

PrimaGIS User Manual  
Version 0.5.1

Kai Hänninen  
<[kai.hanninen@gispython.org](mailto:kai.hanninen@gispython.org)>  
<http://www.primagis.fi/>

2006-01-29

## Contents

<b>1</b>	<b>Introduction</b>	<b>4</b>
<b>2</b>	<b>Components</b>	<b>4</b>
2.1	Python Cartographic Library (PCL)	4
2.2	Cartographic Objects for Zope (ZCO)	4
2.2.1	ZCO Data Store	6
2.2.2	ZCO Layer	6
2.2.3	ZCO Style	6
2.2.4	ZCO Symbolizer	6
2.2.5	ZCO Map Renderer	7
2.3	PrimaGIS	7
2.3.1	PrimaGISMap	7
2.3.2	PrimaGISLayer	7
2.3.3	PrimaGISDataLayer	7
2.3.4	PrimaGISView	9
2.3.5	PrimaGISScalebar	9
2.3.6	GeoArchetypesProxy	9
2.4	Geo Aware Objects	9
<b>3</b>	<b>Object hierarchy, layout and relationship</b>	<b>10</b>
3.1	ZCO hierarchy	10
3.2	PrimaGIS hierarchy	11
<b>4</b>	<b>Installing PrimaGIS</b>	<b>12</b>
4.1	Dependencies	12
4.2	Win32	12
4.2.1	FWTools	12
4.2.2	PROJ.4	13
4.2.3	PCL	13
4.3	Linux	14
4.3.1	Mapserver and Python MapScript	14
4.3.2	PCL	16
4.3.3	ZCO	17
4.3.4	PrimaGIS	18
4.4	Upgrading from previous versions	18
<b>5</b>	<b>Getting started</b>	<b>18</b>
5.1	Creating a demo map	18
5.1.1	Obtaining and configuring the spatial data	19
5.1.2	Customizing the script	19
5.1.3	Running the script in the right context	20
5.1.4	Demo map layers	20
<b>6</b>	<b>Creating a map</b>	<b>21</b>

---

<b>7</b>	<b>Adding new layers</b>	<b>22</b>
7.1	PrimaGISLayer . . . . .	22
7.2	PrimaGISDataLayer . . . . .	24
<b>8</b>	<b>Adding spatial content</b>	<b>24</b>
8.1	IGeoAware objects . . . . .	25
8.2	GeoArchetypesProxy . . . . .	25
<b>9</b>	<b>Exporting data</b>	<b>25</b>
9.1	Geography Markup Language (GML) . . . . .	25
9.2	Keyhole Markup Language (KML) . . . . .	26
<b>10</b>	<b>Customizing the PrimaGIS User Interface</b>	<b>26</b>
10.1	Cascading Style Sheets (CSS) . . . . .	26
10.2	Zope Page Templates (ZPT) . . . . .	26
<b>11</b>	<b>Developing IGeoAware content types</b>	<b>27</b>
11.1	IGeoAware interface . . . . .	27
11.2	Inheriting from GeoAwareMixin . . . . .	27
11.2.1	PointMixin and PointMixinSchema . . . . .	28
11.2.2	LinestringMixin and LinestringMixinSchema . . . . .	28
11.2.3	SimplePolygonMixin and SimplePolygonMixinSchema . . . . .	28
11.3	Building from scratch . . . . .	29
<b>12</b>	<b>Resources and additional information</b>	<b>29</b>

## 1 Introduction

PrimaGIS is a collaborative web mapping application for Plone[15], which allows you to combine spatial data from traditional sources (shapefiles, PostGIS, rasters, WMS<sup>1</sup>, WFS<sup>2</sup>) with spatially enabled content types from within Plone. In addition it provides a developer friendly interfaces and mixin classes for easy adaptation to your own projects.

This document applies to the following versions of the main PrimaGIS components:

- Python Cartographic Library (PCL) 0.10
- Cartographic Objects for Zope (ZCO) 0.7.1
- PrimaGIS 0.5.1

The document assumes that you are familiar with and have the required skills to install (and compile) new software on your system and within Zope/Plone. The intended audience for this document is both Plone users and developers.

## 2 Components

PrimaGIS is based on a stack of components, with PrimaGIS being on the top. The major components and their roles are briefly introduced below. Figure 1 shows the PrimaGIS architecture.

