# CartoWeb Documentation

## 3.0.0 Edition


**Published 2005-04-06**

# 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 :

- Sylvain Pasche
- Yves Bolognini
- Alexandre Saunier
- Pierre Giraud
- Daniel Faivre
- Alexandre Fellay
- Olivier Courtin
- Arnaud Saint Leger
- Claude Philipona
- David Jonglez
- Marc Fournier

# 2. Cartographic Functionalities

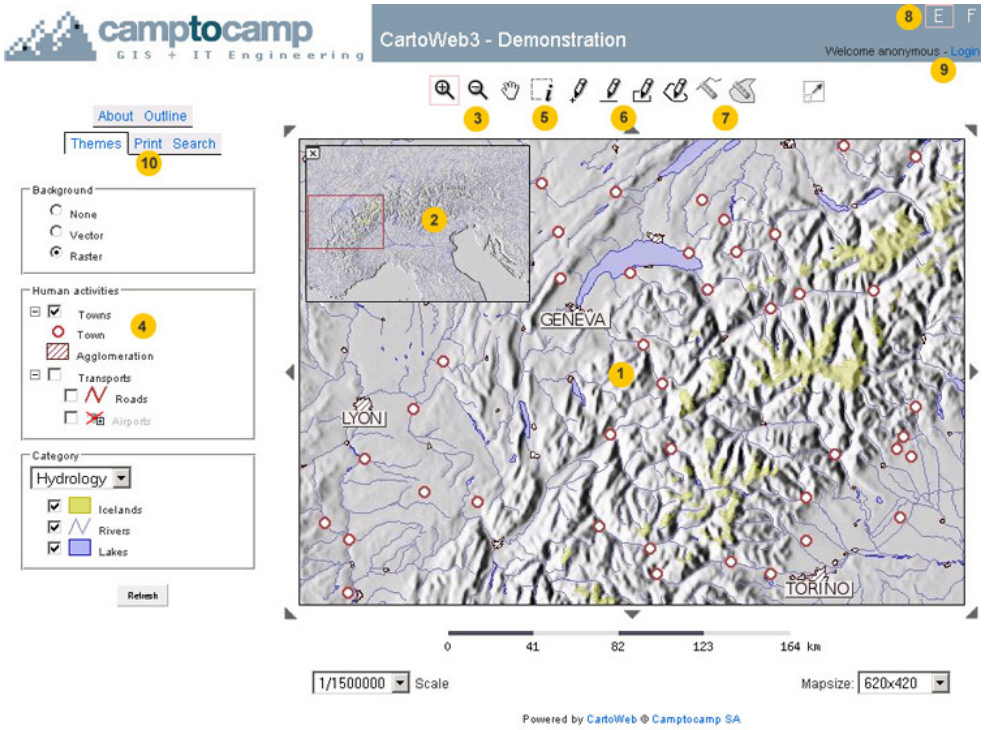
## 2.1. Introduction

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

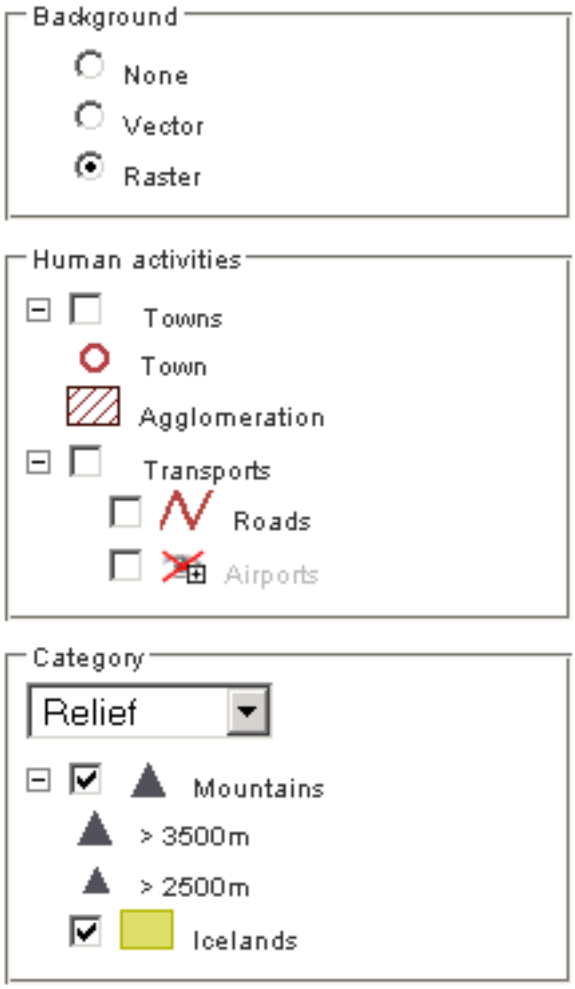
Contrary to Mapserver itself, CartoWeb supports an arbitrarily complex hierarchy of layers, with infinite depth.

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.

