
(3 rows)
```

We can see that it contains NULL values for the cost column. The function update_cost_from_distance can update the cost column with the distance of the lines contained in the geometry table, attached to each edge:

```
$ SELECT update_cost_from_distance('roads_europe');
```

The costs are now:

```
id | source | target |       cost        | reverse_cost
----+--------+--------+-------------------+--------------
  1 |      1 |      2 | 6857.46585793103  |
  2 |      3 |      4 | 37349.9592156392  |
  3 |      5 |      6 | 14040.5673116933  |
(3 lignes)
```

Then you need to add the column which will contain the town labels, which will be shown in the drop-down list for selecting the two points of a path.

The label information is contained in the `roads` table which was originally imported using the **shp2pgsql** utility. The following commands will create a temporary table, fill it with the town information, and then update the vertices table. The towns contained in the shapefile may not be on the exact same point that the intersections of the roads. Thus, all intersections which are in a distance less that 2000 meters are associated to the town label. This distance may be adjusted according to the dataset you are using.

```
$ CREATE TABLE roads_source_town AS
   SELECT DISTINCT t.txt, source_id, distance(PointN(r.the_geom, 1), t.the_geom) AS d
     FROM roads_europe r, town t, roads_europe_vertices
     WHERE t.txt != 'UNK' AND distance(PointN(r.the_geom, 1), t.the_geom) < 2000
      AND geom_id = source_id ORDER BY t.txt, d;
$ CREATE TABLE roads_source_town_uniq AS
   SELECT * FROM roads_source_town a WHERE a.txt IN
     (SELECT b.txt FROM roads_source_town b where a.txt = b.txt LIMIT 1)
     AND a.d IN (SELECT b.d FROM roads_source_town b WHERE a.txt = b.txt LIMIT 1);
$ ALTER TABLE  roads_europe_vertices ADD COLUMN txt character varying(50);
$ SELECT AddGeometryColumn('','roads_europe_vertices','the_geom','-1','POINT',2);
$ UPDATE roads_europe_vertices SET txt = (SELECT DISTINCT txt
   FROM roads_source_town_uniq
    WHERE roads_europe_vertices.geom_id = roads_source_town_uniq.source_id);
-- clean the temporary tables
$ DROP TABLE roads_source_town;
$ DROP TABLE roads_source_town_uniq;
```

The last step is to fill the geometry column of the vertices table:

```
$ CREATE TABLE roads_europe_vertices_geom AS
    SELECT v.id, v.geom_id, v.txt,startPoint(geometryn(r.the_geom, 1)) AS
     the_geom FROM roads_europe r LEFT JOIN roads_europe_vertices v ON v.geom_id = r.source_id;
$ INSERT INTO roads_europe_vertices_geom
    SELECT v.id, v.geom_id, v.txt, endPoint(geometryn(r.the_geom, 1)) AS the_geom
     FROM roads_europe r LEFT JOIN roads_europe_vertices v ON v.geom_id = r.target_id;
```

```
$ DELETE FROM roads_europe_vertices;
$ INSERT INTO roads_europe_vertices SELECT DISTINCT ON (id) * FROM roads_europe_vertices_geom;
$ DROP TABLE roads_europe_vertices_geom;
```

Now, all is set up correctly for using the shortest path function on these data. But to include the routing fonctionnalities in a custom project, we also must respect some rules dictated by the routing plugin.

26.2.2.3. Routing Plugin Database Configuration

The two things to do are to:

- create the routing results table. In this example the table is routing_results.

```
$ CREATE TABLE routing_results (
    results_id integer,
    "timestamp" bigint,
    gid integer
  );
$ SELECT AddGeometryColumn('','routing_results','the_geom','-1',
'MULTILINESTRING',2);
```

- create the 'routing_results_seq' sequence.

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

26.2.2.4. Mapfile Configuration

In the mapfile, you must include the routing layer, its connection to the database, a symbology for the route and a first route using a unique identifier. The data parameter will be overwritten by the routing plugin to draw the route chosen by the end-user. Example:

```
LAYER
  NAME "graph"
  TYPE LINE
  TRANSPARENCY 80
  CONNECTIONTYPE postgis
    CONNECTION "user=@DB_USER@ password=@DB_PASSWD@ host=@DB_HOST@ dbname=@DB_NAME@"
    DATA "the_geom from (SELECT the_geom from routing_results) as foo using unique
    gid using srid=-1"
  TEMPLATE "t"
  CLASS
    NAME "0"
    STYLE
      SYMBOL "circle"
      SIZE 10
      COLOR 90 27 191
```

```
      END
   END
END
```

26.2.2.5. General Configuration

For the demo, we suggest that you select your route by starting from a town until an other town. This is possible because for each object of a european-towns layer, we have identified the nearest object of the roads_europe_vertices table. That is why in the demoRouting configuration there is a client-side configuration. Normally, in the plugin routing, client-side only allows you to type an id of object, from which to start and an other to finish the route. No configuration is needed. So, if you use demoRouting extension, you must specify client-side, the:

- postgresRoutingVerticesTable: vertices table
- stepName: vertices table col containing informations you want to propose a choice on
- dsn: the connexion string to the database

Anyway, server-side, you must specify :

- the routing table (postgresRoutingTable option),
- the routing layer in the mapfile (postgresRoutingResultsLayer option),
- the results routing table (postgresRoutingResultsTable),
- the connexion string to the database (dsn option).

27. Bounding box history plugin

The bboxHistory plugin allow the user to navigate through the bbox he or she visits: just like the back and forward buttons of a web browser.

Note that the ajax mode must be enabled to use the plugin and it has no configuration file.

27.1. Plugin activation

To activate the plugin, add it to *loadPlugins* from CartoClient configuration file: `client_conf/client.ini`. For instance:

```
loadPlugins = bboxHistory
```

You also need to add the plugin's template into the project template `templates/cartoclient.tpl`. For instance:

```
...
{include file="toolbar.tpl" group=1 header=1}
{include file="toolbar.tpl" group=2}
{$bboxHistoryForm}
...
```

Part III. Developer Manual

As is implied by its name, this part of the documentation is aimed at those who need to customize or extend CartoWeb for their specific needs.

1. Calling Plugins

This chapter describes the structure of SOAP calls to CartoWeb server methods in order to obtain cartographic data.

Warning

Due to some CartoWeb evolutions, please note that some parts of the current chapter may need some updates!

Global WSDL code can be found in file `CARTOWEB_HOME/server/cartoserver.wsdl`. WSDL code specific to plugins are located in `PLUGIN_HOME/common/plugin_name.wsdl.inc`. Interesting parts from these files are copied in the following sections.

Complete WSDL code dynamically generated for a map ID is accessible at the URL `CARTOWEB_URL/cartoserver.wsdl.php?mapId=project_name.mapfile_name`.

SOAP method getMapInfo is used to retrieve server configuration information, such as available layers, initial state, etc.. It shouldn't be called each time a map is requested. A mechanism based on a timestamp is available to be sure configuration is up-to-date (see Section 1.2, "Call to getMapInfo").

SOAP method getMap is used each time a new map or related information are needed.

1.1. Standard Structures

1.1.1. Simple Types

These types are used in different other structures.

```
<complexType name="ArrayOfString">
  <complexContent>
    <restriction base="enc11:Array">
      <attribute ref="enc11:arrayType" wsdl:arrayType="xsd:string[]"/>
    </restriction>
  </complexContent>
</complexType>
```

• array - list of character strings

```
<complexType name="Dimension">
```

```
  <all>
    <element name="width" type="xsd:int"/>
    <element name="height" type="xsd:int"/>
  </all>
</complexType>
```

- width - width in pixels

- height - height in pixels

```
<complexType name="GeoDimension">
  <all>
    <element name="dimension" type="types:Dimension"/>
    <element name="bbox" type="types:Bbox"/>
  </all>
</complexType>
```

- dimension - dimensions in pixels
- bbox - bounding box (see Section 1.1.2, "Shapes" for a description of type Bbox)

1.1.2. Shapes

These types define a hierarchy of shapes. As heritage and polymorphism cannot be used, type Shape includes all attibutes of its children types.

```
<complexType name="Bbox">
  <all>
    <element name="minx" type="xsd:double"/>
    <element name="miny" type="xsd:double"/>
    <element name="maxx" type="xsd:double"/>
    <element name="maxy" type="xsd:double"/>
  </all>
</complexType>
```

- minx - minimum x coordinate

- miny - minimum y coordinate

- maxx - maximum x coordinate

- maxy - maximum y coordinate

```
<complexType name="Point">
  <all>
    <element name="x" type="xsd:double"/>
    <element name="y" type="xsd:double"/>
  </all>
</complexType>
```

- x - x coordinate

- y - y coordinate

```
<complexType name="ArrayOfPoint">
  <complexContent>
    <restriction base="enc11:Array">
      <attribute ref="enc11:arrayType" wsdl:arrayType="types:Point[]"/>
    </restriction>
  </complexContent>
</complexType>
```

- array - list of points

```
<complexType name="Shape">
  <all>
    <element name="className" type="xsd:string"/>
    <element name="x" type="xsd:double" minOccurs="0"/>
    <element name="y" type="xsd:double" minOccurs="0"/>
    <element name="minx" type="xsd:double" minOccurs="0"/>
    <element name="miny" type="xsd:double" minOccurs="0"/>
    <element name="maxx" type="xsd:double" minOccurs="0"/>
    <element name="maxy" type="xsd:double" minOccurs="0"/>
    <element name="points" type="types:ArrayOfPoint" minOccurs="0"/>
  </all>
</complexType>
```

- className - shape class name: "Point", "Bbox", "Rectangle", "Line" or "Polygon"
- x - x coordinate (Point)
- y - y coordinate (Point)
- minx - minimum x coordinate (Bbox or Rectangle)
- miny - minimum y coordinate (Bbox or Rectangle)
- maxx - maximum x coordinate (Bbox or Rectangle)
- maxy - maximum y coordinate (Bbox or Rectangle)
- points - list of points (Line or Polygon)

```
<complexType name="ArrayOfShape">
  <complexContent>
    <restriction base="enc11:Array">
      <attribute ref="enc11:arrayType" wsdl:arrayType="types:Shape[]"/>
    </restriction>
  </complexContent>
</complexType>
```

- array - list of shapes

1.1.3. Tables

These types define a table structure, used in particular in Query plugin (see

Section 1.3.6, "Query").

```
<complexType name="TableRow">
  <all>
    <element name="rowId" type="xsd:string"/>
    <element name="cells" type="types:ArrayOfString"/>
  </all>
</complexType>
```

- rowId - row ID
- cells - cell contents (see Section 1.1.1, "Simple Types" for a description of type ArrayOfString)

```
<complexType name="ArrayOfTableRow">
  <complexContent>
    <restriction base="enc11:Array">
      <attribute ref="enc11:arrayType"
                 wsdl:arrayType="types:TableRow[]"/>
    </restriction>
  </complexContent>
</complexType>
```

- array - list of rows

```
<complexType name="Table">
  <all>
    <element name="tableId" type="xsd:string"/>
    <element name="tableTitle" type="xsd:string"/>
    <element name="numRows" type="xsd:integer"/>
    <element name="totalRows" type="xsd:integer"/>
    <element name="offset" type="xsd:integer"/>
    <element name="columnIds" type="types:ArrayOfString"/>
    <element name="columnTitles" type="types:ArrayOfString"/>
    <element name="noRowId" type="xsd:boolean"/>
    <element name="rows" type="types:ArrayOfTableRow"/>
  </all>
</complexType>
```

- tableId - table ID
- tableTitle - table title
- numRows - number of rows in table
- totalRows - total number of rows in context (for future use)
- offset - current position in context rows (for future use)
- columnIds - column IDs (see Section 1.1.1, "Simple Types" for a description of type ArrayOfString)
- columnTitles - column titles (see Section 1.1.1, "Simple Types" for a description of type ArrayOfString)
- noRowId - if true, table rows contain no row IDs

- rows - list of rows

```
<complexType name="ArrayOfTable">
  <complexContent>
    <restriction base="enc11:Array">
      <attribute ref="enc11:arrayType" wsdl:arrayType="types:Table[]"/>
    </restriction>
  </complexContent>
</complexType>
```

- array - list of tables

```
<complexType name="TableGroup">
  <all>
    <element name="groupId" type="xsd:string"/>
    <element name="groupTitle" type="xsd:string"/>
    <element name="tables" type="types:ArrayOfTable"/>
  </all>
</complexType>
```

- groupId - ID of table group
- groupTitle - title of table group
- tables - list of tables

```
<complexType name="ArrayOfTableGroup">
  <complexContent>
    <restriction base="enc11:Array">
      <attribute ref="enc11:arrayType"
                 wsdl:arrayType="types:TableGroup[]"/>
    </restriction>
  </complexContent>
</complexType>
```

- array - list of table groups

```
<complexType name="TableFlags">
  <all>
    <element name="returnAttributes" type="xsd:boolean"/>
    <element name="returnTable" type="xsd:boolean"/>
  </all>
</complexType>
```

- returnAttributes - if true, will return attributes (row cells) in addition to row IDs
- returnTable - if false, won't return any table information. This can be useful for instance when hilighting an object on which no information is needed

1.2. Call to getMapInfo

This method returns server configuration, which includes layers, initial states and other plugin-specific configuration. Variables returned by this method are set in server configuration files described in Part II, "User Manual".

1.2.1. Global Server Configuration

This includes layers configuration and initial states.

```
<complexType name="LayerState">
  <all>
    <element name="id" type="xsd:string"/>
    <element name="hidden" type="xsd:boolean"/>
    <element name="frozen" type="xsd:boolean"/>
    <element name="selected" type="xsd:boolean"/>
    <element name="unfolded" type="xsd:boolean"/>
  </all>
</complexType>
```

- id - layer state ID
- hidden - if true, layer isn't displayed in tree and attribute selected cannot be modified
- frozen - if true, layer is displayed in tree but attribute selected cannot be modified
- selected - if true, layer is displayed as selected in tree
- unfolded - if true, layer tree is displayed unfolded (layer groups)

```
<complexType name="ArrayOfLayerState">
  <complexContent>
    <restriction base="enc11:Array">
      <attribute ref="enc11:arrayType"
                 wsdl:arrayType="types:LayerState[]"/>
    </restriction>
  </complexContent>
</complexType>
```

- array - list of layer states

```
<complexType name="InitialLocation">
  <all>
    <element name="bbox" type="types:Bbox"/>
  </all>
</complexType>
```

- bbox - initial bounding box (see Section 1.1.2, "Shapes" for a description of type Bbox)

```
<complexType name="InitialMapState">
```

```
  <all>
    <element name="id" type="xsd:string"/>
    <element name="location" type="types:InitialLocation"/>
    <element name="layers" type="types:ArrayOfLayerState"/>
  </all>
</complexType>
```

- id - initial state ID
- location - initial location
- layers - list of layer states

```
<complexType name="ArrayOfInitialMapState">
  <complexContent>
    <restriction base="enc11:Array">
      <attribute ref="enc11:arrayType"
                 wsdl:arrayType="types:InitialMapState[]"/>
    </restriction>
  </complexContent>
</complexType>
```

- array - list of initial states

```
<complexType name="MapInfo">
  <all>
    <element name="timestamp" type="xsd:integer"/>
    <element name="mapLabel" type="xsd:string"/>
    <element name="keymapGeoDimension" type="types:GeoDimension"/>
    <element name="initialMapStates"
            type="types:ArrayOfInitialMapState"/>
    ...elements specific to plugins...
  </all>
</complexType>
```

- timestamp - timestamp of last update. This timestamp is transferred each time method getMap is called, so client knows when configuration was modified (see also Section 1.3.1.1, "Global Request")
- mapLabel - name of map as defined in mapfile
- keymapGeoDimension - pixel and geographical dimension information for key map
- initialMapStates - list of initial states

1.2.2. Layers

This includes configuration specific to Layers plugin, ie. list of all available layers and their properties.

```
<complexType name="ArrayOfLayerId">
```

```
  <complexContent>
    <restriction base="enc11:Array">
      <attribute ref="enc11:arrayType" wsdl:arrayType="xsd:string[]"/>
    </restriction>
  </complexContent>
</complexType>
```

- array - list of layer IDs

```
<complexType name="ChildrenSwitch">
  <all>
    <element name="id" type="xsd:string"/>
    <element name="layers" type="types:ArrayOfLayerId" minOccurs="0"/>
  </all>
</complexType>
```

- id - switch ID
- layers - list of layers

```
<complexType name="ArrayOfChildrenSwitch">
  <complexContent>
    <restriction base="enc11:Array">
      <attribute ref="enc11:arrayType" wsdl:arrayType=ChildrenSwitch[]"/>
    </restriction>
  </complexContent>
</complexType>
```

- array - list of children switches

```
<complexType name="Layer">
  <all>
    <element name="className" type="xsd:string"/>
    <element name="id" type="xsd:string"/>
    <element name="label" type="xsd:string"/>
    <element name="children"
             type="types:ArrayOfChildrenSwitch" minOccurs="0"/>
    <element name="minScale" type="xsd:double"/>
    <element name="maxScale" type="xsd:double"/>
    <element name="icon" type="xsd:string"/>
    <element name="link" type="xsd:string"/>
    <element name="aggregate" type="xsd:boolean" minOccurs="0"/>
    <element name="rendering" type="xsd:string" minOccurs="0"/>
    <element name="metadata" type="types:ArrayOfString" minOccurs="0"/>
  </all>
</complexType>
```

- className - layer class name: "LayerGroup", "Layer" or "LayerClass"
- id - layer ID
- label - label to be displayed. This label is not yet translated using internationalization
- children - list of children

- minScale - minimum scale at which layer will be displayed
- maxScale - maximum scale at which layer will be displayed
- icon - filename of the static icon for the layer. Dynamic legends are described in Section 6.4, "Layers Legends"
- link - if set, layer name is clickable
- msLayer - MapServer layer id
- aggregate - if true, children are not displayed and cannot be selected individually
- rendering - how layer will be displayed: "tree", "block", "radio" or "dropdown". See Chapter 6, *Layers* for more details on this option
- metadata - list of meta data defined in server configuration file. Format of each string in list is "variable_name=value"

```
<complexType name="ArrayOfLayer">
  <complexContent>
    <restriction base="enc11:Array">
      <attribute ref="enc11:arrayType" wsdl:arrayType="types:Layer[]"/>
    </restriction>
  </complexContent>
</complexType>
```

- array - list of layers

```
<complexType name="SwitchInit">
  <all>
    <element name="id" type="xsd:string"/>
    <element name="label" type="xsd:string"/>
  </all>
</complexType>
```

- id - switch ID
- label - switch label

```
<complexType name="ArrayOfSwitchInit">
  <complexContent>
    <restriction base="enc11:Array">
      <attribute ref="enc11:arrayType" wsdl:arrayType="types:SwitchInit[]"/>
    </restriction>
  </complexContent>
</complexType>
```

- array - list of switches

```
<complexType name="LayersInit">
  <all>
    <element name="notAvailableIcon" type="xsd:string"/>
    <element name="notAvailablePlusIcon" type="xsd:string"/>
```

```
    <element name="notAvailableMinusIcon" type="xsd:string"/>
    <element name="layers" type="types:ArrayOfLayer"/>
    <element name="switches" type="types:ArrayOfSwitchInit" minOccurs="0"/>
  </all>
</complexType>
```

- notAvailableIcon - filename of icon for not available layer (current scale is above or below this layer maximum/minimum scale)
- notAvailablePlusIcon - filename of icon for not available layer (current scale is above this layer maximum scale)
- notAvailableMinusIcon - filename of icon for not available layer (current scale is below this layer minimum scale)
- layers - list of layers
- switches - list of switches

1.2.3. Location

This includes configuration specific to Location plugin, ie. fixed scales, scales limits and shortcuts.