### 2.1 Python Cartographic Library (PCL)

The Python Cartography Library [18], or PCL, is a package of modules for rendering GIS data from a variety of backends into maps. Its mission is to be the best possible Python interface to open source GIS software such as PROJ.4[17], GEOS[7], GDAL[4], OGR[14], and MapServer[11], and to be easy to use with Python web application frameworks as well as with matplotlib[12].

PCL forms the basis for all cartographic operations and map rendering. Currently it uses MapServer (Cheetah) as the rendering backend, but in the future it will be possible to use other backends as well.

### 2.2 Cartographic Objects for Zope (ZCO)

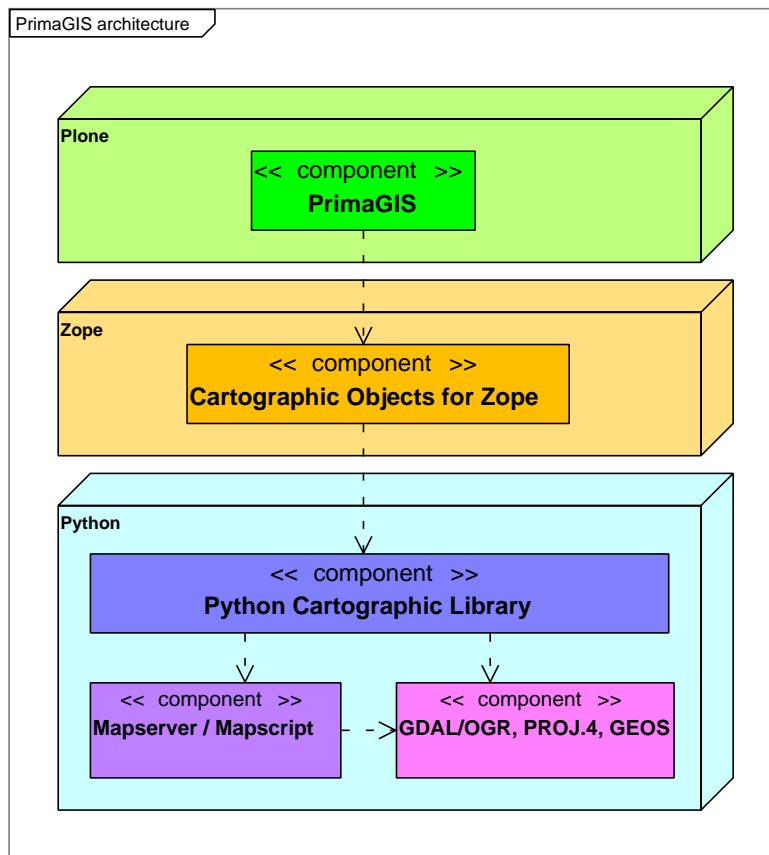
ZCO [1] is a framework for developing web mapping applications and cartographic object management systems in the Zope Application Server. It is based on PCL and provides a group of objects that may be used to implement mapping applications. Each object type is briefly introduced below.

---

<sup>1</sup>Web Map Service

<sup>2</sup>Web Feature Service

Figure 1: PrimaGIS architecture



Created with Poseidon for UML Community Edition. Not for Commercial Use.

### 2.2.1 ZCO Data Store

A ZCO Data Store represents spatial data that is stored in a particular format. ZCO Data Store objects support many different sources for spatial data. Below is a list of the available data store types:

- **Disk data store.** Shapefiles<sup>3</sup>, rasters, and any other format supported by PCL.
- **PostGIS[16] data store.** PostGIS is a spatial extension to the PostgreSQL.
- **WMS data store.** WMS is an OGC<sup>4</sup> standard for publishing map-images (raster) on the Web.
- **WFS data store.** WFS is an OGC standard for communicating feature data (vector) over the Web.

A ZCO Data Store object can be seen as a container for spatial features. For example a single Disk Data Store can represent multiple shapefiles. In addition you can create your own custom data stores by implementing the *ZCO.interfaces.IDataStoreProxy* interface. *PrimaGISDataLayer* is an example of this.

### 2.2.2 ZCO Layer

A Layer defines a collection of features or raster data that is rendered onto a map at the same cartographic “z” level by referring to a Data Store object. A Layer also contains rules for the presentation of the spatial data. The presentation of a layer is dependent on both Style and Symbolizer objects.