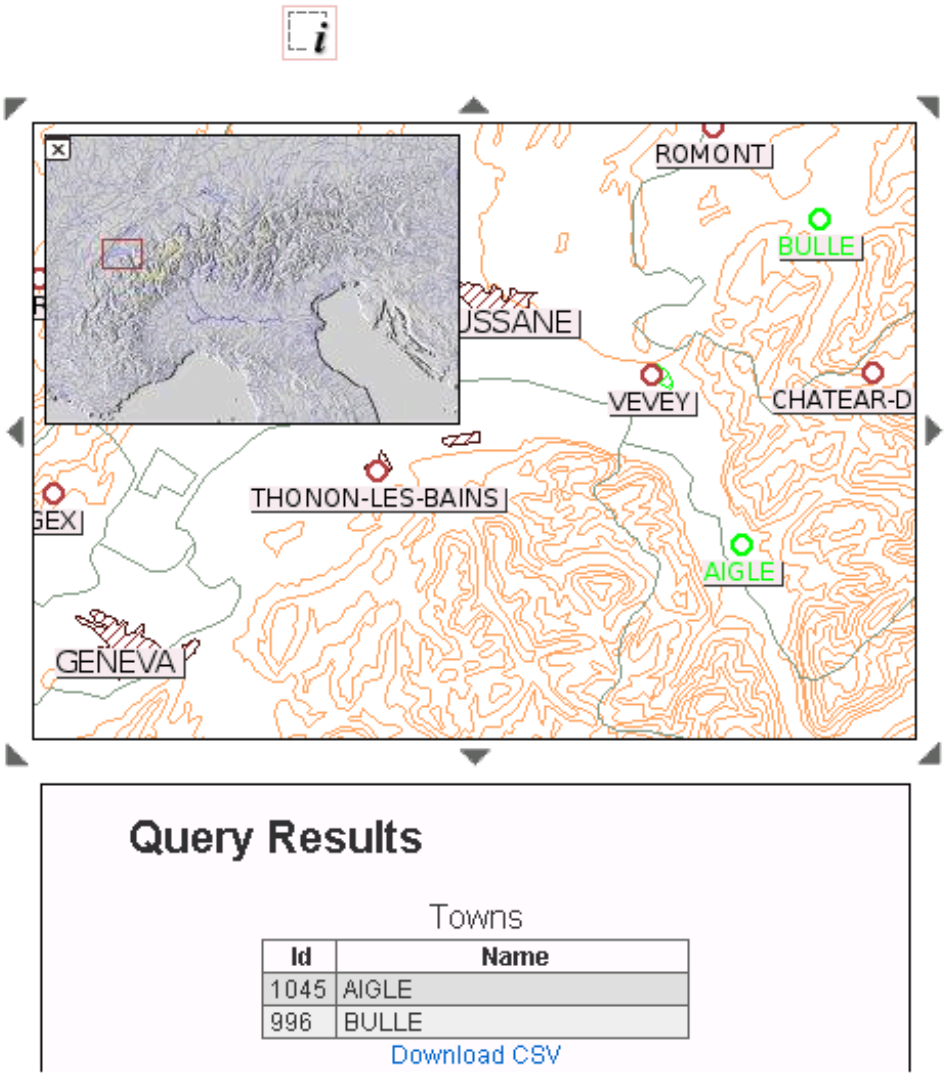



The icons for the classes are automatically drawn, and the out-of-scale layers are grayed out.

## 2.5. Map Queries

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

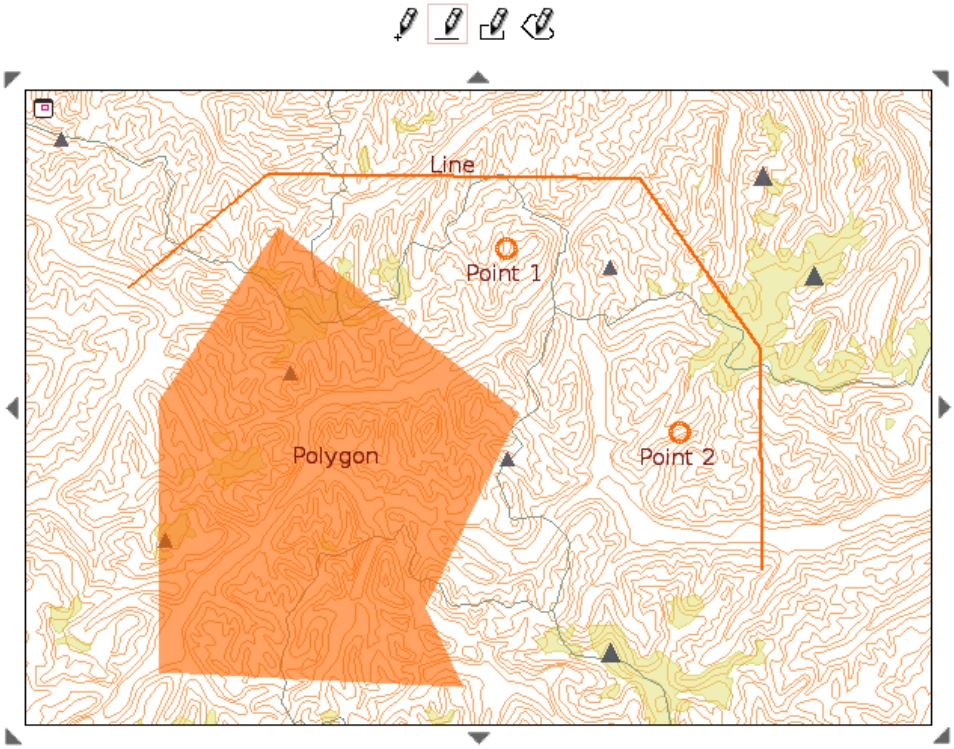
CartoWeb adds many functionalities to the raw queries supported by Mapserver. In particular, the queries may be persistent (i.e. you can add new objects to already selected objects), and the hilighting can be defined on a layer by layer basis.




## 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.

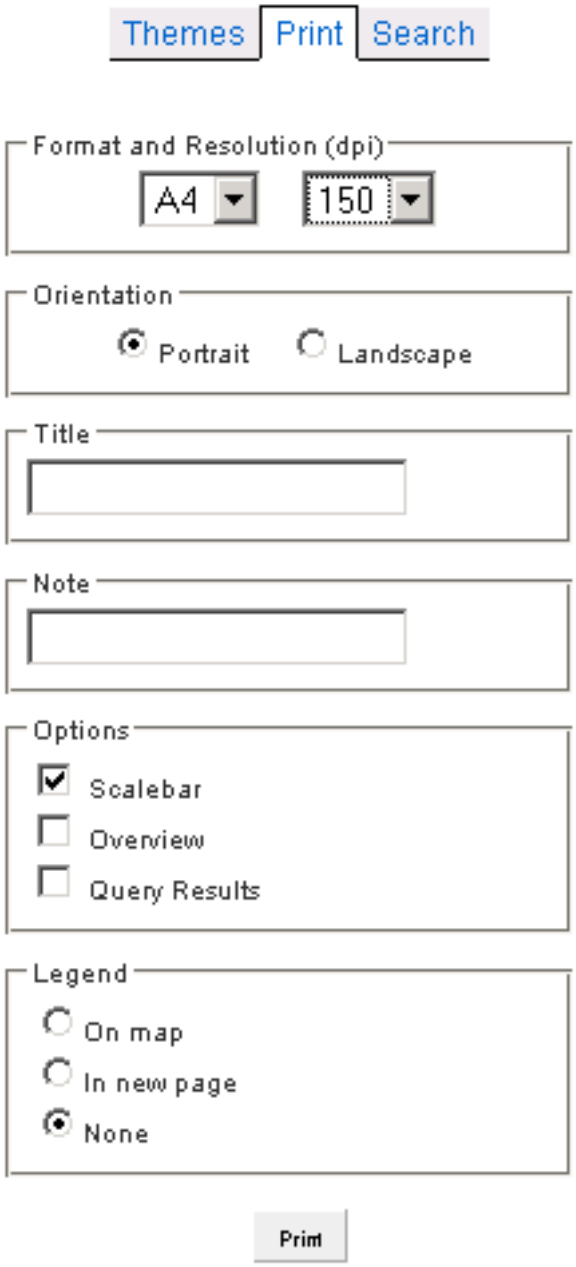
## 2.9. Access Rights

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.