```
<complexType name="LocationScale">
  <all>
    <element name="label" type="xsd:string"/>
    <element name="value" type="xsd:double"/>
  </all>
</complexType>
```

- label - scale caption
- value - scale value to be set when scale is selected

```
<complexType name="ArrayOfLocationScale">
  <complexContent>
    <restriction base="enc11:Array">
      <attribute ref="enc11:arrayType"
                 wsdl:arrayType="types:LocationScale[]"/>
    </restriction>
  </complexContent>
</complexType>
```

- array - list of scales

```
<complexType name="LocationShortcut">
  <all>
    <element name="label" type="xsd:string"/>
    <element name="bbox" type="types:Bbox"/>
  </all>
</complexType>
```

- label - shortcut caption
- bbox - bounding box to recenter on when shortcut is selected

```
<complexType name="ArrayOfLocationShortcut">
  <complexContent>
    <restriction base="enc11:Array">
      <attribute ref="enc11:arrayType"
                 wsdl:arrayType="types:LocationShortcut[]"/>
    </restriction>
  </complexContent>
</complexType>
```

- array - list of shortcuts

```
<complexType name="LocationInit">
  <all>
    <element name="className" type="xsd:string"/>
    <element name="scales" type="types:ArrayOfLocationScale"/>
    <element name="minScale" type="xsd:double"/>
    <element name="maxScale" type="xsd:double"/>
    <element name="shortcuts" type="types:ArrayOfLocationShortcut"/>
  </all>
</complexType>
```

- className - "LocationInit" or extended class name if project implements an extension
- scales - list of fixed scales
- minScale - global minimum scale
- maxScale - global maximum scale
- shortcuts - list of bounding box shortcuts

1.2.4. Layer Reorder

This includes configuration specific to LayerReorder plugin.

```
<complexType name="LayerReorderInit">
  <all>
    <element name="layers" type="types:ArrayOfLayerInit"/>
  </all>
</complexType>
```

- array - layers are ordered from top to bottom of the displayed stack, see below to LayerInit structure overview.

```
<complexType name="LayerInit">
```

```
  <all>
    <element name="layerId" type="xsd:string"/>
    <element name="layerLabel" type="xsd:string"/>
  </all>
</complexType>
```

- layerId - Layer Id
- layerLabel - label to be displayed. This label is not yet translated using internationalization

1.3. Call to getMap

For each plugin, SOAP XML format are described for both server calls (i.e. requests) and server results.

1.3.1. Global Structures

Below is a description of general requests and results which include plugin-specific ones.

1.3.1.1. Global Request

```
<complexType name="MapRequest">
  <all>
    <element name="mapId" type="xsd:string"/>
    ...elements specific to plugins...
  </all>
</complexType>
```

- mapId - map ID, ie. project name and mapfile name separated by a point

1.3.1.2. Global Result

```
<complexType name="Message">
  <all>
    <element name="channel" type="xsd:integer"/>
    <element name="message" type="xsd:string"/>
  </all>
</complexType>
```

- channel - type of message: 1 = end user, 2 = developer
- message - text of the message

```
<complexType name="ArrayOfMessage">
  <complexContent>
```

```
    <restriction base="enc11:Array">
      <attribute ref="enc11:arrayType" wsdl:arrayType="types:Message[]"/>
    </restriction>
  </complexContent>
</complexType>
```

- array - list of messages

```
<complexType name="MapResult">
  <all>
    <element name="timestamp" type="xsd:integer"/>
    <element name="serverMessages"
             type="types:ArrayOfMessage" minOccurs="0"/>
    ...elements specific to plugins...
  </all>
</complexType>
```

- timestamp - timestamp which identifies the server configuration. If this timestamp changes, it means server configuration changed and a call to method getMapInfo is required to get latest version (see Section 1.2, "Call to getMapInfo").
- serverMessages - list of messages returned by server

1.3.2. Images

The Images plugin generates MapServer images. The three types of images are main map, key map and scale bar. Basic parameters, such as image size, are defined in this request/result. More specific parameters, such as map location or content, are defined in other plugins.

1.3.2.1. Images Request

```
<complexType name="Image">
  <all>
    <element name="isDrawn" type="xsd:boolean"/>
    <element name="path" type="xsd:string"/>
    <element name="width" type="xsd:int"/>
    <element name="height" type="xsd:int"/>
  </all>
</complexType>
```

- isDrawn - true if the image should be generated (when used in a request) or if it was generated (when returned in a result)
- path - relative path of generated image. Not used in request
- width - image width
- height - image height

```
<complexType name="ImagesRequest">
  <all>
    <element name="className" type="xsd:string"/>
    <element name="mainmap" type="types:Image"/>
    <element name="keymap" type="types:Image"/>
    <element name="scalebar" type="types:Image"/>
  </all>
</complexType>
```

- className - "ImagesRequest" or extended class name if project implements an extension
- mainmap - main map image information
- keymap - key map image information
- scalebar - scale bar image information

1.3.2.2. Images Result

```
<complexType name="ImagesResult">
  <all>
    <element name="className" type="xsd:string"/>
    <element name="mainmap" type="types:Image"/>
    <element name="keymap" type="types:Image"/>
    <element name="scalebar" type="types:Image"/>
  </all>
</complexType>
```

- className - "ImagesResult" or extended class name if project implements an extension
- mainmap - main map image information (see Section 1.3.2.1, "Images Request" for a description of type Image)
- keymap - key map image information
- scalebar - scale bar image information

1.3.3. Layers

The Layers plugin handles layers selection. Its request object includes list of layers to be displayed on main map. This plugin has no specific result object.

1.3.3.1. Layers Request

```
<complexType name="LayersRequest">
  <all>
    <element name="className" type="xsd:string"/>
    <element name="layerIds" type="types:ArrayOfLayerId"/>
    <element name="resolution" type="xsd:int"/>
    <element name="switchId" type="xsd:string"/>
```

```
   </all>
</complexType>
```

- className - "LayersRequest" or extended class name if project implements an extension
- layerIds - list of layers to include in map generation (see Section 1.1.1, "Simple Types" for a description of type ArrayOfLayerId)
- resolution - MapServer resolution. Set this to null if you want to use default resolution
- switchId - current switch ID

1.3.3.2. Layers Result

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

- className - "LayersResult" or extended class name if project implements an extension

1.3.4. Layer Reorder

The LayerReorder plugin handles layers reorder selection. Its request object includes list of layerIds rightly ordered to be displayed on main map. This plugin has no specific result object.

1.3.4.1. Layer Reorder Request

```
<complexType name="LayersRequest">
  <all>
    <element name="className" type="xsd:string"/>
    <element name="layerIds" type="types:ArrayOfLayerId"/>
  </all>
</complexType>
```

- className - "LayerReorderRequest" or extended class name if project implements an extension
- layerIds - list of layers to include in map generation rightly ordered from top to bottom (see Section 1.1.1, "Simple Types" for a description of type ArrayOfLayerId).

1.3.5. Location

The Location plugin handles position and moves on the map. Its request process includes different position methods, such as recentering on a specific object or moves relative to previous position. It returns the new bounding box and scale.

1.3.5.1. Location Request

```
<simpleType name="LocationType">
  <restriction base="xsd:string">
    <enumeration value="bboxLocationRequest"/>
    <enumeration value="panLocationRequest"/>
    <enumeration value="zoomPointLocationRequest"/>
    <enumeration value="recenterLocationRequest"/>
  </restriction>
</simpleType>
```

- bboxLocationRequest - recenters on a bounding box
- panLocationRequest - moves horizontaly/verticaly (panning)
- zoomPointLocationRequest - recenters on a point, includes relative zoom and fixed scale
- recenterLocationRequest - recenters on mapfile IDs

```
<complexType name="LocationConstraint">
  <all>
    <element name="maxBbox" type="types:Bbox"/>
  </all>
</complexType>
```

- maxBbox - maximum bounding box. If given parameters lead to a larger bounding box, it will be cropped (see Section 1.1.2, "Shapes" for a description of type Bbox)

```
<complexType name="LocationRequest">
  <all>
    <element name="className" type="xsd:string"/>
    <element name="locationType"  type="types:LocationType"/>
    <element name="bboxLocationRequest"
             type="types:BboxLocationRequest" minOccurs="0"/>
    <element name="panLocationRequest"
             type="types:PanLocationRequest" minOccurs="0"/>
    <element name="zoomPointLocationRequest"
             type="types:ZoomPointLocationRequest" minOccurs="0"/>
    <element name="recenterLocationRequest"
             type="types:RecenterLocationRequest" minOccurs="0"/>
    <element name="locationConstraint"
             type="types:LocationConstraint" minOccurs="0"/>
  </all>
</complexType>
```

- className - "LocationRequest" or extended class name if project implements an extension

- locationType - type of location

- bboxLocationRequest - bounding box request parameters (see Section 1.3.5.1.1, "BBox Request")

- panLocationRequest - panning request parameters (see Section 1.3.5.1.2, "Pan Request")

- zoomPointLocationRequest - zoom, recenter on point parameters (see Section 1.3.5.1.3, "Zoom-Point Request")

- recenterLocationRequest - recenter on IDs parameters (see Section 1.3.5.1.4, "Recenter Request")

- locationConstraint - constraint to be respected when location request is executed

1.3.5.1.1. BBox Request

```
<complexType name="BboxLocationRequest">
  <all>
    <element name="bbox" type="types:Bbox"/>
  </all>
</complexType>
```

- bbox - bounding box to be recentered on (see Section 1.1.2, "Shapes" for a description of type Bbox)

1.3.5.1.2. Pan Request

```
<simpleType name="PanDirectionType">
  <restriction base="xsd:string">
    <enumeration value="VERTICAL_PAN_NORTH"/>
    <enumeration value="VERTICAL_PAN_NONE"/>
    <enumeration value="VERTICAL_PAN_SOUTH"/>
    <enumeration value="HORIZONTAL_PAN_WEST"/>
    <enumeration value="HORIZONTAL_PAN_NONE"/>
    <enumeration value="HORIZONTAL_PAN_EAST"/>
  </restriction>
</simpleType>
```

- VERTICAL_PAN_NORTH - panning north
- VERTICAL_PAN_NONE - no vertical panning
- VERTICAL_PAN_SOUTH - panning south
- HORIZONTAL_PAN_WEST - panning west
- HORIZONTAL_PAN_NONE - no horizontal panning

- HORIZONTAL_PAN_EAST - panning east

```
<complexType name="PanDirection">
  <all>
    <element name="verticalPan" type="types:PanDirectionType"/>
    <element name="horizontalPan" type="types:PanDirectionType"/>
  </all>
</complexType>
```

- verticalPan - type of vertical panning
- horizontalPan - type of horizontal panning

```
<complexType name="PanLocationRequest">
  <all>
    <element name="bbox" type="types:Bbox"/>
    <element name="panDirection" type="types:PanDirection"/>
  </all>
</complexType>
```

- bbox - current bounding box (see Section 1.1.2, "Shapes" for a description of type Bbox)
- panDirection - panning directions

1.3.5.1.3. Zoom-Point Request

```
<simpleType name="ZoomType">
  <restriction base="xsd:string">
    <enumeration value="ZOOM_DIRECTION_IN"/>
    <enumeration value="ZOOM_DIRECTION_NONE"/>
    <enumeration value="ZOOM_DIRECTION_OUT"/>
    <enumeration value="ZOOM_FACTOR"/>
    <enumeration value="ZOOM_SCALE"/>
  </restriction>
</simpleType>
```

- ZOOM_DIRECTION_IN - zoom in (default is x2)
- ZOOM_DIRECTION_NONE - no zoom, recenter on point only
- ZOOM_DIRECTION_OUT - zoom out (default is x0.5)
- ZOOM_FACTOR - zoom using a custom factor
- ZOOM_SCALE - zoom to a fixed scale

```
<complexType name="ZoomPointLocationRequest">
  <all>
    <element name="bbox" type="types:Bbox"/>
    <element name="point" type="types:Point"/>
    <element name="zoomType" type="types:ZoomType"/>
    <element name="zoomFactor" type="xsd:float" minOccurs="0"/>
```

```
    <element name="scale" type="xsd:integer" minOccurs="0"/>
  </all>
</complexType>
```

- bbox - bounding box (unused when zoom type = ZOOM_SCALE)

- point - point to recenter on

- zoomType - type of zoom

- zoomFactor - zoom factor (unused when zoom type != ZOOM_FACTOR)

- scale - fixed scale (unused when zoom type != ZOOM_SCALE)

1.3.5.1.4. Recenter Request

```
<complexType name="IdSelection">
  <all>
    <element name="layerId" type="xsd:string"/>
    <element name="idAttribute" type="xsd:string"/>
    <element name="idType" type="xsd:string"/>
    <element name="selectedIds" type="types:ArrayOfString"/>
  </all>
</complexType>
```

- layerId - ID of layer on which query will be executed

- idAttribute - name of ID attribute

- idType - type of ID attribute ("string" or "int")

- selectedIds - list of IDs

```
<complexType name="ArrayOfIdSelection">
  <complexContent>
    <restriction base="enc11:Array">
      <attribute ref="enc11:arrayType"
                 wsdl:arrayType="types:IdSelection[]"/>
    </restriction>
  </complexContent>
</complexType>
```

- array - list of ID selections

```
<complexType name="RecenterLocationRequest">
  <all>
    <element name="idSelections" type="types:ArrayOfIdSelection"/>
  </all>
</complexType>
```

- idSelections - list of ID selections

178

1.3.5.2. Location Result

```
<complexType name="LocationResult">
  <all>
    <element name="className" type="xsd:string"/>
    <element name="bbox" type="types:Bbox"/>
    <element name="scale" type="xsd:double"/>
  </all>
</complexType>
```

*   className - "LocationResult" or extended class name if project implements an extension
*   bbox - new bounding box (see Section 1.1.2, "Shapes" for a description of type Bbox)
*   scale - new scale

1.3.6. Query

The Query plugin allows to search objects, hilight them on map and return text results. Search can be executed from a rectangle selection and/or using a list of object IDs.

Query request object is not mandatory. For more information about Query plugin, see Chapter 9, *Queries*.

1.3.6.1. Query Request

```
<simpleType name="QuerySelectionPolicy">
  <restriction base="xsd:string">
    <enumeration value="POLICY_XOR"/>
    <enumeration value="POLICY_UNION"/>
    <enumeration value="POLICY_INTERSECTION"/>
  </restriction>
</simpleType>
```

*   POLICY_XOR - XOR selection: when selecting a group of objects, already selected ones are unselected and not yet selected ones are selected (default type)
*   POLICY_UNION - union selection: when selecting a group of objects, already selected ones are kept selected and not yet selected ones are selected
*   POLICY_INTERSECTION - intersection selection: when selecting a group of objects, only already selected ones are kept selected

```
<complexType name="QuerySelection">
  <all>
```

```
    <element name="layerId" type="xsd:string"/>
    <element name="idAttribute" type="xsd:string"/>
    <element name="idType" type="xsd:string"/>
    <element name="selectedIds" type="types:ArrayOfString"/>
    <element name="useInQuery" type="xsd:boolean"/>
    <element name="policy" type="types:QuerySelectionPolicy"/>
    <element name="maskMode" type="xsd:boolean"/>
    <element name="hilight" type="xsd:boolean"/>
    <element name="tableFlags" type="types:TableFlags"/>
  </all>
</complexType>
```

- layerId - layer ID on which query will be executed

- idAttribute - name of ID attribute

- idType - type of ID attribute ("string" or "int")

- selectedIds - list of IDs

- useInQuery - if true, will force query to use this layer

- policy - type of selection

- maskMode - if true, will apply a mask instead of a simple selection. This won't work when using MapServer's hilighting feature (see Chapter 9, *Queries*)

- hilight - if false, selection won't be hilighted on map

- tableFlags - table flags (see Section 1.1.3, "Tables" for a description of type TableFlags)

```
<complexType name="ArrayOfQuerySelection">
  <complexContent>
    <restriction base="enc11:Array">
      <attribute ref="enc11:arrayType"
                 wsdl:arrayType="types:QuerySelection[]"/>
    </restriction>
  </complexContent>
</complexType>
```

- array - list of query selections

```
<complexType name="QueryRequest">
  <all>
    <element name="className" type="xsd:string"/>
    <element name="shape" type="types:Shape"/>
    <element name="queryAllLayers" type="xsd:boolean"/>
    <element name="defaultMaskMode" type="xsd:boolean"/>
    <element name="defaultHilight" type="xsd:boolean"/>
    <element name="defaultTableFlags" type="types:TableFlags"/>
    <element name="querySelections" type="types:ArrayOfQuerySelection"/>
  </all>
</complexType>
```

- className - "QueryRequest" or extended class name if project implements an extension

- shape - type of query object dependant. It can be a bounding box, a polygon, a circle or a point (see Section 1.1.2, "Shapes" for a description of differents types)
- queryAllLayers - if true, will execute query on all selected layers, ie. layers sent through Layers request (see Section 1.3.3.1, "Layers Request"). Unused when bounding box is not specified
- defaultMaskMode - mask mode for new layers (returned by query and but not yet in array querySelections). Unused when queryAllLayers = false
- defaultHilight - highlight for new layers (returned by query and but not yet in array querySelections). Unused when queryAllLayers = false
- defaultTableFlags - table flags for new layers (returned by query and but not yet in array querySelections, see Section 1.1.3, "Tables" for a description of type TableFlags). Unused when queryAllLayers = false
- querySelections - list if query selections. It contains all objects that must be hilighted and can be used to maintain persistence of a selection

1.3.6.2. Query Result

```
<complexType name="QueryResult">
  <all>
    <element name="className" type="xsd:string"/>
    <element name="tableGroup" type="types:TableGroup"/>
  </all>
</complexType>
```

- className - "QueryResult" or extended class name if project implements an extension
- tableGroup - group of tables which contains query results (one table per layer)

1.3.7. Outline

The Outline plugin allows to draw shapes on the main map. Shapes can also be drawn as a mask, ie. as holes in a rectangle covering map. It returns total area covered by shapes.

Outline request is not mandatory. As Outline plugin is not a core plugin, it must be activated in order to use the following request/result objects.

1.3.7.1. Outline Request

```
<complexType name="Color">
  <all>
    <element name="r" type="xsd:int"/>
    <element name="g" type="xsd:int"/>
```

```
    <element name="b" type="xsd:int"/>
  </all>
</complexType>
```

- r - red
- g - green
- b - blue

```
<complexType name="ShapeStyle">
  <all>
    <element name="symbol" type="xsd:int"/>
    <element name="size" type="xsd:int"/>
    <element name="color" type="types:Color"/>
    <element name="outlineColor" type="types:Color"/>
    <element name="backgroundColor" type="types:Color"/>
    <element name="transparency" type="xsd:int"/>
  </all>
</complexType>
```

- symbol -index of Mapserver symbol
- size - size of shape
- color - color
- outlineColor - outline color
- backgroundColor - background color
- transparency - transparency

```
<complexType name="LabelStyle">
  <all>
    <element name="font" type="xsd:int"/>
    <element name="size" type="xsd:int"/>
    <element name="color" type="types:Color"/>
    <element name="outlineColor" type="types:Color"/>
    <element name="backgroundColor" type="types:Color"/>
  </all>
</complexType>
```

- font - index of Mapserver font
- size - size of font
- color - color
- outlineColor - outline color
- backgroundColor - background color

```
<complexType name="StyledShape">
  <all>
    <element name="shapeStyle" type="types:ShapeStyle"/>
    <element name="labelStyle" type="types:LabelStyle"/>
    <element name="shape" type="types:Shape"/>
```

```
    <element name="label" type="xsd:string"/>
  </all>
</complexType>
```

- shapeStyle - style of shape

- labelStyle - style of label

- shape - shape (can be a point, rectangle, line or polygon, see Section 1.1.2, "Shapes" for a description of type Shape)

- label - content of label

```
<complexType name="ArrayOfStyledShape">
  <complexContent>
    <restriction base="enc11:Array">
      <attribute ref="enc11:arrayType" wsdl:arrayType="types:StyledShape[]"/>
    </restriction>
  </complexContent>
</complexType>
```

- array - list of shapes with style

```
<complexType name="OutlineRequest">
  <all>
    <element name="className" type="xsd:string"/>
    <element name="shapes" type="types:ArrayOfStyledShape"/>
    <element name="maskMode" type="xsd:boolean"/>
  </all>
</complexType>
```

- className - "OutlineRequest" or extended class name if project implements an extension

- shapes - list of styled shapes

- maskMode - if true, will draw the complement of all shapes merged together

1.3.7.2. Outline Result

```
<complexType name="OutlineResult">
  <all>
    <element name="className" type="xsd:string"/>
    <element name="area" type="xsd:double"/>
  </all>
</complexType>
```

- className - "OutlineResult" or extended class name if project implements an extension

- area - total area for all shapes

1.4. Examples

The following examples show simple SOAP calls using PHP.

To use these examples with your CartoWeb server, you'll have to modify the map ID and the layer names. Please note that in these examples, access to resources uses symbolic links (see Chapter 4, *Configuration Files*).

1.4.1. Retrieving Server Configuration

First thing to do is to declare the SOAP client. Class SoapClient will need the WSDL code dynamically generated by script cartoserver.wsdl.php.