### 2.2.3 ZCO Style

A Style provides rules for the presentation of a Layer. Each rule defines simple constraints based on the available feature attributes and references a Symbolizer object that will be used to symbolize the selected features. Style objects answer to the question “**What** will be styled?”

### 2.2.4 ZCO Symbolizer

A Symbolizer object is responsible for the actual symbolization of a spatial feature. Each spatial feature type (Point, Line, Polygon, Text) has its own Symbolizer with attributes that can be modified to produce a desired symbol. Symbolizer objects answer to the question “**How** will it be styled?”

<sup>3</sup>Shapefile is a widely used spatial data file format published by ESRI. See <http://www.esri.com/library/whitepapers/pdfs/shapefile.pdf> for details.

<sup>4</sup>Open Geospatial Consortium. <http://www.opengeospatial.org/>

### 2.2.5 ZCO Map Renderer

A Map Renderer object is responsible for rendering a map image by composing a single image of a group of layers. There is exactly one map renderer object per ZCO object hierarchy.

ZCO also provides a CMF<sup>5</sup> compatible tool – **portal.gis** – that is a portal wide tool that provides the same services as Map Renderer and can be shared with all maps within the same CMF site.

## 2.3 PrimaGIS

PrimaGIS is an Archetypes based product and the PrimaGIS objects are thus part of the Plone site content as any other content types, such as Page, Rich-Document etc.

Figure 2 shows the UML model of the PrimaGIS classes and the following subsections will elaborate on them.

### 2.3.1 PrimaGISMap

A *PrimaGISMap* object is the main container for a PrimaGIS map and there is always a single *PrimaGISMap* object per map. It contains the main configuration for the map.

It is a containerish (folderish) type that works like a normal Plone Folder. This makes it possible to create a logical hierarchy of PrimaGIS objects that define the map within the *PrimaGISMap* object.

The containerish nature of *PrimaGISMap* is directly used in rendering the map layers. The ordering of *PrimaGISLayer* and *PrimaGISDataLayer* objects within the *PrimaGISMap* object is important because it also defines the rendering order. Layers higher up will be rendered before layers below and you can change the rendering order by simply moving the layer objects within the *PrimaGISMap* object.

For details on the configuration options of *PrimaGISMap* objects please refer to Section 6.

### 2.3.2 PrimaGISLayer

*PrimaGISLayer* objects refer to ZCO Layer objects and represent them within the Plone site. They rely on ZCO Data Stores for data and in this sense can be seen as “static” layers. They are used to inform the *PrimaGISMap* object that we wish to have a given ZCO Layer rendered on the map.