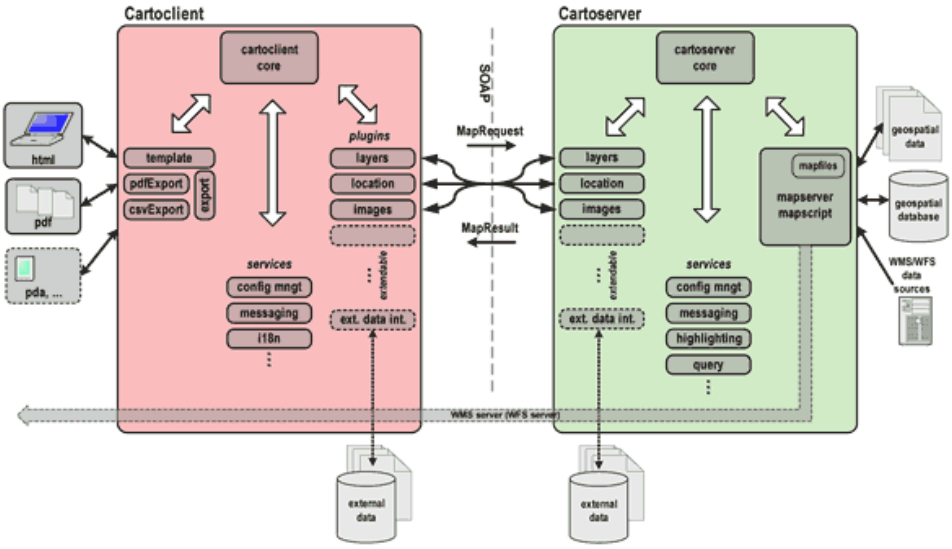
MapServer must be installed prior to any CartoWeb setup.

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

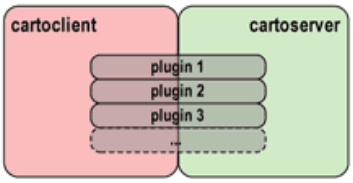
## 3.3. Web-Service Architecture - SOAP

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”).

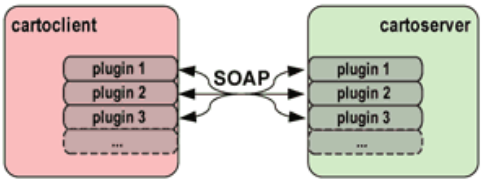



CartoWeb offers two "access" modes :

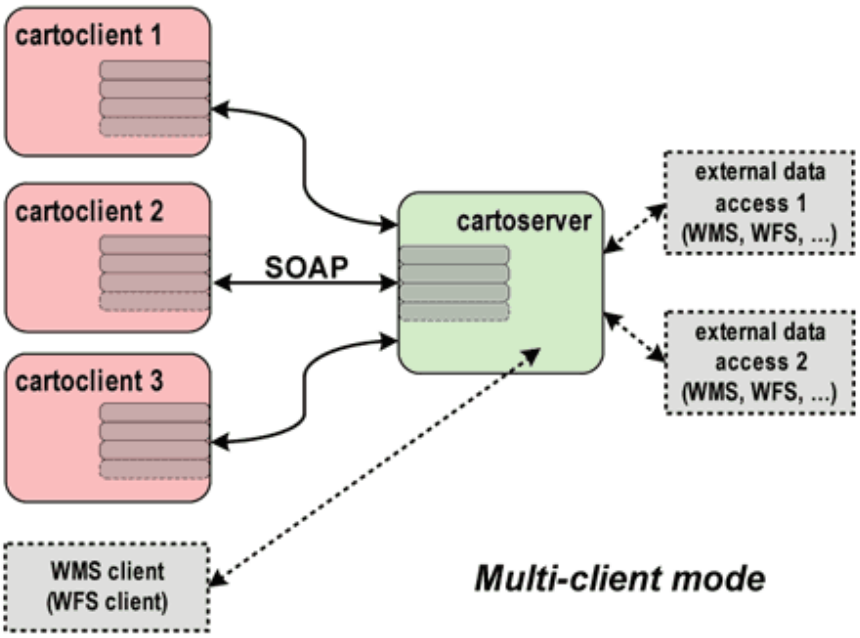
- as a web-service. CartoClient and CartoServer then interact using remote procedures based upon SOAP [http://www.w3.org/TR/soap12/].
- as a standalone application. Procedures are then performed directly between CartoWeb components, bypassing the SOAP calls.