```
<?php

$client = new SoapClient("http://url.to/server/cartoserver.wsdl.php"
                        . "?mapId=swiss_project.swiss");
...
```

Method only needs map ID as argument.

```
try{
    $result = $client->getMapInfo("swiss_project.swiss");
    print_r($result);
} catch (SoapFault $fault) {
    print $fault->faultstring;
}

?>
```

Result is shown below. It includes server configuration for the corresponding project and mapfile.

```
stdClass Object
(
  [timeStamp] => 1107043488
  [mapLabel] => Switzerland
  [keymapGeoDimension] => stdClass Object
    (
      [dimension] => stdClass Object
        (
          [width] => 100
          [height] => 100
        )
      [bbox] => stdClass Object
        (
          [minx] => 485000
          [miny] => 65000
          [maxx] => 835000
          [maxy] => 298000
        )
    )
  [initialMapStates] => Array
    (
      [0] => stdClass Object
        (
```

```
                [id] => default
                [location] => stdClass Object
                  (
                    [bbox] => stdClass Object
                      (
                        [minx] => 470000
                        [miny] => 50000
                        [maxx] => 860000
                        [maxy] => 320000
                      )
                  )
                [layers] => Array
                  (
                    [0] => stdClass Object
                      (
                        [id] => swiss_layer_1
                        [hidden] =>
                        [frozen] =>
                        [selected] => 1
                        [unfolded] =>
                      )
                    [1] => stdClass Object
                      (
                        [id] => swiss_layer_2
                        [hidden] => 1
                        [frozen] =>
                        [selected] => 1
                        [unfolded] =>
                      )
                  )
              )
          )
  )
[layersInit] => stdClass Object
(
[notAvailableIcon] => gfx/icons/swiss_project/swiss/na.png
[notAvailablePlusIcon] => gfx/icons/swiss_project/swiss/nap.png
[notAvailableMinusIcon] => gfx/icons/swiss_project/swiss/nam.png
[layers] => Array
  (
    [0] => stdClass Object
      (
        [className] => LayerGroup
        [id] => root
        [label] => root
        [children] => Array
          (
            [0] => swiss_layer_1
            [1] => swiss_layer_2
          )
        [minScale] => 0
        [maxScale] => 0
        [icon] =>
        [link] =>
        [aggregate] =>
        [rendering] =>
        [metadata] => Array
          (
            [0] => foo=bar
          )
      )
    [1] => stdClass Object
      (
        [className] => Layer
        [id] => swiss_layer_1
        [label] => Swiss Layer 1
        [children] => Array
          (
          )
```

```
                [minScale] => 0
                [maxScale] => 0
                [icon] => gfx/icons/swiss_project/swiss/icon_1.png
                [link] =>
                [metadata] => Array
                  (
                  )
            )
        [2] => stdClass Object
          (
            [className] => Layer
            [id] => swiss_layer_2
            [label] => Swiss Layer 2
            [children] => Array
              (
              )
            [minScale] => 1
            [maxScale] => 20
            [icon] =>
            [link] =>
            [metadata] => Array
              (
              )
          )
      )
  [locationInit] => stdClass Object
    (
        [className] => LocationInit
        [scales] => Array
          (
            [0] => stdClass Object
              (
                [label] => 1/50000
                [value] => 50000
              )
            [1] => stdClass Object
              (
                [label] => 1/100000
                [value] => 100000
              )
            [2] => stdClass Object
              (
                [label] => 1/500000
                [value] => 500000
              )
          )
        [minScale] => 25000
        [maxScale] => 1000000
        [shortcuts] => Array
          (
            [0] => stdClass Object
              (
                [label] => Romandie
                [bbox] => stdClass Object
                  (
                    [minx] => 475000
                    [miny] => 65750
                    [maxx] => 670000
                    [maxy] => 212000
                  )
              )
          )
    )
)
```

1.4.2. Getting a Map Using a Point and a Scale

The simpliest way to obtain a map from CartoWeb server is to send an X-Y location and a scale.

First thing to do is to declare the SOAP client. Class SoapClient will need the WSDL code dynamically generated by script cartoserver.wsdl.php.

```php
<?php

$client = new SoapClient("http://url.to/server/cartoserver.wsdl.php"
                        . "?mapId=swiss_project.swiss");
...
```

The map ID is required also in the request object.

```php
$request->mapId = 'swiss_project.swiss';
...
```

In this example, only main map and scale bar are requested. So key map's isDrawn attribute is set to false.

```php
$request->imagesRequest->className = 'ImagesRequest';

$request->imagesRequest->mainmap->isDrawn = true;
$request->imagesRequest->mainmap->path = '';
$request->imagesRequest->mainmap->width = 500;
$request->imagesRequest->mainmap->height = 500;
$request->imagesRequest->mainmap->format = '';

$request->imagesRequest->keymap->isDrawn = false;
$request->imagesRequest->keymap->path = '';
$request->imagesRequest->keymap->width = 0;
$request->imagesRequest->keymap->height = 0;
$request->imagesRequest->keymap->format = '';

$request->imagesRequest->scalebar->isDrawn = true;
$request->imagesRequest->scalebar->path = '';
$request->imagesRequest->scalebar->width = 100;
$request->imagesRequest->scalebar->height = 100;
$request->imagesRequest->scalebar->format = '';
...
```

Two layers are displayed. Resolution attribute is set to null to keep standard Mapserver resolution.

```php
$request->layersRequest->className = 'LayersRequest';

$request->layersRequest->layerIds = array('swiss_layer_1',
                                          'swiss_layer_2');
$request->layersRequest->resolution = null;
...
```

In this case, the location request object is of type zoom-point, and zoom type is set to ZOOM_SCALE. Bbox is unused but is required.

```php
$request->locationRequest->className = 'LocationRequest';

$request->locationRequest->locationType = 'zoomPointLocationRequest';
```

```
$request->locationRequest
        ->zoomPointLocationRequest->bbox->minx = 500000;
$request->locationRequest
        ->zoomPointLocationRequest->bbox->miny = 100000;
$request->locationRequest
        ->zoomPointLocationRequest->bbox->maxx = 600000;
$request->locationRequest
        ->zoomPointLocationRequest->bbox->maxy = 200000;
$request->locationRequest
        ->zoomPointLocationRequest->point->x = 550000;
$request->locationRequest
        ->zoomPointLocationRequest->point->y = 150000;
$request->locationRequest
        ->zoomPointLocationRequest->zoomType = 'ZOOM_SCALE';
$request->locationRequest
        ->zoomPointLocationRequest->scale = 200000;
...
```

Now request object is ready, SOAP method is called.

```
try{
    $result = $client->getMap($request);
    print_r($result);
} catch (SoapFault $fault) {
    print $fault->faultstring;
}

?>
```

Result is shown below. It includes relative paths to generated images and new bounding box computed from requested scale.

```
stdClass Object
(
  [timestamp] => 1107925732
  [serverMessages] => Array
    (
    )
  [imagesResult] => stdClass Object
    (
      [className] => ImagesResult
      [mainmap] => stdClass Object
        (
          [isDrawn] => 1
          [path] => images/110839565198671.jpg
          [width] => 500
          [height] => 500
          [format] =>
        )
      [keymap] => stdClass Object
        (
          [isDrawn] =>
          [path] =>
          [width] =>
          [height] =>
          [format] =>
        )
      [scalebar] => stdClass Object
        (
          [isDrawn] => 1
          [path] => images/110839565198672.png
          [width] => 300
          [height] => 31
          [format] =>
```

```
            )
        )
    [locationResult] => stdClass Object
        (
            [className] => LocationResult
            [bbox] => stdClass Object
                (
                    [minx] => 536770.840477
                    [miny] => 136770.840477
                    [maxx] => 563229.159523
                    [maxy] => 163229.159523
                )
            [scale] => 200000
        )
)
```

1.4.3. Executing a Query

The following code shows how to use queries to display hilighted selection and to
retrieve corresponding attributes.

First thing to do is to declare the SOAP client. Class SoapClient will need the WSDL
code dynamically generated by script cartoserver.wsdl.php.

```php
<?php

$client = new SoapClient("http://url.to/server/cartoserver.wsdl.php"
                          . "?mapId=swiss_project.swiss");
...
```

The map ID is required also in the request object.

```php
$request->mapId = 'swiss_project.swiss';
...
```

In this example, only main map and key map are requested. So scale bar's isDrawn
attribute is set to false.

```php
$request->imagesRequest->className = 'ImagesRequest';

$request->imagesRequest->mainmap->isDrawn = true;
$request->imagesRequest->mainmap->path = '';
$request->imagesRequest->mainmap->width = 500;
$request->imagesRequest->mainmap->height = 500;
$request->imagesRequest->mainmap->format = '';

$request->imagesRequest->keymap->isDrawn = true;
$request->imagesRequest->keymap->path = '';
$request->imagesRequest->keymap->width = 100;
$request->imagesRequest->keymap->height = 100;
$request->imagesRequest->keymap->format = '';

$request->imagesRequest->scalebar->isDrawn = false;
$request->imagesRequest->scalebar->path = '';
$request->imagesRequest->scalebar->width = 0;
$request->imagesRequest->scalebar->height = 0;
$request->imagesRequest->scalebar->format = '';
...
```

Two layers are displayed. Resolution attribute is set to null to keep standard Mapserver resolution.

```
$request->layersRequest->className = 'LayersRequest';

$request->layersRequest->layerIds = array('swiss_layer_1',
                                           'swiss_layer_2');
$request->layersRequest->resolution = null;
...
```

In this case, the location request object is of type bbox. Only new bounding box is required.

```
$request->locationRequest->className = 'LocationRequest';

$request->locationRequest->locationType = 'bboxLocationRequest';
$request->locationRequest->bboxLocationRequest->bbox->minx = 550000;
$request->locationRequest->bboxLocationRequest->bbox->miny = 100000;
$request->locationRequest->bboxLocationRequest->bbox->maxx = 600000;
$request->locationRequest->bboxLocationRequest->bbox->maxy = 150000;
...
```

The query will be performed on a rectangle, on all selected layers (ie. layers defined in layers request object). IDs and attributes will be returned.

```
$request->queryRequest->className = 'QueryRequest';

$request->queryRequest->shape->className = 'Bbox';
$request->queryRequest->shape->minx = 570000;
$request->queryRequest->shape->miny = 120000;
$request->queryRequest->shape->maxx = 580000;
$request->queryRequest->shape->maxy = 130000;
$request->queryRequest->queryAllLayers = true;
$request->queryRequest->defaultMaskMode = false;
$request->queryRequest->defaultTableFlags->returnAttributes = true;
$request->queryRequest->defaultTableFlags->returnTable = true;
$request->queryRequest->querySelections = array();
...
```

Now request object is ready, SOAP method is called.

```
try{
    $result = $client->getMap($request);
    print_r($result);
} catch (SoapFault $fault) {
    print $fault->faultstring;
}

?>
```

Result is shown below. It includes relative paths to generated images and new scale computed from requested bounding box.

Query results include one table per layer. No results were found in layer swiss_layer_1 and two results in swiss_layer_2. As requested, attributes (here attribute_3 and attribute_4) are returned for each row.

```
stdClass Object
(
  [timestamp] => 1107925732
  [serverMessages] => Array
    (
    )
  [imagesResult] => stdClass Object
    (
      [className] => ImagesResult
      [mainmap] => stdClass Object
        (
          [isDrawn] => 1
          [path] => images/110846607738541.jpg
          [width] => 500
          [height] => 500
          [format] =>
        )
      [keymap] => stdClass Object
        (
          [isDrawn] => 1
          [path] => images/110846607738542.png
          [width] => 150
          [height] => 99
          [format] =>
        )
      [scalebar] => stdClass Object
        (
          [isDrawn] =>
          [path] =>
          [width] =>
          [height] =>
          [format] =>
        )
    )
  [locationResult] => stdClass Object
    (
      [className] => LocationResult
      [bbox] => stdClass Object
        (
          [minx] => 550000
          [miny] => 100000
          [maxx] => 600000
          [maxy] => 150000
        )
      [scale] => 377952.96
    )
  [queryResult] => stdClass Object
    (
      [className] => QueryResult
      [tableGroup] => stdClass Object
        (
          [groupId] => query
          [groupTitle] => Query
          [tables] => Array
            (
              [0] => stdClass Object
                (
                  [tableId] => swiss_layer_1
                  [tableTitle] => swiss_layer_1
                  [numRows] => 0
                  [totalRows] => 0
                  [offset] => 0
                  [columnIds] => Array
                    (
                    )
                  [columnTitles] => Array
                    (
                    )
```

```
                    [noRowId] =>
                    [rows] => Array
                      (
                      )
                )
          [1] => stdClass Object
            (
                [tableId] => swiss_layer_2
                [tableTitle] => swiss_layer_2
                [numRows] => 2
                [totalRows] => 0
                [offset] => 0
                [columnIds] => Array
                  (
                      [0] => attribute_3
                      [1] => attribute_4
                  )
                [columnTitles] => Array
                  (
                      [0] => attribute_3
                      [1] => attribute_4
                  )
                [noRowId] =>
                [rows] => Array
                  (
                      [0] => stdClass Object
                        (
                            [rowId] => 123
                            [cells] => Array
                              (
                                  [0] => Foo
                                  [1] => 84.98
                              )
                        )
                      [1] => stdClass Object
                        (
                            [rowId] => 456
                            [cells] => Array
                              (
                                  [0] => Bar
                                  [1] => 32.47
                              )
                        )
                  )
            )
        )
    )
)
```

2. New Plugins

2.1. What are Plugins

2.1.1. Definition

CartoWeb plugins are modular packages of files (PHP classes, HTML templates, images and other resources) that are used to perform a dedicated action: main map formatting, layers browsing interface, map browsing (zooming, paning etc.), queries, user authentication, search interfaces and many more.

2.1.2. Plugins and Coreplugins

There are two kinds of plugins:

*   *coreplugins*: fundamental plugins that perform "low-level" actions such as map size handling, browsing tools, layers selection. Plugins that are frequently used in many CartoWeb applications may be included in this category as well. They are always available and activated. As a result, other plugins may interact with them. Coreplugins files are grouped in the coreplugins/directory.
*   *plugins*: "normal" plugins perform more specificactions and are not always activated. Normal plugins activation is done by setting the *loadPlugins* parameter in client_conf/client.ini for CartoClient plugins and in server_conf/<mapId>/<mapId>.ini for CartoServer ones. For instance:

    ```
    loadPlugins = auth, outline, exportHtml
    ```

    Since they are not always available, simple plugins usually do not rely on each other. On the other hand, it is not a problem for them to call some coreplugins functionalities if the latter are publicly accessible. Simple plugins files are grouped in the plugins/directory.

The general philosophy is to gather all files of a given plugin in the same dedicated directory, including files from both CartoClient and CartoServer sides of the plugin. Thus it is easy to "plug" a new module in CartoWeb architecture by simply pasting it in the plugins/or coreplugins/parent directories. Note however that plugins configuration files (named <pluginName>.ini) are placed in the client_conf/ and/or server_conf/<mapId>/depending if those plugins have CartoClient/CartoServer components.

2.1.3. Plugins Structure

Plugins and coreplugins have the following general structure:

```
<pluginName>/
<pluginName>/client/
<pluginName>/server/
<pluginName>/common/
<pluginName>/templates/
<pluginName>/htdocs/
<pluginName>/htdocs/gfx/
<pluginName>/htdocs/js/
<pluginName>/htdocs/css/
```

- `client/` contains all specific CartoClient-side PHP files.
- `server/` contains all specific CartoServer-side PHP files.
- `common/` contains PHP files shared by both CartoClient and CartoServer sides, or at least files that are not specific to one side or the other.
- `templates/` contains all the plugin-specific Smarty templates. Since HTML templates are only used in CartoClient, files from `templates/` are only called by `client/` code.
- `htdocs/` contains all files (PHP pages, images, JavaScript or CSS files, etc.) that may be web-accessed when running the plugin. Those files are dispatched in various directories depending on their nature. If necessary, you can create additional subdirectories. For instance `java/` if your plugin uses a Java applet. To preserve the plugin independence, it is strongly recommended not to add your CSS styles in the general CartoClient style sheet but to create a specific file here that will be called separately.

Note that it is not required to actually create the whole structure described above. Only directories that contain files are necessairy. For instance if a plugin only perform CartoServer actions, it is no use to create `client/`, `templates/` and `htdocs/` directories. `common/` may be usefull if not-CartoServer-specific classes have to be defined.

There are two ways to add a plugin/coreplugin to CartoWeb: writing a brand new one or overriding/extending an existing one.

2.2. Writing a Plugin

2.2.1. Introduction

If no existing plugin or coreplugin fulfils your requirements and if none offers close enough functionalities to justify an adaptation, you can write a new plugin.

Plugins main classes (client and/or server if any) must extend CartoWeb defined*ClientPlugin* and/or *ServerPlugin* classes which provide base plugin tools.

For instance:

```
class ClientYourPlugin extends ClientPlugin {

    /* here comes your plugin client class definition */

}
```

2.2.2. Plugin or Coreplugin?

First of all you have to determine if you are about to design a simple plugin or a coreplugin. To be a coreplugin, your plugin must be really generic and present a great interest to the CartoWeb users community since it might be included in the upstream distribution. Contact CartoWeb development team for more info. In most cases it is better and sufficient to create a simple plugin.

To activate a coreplugin, update the *Cartoclient::getCorePluginNames()* method in `/client/Cartoclient.php` and/or the *ServerContext::getCorePluginNames()* one in `/server/ServerContext.php`. For instance:

```
private function getCorePluginNames() {
return array('images', 'location', 'layers', 'query', 'mapquery',
             'tables', 'yourPluginName');
}
```

To load a regular plugin, update the *loadPlugins* parameter from `client_conf/client.ini` and/or `server_conf/<mapId>/<mapId>.ini` as in following example:

```
loadPlugins = auth, outline, exportHtml
```

2.2.3. How Plugins Are Called

As explained in Section 2.1, " What are Plugins", plugins are independent aggregations of PHP code that are called by the CartoWeb core classes to perform dedicated actions. Plugins are called several times during the program execution (entry points). Thus they can interact at various level of the application.

To determine what plugins must be called at what moment and to perform what action, plugins must implement one or more of the CartoWeb plugin interfaces (according to the object-oriented programing meaning). The interfaces define methods that will be triggered by the main program during its execution. For example, you can take a look at the following simplified *Cartoclient::doMain()* method ("main program") defined in `/client/Cartoclient.php`:

```
private function doMain() {
    $this->callPluginsImplementing('InitUser', 'handleInit',
                               $this->getMapInfo());

    if ($this->isRequestPost()) {
```

```
        $this->cartoForm =
            $this->httpRequestHandler->handleHttpRequest(
                                            $this->clientSession,
                                            $this->cartoForm);

        $request = new FilterRequestModifier($_REQUEST);
        $this->callPluginsImplementing('FilterProvider',
                                       'filterPostRequest', $request);
        $this->callPluginsImplementing('GuiProvider',
                                       'handleHttpPostRequest',
                                       $request->getRequest());
    } else {
        $request = new FilterRequestModifier($_REQUEST);
        $this->callPluginsImplementing('FilterProvider',
                                       'filterGetRequest', $request);
        $this->callPluginsImplementing('GuiProvider',
                                       'handleHttpGetRequest',
                                       $request->getRequest());
    }

    $mapRequest = $this->getMapRequest();
    $this->callPluginsImplementing('ServerCaller', 'buildRequest',
                                   $mapRequest);

    $this->mapResult = $this->getMapResultFromRequest($mapRequest);

    $this->callPluginsImplementing('ServerCaller', 'initializeResult',
                                   $this->mapResult);

    $this->callPluginsImplementing('ServerCaller', 'handleResult',
                                   $this->mapResult);

    $this->formRenderer->showForm($this);

    $this->callPluginsImplementing('Sessionable', 'saveSession');
    $this->saveSession($this->clientSession);
}
```

callPluginsImplementing($interfaceName, $methodName, $argument) is run at various points of the program and make plugins implementing given <interfaceName> interface execute given <methodName> with given <argument> argument.

Of course interface-defined methods must be implemented in the matching plugins. Plugins can implements one or more CartoWeb interfaces.

Implementing interfaces is not mandatory when writing a plugin but not doing so will keep plugins from being implicitely called by the main program. As a result, methods from plugins with no interface implementation - also called "service plugins" - must be explicitely called by another piece of code (generally an other plugin).

```
class ClientYourPlugin extends ClientPlugin
                    implements Sessionable, GuiProvider {

    /* here comes your plugin client class definition */

}
```

For a comprehensive list of available client and server interfaces, see /client/

`ClientPlugin.php` and `/server/ServerPlugin.php` files or take a look at the CartoWeb PHP API documentation.
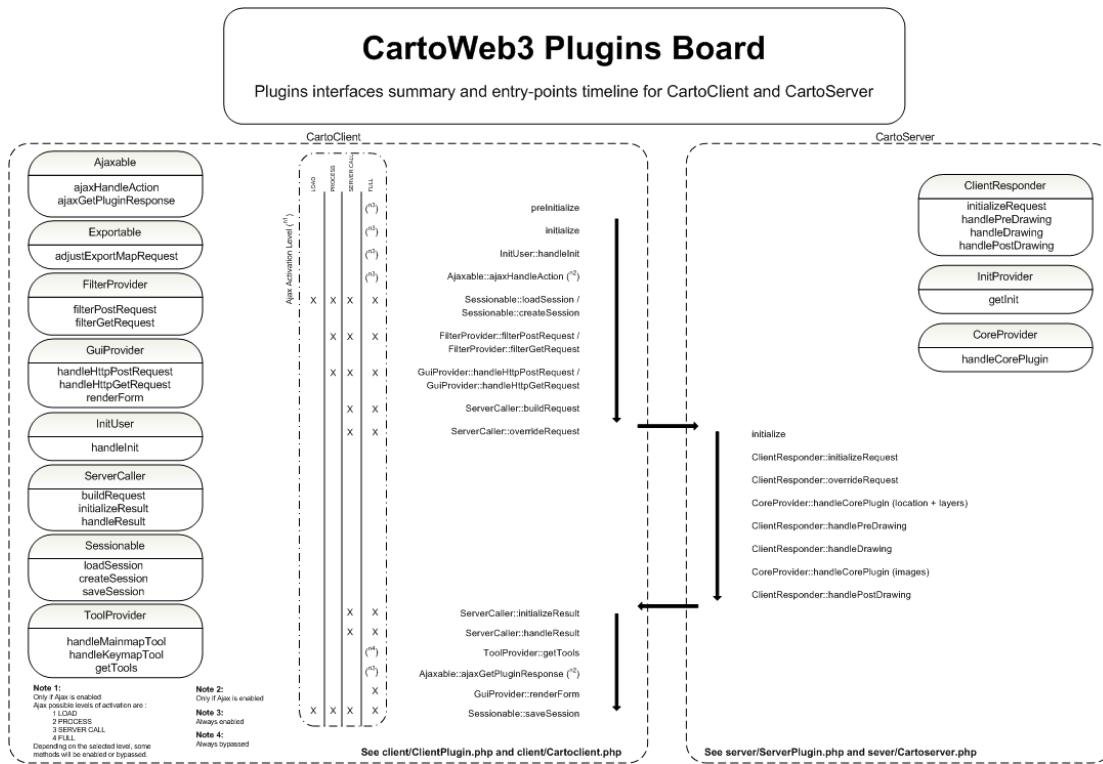
(Get a high quality version of this diagram here [http://www.cartoweb.org/doc/misc/plugins_diagram.pdf].)

2.2.4. Plugin Creation Check-List

1. Determine if you will write a plugin or a coreplugin.

2. Create a `<yourPlugin>/` directory in `/projects/<yourProject>/plugins/` if you need a simple plugin. Directory name will be the plugin name. You can use whatever name you want except of course names of already existing plugins or coreplugins. Yet it is recommended to use lowercase letters, capitalizing only the first letter of each word that composes the name (eg. "yourPluginName").

   In case of a coreplugin, there is no way to create a coreplugin in a project context. Coreplugins can only be integrated in the upstream application. It is not recommended to do so without CartoWeb developers agreement because of compatibility troubles that may occur when upgrading, etc.

3. Create subdirectories to store all plugin-related resources files and templates.

4. Create `client/`, `server/`, `common/` if your plugin as CartoClient, CartoServer and respectively common parts.

5. Create your main PHP classes files. Those files must be named using the first-letter-capitalized name of your plugin, prefixing it with "Client" or "Server"

for client or server components (eg. `ClientYourPlugin.php`, `ServerYourPlugin.php`, `YourPlugin.php`).

6. Extend *ClientPlugin* and/or *ServerPlugin* CartoWeb classes in the matching above files and name the result classes using their files names (with no ".php"). For instance:

```php
<?php
/**
 * @version $Id: ServerYourPlugin.php,v 1.8 2005/02/23 11:52:43 johndoe Exp $
 */

class ServerYourPlugin extends ServerPlugin {
```

7. Make your classes implement needed interfaces and redefine corresponding methods. Note that the `common/`part class "YourPlugin" does not have to extend or implement any CartoWeb class or interface. It is used as a container for common data and treatment used by client and server classes.

8. If your plugin is implementing *ServerCaller* and *ClientResponder*, you will need to write the WSDL code for the plugin request and response. See Section 2.1, " What are Plugins" for examples on how to write such WSDL documents.

Note

You need to add a *className* element in the WSDL file for a proper parsing from CartoWeb

---

9. Activate your plugin by adding its name to the *loadPlugins* of the matching project configuration files.

2.2.5. Automatic Files Inclusion

Main plugin PHP files (eg. `ClientYourPlugin.php`, `ServerYourPlugin.php`, `YourPlugin.php`) are automatically included and their contained classes and objects are directly accessible. Other files in `client/`, `server/`or `common/`are not and must be included explicitely in the main plugin PHP files.

Templates stored in the plugin `templates/`directory are also accessible directly by using PHP code similar to the followin one:

```php
$smarty = new Smarty_Plugin($this->getCartoclient(), $this);
$smarty->assign('foo', 'bar');
return $smarty->fetch('yourPlugin.tpl');
```

2.3. Adapting a Plugin

2.3.1. Approaches

If an already available plugin or coreplugin offers close functionalities to the ones you need, if you wish to slightly modify its behavior or simply want to adapt its output to your website layout, it is far easier to adapt it then to build a new one from scratch.

There are two levels of plugin adaptation. You can:

- override its HTML templates, resources (pictos, CSS or JS files) and its configuration as well. This approach is generally sufficient when you only need to adapt the layout.
- extend the main PHP classes to add your own methods or overload existing ones. This approach is required when you need to add some PHP code to the plugin.

Both approaches are not incompatible and may be combined to obtain desired result. See Section 2.3.4, "Combining Both Approaches" for more explanations.

2.3.2. Overriding a Plugin

Overriding a plugin is the simplest way to adapt it to your needs. It is done by duplicating the plugin files (at least the ones you want to adapt) in your project frame. For more information about projects handling, see Section 2.4, "Projects".

This approach is recommended when you want to use your own versions of the plugin templates or resources. Moreover you can add any new resources files that will be called in your customized templates. However you will not be able to replace or add PHP files (except PHP pages in the plugin `htdocs/`. To adapt a plugin server-side behavior (PHP classes), you have to extend the plugin, which is explained in Section 2.3.3, "Extending a Plugin".

Say for instance, you want customize the *layers* coreplugin by modifying its template `layers.tpl` and rewriting some of its JS tools (stored in `layers.js`). Than your project-adapted coreplugin will look like:

```
/projects/<yourProjectName>/coreplugins/layers/
/projects/<yourProjectName>/coreplugins/layers/templates/
/projects/<yourProjectName>/coreplugins/layers/templates/layers.tpl
/projects/<yourProjectName>/coreplugins/layers/htdocs/
/projects/<yourProjectName>/coreplugins/layers/htdocs/js/
/projects/<yourProjectName>/coreplugins/layers/htdocs/js/layers.js
```

If you don't need to override the CSS file, it is no use to create a `css/` directory containing a copy of the upstream `layers.css`.

If you want to neutralize a file, you can simply override it with a blank version. For instance to have a void output, create a template file with no content.

It is also possible to override the plugin configuration files by adding `<pluginName>.ini` files in the project configuration directories `client_conf/` and/or `server_conf/<mapId>/`. When the plugin is launched, upstream and project configuration files are merged so you don't need to duplicate the configuration parameters that stay unchanged with your adapted plugin.

2.3.3. Extending a Plugin

Extending a plugin is required when your adaptations involve deep changes such as additions or overloadings of methods in the plugin PHP classes.

To do so you will have to extend the plugin PHP classes in the object-oriented programing definition. Since plugin main classes are named using a plugin-name based convention (eg. *ClientLayers* and *ServerLayers* for the CartoClient and CartoServer main classes of the *layers* coreplugin) and since extended classes cannot have the same name than their parent class, you will have to rename your plugin. Any plugin name is OK (as long as it is not already used!) but it is recommended to use a <projectName><initialPluginName> separating words with caps.

Extended coreplugins directories and files must be saved in `/projects/<projectName>/coreplugins/<extendedPluginName>/` whereas extended simple plugins ones will be located in `/projects/<projectName>/plugins/<extendedPluginName>/`.

When writing your extended class, the first thing to do is to specify what plugin is replaced by the new one. This is done by overloading the *replacePlugin()* method. It should return the replaced plugin name. For instance, to extend the *layers* coreplugin CartoClient part in your *myProject* project, create a `ClientMyProjectLayers.php` as follows:

```php
<?php
/**
 * @package CorePlugins
 * @version $Id: ClientMyProjectLayers.php,v 1.8 2005/02/23 11:52:43 johndoe Exp $
 */

class ClientMyProjectLayers extends ClientLayers {

    public function replacePlugin() {
        return 'layers';
    }

    /* Add or overload methods here */

}
?>
```

To be activated, extended plugins AND coreplugins must be explicitely be declared in the *loadPlugins* parameter of your project general configuration files:

```
loadPlugins = exportPdf, auth, myProjectLayers
```

With no surprise, extended classes can take advantage of the tools provided by the interfaces their parent classes implement. By implementing additional interfaces, they will have access to complementary interactions as well. In that case, don't forget to overload the matching interface-defined methods in your extended class. For instance:

```
ClientMyProjectLayers extends ClientLayers
                      implements ToolProvider {

/* ... */

}
```

Tip

To use new config parameters in your extended plugin, add them in the .ini file of your project original plugin (eg. `/projects/<your_project>/client_conf/<original_plugin>.ini`).

2.3.4. Combining Both Approaches

If you need to modify/add templates or resources (overrriding) AND PHP classes (extension), you can combine both approaches by following the instructions of the two last sections.

Say you would like to customize the *images* coreplugin (mainmap size and formats management) to:

- update the layout (new pictos, new texts, new CSS),
- add some JS processing,
- add a new form field in a separated area of the CartoWeb interface.

First point is achieved by creating a `images/` directory in `/projects/yourProject/coreplugins/` and filling it with an overriden template `mapsizes.tpl`, a new `images.css` and some pictos:

```
/projects/yourProject/coreplugins/images/
/projects/yourProject/coreplugins/images/templates/mapsizes.tpl
/projects/yourProject/coreplugins/images/htdocs/css/images.css
/projects/yourProject/coreplugins/images/htdocs/gfx/button.png
```

```
<!-- mapsizes.tpl -->
<p>{t}Mapsize:{/t}
<select name="mapsize" id="mapsize" onchange="javascript:checkMapsize();">
{html_options options=$mapsizes_options selected=$mapsize_selected}
```

```
</select>
<input type="image" src="{r type=gfx plugin=images}button.png{/r}"
alt="{t}Ok button{/t}" id="imagesButton" /></p>
```

For details about template *{r}* (resource) and *{t}* (translation) tags, see Section 17.2, "Internationalization" and Section 17.3, "Resources".

checkMapsize() JavaScript function is not defined in the upstream coreplugin. So we have to add a JS file in our overriden plugin:

```
/projects/yourProject/coreplugins/images/htdocs/js/
/projects/yourProject/coreplugins/images/htdocs/js/images.js
```

```
/* images.js */

function checkMapsize() {
  alert('foobar');
}
```

To add a new form field in a separated area and consequently in a separated template, there is no other way than to modify the *ClientImages* PHP class in order to call the additional template in a special method. The extension approach is thus required. Then create a `yourProjectImages/` directory in `/projects/yourProject/coreplugins/` as follows:

```
/projects/yourProject/coreplugins/yourProjectImages/
/projects/yourProject/coreplugins/yourProjectImages/client/
/projects/yourProject/coreplugins/yourProjectImages/client/ClientYourProjectImages.php
```

It can seem a little tricky but the new template file (say `yourImages.tpl`) will not be stored in `/projects/yourProject/coreplugins/yourProjectImages/templates/` as one can expect it but in `/projects/yourProject/coreplugins/images/templates/` with the templates of the "overriden part" of the coreplugin. Remember: templates are stored in the overriden part and PHP classes in the extended part.

```
<!-- yourImages.tpl -->

<input type="text" name="testField" value="{$imagesTest}" />
```

```php
<?php
/**
 * @version $Id: ClientYourProjectImages.php,v 1.8 2005/02/23 11:52:43 johndoe Exp $
 */

class ClientYourProjectImages extends ClientImages {

    // indicates that we want to use current plugin instead of
    // regular images plugin
    public replacePlugin() {
        return 'images';
    }

    // overloaded method
```

202

```
    public function renderForm(Smarty $template) {
        // a {$image2} Smarty var must have been added in cartoclient.tpl
        $template->assign('image2', $this->drawNewImagesField());
        parent::renderForm($template);
    }

    // additional method
    private function drawNewImagesField() {
        $smarty = new Smarty_Plugin($this->getCartoclient(), $this);
        $smarty->assign('imagesTest', 'Foobar');
        return $smarty->fetch('yourImages.tpl');
    }
}
?>
```

Don't forget to activate the extended plugin in `/projects/yourProject/`
`client_conf/client.ini`:

```
loadPlugins = yourProjectImages
```

2.4. Special Plugins

2.4.1. Export Plugins

Export plugins allow to export maps and data. Concepts described below provide
tools to help writing such plugins.

In brief, export plugins follow these steps in order to generate an output:

- Retrieve last request that was sent to server
- Call all plugins to ask for request modification (eg. map resolution changes,
  keymap generation enabled/disabled, etc.)
- Call server to get a new modified result
- Use the result to generate output
- Return output in a standardized form

2.4.1.1. Export Plugin Naming Convention

Warning

Export plugins MUST have names that begin with the string "export", for instance
`exportPdf`, `exportCsv` or `exportHtml`.

This rule does not apply to extended plugins since only the original plugin name is
detected.

2.4.1.2. ExportPlugin

Class `ExportPlugin` implements a special type of client plugin, with some specific functionnalities for export. It implements interface `GuiProvider` so child classes must implement corresponding methods. Class methods are:

- `getLastMapRequest`: returns last used request object. This is useful to prepare a new call to server in order to obtain data specific to export. This call is done in method `getExportResult`
- `getLastMapResult`: This can also be useful in some cases to have the last returned result obect
- `getExportResult`: executes call to server in order to obtain a modified result suitable for export generation. Calls all exportable plugins in order to modify request (see Section 2.4.1.4, "Exportable Interface")
- `getExport` (abstract): contains export generation itself. Should prepare export configuration, call `getExportResult` and generate export in an `ExportOutput` object

2.4.1.3. ExportConfiguration

Export configuration objects contain information on what is needed by export plugin to generate output. For instance, for a CSV export, no images are needed and it would be a waste of time to generate them.

Configuration is set in method `getExport`, then passed to method `getExportResult` in order to get modified result. Configuration is used by plugin to know how to modify request to retrieve useful data.

2.4.1.4. Exportable Interface

Exportable interface declares a method `adjustExportMapRequest` which modifies a standard map request to a special export request. For instance, plugin Image uses `ExportConfiguration` object to know if maps are needed by export plugin. If not, image request is modified.

2.4.1.5. Example

Plugin `exportCsv` is a good, simple example of export plugin.

Configuration is filled in method `getConfiguration`. No images are required to output a CSV file:

```
protected function getConfiguration() {
    $config = new ExportConfiguration();
    $config->setRenderMap(false);
    $config->setRenderKeymap(false);
    $config->setRenderScalebar(false);
```

```
}
```

Result rendering is done in method `getExport`. Note that no calls to methods `getLastMapRequest` or `adjustExportMapRequest` are needed, as those calls are handled by method `getExportResult`:

```
protected function getExport() {
    $this->getExportResult($this->getConfiguration());

    // ...