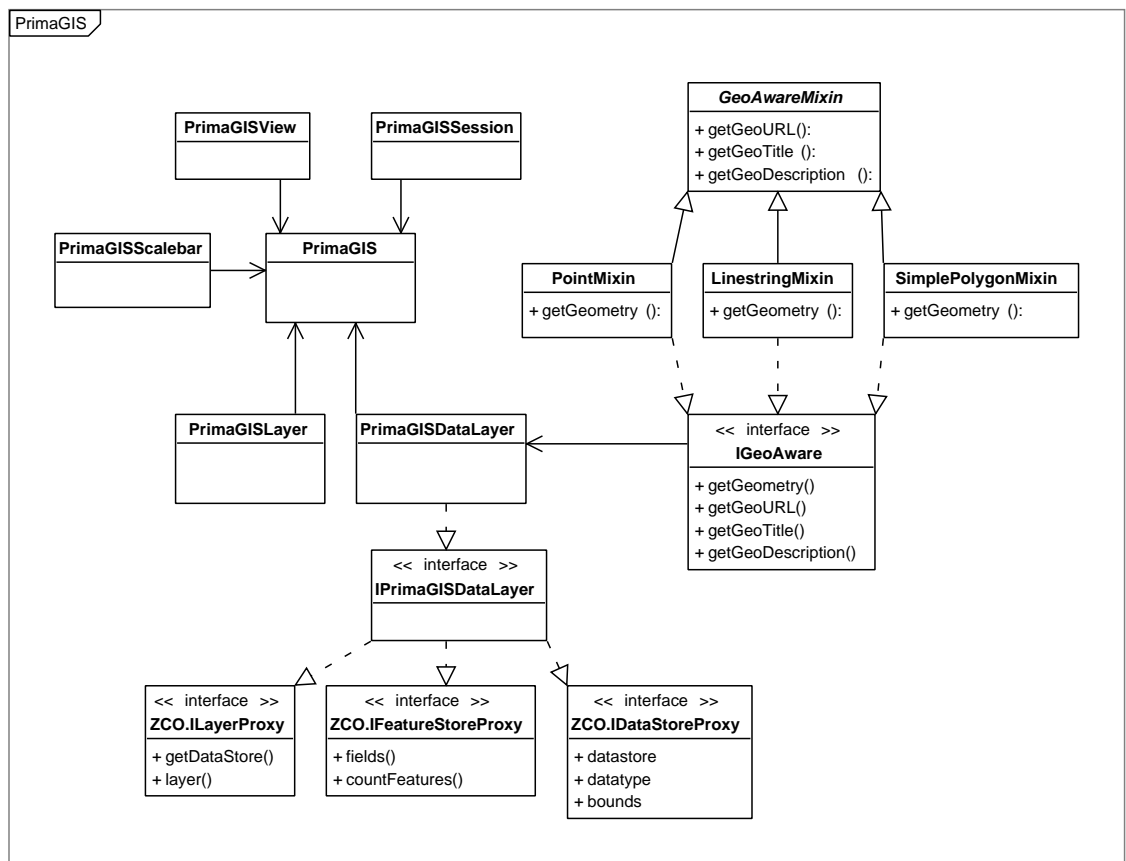
### 2.3.3 PrimaGISDataLayer

*PrimaGISDataLayer* extends *PrimaGISLayer* by implementing the *ZCO.IDataStoreProxy* interface which allows it to function as a data store for ZCO. *PrimaGISDataLayer* provides spatial data from the Plone site through the use of content

---

<sup>5</sup>Content Management Framework

Figure 2: PrimaGIS UML model



Created with Poseidon for UML Community Edition. Not for Commercial Use.

types implementing the IGeoAware interface. In this sense it can be seen as a “dynamic” layer.

### 2.3.4 PrimaGISView

PrimaGISView objects allow you to save a map extent (view of the map) and attach a name and description to it. These objects can be used to provide quick access to specific areas of interest on the map. PrimaGIS provides a drop-down box in the map user interface where the available PrimaGISViews are listed.

### 2.3.5 PrimaGISScalebar

PrimaGISScalebar is an Archetypes wrapper around *Javascript Scalebar for Mapserver* [9] created by Tim Schaub. It allows you to configure the way the scalebar is rendered within Plone.

### 2.3.6 GeoArchetypesProxy

GeoArchetypesProxy is a geo-aware proxy object for Archetypes based content types that makes it possible to use any Archetypes based object with a PrimaGIS map regardless of whether it implements IGeoAware.

GeoArchetypesProxy stores the spatial data within itself, uses the AT reference mechanism to link an arbitrary number of content objects and implements the IGeoAware interface on behalf to the referenced objects. In practice this means that you can geocode arbitrary AT objects within your site.

## 2.4 Geo Aware Objects

GeoAwareObjects [5] is a companion product to PrimaGIS whose primary purpose is to provide concrete examples on how to implement new content types or extend existing content types so that they can be used with PrimaGIS. Many of the content types are usable as such in production environments. Currently available content types are:

- **GeoRSSFeed.** A geo-aware wrapper around CMFSin [2] that allows you to display RSS feeds on your map. GeoRSSFeed is about geo-localizing RSS feeds themselves and has nothing to do with the *georss* (<http://georss.org/>) microformat used to add spatial information to the RSS feed content.
- **GeoATImage.** A simple extension to ATImage from ATContentTypes that allows you to display images directly in the hotspot pop-up windows.
- **GeoATFile.** A simple extension to ATFile from ATContentTypes that allows you to add links to downloadable files directly in the hotspot pop-up windows.

- **GPSTrack**. A content type that allows you to upload track files created by GPS receivers and render them on PrimaGIS maps.
- **GeoLink**. A simple content type that allows you to create geo-localized hyperlinks.
- **SimplePolygon**. Example content type for creating custom POLYGON features.
- **RandomSquare**. Example content type for creating custom POLYGON features.