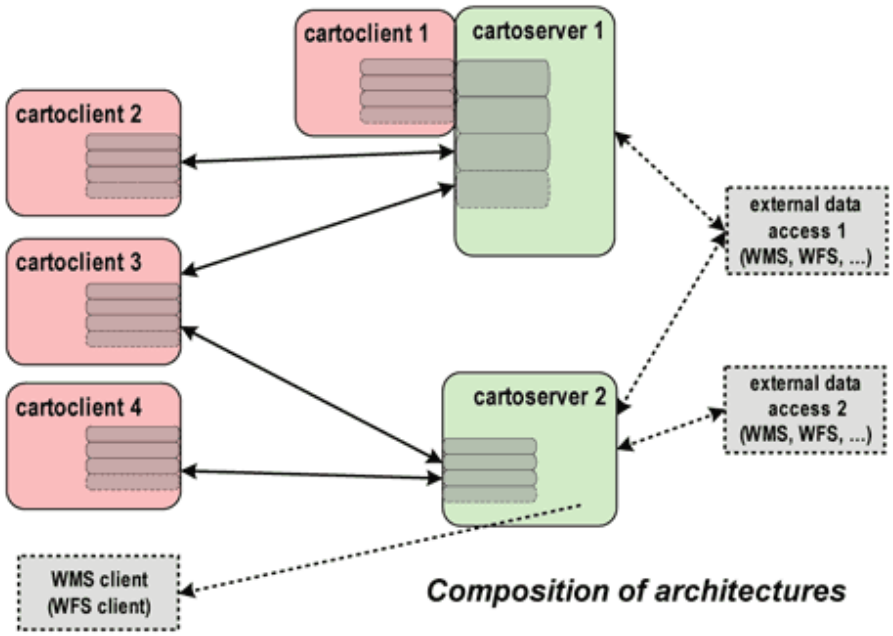

*Direct mode*




*SOAP mode*




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

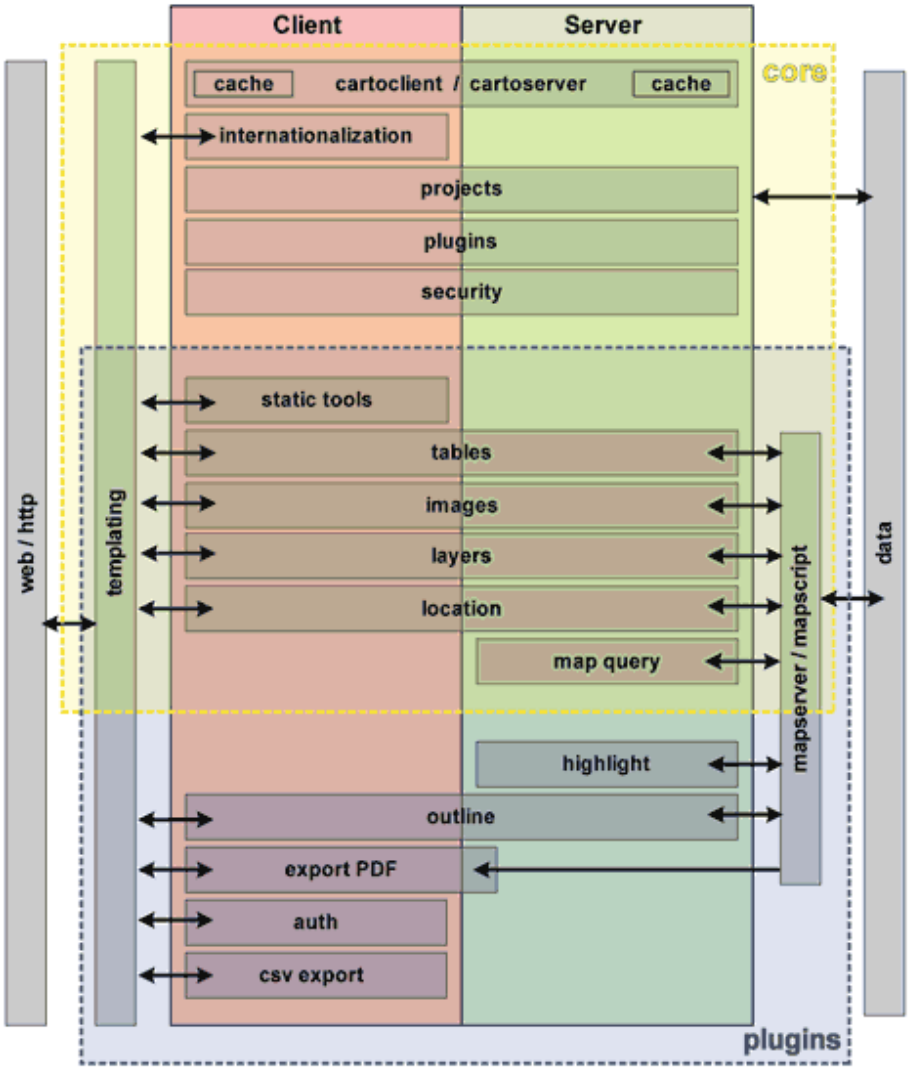



## 3.4. Modularity: Projects and Plugins

### *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, exportations 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 15, *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 13, *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. Prerequisite for Installing CartoWeb

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

### Required software

| | |
|---|---|
| PHP >= 5.0.3[1] | See http://www.php.net for more informations. You will need to have some features available in PHP<br>• SOAP: This is required by CartoWeb as it uses SOAP webservices.<br>• Gettext (optional): You need the Gettext module if you want to enable the Internationalisation in CartoWeb. See Chapter 14, *Internationalization* for configuration.<br>**Note**<br>If you are using the demo, you need to have Gettext support installed, as it uses Gettext by default.<br>• Curl (optional): Some scripts (like i18n) need Curl to fetch files remotely. |
| Mapserver PHP MapScript (from MapServer >= 4.4) | See http://www.maptools.org/php_mapscript/for more information and installation instructions. |
| PostgreSQL with PostGIS supppport (optional) | If you want spatial database support in CartoWeb you will need to install |

[1] This implies that you have a Web Server configured to execute PHP scripts, such as Apache (http://httpd.apache.org)

PostGIS of the PostgreSQL database. See http://postgis.refractions.net/for more information.

## 1.2. Installation Instructions for Version 3.0.0

For version 3.0.0, the installer is not yet completely ready. These are instructions about how to install CartoWeb with this version. These notes are temporary and will be removed once the installer is totally fonctional.

Step by step guide:

1. Download CartoWeb source from http://www.cartoweb.org/. It is recommended that you download the version with demo for a better user experience.
2. Uncompress the archive somewhere in your path accessible by your Web server.
3. 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 a section called MapScript in this page. If it is not the case, CartoWeb will not run correctly. You should install and setup PHP and PHP MapScript correctly. See Section 1.1, “Prerequisite for Installing CartoWeb”.

   If you intend to use the CartoWeb demo, you need to have a section called Gettext in the previous output.
4. CartoWeb needs some file to be writable by the Web Server such as the directory where generated images are stored. To set up these permissions, you need to launch the PHP installer with an argument. This needs to be done on the command line (launch `cmd.exe` on Windows). You also need to find the path to your PHP executable. For instance `c:\wamp\php\php.exe` on Windows, or `/var/lib/cgi-bin/php` on Unix (just example, they may differ). Once located, launch the following command:

   ```
   php cw3setup.php perms
   ```

   Replacing `php` with the path to your PHP.
5. You can now configure CartoWeb with your desired options. If you want to quickly test the demo, you need only one option to be

modifed. Open in your favorite editor the file named `client_conf/client.ini`. Find the line containing the value `cartoclientBaseUrl`. Uncomment this line (removing the first `;` character and set on the right side the URL which is used to access the `htdocs`directory inside CartoWeb. For instance, if you need to type `http://localhost/cartoweb3/htdocs/info.php` to access the PHP information page, then the value to put there will be `http://localhost/cartoweb3/htdocs/`.

Then, do the same for the option `cartoserverBaseUrl` just below.

6. (Optional) If you did not download CartoWeb with the demo included, and want to install it afterward, you can get the demo data and extract it inside the `projects/demo/server_conf/demo` directory (this will give you a `projects/demo/server_conf/demo/data` new directory with the data.

   On Unix, you can type the following in a shell:

   ```
   cd projects/demo/server_conf/demo
   wget http://www.cartoweb.org/downloads/cartoweb-demodata-3.0.0.tar.gz
   tar xvzf cartoweb-demodata-3.0.0.tar.gz
   ```

   On Windows, you can use a graphical extractor to do the same (you can use http://www.7-zip.org/).

7. You should now be able to point your browser to the `htdocs/demo.php` file, and see CartoWeb in action (if you have the demo data). If you don't have the demo, you can use the more simple test project with the `htdocs/client.php` URL.

## 1.3. UNIX-like Install[2]

**Note**

For version 3.0.0 installation, please refer to Section 1.2, “Installation Instructions for Version 3.0.0”.

[2] 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 the Debian distribution, you can easily install PHP and Mapserver. See Appendix A, *Mapserver Debian Installation*.

CartoWeb installer is "cw3setup.php", located in the root directory of the application: Simply run

```
php -f cw3setup.php
```

to setup the application.