    $output = new ExportOutput();
    $output->setContents($contents);
    return $output;
}
```

Final output (headers + content) is done in method `output`. For instance:

```
public function output() {
    header('Content-Type: text/html');
    print $this->getExport()->getContents();
    return '';
}
```

2.4.2. Filters

Filter plugins can be used to modify parameters transfered from browser to CartoWeb client. These parameters can be part of a POST request (HTML forms) or a GET request (URL query string).

Once a new filter plugin has been developed, it can be activated by adding it to the *loadPlugins* variable in file `client_conf/client.ini`.

2.4.2.1. Interface and Classes

Interface FilterProvider declares following methods:

* `filterPostRequest(FilterRequestModifier $request)`: modifies parameters transfered via a POST request
* `filterGetRequest(FilterRequestModifier $request)`: modifies parameters transfered via a GET request

Class `FilterRequestModifier` is used to get old values from the request and set new ones. It implements two main methods:

* `getValue($key)`: retrieves old value
* `setValue($key, $value)`: sets new value

2.4.2.2. Available Parameters

This is the list of global parameters accepted by CartoWeb via GET

- `reset_session` : This will reset all session parameters to default value. Useful when you want to go back to the initial stats of the application

    ```
    http://www.yoursite.org/cartoweb3/htdocs/client.php?reset_session
    ```

- `prevent_save_session` : This will bypass the session save. Useful when you want to performe an action without effect on the actual state of the application. This was added specifically when using the mode=image global parameter

    ```
    http://www.yoursite.org/cartoweb3/htdocs/client.php?prevent_save_session
    ```

- `mode` : This will define the kind of output of CartoWeb. By default it will be html. Another possible output is `image`. In this case, CartoWeb will only output an image.

    ```
    http://www.yoursite.org/cartoweb3/htdocs/client.php?mode=image
    ```

    The image mode can be used with other plugin specific parameters:

    ```
    http://www.yoursite.org/cartoweb3/htdocs/client.php?mode=image&recenter_x=300000&recenter_y=51500(
    ```

This is the list of parameters that can be set in a filter plugin using function `setValue()` (in php file) or GET request (URL query string):

- Images plugin
    - `mapsize` - ID of the selected map size (see Section 8.1, " Client-side Configuration ")
    - `customMapsize` - this string parameter enables to specify a customized map size using the syntax `[width]x[height]`, `[width]` and `[height]` being positive integers (size in pixels). You may specify dimensions limitations in `client_conf/images.ini` (see Section 8.1, " Client-side Configuration ")
    - `drawMainmap` - boolean indicating if the mainmap must be drawn or not.
    - `drawKeymap` - boolean indicating if the keymap must be drawn or not.
    - `drawScalebar` - boolean indicating if the scalebar must be drawn or not.
- Location plugin
    - `recenter_bbox` - new bounding box, comma-separated coordinates, eg.

"10.5,20,15.5,28"

- o *recenter_x* - re-centering: new x-coordinate
- o *recenter_y* - re-centering: new y-coordinate
- o *show_crosshair* - '0' or '1', default '0'. If '1' display a crosshair on (recenter_x, recenter_y).
- o *recenter_scale* - new scale
- o *id_recenter_layer* - re-centering on objects: layer to look for IDs
- o *id_recenter_ids* - re-centering on objects: list of IDs, comma-separated
- o *shortcut_id* - ID of the selected map size (see Section 7.2, "Server-side Configuration")

- • Query plugin
  - o *query_layer* - layer to look for IDs
  - o *query_select* - IDs of objects to add to selection
  - o *query_unselect* - IDs of objects to remove from selection
  - o *query_policy* - selection policy: 'POLICY_XOR', 'POLICY_UNION' or 'POLICY_INTERSECTION', default is 'POLICY_XOR'
  - o *query_maskmode* - '0' or '1', default is '0'. If '1', will show selection as a mask.
  - o *query_hilight* - '0' or '1', default is '1'. If '0', won't shows selection hilighted.
  - o *query_return_attributes* - '0' or '1', default is '1'. If '0', won't return attributes other than IDs.
  - o *query_return_table* - '0' or '1', default is '1'. If '0', won't return any table results.
  - o *query_clear* - '0' or '1', default is '0'. If '1', previous query results are purged even if they were marked as persistent.

- • Layers plugin
  - o *switch_id* - id of the switch to be used when the children switching is activated (see Section 6.2.4, "Children Switching").
  - o *layer_select* - comma-separated list of layers or layerGroups id's that must be added into the activated layers list.
  - o *layer_unselect* - comma-separated list of layers or layerGroups id's that must be removed from the activated layers list. In some cases, layerGroups children might have to be explicitly listed to actually be removed.

Warning

Note that for Query plugin, display of extended selection must be disabled in client's `query.ini` in order to use above parameters (see Section 9.1, "Client-side Configuration").

Tip

To make CartoWeb outputs a map as a raw image file (no HTML generated), you may use the GET parameter `mode=image`. For instance:

```
http://example.com/cartoweb3/client.php?mode=image&mapsize=2
```

2.4.2.3. Example

The following class implements a filter which allows to recenter on an object while hilighting it:

```
class ClientFilterIdrecenter extends ClientPlugin
                             implements FilterProvider {

    public function filterPostRequest(FilterRequestModifier $request) {}

    public function filterGetRequest(FilterRequestModifier $request) {

        $id = $request->getValue('id');
        if (!is_null($id)) {
            $layer = 'grid_classhilight';
            $request->setValue('query_layer', $layer);
            $request->setValue('query_maskmode', '1');
            $request->setValue('query_select', $id);

            $request->setValue('id_recenter_layer', $layer);
            $request->setValue('id_recenter_ids', $id);
        }
    }
}
```

2.4.3. Tables

Tables plugin is responsible for table formatting and display.

2.4.3.1. Tables Structures

Tables plugin declares several structures to help plugin developer manage tables. These structures are:

- Class `Table` which includes in particular a list of rows (class `TableRow`)
- Class `TableGroup` which includes in particular a list of tables. Table groups are used for instance to separate table results coming from several plugins
- Class `TableFlags` which defines parameters that will be useful for a plugin using tables

Typically, a plugin using table will include a `TableFlags` in its request and a `TableGroup` in its result. This is the case for Query plugin, which is the only core

plugin which uses tables.

2.4.3.2. Setting Rules

Tables plugin maintains an object called the registry (one on client and one on server). This object allows to add table rules, which will describes how tables must be displayed.

It is recommended to add rules in plugin's `initialize()` method, so they are ready at the earliest stage. To obtain the registry object, first you have to get the Tables plugin object.

On client:

```
public function initialize() {

    $tablesPlugin = $this->cartoclient->getPluginManager()->tables;
    $registry = $tablesPlugin->getTableRulesRegistry();

    // Add rules here
}
```

On server, plugin manager is stored in `ServerContext` object:

```
    // ...
    $tablesPlugin = $this->serverContext->getPluginManager()->tables;
    // ...
```

Now you are ready to add rules. Next sections describe the different types of rules. Registry's method signature is explained for each type.

Once rules have been added in registry, they must be executed on tables. See Section 2.4.3.4, "Executing Rules" for a description of table rules execution.

2.4.3.2.1. Column Selector

```
public function addColumnSelector($groupId, $tableId, $columnIds)
```

Column selector rules allow to keep only a subset of columns from the source table. Parameter *$columnIds* should contain an array of column IDs determining which columns to keep.

2.4.3.2.2. Column Unselector

```
public function addColumnUnselector($groupId, $tableId, $columnIds)
```

Column unselector rules allow to keep only a subset of columns from the source table, by removing a list of columns. Parameter $columnIds$ should contain an array of column IDs determining which columns to remove.

2.4.3.2.3. Group Filter

```
public function addGroupFilter($groupId, $callback)
```

Group filter rules allow to modify group title. Parameter $callback$ should contain a pointer to a callback method with the following signature:

```
static function myCallbackMethod('group_id', 'group_title')
    return 'group_new_title'
```

2.4.3.2.4. Table Filter

```
public function addTableFilter($groupId, $tableId, $callback)
```

Table filter rules allow to modify table title. Parameter $callback$ should contain a pointer to a callback method with the following signature:

```
static function myCallbackMethod('table_id', 'table_title')
    return 'table_new_title'
```

2.4.3.2.5. Column Filter

```
public function addColumnFilter($groupId, $tableId,
                                $columnId, $callback)
```

Column filter rules allow to modify column title. Parameter $callback$ should contain a pointer to a callback method with the following signature:

```
static function myCallbackMethod('table_id', 'column_id', 'column_title')
    return 'column_new_title'
```

2.4.3.2.6. Cell Filter

```
public function addCellFilter($groupId, $tableId, $columnId,
                              $inputColumnIds, $callback)
```

Cell filter rules allow to modify content of a cell. Values of columns given in parameter $inputColumnIds$ will be transfered to the callback method for cell content calculation. Parameter $callback$ should contain a pointer to a callback method with the following signature:

```
static function myCallbackMethod('table_id', 'column_id',
                                array ('column_1' => 'value_1',
                                       'column_2' => 'value_2'))
    return 'cell_value'
```

Note

To return all column's data to the callback function, you can use *NULL*.

```
$registry->addCellFilter('query', 'table_name', '*', NULL,
            array('TableRulesClassName', 'callbackFunctionName'));
```

This will apply the *callbackFunctionName* function to all column of table *table_name* and the callback function will receive an array containing the values of all columns.

2.4.3.2.7. Cell Filter (Batch)

```
public function addCellFilterBatch($groupId, $tableId, $columnId,
                                   $inputColumnIds, $callback)
```

Cell filter rules used in batch mode allow to modify content of all cells of a given column. Values of columns given in parameter $inputColumnIds will be transfered to the callback method for cells content calculation. Values for all rows are transfered at the same time. Parameter $callback should contain a pointer to a callback method with the following signature:

```
static function myCallbackMethod('table_id', 'column_id',
                                array (
                                '0' => array (
                                        'column_1' => 'value_1_row_1',
                                        'column_2' => 'value_2_row_1'),
                                '1' => array (
                                        'column_1' => 'value_1_row_2',
                                        'column_2' => 'value_2_row_2')))
    return array ('0' => 'cell_value_row_1', '1' => 'cell_value_row_2')
```

2.4.3.2.8. Row Unselector

```
public function addRowUnselector($groupId, $tableId,
                                 $columnId, $rowIds)
```

Row unselector rules allow to remove some rows from a table. Parameter rowIds contains IDs of row that must be removed.

2.4.3.2.9. Row Selector

```
public function addRowSelector($groupId, $tableId,
                               $columnId, $rowIds)
```

Row selector rules allow to keep only some rows from a table. Parameter $rowIds$ contains IDs of row that must be kept.

2.4.3.2.10. ColumnAdder

```
public function addColumnAdder($groupId, $tableId,
                               $columnPosition, $newColumnIds,
                               $inputColumnIds, $callback)
```

Column adder rules allow to add one or more columns to the table. Parameter $newColumnIds$ should contain the list of new column IDs. Values of columns given in parameter $inputColumnIds$ will be transfered to the callback method for cell content calculation. Parameter $callback$ should contain a pointer to a callback method with the following signature:

```
static function myCallbackMethod('table_id',
                                 array ('column_1' => 'value_1',
                                        'column_2' => 'value_2'))
    return array ('new_column_1' => 'cell_value_1',
                  'new_column_2' => 'cell_value_2')
```

Parameter $columnPosition$ indicates where the new columns must be inserted. It should be an instance of class ColumnPosition. Positions can be absolute or relative, with a positive or negative offset:

-
    ```
    $position = new ColumnPosition(ColumnPosition::TYPE_ABSOLUTE, 1);
    ```

    The new columns will be added after the first column

-
    ```
    $position = new ColumnPosition(ColumnPosition::TYPE_ABSOLUTE, -2);
    ```

    The new columns will be added just before the last column

-
    ```
    $position = new ColumnPosition(ColumnPosition::TYPE_RELATIVE,
                                   0, 'column_1');
    ```

    The new columns will be added just before column 'column_1'

-
    ```
    $position = new ColumnPosition(ColumnPosition::TYPE_RELATIVE,
                                   1, 'column_1');
    ```

    The new columns will be added just after column 'column_1'

2.4.3.2.11. Column Adder (Batch)

```
public function addColumnAdderBatch($groupId, $tableId,
                                    $columnPosition, $newColumnIds,
                                    $inputColumnIds, $callback)
```

Column adder rules used in batch mode allow to add one or more columns to the table, while calculating values for all newly added cells. Parameter $newColumnIds should contain the list of new column IDs. Values of columns given in parameter $inputColumnIds will be transfered to the callback method for cells content calculation. Values for all rows are transfered at the same time. Parameter $callback should contain a pointer to a callback method with the following signature:

```
static function myCallbackMethod('table_id',
                                 array (
                                 '0' => array (
                                        'column_1' => 'value_1_row_1',
                                        'column_2' => 'value_2_row_1'),
                                 '1' => array (
                                        'column_1' => 'value_1_row_2',
                                        'column_2' => 'value_2_row_2')))
   return array (
         '0' => array (
                'new_column_1' => 'cell_value_1_row_1',
                'new_column_2' => 'cell_value_2_row_1'),
         '1' => array (
                'new_column_1' => 'cell_value_1_row_2',
                'new_column_2' => 'cell_value_2_row_2')))
```

See Section 2.4.3.2.10, "ColumnAdder" to know more about parameter $columnPosition.

2.4.3.2.12. Column reorder

```
public function addColumnReorder($groupId, $tableId, $columnIds)
```

Column reorder rule allow you to reorder the columns and their contents. Parameter $columnIds should contain an array of column IDs given the new column's order. Note that all the IDs must appear in $columnIds even they don't move.

2.4.3.3. Precedence of Rules

Depending on rule type, rules are set for a group, a table or a column. Parameters ($groupId, $tableId or $columnId) can point to one object or to a group of object, using wildcard '*':

- 'column_1': rule will be executed on columns called 'column_1' only

- 'col*': rule will be executed on columns with name starting with 'col'
- '*': rule will be executed on any columns

For instance, following rule may be executed on groups with name starting with 'myGr', tables called 'myTable' and all columns:

```
$registry->addColumnFilter('myGr*', 'myTable', '*',
                              array($this, 'myCallbackMethod));
```

Only one rule of each type may be executed on one item. If two or more rules apply, most specific rule will be choosen. In the following rule definition, only the third rule will be executed on a table 'myTable' in a group 'myGroup':

```
$registry->addColumnSelector('*', '*', array('column_1', 'column_2'));
$registry->addColumnSelector('myGr*', '*', array('column_1'));
$registry->addColumnSelector('myGr*', 'myTable', array('column_2'));
$registry->addColumnSelector('myGroup', 'toto', array('column_3'));
```

2.4.3.4. Executing Rules

2.4.3.4.1. On Client

Each time a table group is created, it must be stored in Tables plugin in order to display it:

```
$tablesPlugin = $this->cartoclient->getPluginManager()->tables;
$tablesPlugin->addTableGroups($newTableGroup);
```

Tables rules are executed automatically at the same time.

2.4.3.4.2. On Server

Rules execution must be done explicitly on server. A call to Tables plugin `applyRules` method is needed for each new table group before returning it to client:

```
$tablesPlugin = $this->serverContext->getPluginManager()->tables;
readyForClientTableGroups = $tablesPlugin->applyRules($newTableGroup);
```

3. Dynamic mapfile modifications

3.1. Introduction

This chapter describes the mapOverlay plugin. This plugin allows dynamic modifications of the mapfile, it can be used for example to: let the user add a WMS layer, restrict data to be exported to the user, change the color of layers, etc ...

The plugin has only a server part and only a public method:

```
public function updateMap(BasicOverlay $overlay)
```

The `$overlay` argument is either a `MapOverlay` or a `LayerOverlay` instance, it represent respectively a mapfile or a layer.

Customizable elements are:

* `ColorOverlay`
* `StyleOverlay`
* `LabelOverlay`
* `ClassOverlay`
* `MetadataOverlay`
* `LayerOverlay`
* `MapOverlay`

The organization of these elements are the same as the mapfile structure: a `MapOverlay` contains zero or many `LayerOverlay` who cantains zero or many `MetadataOverlay` etc. None of theses elements or instance variables are mandatory, you just need to provide informations about property you want to search, update, insert or delete.

For each of theses mapfile elements the developer have to provide an action by setting the `action` instance variable. The action set the behavior of the element during the mapfile update process. `action` is one of:

* `BasicOverlay::ACTION_SEARCH`: this action search or create the element.
* `BasicOverlay::ACTION_INSERT`: insert the element at a given position or at the end if not specified.
* `BasicOverlay::ACTION_REMOVE`: remove the element.
* `BasicOverlay::ACTION_UPDATE`: update or create the element.

If no action is specified, the default action is `BasicOverlay::ACTION_UPDATE`.

The standards steps to use this mechanism are:

1. Create all the needed elements and combine them together in a `MapOverlay` or a `LayerOverlay` instance.
2. Call the `updateMap()` function. The function returns a `MapOverlay` instance.
3. If needed, use the result instance to retrieve an index or a element name.

Note that the modification are not saved in the project's mapfile: the result of the mapfile modifications are specific to a session.

3.2. Plugin usage

This chapter contains some basic examples of the plugin usage. All the code listed here must be placed in the server part of a plugin.

3.2.1. Class creation

Consider this code excerpt:

```
$color = new ColorOverlay();
$color->red = 255;
$color->green = 0;
$color->blue = 0;

$style = new StyleOverlay();
$style->index = 0;
$style->color = $color;

$label = new LabelOverlay();
$label->outlineColor = $color;

$class = new ClassOverlay();
$class->action = BasicOverlay::ACTION_INSERT;
$class->name = 'foobar';
$class->styles = array($style);
$class->label = $label;
```

A `ColorOverlay` and a `StyleOverlay` are created. Create a `LabelOverlay` and set the text color. Note that those three instances ($color, $style and $label) have only a subset of theirs variable who are set.

```
$layer1 = new LayerOverlay();
$layer1->name = 'cartoweb_point_outline';
$layer1->classes = array($class);
```

Put the class in the layer. The behavior of `$layer1` can be resumed as: Update or create a layer named 'cartoweb_point_outline'. In this layer, insert a new class an call it 'foobar', the first style of this class render as red.