### 3 Object hierarchy, layout and relationship

Both PrimaGIS and the underlying ZCO provide many different types of objects that are used together to create map images. These objects need to be organized in a certain manner for everything to work as expected.

The ZCO product itself is very flexible and allows administrators to organize the objects involved in many different ways, but PrimaGIS has adopted a certain structure that it expects the ZCO objects to follow<sup>6</sup>.

The following subsections will explain the expected ZCO and PrimaGIS object hierarchies. Please note that ZCO is a pure Zope product and needs to be managed through ZMI whereas PrimaGIS should be managed through Plone.

#### 3.1 ZCO hierarchy

The ZCO hierarchy is organized using Zope Folders which are not visible within the Plone site. This makes it possible to have the ZCO objects stored within the Plone site while hiding them from normal users. Alternatively you may store the ZCO hierarchy outside your Plone site (possibly sharing it with multiple Plone sites within the ZODB) as long as it is acquirable from Plone.

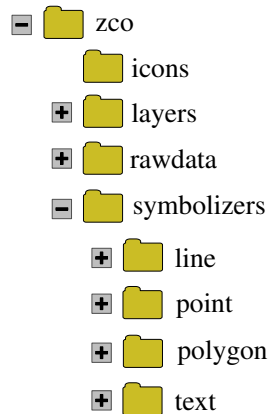
Figure 3 shows the layout of the ZCO hierarchy expected by PrimaGIS. Each folder and its purpose is described below. All folders except the the top-most should be named exactly as shown in Figure 3. The top-most folder may be named arbitrarily because it is configured on a per map basis in the attributes of *PrimaGISMap* objects. This makes it possible to have multiple ZCO hierarchies under the same container.

- **zco**. The name of the top-most container. May be chosen arbitrarily.
- **rawdata**. The container for *ZCO Data Store* objects.
- **layers**. The container for *ZCO Layer* objects.

---

<sup>6</sup>Both PrimaGIS and ZCO use the Zope acquisition mechanism to access objects so it is possible in some special cases to deviate from the expected structure although this is not recommended.

Figure 3: ZCO object hierarchy



- **symbolizers.** The container for *ZCO Symbolizer* objects. Each type of symbolizer is stored under a specified sub-folder.
- **text.** The container for *TEXT* symbolizers used for annotation.
- **point.** The container for *POINT* symbolizers.
- **line.** The container for *LINE* symbolizers.
- **polygon.** The container for *POLYGON* symbolizers.
- **icons.** The container for Zope Image objects used in symbolizing *POINT* features.

The *ZCO Style* objects are stored within their corresponding *ZCO Layer* objects.

PrimaGIS comes with a script named *createPrimaGISDemo.py* that will create a fully working ZCO and PrimaGIS object hierarchies, see section 5.1 for details.

### 3.2 PrimaGIS hierarchy

A PrimaGIS map is organized around a single *PrimaGISMap* object with all the other PrimaGIS objects contained within it. All *PrimaGISLayer* and *PrimaGISDataLayer* objects should be stored directly under the *PrimaGISMap* object. PrimaGIS does not currently support layer grouping.

The order of the *PrimaGISLayer* and *PrimaGISDataLayer* objects defines also the rendering order of the layers and you can change this by using the default Plone mechanism for ordering folder contents. You will need to have the appropriate access rights to be able to manage the contents of the *PrimaGISMap* object.

PrimaGISView objects make an exception in this regard because they can be located anywhere within the *PrimaGISMap* object (using a catalog query). This makes it possible to store them within a separate folder or even putting them within a *PrimaGISDataLayer* and rendering them.

To use a scalebar with your map, simply place a single *PrimaGISScalebar* object within your *PrimaGISMap* object and it will be rendered in your map UI.

## 4 Installing PrimaGIS

PrimaGIS builds upon many different components which must be properly installed and configured. Installing PrimaGIS itself is a fairly simple process, but satisfying all dependencies may require some domain specific information on the specific components. Please refer to the documentation of each such component for up-to-date and detailed information.

### 4.1 Dependencies

PrimaGIS depends on the following major components. Each component may have its own dependencies, which you need to also satisfy.