### 1.3.1. List of Available Commands

```
(no command, by default): install cartoweb3
cvs                     : install cartoweb3 from cvs
check                   : check configuration
get[=user]              : CVS checkout with user name [anonymous]
get_libs                : get required libraries
dirs                    : create user directories
perms[=www-data]        : set writing perms for web-user [www-data]
                          - set ownership if ran as superuser,
                          - give write permission instead.
createConfig            : create the new configuration (install .dist files)
setupLinks              : link or copy paths for web browser
removeLinks             : remove all links created by setupLinks.
                          - super user rights required to remove dynamic content
setup[=path]            : setup a new project in existing installation
mkDirs                  : create all cache directories
rmDirs                  : remove all temporary files AND directories
remove                  : remove cartoweb3
```

### 1.3.2. Commands Details

- (no command, by default): install CartoWeb from a download.
- cvs: install CartoWeb from a cvs checkout. You must have cvs access configured and cvs login done with the right user name.
- check: check if the configuration meet requirements.
- get[=user]: CVS checkout with user name "user". You must have done succesfully a cvs login before.
- get_libs: fetch required libraries. You need an internet connection.
- perms[=www-data]: set writing perms for web-user [www-data]. Set the ownership if ran as superuser, and give write permission instead. Warning: the second choice is secureless.
- createConfig: create configuration from .dist files.

- setupLinks: setup symbolic links for a SymLink configuration.
- removeLinks: remove links created by setupLinks.
- setup[=path]: setup a new CartoWeb project from project sources located in path.
- mkDirs: make user-writeables directories.
- rmDirs: remove user-writeable directories.
- remove: try to completely remove CartoWeb. Super-user rights may be required for user-writed directories removal.

## 1.4. Windows Install

---

**Note**

For version 3.0.0 installation, please refer to Section 1.2, “Installation Instructions for Version 3.0.0”.

---

The process is as simple as for UNIX, but the windows installer is not available yet.

# 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 14,

*Internationalization*

- `log`: logs, mainly used for development and debug purposes.
- `plugins`: Standard, but not mandatory plugins, see Section 2.3, “Plugins”
  - `auth`: authentification plugin, see Chapter 13, *Security Configuration*
  - `exportCsv`: Csv export plugin, see Section 11.3, “CSV Export”
  - `exportHtml`: HTML export plugin, see Section 11.2, “HTML Export”
  - `exportPdf`: PDF export plugin, see Chapter 12, *PDF Export*
  - `hello`: test plugin
  - `hilight`: hilight plugin, see Section 9.3.2, “Hilight Configuration”
  - `outline`: redlining and annotations, see Chapter 10, *Annotation and Redlining*
- `po`: PO templates files, used for gettext translation system, see Section 14.1.2, “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 15, *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
  - `icons`: Generated icons
  - `images`: Mapserver images
  - `mapinfo_cache`: Client-side server configuration cache, see Section 5.6, “Caches Configuration”
  - `mapresult_cache`: Client requests and associated server results cache, see Section 5.6, “Caches Configuration”
  - `pdf`: Pdf generated cache files
  - `soapxml_cache`: Client SOAP XML requests and associated

server results cache, see Section 5.6, “Caches Configuration”

- `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 overriden. 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 formular.

### 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
$_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'll 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.

- urlProvider = Miniproxy|Symlink: The kind of URL generated for resources, see Section 15.3, “Resources”
- 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.

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.
- cartoclientUrl = "url" : base URL of the cartoclient-
- cartoserverUrl = "url" : base URL of the cartoserver (i.e. path containing the `cartoserver.wsdl.php` file.

Mapfile configuration:

- mapId = string
- initialMapStateId = string

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: zoom_in, zoom_out, pan, query, distance, surface.

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 = I18nDummy|I18nGettext. See Chapter 14, *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 ...)

## 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. Using this will bypass Miniproxy !
- 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 is located in

the same directory as the `myMap.map` and `myMap.ini`. 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.

## 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.

---

**Note**

It is recommended that you set these parameters accordingly if you are doing development.

---

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
- allowTests = true|false: allows tests running through Web interface

# 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**

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.

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

## 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 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.

---

**Tip**

During development, it is recommended that you turn the caches off (set their parameter to `true`), so that your modifications will not be hidden by cached values.

---

### *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 path is relative to `layers.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.

### 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.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.

## 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,security_edit"
    "security_view" "admin"
    "security_edit" "admin,editors"

    "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
```

## 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.

# 7. Navigation

### *Plugin Location*

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.
- 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 zoom_in tool icon in toolbar (if not specified, default to 10). A negative weight disables the tool.
- weightZoomOut: see weightZoomIn (default to 11)
- weightPan: see weightZoomIn (default to 12)

## 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 whan scaleModeDiscrete is set to false.
- 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.
- recenterDefaultScale: scale to use when the centered-on object is a point.

## 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

### *Plugin Images*

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

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.

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

## 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

### *Plugins Query, MapQuery, Hilight and Tables*

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 rectangle selection or using a list of object IDs. This 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 9.3.2, “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)

- weightQuery: integer which defines display order of the query tool icon in toolbar (default: 40). Negative weighted tools are disabled

### 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 thios cas, Hilight plugin must be loaded on server. If false, objects will be hilighed using MapServer query functionality. See also Section 9.3.2, “Hilight Configuration” (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. 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 "integer" 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

Example:

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

### 9.3.2. 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.

#### 9.3.2.1. 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
    NAME "hilight"
    STYLE
      ...
    END
  END
END
```

Hilight using dynamically generated layer

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

#### *9.3.2.2. 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

### *Plugin Outline*

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. If not set, the polygon layer is automaticaly used instead.
- lineLayer: defines the name of the mapserver LAYER set in the mapfile to display lines.
- polyLayer: 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.

## 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
```

The point layer is optional as it can be replaced by polygon layer when missing.

Don't forget that layers names must be set in the client-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.

# 11. Export Plugins

### *exportCsv, exportHtml 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. Three formats are available: PDF, CSV, HTML.

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
```

# 12. PDF Export

### *Plugin exportPdf*

## 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.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).
- *general.mapServerResolution* (integer): resolution set in mapfile. Generally it is no use to set this parameter since its value is retrieved directly from the CartoServer. However you can use it to override this value (but this will not change the effective MapServer resolution).
- *general.defaultResolution* (integer): default selected/used resolution. Must be one of the above list items.
- *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.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 *`urlProvider = Miniproxy`* in CartoServer settings and/or physically separated CartoClient and CartoServer (See Chapter 4, *Configuration Files*).

PHP built-in default values:

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

- output: redirection
- allowedRoles: (empty ie. nobody)
- importRemotePng: false

### 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 14, *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.

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

## 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.

Note 2: 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 (").