```
$layer2 = new LayerOverlay();
$layer2->action = BasicOverlay::ACTION_SEARCH;
$layer2->name = 'cartoweb_point_outline';
$layer2->transparency = 30;
```

Search a layer named 'point_point_outline' with a 30% transparency. If this layer don't exist in the mapfile it will be created

```
$map = new MapOverlay();
$map->layers = array($layer1, $layer2);

$plugins = $this->serverContext->getPluginManager();
$result = $plugins->mapOverlay->updateMap($map);
```

The two layers are inserted in a `MapOverlay` and the mapfile is updated.

3.2.2. Add a feature

The goal of this example is to add a feature with a specific style and transparency. The layer name and transparency are given ('foo' and 40%).

```
$layer = new LayerOverlay();
$layer->name = 'foo';
$layer->action = BasicOverlay::ACTION_SEARCH;
$layer->transparency = 40;

$mapOverlay = $this->serverContext->getPluginManager()->mapOverlay;
$result = $mapOverlay->updateMap($layer);
```

At this point, the layer and its classes are created or updated. Now we need to insert the feature:

```
$f = ms_newShapeObj(MS_SHAPE_POINT);
$p->addXY(30000, 20000);
$f->set('classindex', $result->layers[0]->classes[0]->index);
```

A `ms_newShapeObj` is created and the class index is fetched from the `updateMap()` return instance.

```
$msLayer = $this->serverContext->getMapObj()->getLayer($result->layers[0]->index);
$msLayer->addFeature($f);
$msLayer->set('status', MS_ON);
```

This example also show that you can mix standard phpMapscript and mapOverlay code.

3.2.3. Filter data

```
$layer = new LayerOverlay();
$layer->name = "field";
$layer->data = "geom FROM (SELECT gid, geom, name FROM fields WHERE farm_id = {$farmId}) " .
               "AS foo USING UNIQUE gid USING SRID=-1";
```

217

```
$mapOverlay = $this->serverContext->getPluginManager()->mapOverlay;
$mapOverlay->updateMap($layer);
```

The DATA field of the field layer is update to only display some informations. $farmId is defined elsewhere.

In this case, a much more clever solution is to use the filter attribute of the LayerOverlay object.

```
$layer = new LayerOverlay();
$layer->name = "field";
$layer->filter = "farm_id = {$farmId})";

$mapOverlay = $this->serverContext->getPluginManager()->mapOverlay;
$mapOverlay->updateMap($layer);
```

3.2.4. Debugging

At any time you can see the result of the modifications by calling:

```
$this->serverContext->getMapObj()->save('/tmp/debug.map');
```

This will save the updated mapfile to /tmp/debug.map.

4. Using the Security Infrastructure

4.1. Introduction

This chapter describes the security infrastructure from the developer point of view. For general details about security and its configuration see Chapter 15, *Security Configuration*.

The security management in cartoweb is separated in the following parts:

- Management of the user/password/roles database. (`SecurityContainer` class in `common/SecurityManager.php`).
- Management of user authentication (calling `checkUser` and `setUser/setUserAndRoles` in `SecurityManager`).
- Granting access to objects based on the current roles.

4.2. Plugins Managing Security Database and Authentication

Point 1. and 2. in the previous section are the responsibility of specific plugins. For an example, see the `auth` plugin.

4.3. Plugins Granting or Denying Access to Objects/Features in CartoWeb

This point is the most important for plugin developers wanting to use the CartoWeb security mechanisms to allow or deny an access to a feature/object.

The plugin can call the method `hasRole($roles)` on the current security manager.

For an example, let's take the pdf plugin which has to restrict printing some formats only to allowed users.

in the .ini file, we could have:

```
formats.A4.allowedRoles = printers, admin
```

In the plugin, we can then check the permissions with:

```
in the routine building the available format list:
```

```
foreach($formats as $format) {

 ... add the format to the list ...
 $roles = $this->getRolesForFormat($format); //this should get it from the .ini
 if (!SecurityManager::getInstance()->hasRole($roles))
     continue; // skips unauthorized resolution for this user

  .. do the work with the format ...
}

in the routine handling the user passed parameters:

.. to the same check as above ..
```

5. Internationalization

5.1. Translations

Texts to be translated can be found in:

- Smarty templates (see Section 17.2, "Internationalization")
- Client and server .ini files and map files (see Chapter 4, *Configuration Files*)
- Client and server PHP code

In the last case, the script which finds strings to be translated (see Section 16.1.2, "PO Templates") looks for calls to `gt()` functions. There are two different `gt()` functions:

- `I18n::gt()`: tries to translate the string given as argument and returns the translation. This function assumes that string is UTF-8 encoded and returns a string ready for output (see Section 5.2, "Character Set Encoding"). It can be used on client side only
- `I18nNoop::gt()`: does nothing during runtime ("noop" stands for "no operations"). Call to this function is only needed to indicates that the string must be translated. This function can be used on client and server side

Below is an example on how to use **I18nNoop::gt()**:

```
/**
 * Example for use of I18n gt() functions (client side)
 */
class ClientMyPlugin extends ClientPlugin
                     implements GuiProvider, ServerCaller {

    // ...

    public function initializeResult($myPluginResult) {

        // Retrieves message
        $this->message = $myPluginResult->message;
    }

    public function renderForm(Smarty $smarty) {

        // Translation and display
        $smarty->assign('message', I18n::gt($this->message));
    }
}
```

```
/**
 * Example for use of I18n gt() functions (server side)
 */
class ServerMyPlugin extends ClientResponderAdapter {
```

```
    // ...

    public function handlePreDrawing($request) {

        $myPluginResult = new MyPluginResult();

        // Message must be translated, but not now!
        $myPluginResult>message = I18nNoop::gt('hello, world');

        return $myPluginResult;
    }
}
```

In this example, message sent by server has to be translated. But as translation process is always done on client, we only indicates to the script that there is a text to add to the translation template.

5.2. Character Set Encoding

As already described in Section 16.2, "Character Set Encoding Configuration", character set encoding is done using `Encoder` set of classes. It uses functions `Encoder::encode()` and `Encoder::decode()`:

- `Encoder::encode($text, $context)`: converts text from context's character set to CartoWeb's internal character set (UTF-8)
- `Encoder::decode($text, $context)`: converts text from CartoWeb's internal character set (UTF-8) to context's character

Context can be either 'config' or 'output', default is 'output'. Corresponding configuration is set in `server.ini` and `client.ini` (see Section 16.2, "Character Set Encoding Configuration").

Function `Encoder::encode()` must be used in the following situation:

- on client or server when reading a text from a configuration file:

```
$encodedText = Encoder::encode($readText, 'config');
```

Function `Encoder::decode()` must be used in the following situations:

- on client when outputing a text without calling `I18n::gt()`:

```
$textToDisplay = Encoder::decode($encodedText);
```

- on client or server when calling an external module, eg. Mapserver for a query:

```
$textToUseInMapserver = Encoder::decode($encodedText, 'config');
```

Note that function `I18n::gt()` takes an encoded text as argument and already prepares texts for output. It means you don't need to call `Encoder::decode()` after a call to `I18n::gt()`.

6. Code Convention

6.1. Introduction

As an Open-Source software, CartoWeb needs to be written following some coding guidelines and/or rules. It is the required condition unless the developpers community isn't able to share new features and enhancements.

Some of those advises may seem obvious, others less. For all, it is principaly a good way to produce more readable, maintainable and portable code.

6.2. General Coding Rules

6.2.1. Paths

It is highly recommended to be avoid absolute paths as much as possible. CartoClient and CartoServer may be relocated with very minimal even none reconfiguration.

6.2.2. Extract and Run Deployement

It should be possible to extract the archive, launch a script, edit few options, and be ready to use the application with the built-in data set (see test mapfile).

6.2.3. Development Configuration

Developers should absolutely set the following variables to true in their config (on both client and server sides):

Warning

These parameters will be overrided by the `profile` parameter (See Section 4.1, "Common client.ini and server.ini Options").

- showDevelMessages = true
- developerIniConfig = true

6.2.4. Unit Tests

Code should produce no notices before any CVS commit. Code should pass all tests.

This also means that the developpers should add and run unit tests for every new

feature they add.

See Chapter 7, *Unit Tests*.

6.3. PHP

6.3.1. Coding Style

Developers should use the PEAR coding standards as the coding style reference in CartoWeb, with some exceptions. Have a look at http://pear.php.net/manual/en/standards.php.

Following are briefly described some guidelines to respect.

6.3.1.1. Indent

Developers should respect some indentation conventions when writing PHP code in CartoWeb:

- 4 spaces indentations are recommended,
- the use of tabs for indentation is prohibited, use space instead (select the appropriate preferences in your favorite code editor if needed).

6.3.1.2. Control Structures

Control statements should have one space between the control keyword and opening parenthesis.

It is also recommended to use curly braces even when they are optional.

This is correct:

```
if (condition) {
  ...
}
```

6.3.1.3. Function Calls

Functions should be called with no spaces between the function name, the opening parenthesis, and the first parameter; spaces between commas and each parameter, and no space between the last parameter, the closing parenthesis, and the semicolon.

This is correct:

```
$var = foo($bar, $baz, $quux);
```

6.3.1.4. Function Definitions

Function declarations follow the "one true brace" convention.

Arguments with default values go at the end of the argument list. Always attempt to return a meaningful value from a function if one is appropriate.

6.3.1.5. Class Modifiers

Modifiers keywords (public, protected, private) are mandatory in class members and methods definitions. It is recommended to use `private` for most members and properties. Use `public` only for methods that needs to be accessible from outside the class.

In coreplugins or plugins classes, it is generally better to use `protected` instead of `private` since most methods may have to be redeclared in extended classes (projects plugins). Then only use `private` for members/properties you do not want to be redeclared.

6.3.1.6. Nesting

Avoid deep blocks nesting:

This is correct:

```
for ($i = 0; $i < 10; i++) {
  if (! something ($i))
    continue;
  doMore();
}
```

6.3.1.7. PHP Code Tags

Always use

```
<?php ?>
```

to delimit PHP code, not the

```
<? ?>
```

shorthand.

6.3.1.8. Naming Conventions

6.3.1.8.1. Classes

Classes should be given descriptive names that should always begin with an uppercase letter. Avoid using abbreviations.

6.3.1.8.2. Functions and Methods

Functions and methods should be named using the "studly caps" style (also referred to as "bumpy case" or "camel caps").

```
function handleKeymapTool()
```

Functions and methods names should always describe actions.

Developpers should declare the access modifiers (public, private, protected) for each function or method.

6.3.1.8.3. Constants

Constants should always be all-uppercase, with underscores to seperate words.

6.3.2. Comments

6.3.2.1. Php Doc Comments

To improve php code and object structure readability, automatic code documentation is implemented in CartoWeb. It is based on specific comments describing classes, methods, interfaces and objects. See Chapter 8, *Code Documentation* for more information.

6.3.2.2. Inline Comments

As often as necessary, the developers should add code comments to explain verbosly the purposes of commands.

6.4. HTML Coding Standards

In CartoWeb, mainly for the templates, HTML coding should respect some rules.

To take benefits of recent browsers enhancements and, above all, to make HTML codes easier to read and maintain, some HTML-coding guidelines should be followed.

- for indentation : preferably use 2 white spaces (such an indentation might be used for javascript coding as well).
- Generated HTML pages should be XHTML 1.0 (say Transitional for now) valid and pass matching W3C validation: http://validator.w3.org/ [http://validator.w3.org/]

XHTML (standing for eXtensible Hypertext Markup Language) was chosen vs simple HTML for following reasons:

- XHTML is aimed to replace HTML
- XHTML is a stricter and cleaner version of HTML
- XHTML is HTML defined as an XML application

For more information on XHTML, reference and tutorial are available here : http://www.w3schools.com/xhtml/ Specially useful pages are:

- Differences between HTML and XHTML: http://www.w3schools.com/xhtml/xhtml_html.asp
- XHTML syntax: http://www.w3schools.com/xhtml/xhtml_syntax.asp

But, here are some things people must know to get XHTML valid generated pages.

6.4.1. Nesting

In HTML some elements can be improperly nested within each other like this:

```
<b><i>This text is bold and italic</b></i>
```

In XHTML all elements must be properly nested within each other like this:

```
<b><i>This text is bold and italic</i></b>
```

All XHTML elements must be nested within the <html> root element. All other elements can have sub (children) elements. Sub elements must be in pairs and correctly nested within their parent element. The basic document structure is:

```
<html>
  <head> ... </head>
  <body> ... </body>
</html>
```

6.4.2. Lower Case

This is because XHTML documents are XML applications. XML is case-sensitive. Tags like <br> and <BR> are interpreted as different tags.

This is wrong:

```
<BODY>
<P>This is a paragraph</P>
</BODY>
```

This is correct:

```
<body>
<p>This is a paragraph</p>
</body>
```

This is wrong:

```
<table WIDTH="100%">
```

This is correct:

```
<table width="100%">
```

6.4.3. Closing

6.4.3.1. All Elements

Non-empty elements must have an end tag.

This is wrong:

```
<p>This is a paragraph
<p>This is another paragraph
```

This is correct:

```
<p>This is a paragraph</p>
<p>This is another paragraph</p>
```

6.4.3.2. Empty Elements

Empty elements must either have an end tag or the start tag must end with />

This is wrong:

```
This is a break<br>
Here comes a horizontal rule:<hr>
Here's an image <img src="happy.gif" alt="Happy face">
```

This is correct:

```
This is a break<br />
Here comes a horizontal rule:<hr />
Here's an image <img src="happy.gif" alt="Happy face" />
```

IMPORTANT Compatibility Note:

To make your XHTML compatible with older browsers, you should add an extra space before the "/" symbol like this: <br />, and this: <hr />.

6.4.4. Minimization

This is wrong:

```
<dl compact>
<input checked>
<input readonly>
<input disabled>
<option selected>
<frame noresize>
```

This is correct:

```
<dl compact="compact">
<input checked="checked" />
<input readonly="readonly" />
<input disabled="disabled" />
<option selected="selected" />
<frame noresize="noresize" />
```

Here is a list of the minimized attributes in HTML and how they should be written in XHTML:

```
<dl compact="compact">
<input checked="checked" />
<input readonly="readonly" />
<input disabled="disabled" />
<option selected="selected" />
<frame noresize="noresize" />
```

6.4.5. Id vs Name

HTML 4.01 defines a name attribute for the elements a, applet, frame, iframe, img, and map. In XHTML the name attribute is deprecated. Use id instead.

This is wrong:

```
<img src="picture.gif" name="picture1" />
```

This is correct:

```
<img src="picture.gif" id="picture1" />
```

Note: To interoperate with older browsers for a while, you should use both name and id, with identical attribute values, like this:

```
<img src="picture.gif" id="picture1" name="picture1" />
```

IMPORTANT Compatibility Note:

To make your XHTML compatible with today's browsers, you should add an extra space before the "/" symbol.

6.4.6. Image "alt"

"alt" attribute is mandatory for <img /> tag in XHTML. But it can - and sometimes should - have a void value. "alt" is used to specify a replacement text when image is not loaded (image unavailable, not yet loaded or user deactivated images...). For "data" images, a convenient alternative text should be specified but for layout-only pictos it is no use to display a replacement message.

For instance:

```
<img src="mainmap.gif" width="400" height="200"
   alt="Main map" />
<img src="pixel.gif" width="50px" height="1px" alt="" />
```

7. Unit Tests

7.1. Introduction

Unit tests are an important componant in the CartoWeb development environment. The framework used for unit testing is based on PHPUnit2 [http://www.phpunit.de/en/index.php], a PEAR package. For more informations abouth PHPUnit2, see http://pear.php.net/reference/PHPUnit2-2.0.3/

PHPUnit2 is included in the libraries shipped with CartoWeb. Thus, no installation is needed to run and write new tests.

7.2. Writing Tests

Information about writing tests for CartoWeb can be separated into two parts. First part about writing Unit tests in general, useful for people new to PhpUnit. Then a part more specific about the infrastructure which is available in CartoWeb for writing tests.

7.2.1. General Information About Writing Tests

Test cases are located in directory `tests` for testing the core of CartoWeb and in the directories `<project>/tests` for for tests specific to project `<project>`. Inside these `tests` directories, path hierarchy mirrors the existing hierarchy they are testing. For instance, the `tests` hierarchy for the core CartoWeb testing is the following:

```
client
    CartoclientTest.php
    CartoserverServiceTest.php
    AllTests.php
common
    BasicTypesTest.php
    AllTests.php
coreplugins
    AllTests.php
    ...
plugins
    AllTests.php
    ...
```

For the `test_main` project, the hierarchy is the following:

```
coreplugins
    AllTests.php
    ...
plugins
    AllTests.php
    ...
misc
    AllTests.php
    < misc tests >
    ...
```

```
...
```

Each directory including tests root directory has a file named `AllTests.php`. This is called a test suite. It is used to run all tests of a specific directory (ie "package").

Warning

All test case and test suite classes must have the name of their file relative path without extension, with '/' replaced by '_'. For instance, class `client_CartoclientTest` file name must be `<cartoweb3_root>/tests/client/CartoclientTest.php`.

The following example shows a test in `common/BasicTypeTest.php` file:

Example 7.1. Simple test case (BasicTypesTests.php)

```php
<?php
require_once 'PHPUnit2/Framework/TestCase.php';
require_once(CARTOWEB_HOME . 'common/basic_types.php');

class common_BasicTypesTest extends PHPUnit2_Framework_TestCase {

        public function testBboxFrom2Points() {

                $bbox = new Bbox();
                $point1 = new Point(12, 34);
                $point2 = new Point(56, 78);
                $bbox->SetFrom2Points($point1, $point2);

                $this->assertEquals(12, $bbox->minx);
                $this->assertEquals(34, $bbox->miny);
                $this->assertEquals(56, $bbox->maxx);
                $this->assertEquals(78, $bbox->maxy);
        }
}
?>
```

Each function with name starting with 'test' will be considered as a test case by the automated test runner. You may also want to use functions `setUp()` and `tearDown()` to initialize and clean a test environment.

Method `assertEquals` tests if two values have the same values. If not, the test will be added to the final report as failure.

As stated previously, all test classes have to belong to a test suite. The next example shows how such a test suite is built, by grouping all tests together in the `suite()` method.

Example 7.2. Test suite (AllTests.php)

```php
<?php

require_once 'PHPUnit2/Framework/TestSuite.php';
require_once 'CartoclientTest.php';
require_once 'CartoserverServiceTest.php';

class client_AllTests {

        public static function suite() {

                $suite = new PHPUnit2_Framework_TestSuite;

                $suite->addTestSuite('client_CartoclientTest');
                $suite->addTestSuite('client_CartoserverServiceTest');

                return $suite;
        }
}
?>
```

All test suites are then grouped together into the root test suite. It is shown there for information.

Example 7.3. Root directory's AllTests.php:

```php
<?php

require_once 'PHPUnit2/Framework/TestSuite.php';
require_once 'client/AllTests.php';
require_once 'common/AllTests.php';

class AllTests {

        public static function suite() {

                $suite = new PHPUnit2_Framework_TestSuite;