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:

```
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.formats = A4, A3
general.defaultFormat = A4
general.resolutions = 96, 150, 300
general.mapServerResolution = 96
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
```

#### *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 14, *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. Blocks Configuration

Whatever their types (image, text, table, legend), 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. 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
```

#### 12.3.3.1. 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
```

#### *12.3.3.2. 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.

#### 12.3.3.3. 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
```

#### *12.3.3.4. 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
```

#### *12.3.3.5. 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
```

#### 12.3.3.6. 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 = legendp
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. 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 13,

*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. Security Configuration

## 13.1. Introduction

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

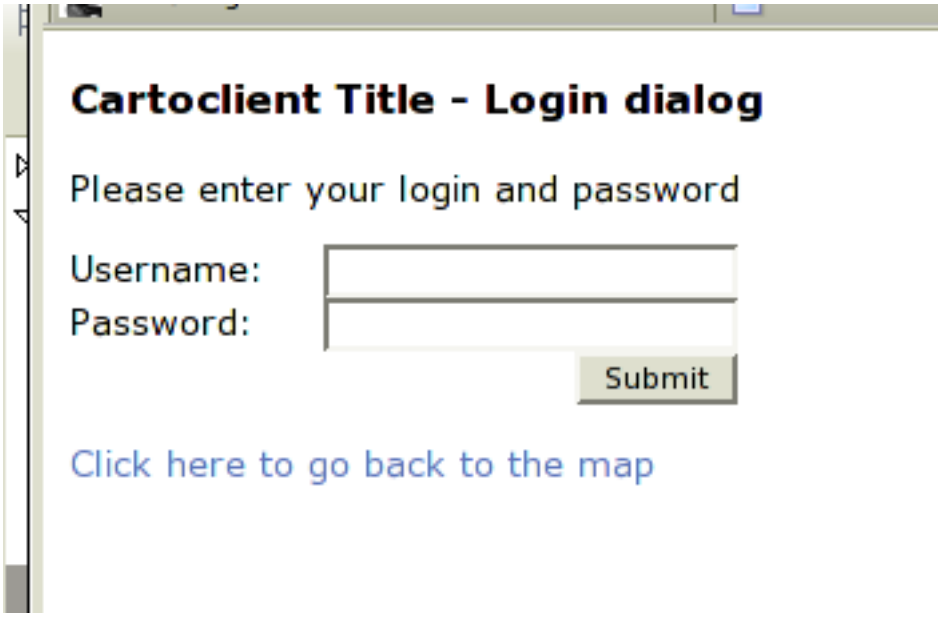
The following concepts are used in this chapter.

### 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. |

## 13.2. 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.

### 13.2.1. 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 Authentification login/logout buttons. (note: this is not related to the fact the authentification 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 13.2.2, “Special Role Names”

### 13.2.2. 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 13.1. Special Role Name Usage

```
general.allowedRoles = loggedIn
```

## 13.3. 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.

## 13.4. 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.

### *13.4.1. 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 `<mapfile>.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,security_edit"
```

```
  "security_view" "admin"

  "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 `<mapfile>.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.

---

### *13.4.2. PDF Printing Permissions*

The roles management in Pdf printing is explained in detail in Chapter 12, *PDF Export*. In particular, see Section 12.3.4, “Roles Management”.

# 14. Internationalization

## 14.1. 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.

### *14.1.1. 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.

Choosen translation system is set in client configuration file `client_conf/client.ini`:

```
### Internationalization ###

# I18n class
# use I18nDummy for no translation management
# use I18nGettext for gettext (will need PHP gettext module)
I18nClass = I18nDummy
```

#### *14.1.1.1. 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
```

#### 14.1.1.2. Win32

TODO

### 14.1.2. PO Templates

Texts to be translated can be found in:

- Smarty templates using SmartyGettext [http://smarty.incutio.com/?page=SmartyGettext] (see Section 15.2, “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 4.1, “Translations”)

To generate PO templates, you will need to launch scripts on server and on client. Templates are generated in directory `<cartoweb_home>/po`. If translation files (see Section 14.1.3, “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
  ./server2pot.php
  ```

  For each mapfile, two templates will be generated: `server-<project_name>.po` and `server-<project_name>.<mapfile_name>.po`
- generate project template on client:

  ```
  cd <cartoweb_home>/scripts
  ./client2pot.php
  ```

  For each project, one template will be generated: `client-<project_name>.po`

### 14.1.3. 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 default project will have three PO files for a complete french translation:

- `server-default.po`
- `server-default.test.po`
- `client-default.po`

### 14.1.4. 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
./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.

### 14.1.5. Example

To translate texts in french for project testproject and map file projectmap, follow these steps:

- On server:

  ```
  cd <cartoweb_home>/scripts
  ./server2pot.php
  ```

  Copy `<cartoweb_home>/po/server-testproject.po` to `<cartoweb_home>/po/server-testproject.fr.po` and `<cartoweb_home>/po/server-testproject.projectmap.po` to `<cartoweb_home>/po/server-testproject.projectmap.fr.po`. Edit french files with Poedit (or any editor).
- On client:

  ```
  cd <cartoweb_home>/scripts
  ./client2pot.php
  ```

Copy `<cartoweb_home>/po/client-testproject.po` to `<cartoweb_home>/po/client-testproject.fr.po`. Edit french file with Poedit (or any editor).

Merge and compile files with the following commands:

```
cd <cartoweb_home>/scripts
./po2mo.php
```

Now you should have the file `testproject.projectmap.mo` in directory `<cartoweb_home>/locale/fr/LC_MESSAGES`. The directory `fr/LC_MESSAGES` will be created if it does not exist.

## 14.2. 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

# 15. Templating

## 15.1. 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.

## 15.2. 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>
```

See also Chapter 14, *Internationalization*

## 15.3. 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="" />
```

Generated URLs depend on what *`urlProvider`* has been set in the general client configuration file `client.ini`. See Section 4.2, “ `client.ini` ” for details about *`urlProvider`*.

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`
  - generated URL (Miniproxy mode): `r.php?t=css&r=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`
  - generated URL (Miniproxy mode): `r.php?t=css&pl=layers&r=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`
  - generated URL (Miniproxy mode): `r.php?t=css&pl=hello&r=toto.css`
- Projects
  - path: `/projects/myproject/htdocs/css/toto.css`
  - Smarty: `{r type=css}toto.css{/r}`
  - generated URL: `myproject/css/toto.css`
  - generated URL (Miniproxy mode): `r.php?t=css&pr=myproject&r=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`
    - generated URL (Miniproxy mode): `r.php?t=css&pl=layers&pr=myproject&r=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`
    - generated URL (Miniproxy mode): `r.php?t=css&pl=hello&pr=myproject&r=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`
    - generated URL (Miniproxy): `r.php?t=css&pl=myplugin&pr=myproject&r=toto.css`

# 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.

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_nam`

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 1.3.5, “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="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:ArrayOfLayerId" 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 (layer IDs) separated by commas
- 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
- 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="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"/>
  </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

### 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

## 1.3. 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"/>
  </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

### 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. 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.

#### *1.3.4.1. 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 1.3.4.1.1, “BBox Request”)
- panLocationRequest - panning request parameters (see Section 1.3.4.1.2, “Pan Request”)
- zoomPointLocationRequest - zoom, recenter on point parameters (see Section 1.3.4.1.3, “Zoom-Point Request”)
- recenterLocationRequest - recenter on IDs parameters (see Section 1.3.4.1.4, “Recenter Request”)
- locationConstraint - constraint to be respected when location request is executed

#### *1.3.4.1.1. 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)

#### *1.3.4.1.2. 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

#### *1.3.4.1.3. 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)

#### *1.3.4.1.4. 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

#### 1.3.4.2. 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

### 1.3.5. 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*.

#### 1.3.5.1. 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="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*)
- 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="bbox" type="types:Bbox"/>
    <element name="queryAllLayers" type="xsd:boolean"/>
    <element name="defaultMaskMode" 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
- bbox - bounding box when querying with a rectangle (see Section 1.1.2, “Shapes” for a description of type Bbox)
- 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
- 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

#### *1.3.5.2. 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)

### 1.3.6. 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.

#### 1.3.6.1. Outline Request

```
<complexType name="OutlineRequest">
  <all>
    <element name="className" type="xsd:string"/>
    <element name="shapes" type="types:ArrayOfShape"/>
    <element name="maskMode" type="xsd:boolean"/>
  </all>
</complexType>
```

- className - "OutlineRequest" or extended class name if project implements an extension
- shapes - list of shapes (can include points, rectangles, lines or polygons, see Section 1.1.2, “Shapes” for a description of type Shape)
- maskMode - if true, will draw the complement of all shapes merged together

#### 1.3.6.2. 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

$client = new SoapClient("http://url.to/server/cartoserver.wsdl.php"
                         . "?mapId=swiss_project.swiss");
...
```

The map ID is required also in the request object.

```
$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.

```
$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.

```
$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.

```
$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

$client = new SoapClient("http://url.to/server/cartoserver.wsdl.php"
                         . "?mapId=swiss_project.swiss");
...
```

The map ID is required also in the request object.

```
$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.

```
$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->bbox->minx = 570000;
$request->queryRequest->bbox->miny = 120000;
$request->queryRequest->bbox->maxx = 580000;
$request->queryRequest->bbox->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', 'buildMapRequest',
                                   $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.

### 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
/**
 * @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. 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:

```
$smarty = new Smarty_CorePlugin($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
/**
 * @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 {

/* ... */

}
```

### 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 15.2, “Internationalization” and Section 15.3, “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
/**
 * @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
    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() {
        $smarty = new Smarty_CorePlugin($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. `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 2.4.1.3, “`Exportable` Interface”)
- `getExportScriptPath`: returns path to PHP export script (see Section 2.4.1.4, “PHP Export Script”). Default is `export.php`, located in plugin's `htdocs` directory
- `getExport` (abstract): contains export generation itself. Should prepare export configuration, call `getExportResult` and generate export in an `ExportOutput` object

#### 2.4.1.2. `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.

#### 2.4.1.3. `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.

#### 2.4.1.4. PHP Export Script

In most cases, export plugins generate outputs to be downloaded through the browser. This needs an external script that is not standard `client.php`. Default export script is named `export.php` and is located in plugin's `htdocs` directory.

When a plugin needs to know where this script is located (for instance to display a 'Print' link), it can use plugin's method `getExportScriptPath`.

#### 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:

```
private function getConfiguration() {
    $config = new ExportConfiguration();
    $config->setRenderMap(false);
    $config->setRenderKeymap(false);
    $config->setRenderScalebar(false);
}
```

Output 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`:

```
public function getExport() {
    $this->getExportResult($this->getConfiguration());

    // ...

    $output = new ExportOutput();
    $output->setContents($contents);
    return $output;
}
```

File `export.php` is also quite simple:

```
// CartoWeb Initialization
// ...

// HTTP request is handled in export plugin
$plugin = $cartoclient->getPluginManager()->getCurrentPlugin();
$plugin->handleHttpGetRequest($_REQUEST);

// Generated output will be in CSV format
header('Content-Type: application/csv-tab-delimited-table; charset='
                                . Encoder::getCharset());

// File name is generated in export plugin
header('Content-disposition: filename=' . $plugin->fileName);

// Exports CSV content
print $plugin->getExport()->getContents();
```

### 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 parameters that can be set in a filter plugin using function `setValue()`:

- Images plugin
  - *`mapsize`* - ID of the selected map size (see Section 8.1, “Client-side Configuration”)
- Location plugin
  - *`recenter_bbox`* - new bounding box, comma-separated coordinates, eg. "10.5,20,15.5,28"
  - *`recenter_x`* - re-centering: new x-coordinate
  - *`recenter_y`* - re-centering: new y-coordinate
  - *`recenter_scale`* - new scale
  - *`id_recenter_layer`* - re-centering on objects: layer to look for IDs
  - *`id_recenter_ids`* - re-centering on objects: list of IDs, comma-separated
  - *`shortcut_id`* - ID of the selected map size (see Section 7.2, “Server-side Configuration”)
- Query plugin
  - *`query_layer`* - layer to look for IDs
  - *`query_select`* - IDs of objects to add to selection
  - *`query_unselect`* - IDs of objects to remove from selection
  - *`query_policy`* - selection policy: 'POLICY_XOR', 'POLICY_UNION' or 'POLICY_INTERSECTION', default is 'POLICY_XOR'
  - *`query_maskmode`* - '0' or '1', default is '0'. If '1', will show selection as a mask
  - *`query_hilight`* - '0' or '1', default is '1'. If '0', won't shows selection hilighted
  - *`query_return_attributes`* - '0' or '1', default is '1'. If '0',

won't return attributes other than IDs

- *`query_return_table`* - '0' or '1', default is '1'. If '0', won't return any table results

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”).

#### 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'
```

#### 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` contain IDs of row that must be removed.

#### 2.4.3.2.9. 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'

#### 2.4.3.2.10. 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 2.4.3.2.9, “ColumnAdder” to know more about parameter `$columnPosition`.

### 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. Using the Security Infrastructure

## 3.1. Introduction

This chapter describes the security infrastructure from the developer point of view. For generatil details about security and its configuration see Chapter 13, *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.