                $suite->addTest(client_AllTests::suite());
                $suite->addTest(common_AllTests::suite());

                return $suite;
        }
}
?>
```

7.2.2. Specific Information for Tests

This section describes specific features developed in CartoWeb for running tests, and infrastructure classes for more simple test case writing.

7.2.2.1. HttpUnit Based Tests

To test features of the cartoclient, the HttpUnit [http://httpunit.sourceforge.net/] software is used. It is written in Java, and there is no Php port. The http unit tests are run if you have a JVM in you path.

For more information about running HttpUnit tests, see the `tests/client/httpunit/README` file in the CartoWeb distribution.

7.2.2.2. Testing CartoWeb Plugins

Plugins are a main component in CartoWeb architecture. That's why there is support to maintain common code used for testing plugins. As described in Section 7.2.1, "General Information About Writing Tests" the tests for plugins have to mirror the file hierarchy of the base application. That's why there are `coreplugins` and `plugins` directories where test for core plugins and plugins are stored respectively.

Note

Tests are also available for projects. So, to test a plugin in a project, the path of the test will be `<cartoweb3>/<projectname>/tests/plugin/<pluginname>`

Testing plugins is separated into two tasks:

1. Locally testing client, common or server classes. For plugin client classes, it requires a CartoClient environment available, and identically a CartoServer environment for testing server classes.
2. Remote plugin tests, throught the webservice API. This kind of tests are related to the server plugins, that's why we chose to put them in the `server` folder of plugins.

For the first point mentionned above, general Unit tests rules apply, as described in Section 7.2.1, "General Information About Writing Tests".

For the second point stated, a help class named `client_CartoserverServiceTest` can be extended by the testing classes. In turn, `client_CartoserverServiceTest` extends other classes which offer additional helpful methods. For the complete list of available methods, please have a look at the generate API docs (as more may be added in future). The main useful methods are `createRequest()` for initializing a new request, `getMap()` for lanching the request.

Tip

Having a look at an existing plugin test case is the best starting point for writing new tests.

7.2.2.3. Tests for projects

Each CartoWeb project can have its set of tests directly inside the project, in a directory `tests`. Inside this directory, you have to use the same hierarchy as in the main `tests` directory.

Inside these `tests` directories, the same mechanism is used for testing as the main `tests` directory. The best way to get started is to have a look at the `test_main` project for a starting point.

7.2.2.3.1. Invoking tests for a specific project

For now, only the command line can be used for invoking tests for a specific project. An environment variable CARTOWEB_TEST_PROJECT can be used to define which test to launch. Under a Unix like shell enviroment, the command line to use is:

```
CARTOWEB_TEST_PROJECT=<myproject> <php-bin> phpunit.php projects_AllTests projects/AllTests.php
```

7.3. Running Tests

Unit tests are runned using the command line interface (CLI). To run a test case or a test suite, type the following command in directory `<cartoweb3_root>/tests`:

```
<php-bin> phpunit.php <test-class> <php-file>
```

Where <php-bin> is the PHP binary, <test-class> is the name of the test class (`AllTests`, `client_AllTests`, `client_CartoclientTest`, etc.) and <php-file> is the name of the PHP file containing the test case (`client/CartoclientTest.php`).

Result should look like this:

```
PHPUnit 2.0.3 by Sebastian Bergmann.

.......F.....

Time: 0.0410950183868
There was 1 failure:
1) testpointtobbox
expected same: <113> was not: <123>
/home/yves/cartoweb3-proto2/tests/common/BasicTypesTest.php:59
/home/yves/cartoweb3-proto2/tests/phpunit.php:24

FAILURES!!!
Tests run: 12, Failures: 1, Errors: 0, Incomplete Tests: 0.
Content-type: text/html
X-Powered-By: PHP/5.0.1
```

In this case, 12 tests were run with one failure.

236

8. Code Documentation

CartoWeb code documentation is generated using PhpDocumentor [http://www.phpdoc.org/], a JavaDoc-style doc generator. CartoWeb already includes version 1.3.0rc3 of PhpDocumentor.

8.1. Generating Documentation

Documentation is generated using script `makedoc.php`:

```
cd scripts
php makedoc.php
```

This will generate documentation in directory `CARTOWEB_HOME/documentation/apidoc`.

8.2. DocBlocks

DocBlocks are comments located at the beginning of a file, or just before a class, a method, a function outside a class or a variable declaration. These comments will be parsed by PhpDocumentor to generate documentation.

For a full description of DocBlocks, see official PhpDocumentor documentation [http://phpdoc.org/docs/HTMLSmartyConverter/default/phpDocumentor/tutorial_phpDocumentor.howto.pkg.html#basics.docblock].

8.2.1. DocBlocks Types

In CartoWeb we use:

- Page-level DocBlocks: one DocBlock for each PHP file.
- Class, method, class variable and function (outside a class) DocBlocks: one DocBlock for each.
- Require, include, define: if needed, one DocBlock for each or all.

8.2.2. DocBlocks Contents

- Short description: if needed, a one line description.
- Long description: if needed, a longer description.

- @package <package> (file, class): we use one package for each directory which

contains PHP files, it means there are the following packages: Client, Server, Common, CorePlugins, Plugins, Scripts, Tests.

- @author <author> (file): author with email address.
- @version $Id:$ (file): always '$Id:$', content automatically set by CVS.
- @param <type> [<description>] (method): type mandatory, description if needed.
- @return <type> [<description>] (method): type mandatory, description if needed.
- @var <type> [<description>] (variable): type mandatory, description if needed.
- {@link [<class>|<method>]} (anywhere): to add a hyperlink to a class or method.
- @see [<class>|<method>] (anywhere): to add a reference to a class or method. @see is also used for interface implementation: Because PhpDocumentor doesn't inherit tags @param, @return, etc. and because we don't want to copy/paste these tags, we add a simple @see tag to interface method definition. See example below.

8.2.3. Example

Here is a code example. Please note:

- $simpleVariable doesn't need a description, but @var tag is mandatory.
- here constructor doesn't need a description.
- getters and setters are too simple to have a description, but don't forget the @param and @return!
- use (but not abuse) of {@link} and @see. This can be really useful to navigate through documentation.

```php
<?php
/**
 * Test file
 *
 * The purpose of this file is to show an example of how to use
 * PhpDocumentor DocBlocks in CartoWeb.
 * @package MyPackage
 * @author Gustave Dupond <gustave.dupond@camptocamp.com>
 * @version $Id:$
 */

/**
 * This is a require description
 */
require_once('required_file.php');

/**
 * This is a short description of MyClass
 *
 * MyClass long descrition.
 * @package MyPackage
 */
class MyClass extends MySuperClass {
```

```
/**
 * @var int
 */
public $simpleVariable;

/**
 * @var MyVarClass
 */
public $simpleObjectVariable;

/**
 * This variable needs a description
 * @var string
 */
public $notSoSimpleVariable;

/**
 * @param int
 */
function __construct($initialValue) {
        parent::__construct();
        $this->simpleVariable = $initialValue;
        $this->simpleObjectVariable = NULL;
        $this->notSoSimpleVariable = '';
}

/**
 * @param int
 */
function setSimpleVariable($newValue) {
        $this->simpleVariable = $newValue;
}

/**
 * @return int
 */
function getSimpleVariable() {
        return $this->simpleVariable;
}

/**
 * This is a short description for method
 *
 * This is a longer description. Don't forget to have a
 * look here {@link MyLinkClass::myLinkMethod()}. blah blah.
 * @param string description of first parameter
 * @param MyParamClass description of second parameter
 * @return boolean true if everything's fine
 * @see MyInterestingClass
 */
function myMethod($myFirstParameter, $mySecondParameter) {
        // blah blah

        return true;
}

/**
 * @see MyInterface::myImplementingMethod()
 */
function myImplementingMethod($myParameter) {
        // blah blah

        return true;
}

function myOverridingMethod($myParameter) {
        // blah blah
```

```
            return true;
        }
}
?>
```

9. Logging and Debugging

9.1. Introduction

This chapter is about the logging framework used in CartoWeb, and gives some tips for debugging the application. The two concepts are somewhat related, as logging is often used as a way to ease debugging.

9.2. Logging

Logging is an important feature for being able to see what happens, debug more easily the application, and track invalid or unexpected situations.

The logging framework used for CartoWeb is Log4php [http://logging.apache.org/log4php/]. Log4php is a portage to Php of the famous Log4j Java logging library. Thus, Log4php has lots of similarities with Log4j, and users familiar with it will have no problems understanding it.

9.2.1. Log4php Activation

To activate the logging with the default settings, uncomment the line :

```
;log4php.rootLogger=DEBUG, A1
```

in `client_conf/cartoclientLogger.properties`

9.2.2. Log4php Configuration Files

Log4php settings are customizable in the `client_conf/cartoclientLogger.properties` configuration file on the CartoClient and `server_conf/cartoserverLogger.properties` on the CartoServer.

For the detailed syntax of the configuration file, see the Log4php documentation. A very short introduction is given there. The line `"log4php.rootLogger=DEBUG, A1"` can be uncommented, which will activate the loggers defined in the lines starting with `log4php.appender.NAME`, (where `NAME` is the name in the list `"DEBUG, A1"`). After the loggers are activated, the the log output will be redirected to the corresponding location. In the line `log4php.appender.A1.file="LOG_HOME/cartoclient.log"` the `LOG_HOME` variable has a special meaning: it is expanded to the `log` directory of the CartoWeb distribution.

One powerful feature of Log4php, among others, is the ability to filter log message

according to their severity. Each log message has a severity which may be ALL, DEBUG, INFO, WARN, ERROR, FATAL, OFF. As described in the configuration file comments, the lines like log4php.logger.CLASSNAME can be used to apply a filtering of the log message for the class CLASSNAME. For instance, adding a line "log4php.logger.Cartoclient = INFO", means that only log message of severity INFO or above will be printed. This is useful to avoid displaying unwanted log messages.

9.2.3. Default Log File Location

Default log messages will be written to log/cartoclient.log and the CartoServer ones to log/cartoserver.log. Of course, the Log4php configuration files can be adapted to write messages elsewhere.

9.2.4. Using Log4php in Source Files

The Log4php usage in code is quite easy.

1. For objects, it is advised to store a Logger object as an instance variable

```
class MyClass {
    /**
     * @var Logger
     */
    private $log;

    [...]

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

For non object, a reference to a Logger object can be obtained this way:

```
  $log =& LoggerManager::getLogger(__METHOD__);
```

Tip

Using __METHOD__ allows the same line to be used independently of the method where it is located.

2. On the Logger object, several methods can be used to log messages: debug(), info(), warn(), error(), fatal() They take a string as argument, which is the message to log. Example:

```
$this->log->debug('My Message'); // Inside objects
$log->warn('My Message');        // Outside objects
```

9.3. Debugging

Debugging is a large topic. Everyone has its preference over the tool to be used like using an integrated debugging tool inside an IDE, using print statements, code printing and reading, ... Because of this, this section does not tells what tools to use, but rather gives some tips when debugging.

9.3.1. Understanding Exceptions and Stack Traces

When a failure is encountered in CartoWeb the Php5 mechanism for exceptions handling is used to manage exception and display stack traces. People knowing the Java™ language will be familiar with such stack traces. The following example shows such a stack trace display. It is easily understood as the list of functions called, and the line numbers where the call happened.

```
Failure

class:   SoapFault
message:   Error [8, Undefined property:  ServerMapquery::$currentQeury,
        /var/www/cartoweb3/coreplugins/mapquery/server/ServerMapquery.php, 222]
Backtrace:

file: 182 - /var/www/cartoweb3/common/Common.php
call: Common::cartowebErrorHandler()

file: 222 - /var/www/cartoweb3/coreplugins/mapquery/server/ServerMapquery.php
call: Common::cartowebErrorHandler()

file: 222 - /var/www/cartoweb3/coreplugins/mapquery/server/ServerMapquery.php
call: ServerMapquery::queryByBbox()

file: 248 - /var/www/cartoweb3/coreplugins/query/server/ServerQuery.php
call: ServerMapquery->queryByBbox(8, "Undefined property:
     ServerMapquery::$currentQeury",
     "/var/www/cartoweb3/coreplugins/mapquery/server...", 222, Array(2))

file: 369 - /var/www/cartoweb3/coreplugins/query/server/ServerQuery.php
call: ServerQuery->queryLayer("polygon", Object(Bbox))

file: 58 - /var/www/cartoweb3/server/ServerPluginHelper.php
call: ServerQuery->handlePreDrawing(Object(Bbox), Object(QuerySelection))

file: 96 - /var/www/cartoweb3/server/ServerPluginHelper.php
call: ClientResponderHelper->callHandleFunction(Object(QueryRequest))
```

9.3.2. Using Direct for More Verbosity

In some situations, a fatal error on the server will display a message with not much verbosity:

```
Failure

class: SoapFault
message: parse error, unexpected T_VARIABLE
```

The fact no line number and Php file is displayed is a limitation of the Php SOAP implementation (workarounds are welcomed ;-)).

In such a situation, a solution for this problem is to enable the CartoWeb direct access mode of operation. Direct access mode is set with the `cartoserverDirectAccess` parameter of the `client_conf/client.ini` configuration file. For more details about this parameter, see Section 4.2, " client.ini ".

10. Performance Tests

10.1. Main Parameters

This is a non-exhaustive list of interesting parameters for CartoWeb performance tests. You may want to vary these parameters values when testing performance before and after development of new functionalities.

- Cartoweb configuration
    - Local SOAP, distant SOAP or direct mode
    - Data on PC or through NFS
    - Map size
    - Number of active layers
    - Dynamic legends or not
- Logs and cache
    - MapInfo cached or not
    - MapResult cached or not
    - SOAP XML cached or not
    - Logs activated or not
- Map data
    - Number of layers (10, 50, 250)

10.2. Executing Tests

This section describes performance tests execution using APD, a debugging/profiling tool available as a Zend PHP module.

10.2.1. APD Module Installation

First thing to do is to install APD's PHP Zend module. You can download archive here [http://pecl.php.net/package/apd].

Follow instructions to compile APD. Then load the module by adding the following two lines in php.ini:

```
zend_extension = <php_home>/lib/php/extensions/no-debug-non-zts-20040412/apd.so
apd.dumpdir = /tmp/apd
```

On a win32 installation:

```
zend_extension_debug_ts = <php_lib_home>\apd.dll
apd.dumpdir = c:\apd\traces
```

Path to `apd.so` may vary. See also README file in APD archive.

You may now activate tracing by adding an empty file trace.apd in directories
`<cartoweb_home>/client` and `<cartoweb_home>/server`:

```
touch trace.apd
```

When using Cartoweb in direct mode, only one trace file will be generated. When
using Cartoweb in SOAP mode, two trace files will be generated, one for client and
one for server. These files can be found in directory set in `apd.dumpdir` variable
(`php.ini`, see above).

10.2.2. Simple Execution Times

To get global execution times, use script `cwprof.php`:

- First usage: execute script on each trace file. This could be useful to re-parse an
  old trace file. If you have separated trace files for client and server, you will need
  to execute the script twice.

  ```
  cd <cartoweb_home>/scripts
  php cwprof.php <trace_file>
  ```

- Second usage: execute script on a directory. The script will parse the most recent
  trace file. The `-local` option is used when client and server trace files are located
  in same directory. In this case, the two most recent trace files are parsed and
  results for client and server are merged.

  ```
  cd <cartoweb_home>/scripts
  php cwprof.php [-local] <trace_directory>
  ```

Script output will look like this (times in milliseconds):

```
Exec client       = 451
Exec server total = 707
Exec MS obj       = 472
Exec MS other     = 85
Exec total        = 1524
```

- Exec client: time elapsed on client. Will be empty if script is executed on a
  server-only trace file

- Exec server total: time elapsed on server. It includes `Exec MS obj` and `Exec MS
  other` times. Will be empty if script is executed on a client-only trace file, or if

246

direct mode is on

- Exec MS obj: time elapsed while creating Mapserver main object. It includes reading the mapfile. Will be empty is script is executed on a client-only trace file

- Exec MS other: time elapsed in other Mapserver tasks. Will be empty is script is executed on a client-only trace file

- Exec total: time elapsed in total. If direct mode is off, it also includes time elapsed in SOAP data transmission

10.2.3. Graphical Interface (Unix-like)

To have more information about execution times and calls stack, you can use a powerful graphical viewer called KCachegrind. This tool is available on Unix-like environments only. On Win32, it can be used via KDE on CygWin.

KCachegrind is included in KDE (package kdesdk). To install it on a Debian distribution, type:

```
apt-get install kcachegrind
```

APD package includes a script called `pprof2calltree` that can translate a trace generated by APD to a file in KCachegrind format. To translate a pprof file, type:

```
./pprof2calltree -f <pprof_file> >/dev/null
```

Redirecting to `/dev/null` is needed because script generates a large number of PHP notices. Then you can open the resulting file in KCachegrind.

11. Upgrading Views Data

11.1. Introduction

Views saves given CartoWeb states by recording the plugins session data. Only client-side plugins implementing `Sessionable` interface may be saved in views.

Because CartoWeb and its plugins evolve, views might become outdated after some time. Plugin session containers may change: properties may be added, removed, renamed, etc. and thus shifts between old views and new plugin session formats appear. Those shifts then prevent some views from correctly displaying or even make CartoWeb crash.

However CartoWeb offers upgrade tools to read outdated views.

11.2. Upgrade Tools

11.2.1. Views Versions

Plugins session data are saved in views in separated XML elements. Each element records the plugin session container version. Plugin session container version must be incremented whenever a change is applied on the container format (new property, name change, etc.). Container version, if not specified, defaults to 1. To explicitely update it, add following constant definition before container declaration:

```
define('PLUGINNAME_SESSION_VERSION', 3);
```

where *PLUGINNAME* is the uppercased plugin name. Container versions are always integers.

11.2.2. Upgrade Filters

Upgrading a view is done by sequentially applying filters, each one updating it to the following version (from version N to version N+1). If matching sequence cannot be detected (missing filters), upgrade is aborted and outdated data is discarded. Current plugin data is then used instead.

Upgrade filters are PHP classes extending *ViewUpgrader*, the latter class being defined at the end of `client/Views.php`. They are saved in a `ViewsUpgrade.php` file, located in the same directory than the plugin client-side class file (for instance `coreplugins/location/client/`). Filters naming convention is *<UppercasedPluginName>V<initialVersion>ToV<initialVersion+1>*.

Some basic methods (rename, remove, add) are available in every filters but you may

define your own transformers methods. Details about basic methods are given in
`ViewUpgrader` class documentation (CartoWeb API manual). At least an upgrade
filter class must redefine the `callFilters()` method that indicates the sequence of
transformations to apply.

An example of filters might be:

```php
<?php
/**
 * coreplugins/location/client/ViewsUpgrade.php
 */

class test {
var $game = 'arkanoid';
var $i = 12;
}

/**
 * Upgrades from V1 to V2
 */
class LocationV1ToV2 extends ViewUpgrader {
    protected function callFilters() {
        $this->add('test', 'toto');
    }
}

/**
 * Upgrades from V2 to V3
 */
class LocationV2ToV3 extends ViewUpgrader {
    protected function callFilters() {
        $this->rename('test', 'foo');
        $this->add('bar', new test);
        $this->add('someProperty', 2);
        $this->add('caramba', 'ole');
        $this->remove('someProperty');
    }
}
?>
```

Warning

It is the responsibility of each developer to provide adapted filters when he/she
updates plugin session containers!