## 3.2. 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.

## 3.3. 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 ..
```

# 4. Internationalization

## 4.1. Translations

Texts to be translated can be found in:

- Smarty templates (see Section 15.2, “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 14.1.2, “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 4.2, “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.

## 4.2. Character Set Encoding

As already described in Section 14.2, “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 14.2, “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()`.

# 5. Code Convention

## 5.1. 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.

## 5.2. General Coding Rules

### *5.2.1. 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.

### *5.2.2. 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).

### *5.2.3. Development Configuration*

Developers should absolutely set the following variables to true in their config (on both client and server sides):

- showDevelMessages = true
- developerIniConfig = true

### *5.2.4. 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 6, *Unit Tests*.

## 5.3. PHP

### 5.3.1. 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.sourceforge.net/en/standards.php.

Following are briefly described some guidelines to respect.

#### 5.3.1.1. 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).

#### 5.3.1.2. 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) {
  ...
}
```

#### 5.3.1.3. 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);
```

#### 5.3.1.4. 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.

#### 5.3.1.5. Nesting

Avoid deep blocks nesting:

This is correct:

```
for ($i = 0; $i < 10; i++) {
  if (! something ($i))
    continue;
  doMore();
}
```

#### 5.3.1.6. PHP Code Tags

Always use

```
<?php ?>
```

to delimit PHP code, not the

```
<? ?>
```

shorthand.

#### 5.3.1.7. Naming Conventions

##### 5.3.1.7.1. Classes

Classes should be given descriptive names that should always begin with an uppercase letter. Avoid using abbreviations.

##### 5.3.1.7.2. 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.

#### 5.3.1.7.3. Constants

Constants should always be all-uppercase, with underscores to seperate words.

### 5.3.2. Comments

#### 5.3.2.1. 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 7, *Code Documentation* for more information.

#### 5.3.2.2. Inline Comments

As often as necessary, the developers should add code comments to explain verbosly the purposes of commands.

## 5.4. 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.

### 5.4.1. Nesting

#### Elements Must Be Properly Nested

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>
```

### 5.4.2. Lower Case

#### Tag Names and Attribute Names Must Be in 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%">
```

### 5.4.3. Closing

#### 5.4.3.1. All Elements

##### All XHTML Elements Must Be Closed

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>
```

#### 5.4.3.2. Empty Elements

##### Empty Elements Must Also Be Closed

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 />.

### 5.4.4. Minimization

#### Attribute Minimization Is Forbidden

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" />
```

### 5.4.5. Id vs Name

#### The id Attribute Replaces The name Attribute

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.

### 5.4.6. Image "alt"

Tip: Image "alt" Attribute

"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="" />
```

# 6. Unit Tests

## 6.1. Introduction

Unit tests are an important componant in the CartoWeb development environment. The framework used for unit testing is based on PHPUnit2 [http://www.sebastian-bergmann.de/en/phpunit.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.

## 6.2. 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.

### *6.2.1. General Information About Writing Tests*

Test cases are located in directory tests, in a similar structure as the root directory:

```
client
    CartoclientTest.php
    CartoserverServiceTest.php
    AllTests.php
common
    BasicTypesTest.php
    AllTests.php
coreplugins
    AllTests.php
    ...
plugins
    AllTests.php
    ...
projects
    < all test for projects >
    ...
...
AllTests.php
```

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 6.1. Simple test case (`BasicTypesTests.php`)

```
<?php
require_once 'PHPUnit2/Framework/TestCase.php';
require_once(CARTOCLIENT_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 6.2. Test suite (`AllTests.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 6.3. Root directory's AllTests.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;
        }
}
?>
```

## *6.2.2. Specific Information for Tests*

This section describes specific features developped in CartoWeb for running tests, and infrastructure classes for more simple test case writing.

#### 6.2.2.1. 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.

#### 6.2.2.2. 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 6.2.1, “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 `<projectname>/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 6.2.1, “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.

---

## 6.3. Running Tests

Running tests is possible in two different ways:

1. With command line.
2. Through the web interface.

### *6.3.1. Command Line Interface*

To run a test case or a test suite, type the following command in directory `<cartoweb3_root>/tests`:

```
<php-bin> phpunit.php <test-class>
```

Where <php-bin> is the PHP binary and <test-class> is the name of the test class (AllTests, client_AllTests, client_CartoclientTest, etc.).

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.

### 6.3.2. Web Interface

The web interface for running test is available at the address `/runtests.php` from the CartoWeb base path. It is possible to give a GET parameter named `testsuite`, which accepts the same parameter as the command line argument described in Section 6.3.1, “Command Line Interface”.

---

**Important**

To allow the web interface to be used, the configuration parameter `allowTests` has to be set to `true`. See FIXME_ADD_LINK about the configuration of these parameters.

---

# 7. 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.

## 7.1. Generating Documentation

Documentation is generated using script `makedoc.php`:

```
cd scripts
php ./makedoc.php
```

This will generate documentation in directory `CARTOWEB_HOME/documentation/apidoc`.

## 7.2. 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].

### *7.2.1. 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.

### *7.2.2. 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.

### 7.2.3. 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
/**
 * 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;
        }
}
?>
```

# 8. Logging and Debugging

## 8.1. 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.

## 8.2. 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://www.vxr.it/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 understaning it.

### *8.2.1. Log4php Configuration Files*

Log4php settings are cusomizable 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, amon 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.

### 8.2.2. Default Log File Location

From what was said in the previous section, just uncommenting the line `log4php.rootLogger=...` means that the CartoClient 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.

### 8.2.3. 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
```

## 8.3. Debugging

Debugging is a large topic. Everyone has its preference over the tool to be used like using an integraded 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.

### *8.3.1. Understanding Exceptions and Stack Traces*

When a failure is encoutered 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))
```

### 8.3.2. 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` ”.

# 9. Performance Tests

## 9.1. 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)

## 9.2. Executing Tests

This section describes performance tests execution using APD, a debugging/profiling tool available as a Zend PHP module.

### *9.2.1. 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).

### 9.2.2. 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 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

### *9.2.3. Graphical Interface (Unix-like)*

To have more information about execution times and calls stack, you can use a powerfull 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.

# Appendix A. Mapserver Debian Installation

Some Debian packages are available for Php5 and Mapserver[1]. For now only the cgi version of Php is available because of threading issues with Mapserver.

## A.1. Prerequisites for Debian Woody

These packages have some dependencies on packages available in the so-called Sarge Debian repository version. Thus, if you are using Debian Woody, you will need to add the following lines to your `/etc/apt/sources.list`:

```
#debian sarge
deb http://ftp.<country>.debian.org/debian/ sarge main contrib
```

Of course, replace <country> by two-letter country code of your nearest mirror. See http://www.debian.org/mirror/list for the full mirrors list.

## A.2. Setting Up Your Repository and Preferences File

Add the following to your `/etc/apt/sources.list` :

```
# cartoweb
deb http://dev.camptocamp.com/packages/debian/ sarge main
deb-src http://dev.camptocamp.com/packages/debian/ sarge main
```

You may want to pin c2c packages higher than 1000 to have them preferred over the upstream ones, even if downgrading. Add in `/etc/apt/preferences`:

```
Package: *
Pin: release o=c2c
Pin-Priority: 1100
```

## A.3. Installing the packages

To install Mapscript and PHP 5, type the following:

[1]Some of the packages were inspired from http://agrogeomatic.educagri.fr/

```
apt-get update
apt-get install  php5-mapscript
```

If want to install PostGIS and PostgreSQL:

```
apt-get install  postgis
```

## A.4. Additional Steps

1. Update your `/etc/php5-c2cms-cgi/php.ini` to load the `php_mapscript.so` extension:

   ```
   extension=php_mapscript.so
   ```

1. you may want to raise the memory limit:

   ```
   memory_limit = 30M
   ```

# Index

## O



## P



## Q



## R



## S



## T



## U



## W



## Z