Warning

Upgrade filters do not support non object-typed plugin session containers.

11.2.3. Upgrade Configuration

See Section 13.2.2, "Main Views Controller". Only two of these parameters are
related to views upgrading. `viewLogErrors` enables to log outdated views loadings
whereas `viewUpgradeOutdated` is useful to activate or not the upgrading device (if

deactivated, outdated views parts are discarded).

11.3. Customizing Views Processing Using Project Hooks

CartoWeb offers the ability to add some project-specific controls ("hooks") within the views processing.

For now only one hook is available, right before the data of the loaded view is merged with the current session data. See method *ViewManager::handleView()* in `client/Views.php`.

To create hooks, write a PHP class extending the basic *ViewHooks* class defined in `client/Views.php`. The extended class must be saved in a file with the same name, located in your project `plugins/views/client/` directory. If the latter directory does not exist, simply create it. The standard "views" plugin is not required to have hooks working. Finally, set the *viewHooksClassName* with your extended class name in `client.ini`.

```ini
; configuration in client.ini
viewOn = true
viewStorage = db
viewablePlugins = location, layers, query, outline
viewMetas = author
...
viewHooksClassName = FoobarViewHooks
```

```php
<?php
/**
 * projects/<your_project>/plugins/views/client/FoobarViewHooks.php
 */
class FoobarViewHooks extends ViewHooks {

    /**
     * @see ViewHooks::handlePreLoading()
     */
    public function handlePreLoading($data, $viewId) {
        // Your customized code...
    }
}
?>
```

12. AJAX Implementation

12.1. Introduction

CartoWeb implements an AJAX layer, enabling asynchronous update of the HTML GUI.

12.2. AJAX Implementation Outline

The implementation of AJAX in CartoWeb is achieved by extending the Cartoclient and creating a lightweight Javascript engine. It introduces a few new concepts:

- Synchronous and asynchronous modes
- Ajaxable interface
- Plugins Javascript side
- Plugins actions
- Plugins enablement level

12.2.1. Synchronous And Asynchronous Modes

In addition to the implicit synchronous mode (traditional until-now Cartoclient execution flow), an asynchronous mode has been implemented to process AJAX calls.

- The synchronous mode is the default Cartoclient execution flow, that processes each plugin's logic and renders a monolithic HTML page. Non-AJAX projects will run the synchronous mode without any alteration.
- The asynchronous mode is the execution flow specifically built to process AJAX calls. In this mode, a subset of the plugins features is processed and an XML

document, containing plugins output, is rendered - to be processed by the Javascript part.

12.2.2. Ajaxable Interface

Client plugins can now implement an interface called Ajaxable that enables a plugin to handle an asynchronous request and render XML.

See CartoWeb 3.3 API documentation, interface Ajaxable [http://www.CartoWeb.org/doc/cw3.3/apidoc/Client/Ajaxable.html].

12.2.3. Plugins Javascript Side

Each Ajaxable plugin is made of a Javascript part that handles:

• HTTP POST/GET parameters generation for AJAX requests
• GUI refresh logic: processing of the plugin XML response and DOM injection/manipulation.

12.2.4. Plugins Actions

AJAX introduces the concept of actions. An action is provided by a plugin and is used to define:

• From the Javascript side, what are the HTTP parameters (GET & POST) to be sent to the Cartoclient
• From the Caroclient side, what subset of each plugin's logic is to be processed

In example, the Location coreplugin provides the following actions:

• Location.Pan
• Location.Zoom
• Location.FullExtent
• Location.Recenter

12.2.5. Plugins Enablement Level

When the Cartoclient's asynchronous mode is triggered, the action is used to determine what subset of each plugin's logic should be processed. Processing a subset of a plugin's logic is achieved by changing it's *enablement level*. Enablement levels are by default set to *Full* (i.e. the whole logic is processed).

Here are the yet available enablement levels:

- *Load*: The plugin is loaded
- *Process*: The plugin is loaded and handles GET/POST Requests
- *ServerCall*: The plugin handles GET/POST requests, builds/overrides server request and handles server result
- *Full*: The plugin's whole logic is executed

Note that all enablement levels perform a Sessionable::saveSession().

12.3. Making Your Plugin Ajaxable

An ajaxable plugin has the ability to refresh the CartoWeb GUI asynchronously, if the project using it enables the AJAX mode (see how to enable AJAX in a project: Section 20.2, "Make Your Project AJAX Enabled"

Please have a look at the wiki for an up-to-date howto explaining how to make a plugin ajaxable: http://www.cartoweb.org/cwiki/HowToAjaxablePlugin

Appendix A. Mapserver Debian Installation

See CartoWeb wiki at http://cartoweb.org/cwiki/DebianPackages.

Appendix B. Apache & Mapserver Windows Manual Installation

This page describes the step-by-step procedure to install Cartoweb3 on Windows, using WAMP, Mapscript and Cartoweb.

The following configuration was tested:

- Windows 2000
- WAMP 1.4.4 (Apache 1.3.33 + PHP 5.0.4)
- Mapserver 4.5
- Cartoweb 3.0.0 and Cartoweb3 CVS-version

B.1. Apache/PHP5 Install

Cartoweb requires php5. MS4W only supports php4.3 so we will need WAMP (1.4.4 or greater). You can get the installation package here: http://www.wampserver.com/download.php

Launch the setup by clicking on the executable file you just downloaded (here `wamp5_1.4.4.exe`). Keep default install options:

- Install WAMP in `C:\wamp`
- Set www root folder as `C:\wamp\www`

Once installation is completed:

- If WAMP didn't loaded automatically (icon in the system tray in the lower right of the screen), load it (Stat menu, Programs, WampServer, Start Wampserver)
- Type http://localhost in a browser
- Once the page has loaded, click on Tools > phpinfo() (http://localhost/exemples/phpinfo.php) to display the complete PHPINFO information.
- Take note of the path to the `php.ini` file (Configuration File (php.ini) Path). It should look like `C:\wamp\apache\php.ini`
- Open the file `php.ini` in any text editor and go to the Dynamic Extensions section. Activate the SOAP, Curl and Gettext extensions. Simply remove the ";" at the beginning of the lines.

If you can not see the extension in the list, you will need to add it manually:

```
extension=php_soap.dll
extension=php_curl.dll
extension=php_gettext.dll
```

The corresponding DLL files must be present in the extensions folder of PHP. It should be `C:\wamp\php\ext` by default. If you can not find it, look in the PHPINFO ("local value" for parameter extension_dir, visible in `php.ini` as well), you will notice something like `C:/wamp/php/ext/`.

- Restart Apache (left-click on the WAMP icon in the systray > Apache > Restart Service)
- Check in the PHPINFO if the new extensions loaded correctly (look for a section with their name).

B.2. Mapserver/Mapscript Install

You need a Mapscript version compiled for PHP5/Win32. You can get one here: http://www.maptools.org/dl/mapserver-4.5-win32-php5.0.3.zip

If the link doesn't work, go to http://dl.maptools.org/dl/and look for mapserver-4.5-win32-php5.0.3.zip or newer.

Note

MapServer for Windows/PHP5 archives are also available on CartoWeb website at http://cartoweb.org/downloads.html#msw. Additional instructions for installing DLL's are given in the matching README.txt files.

- Once you have downloaded the archive, uncompress it in any directory.
- The README.txt included in the archive tells you where you must place the various files:
    ○ Unzip gdal-1.2.5.zip, libcurl-7.10.7_dll.zip, xerces_dll.zip, ECW_DLL.zip, pdfdll.zip, libpq.zip in `C:\WINNT\System32` if you are using Windows 2000/NT (`C:\Windows\System32` for XP and `C:\Windows\System` for 95/98/Me)
    ○ Move `php_mapscript_45.dll` in the extensions folder of PHP (see PHPINFO, default is `C:\wamp\php\ext`). The file `php_proj.dll` is outdated, you can ignore it.
    ○ Open `php.ini` and add a call to this extension:

```
extension=php_mapscript_45.dll
```

- Restart Apache and check that the mapscript module is loaded correctly (look in PHPINFO)

- Install Proj4 by following the instructions in URL indicated in the README.txt: http://mapserver.gis.umn.edu/cgi-bin/wiki.pl?WindowProjHowTo > Using Prebuilt Binaries

    ○ Download the archive Proj4 here: ftp://ftp.remotesensing.org/proj/ proj446_win32_bin.zip

    ○ Uncompress it in C:\. If you want to install it elsewhere, look at point 2 of Using Prebuilt Binaries

    ○ Point 3 in the README is irrelevant in our case.

- CartoWeb setup script `cw3setup.php` requires an "unzip" program to uncompress libraries and demo data archives. If you don't have one installed, download one on the Web (type "unzip.exe" in your favorite search engine) and put it for instance in your `C:\Windows` directory.

Note

The .exe in the Mapserver archive are not needed.

Note

If you are using Windows XP, it may be necessary to modify the extension_dir value in php.ini by using "\" instead of "/"

Appendix C. Create a New Project

This appendix describes how to configure your own project. The main steps are detailed here.

C.1. Project Directory

Get in the `projects` folder of Cartoweb. Rename the `sampleProject` directory to the name of your project.

Edit the `client.ini.in` of this new project, and modify the mapId parameter value with the name of your project:

```
mapId = your_project_name
```

Open the `server_conf` directory and rename the `sampleProject` directory to the name of your project.

In this folder, two more files need to be renamed (`sampleProject.map` and `sampleProject.ini`) with the name of the new project.

C.2. Setup Your Project

In a command line prompt (DOS, shell), launch the `cw3setup.php` setup script.

```
<PHP-INTERPRETER> cw3setup.php --install
    --base-url http://localhost/cartoweb3/htdocs/ --profile development
    --project <nameOfYourProject>
```

See Chapter 1, *Installation* for more info on the installation script.

If no error occured, you should be able to access your project by typing http://localhost/cartoweb3/htdocs/client.php in your browser and selecting your project from the projects drop-down menu.

C.3. Project Shortcut

To make development easier, you can create a shortcut file to access your project directly. Go in `cartoweb3/htdocs/` and copy the file `demoCW3.php`.

Rename the file to any name (usually your project name). Edit the file and replace *'demoCW3'* by *'your_project_name'*. Your project is now directly available with the URL http://localhost/cartoweb3/htdocs/your_project_name.php. (the URL given above depends on your installation, see the previous comments about

cartoclientBaseUrl and --base-url).

C.4. Loading Data

As you can see, the project you created is hopelessly empty. Then, we'll load spatial data in it.

Add the following in the mapfile (`your_project_name.map`) :

```
LAYER
  NAME "region"
  TYPE POLYGON
  DATA "reg_france"
  TEMPLATE "ttt"
  CLASS
    NAME "region"
    STYLE
      COLOR 240 240 240
      OUTLINECOLOR 255 165 96
    END
  END
  METADATA
    "id_attribute_string" "CODE"
  END
END
```

And in the `layers.ini` file (`server_conf/your_project_name/`) add the following :

```
layers.region.className = Layer
layers.region.label = Régions
layers.region.msLayer = region
```

And add the new layer id in the root layer children list :

```
layers.root.children = region
```

For more details on how to fill the layers.ini file, see Chapter 6, *Layers*.

Also edit the `your_project_name.ini` file in a text editor in order to set some initial mapstates such as default selected layers.

Add the following :

```
mapInfo.initialMapStates.default.layers.region.selected = true
```

See Section 4.3.3.3, "Initial Mapstates" for more info about initialMapStates.

In your browser, click on the *reset_session* button and you should now see the France regions.

C.5. Location Parameters

Move the location.ini file you will find in the sampleFiles folder into the `server_conf/your_project_name/`one.

This file defines some parameters such as scales or shortcuts. For more details on how to configure those parameters, see Chapter 7, *Navigation*

Click on the reset_session button in your browser page. New elements should appear. You should be able to choose a scale and a shortcut.

C.6. Your Own Data

After that the next point is to add your own data. You should just have to :

- get your data available, by default in the data folder or somewhere else on your filesystem,
- modify the mapfile extent,
- modify the extent defined in the initialMapstates (`your_project_name.ini`),
- add layers definitions in the mapfile (`your_project_name.map`),
- add layers definitions in the `layers.ini`.
- reset the session in the browser, and get your data displayed.

C.7. Outline

Let's see how to load a plugin. Edit the `client.ini.in` file (`client_conf/`) and add the outline plugin in the list of plugins to load:

```
loadPlugins = outline
```

Do the same in the `your_project_name.ini` file on server-side (`server_conf/your_project_name/`) but also add the mapOverLay plugin :

```
mapInfo.loadPlugins = outline, mapOverlay
```

Move the `outline.ini` file from `sampleFiles` to `server_conf/your_project_name/`.

In the mapfile (`your_project_name.map`), add the following at the bottom of the layers definition section :

```
  LAYER
    NAME "cartoweb_point_outline"
    TYPE POINT
    CLASS
```

```
      STYLE
        SYMBOL "circle"
        COLOR 0 0 205
        SIZE 10
      END
      LABEL
        TYPE TRUETYPE
        FONT "Vera"
        SIZE 7
        COLOR 0 0 0
        OUTLINECOLOR 255 255 255
        POSITION lc
      END
    END
END

LAYER
  NAME "cartoweb_line_outline"
  TYPE LINE
  TRANSPARENCY 100
  CLASS
    STYLE
      OUTLINECOLOR 255 0 0
      SYMBOL "line-dashed"
      SIZE 3
    END
    LABEL
      TYPE TRUETYPE
      FONT "Vera"
      SIZE 7
      COLOR 0 0 0
      OUTLINECOLOR 255 255 255
      ANGLE auto
      POSITION uc
    END
  END
END

LAYER
  NAME "cartoweb_polygon_outline"
  TYPE POLYGON
  TRANSPARENCY 60
  CLASS
    STYLE
      COLOR 255 153 0
      OUTLINECOLOR 0 0 0
    END
    LABEL
      TYPE TRUETYPE
      FONT "Vera"
      SIZE 7
      OUTLINECOLOR 255 255 255
      COLOR 0 0 0
      POSITION cc
    END
  END
END
```

Launch the setup script. This step is compulsary to convert `.ini.in` files into `.ini` configuration files read by Cartoweb.

By clicking on the *reset_session* button in the browser window, you should see new buttons in the toolbar and the corresponding folder in the leftbar.

C.8. Pdf Printing

As done for the outline plugin, add *"exportPdf"* to the list of plugins to load on both client and server-side.

After launching the setup script and refreshing the application (reset_session), you should be able to test the new functionality as a new folder appeared in the interface. But a warning printed in the pdf file advices you to edit your `exportPdf.ini` file.

So move `exportPdf.ini` from `sampleFiles` to `client_conf`. Take care that this is a client-side configuration file. Try a new print (no need to reset).

For more information on how to edit your `exportPdf.ini` file, see Chapter 12, *PDF Export*.

C.9. Templating and Layout

Create a new `templates` folder in your project directory and move the `cartoclient.tpl` file from `sampleFiles` to this new folder.

This file is the main template for CartoWeb.

Refresh the application in your browser (refresh button) and enjoy!

To customize your project even more, you may apply styles sheets. To do so, first create a `htdocs` folder in your project directory. In this folder, create a new folder named `css`. Then move `cartoweb.css` and `folders.css` from `sampleFiles` to this new folder.

At this step, you need to launch the setup script so that the new ressources you just created are copied or linked (depending on your OS) in a web reachable folder.

You can now refresh the browser and see the new colors and styles.

Appendix D. DocBook Documentation Generation

CartoWeb documentation source are in DocBook XML [http://www.docbook.org] format and located in `documentation/user_manual/source` in CartoWeb archive.

You may want to generate it by yourself in order to produce XHTML or PDF output.

D.1. Documentation Generation on UNIX-like System

D.1.1. Tiny DocBook Install

Uncompress the tiny-docbook-1-6-17.tar.bz2 [http://www.cartoweb.org/downloads/ docbook/tiny-docbook-1-6-16.tar.bz2] archive somewhere in your filesystem.

Execute:

```
$ ./configure --enable-install
$ make install
```

to install and configure Tiny DocBook environment.

D.1.2. Create a Symbolic Link to Documentation Source

Create a symbolic link in the Tiny DocBook install directory pointing to the CartoWeb documentation sources `<cartoweb_home>/documentation/ user_manual/source`, like that:

```
$ ln -s CARTOWEB_HOME/documentation/user_manual/source
```

D.1.3. XHTML Generation

Then, to generate an XHTML version of the documentation, execute as follow:

```
$ ./configure
$ make xhtml
```

Output result will be generated in `xhtml/` directory.

D.1.4. PDF Generation

To generate a pdf version of the documentation, execute instead:

```
$ ./configure
$ make pdf
```

Resulting pdf will be generated in `book.pdf` file.

Note

You need to have at least a JRE (or JDK) installed on your system in order to generate PDF . The environment variable JAVA_HOME must also be rightly set, e.g

```
$ export JAVA_HOME=dir_path_where_jre_is_put_on_your_system
```

D.2. Documentation Generation on Windows using Cygwin

At this time, the only known way to generate xhml or pdf version of the documentation on Windows is by using Cygwin.

You will need to install the following Cygwin packages: *libxml2, libxslt, make* .

Then follow the same instructions as for UNIX-like system Section D.1, " Documentation Generation on UNIX-like System ".

Index

A

Accounting, 123
AJAX, 127
all, 112
anonymous, 112
applySecurity, 113
areaFactor, 69
attributes, 131
authActive, 110
autoClassLegend, 50

B

basedn, 111
bbox (initial), 58
block positioning, PDF, 93
Blocks configuration, PDF, 86, 92
bottomLayers, 55

C

charsetUtf8, CSV, 77
collapsibleKeymap, 59
Colors, PDF, 89
Column reorder, 213
Coreplugins, 193
crosshairSymbol, 57
crosshairSymbolColor, 57
crosshairSymbolSize, 57
CSV export plugin, 76

D

dbSecurityDsn, 111
dbSecurityQueryRoles, 111
dbSecurityQueryUser, 111
dbTableName, 131
defaultAttributes, 63
defaultHilight, 62
defaultMaskmode, 62
defaultPolicy, 62

defaultTable, 63
displayExtendedSelection, 62
drawQueryUsingHilight, 63
dsn, 131
DXF export, 77

E

editDisplayAction, 106
editLayers, 106
editResultNbCol, 106
edit_attributes, 107
edit_geometry_column, 106
edit_geometry_type, 106
edit_srid, 107
edit_table, 106
enableTransparency, 55
Export plugins, 76, 203
exportDxfContainerName, DXF, 78
extent, 58

F

filename, CSV, 77
fileName, DXF, 78
Filters, 205
force_imagetype , 61
Formats configuration, PDF, 85
formats object, PDF, 91
freeScaleActive, 56

G

General configuration, PDF, 82
general object, PDF, 90
Geostatistics, 129
groupattr, 111
groupdn, 111
groupfilter, 111
groupscope, 111

H

Hilight, 64
hilight_color, 65
hilight_createlayer, 65

Z