- **Python Cartographic Library (PCL)**. Required version 0.10. PCL has some additional dependencies of its own (MapServer, GDAL, and PROJ.4).
- **Zope**. Works with both Zope 2.7 and 2.8 series. Recommended version 2.7.8 or 2.8.4.
- **Cartographic Objects for Zope (ZCO)**. Required version 0.7.1.
- **Plone**. Works with both 2.0 and 2.1 series. Recommended version 2.0.5 or 2.1.1.
- **Archetypes**. Required version  $\geq$  1.3.2-final.
- **ATReferenceBrowserWidget**. Optional for Plone 2.0. Plone 2.1 comes with ATReferenceBrowserWidget and it is automatically used by PrimaGIS.
- **simplejson** [19]. A python JSON implementation. Required version 1.1.

### 4.2 Win32

#### 4.2.1 FWTools

“FWTools is a set of Open Source GIS binaries for Windows (win32) and Linux (x86) systems produced by Frank Warmerdam (ie. FW). It was previously known as OpenEV\_FW. The kits are intended to be easy for end users to install and get going with. No fudging with building from source, or having to collect

lots of interrelated packages. FWTools includes OpenEV, GDAL, MapServer, PROJ.4 and OGD I as well as some supporting components.” [3].

The FWTools package is the easiest way to satisfy PCL’s MapServer and GDAL/OGR dependencies.

1. Download FWTools (1.0.0-a7 will do) from <http://www.gdal.org/dl/fwtools/>
2. Run the installer, installing to *C:\Program Files\FWTools1.0.0a7*.
3. Modify your system environment, setting up the following variables.

```
GDAL_DATA=C:\Program Files\FWTools1.0.0a7\data
PROJ_LIB=C:\Program Files\FWTools1.0.0a7\proj_lib
```
4. Modify your system PATH, adding *C:\Program Files\FWTools1.0.0a7\bin* to the head of the path
5. Modify your system PYTHONPATH, adding *C:\Program Files\FWTools1.0.0a7\pymod* to the head of the path

#### 4.2.2 PROJ.4

PROJ.4 [17] is required by the PCL-Referencing module. If you don’t already have a proj.dll on your system (version  $\geq 4.4.8$ ), download one from <http://www.gispython.org/downloads/gispy/proj.dll>.

#### 4.2.3 PCL

1. Download <http://www.gispython.org/downloads/gispy/PCL-0.10.0-win32.zip>.
2. Extract the package to a temp location, *C:\temp* for example
3. cd to *C:\temp\PCL-0.10.0-win32*
4. Extract *PCL-Referencing-0.10.0.win32.zip* to a location such as *C:\software*
5. Extract *PCL-0.10.0.win32.zip* to a location such as *C:\software*
6. Result is *C:\software\python23\Lib\site-packages\cartography\...*
7. Modify the system pythonpath again, putting *c:\software\python23\Lib\site-packages* at the head

Proceed to Section 4.3.3 for instructions on installing ZCO and PrimaGIS.

## 4.3 Linux

### 4.3.1 Mapserver and Python MapScript

Python Cartographic Library uses the Python bindings to MapServer's scripting interface called MapScript. This is the only way Mapserver is used by PCL (and thus PrimaGIS) and you do not need to install the MapServer CGI program.

In the examples in the following sections we will use \$ (dollar sign) to symbolize a shell prompt and any following text something that you should type into your terminal.

#### Acquiring and compiling Mapserver / Mapscript

Download the latest MapServer release from <http://ms.gis.umn.edu/download/current/> and unpack the source tree. We will refer to the location where the MapServer source is unpacked as \$MAPSERVER.

Go to the \$MAPSERVER directory and use the *configure* script to select the features you wish to compile into your build. Below is an example of a minimal configuration that will work with PrimaGIS. You may wish to modify it to better suit your needs. Use *configure --help* to learn about the available options.

```
$ ./configure --with-tiff --with-jpeg --with-png
--with-freetype --with-zlib --with-threads --with-proj
--with-gdal --with-wcs --with-ogr --with-wmsclient
--with-wfsclient
```

You will need the development packages for the features you wish to build into MapServer. If you are running a Debian (or Debian based) distribution you can use *apt-get* to install the following packages<sup>7</sup>.

- python-dev
- python-gdal
- proj
- libgd2-dev
- libgdal1-dev
- libtiff4-dev
- libfreetype6-dev
- libcurl3-dev

---

<sup>7</sup>You should also check out the DebianGIS (<http://pkg-grass.aliioth.debian.org/cgi-bin/wiki.pl>) and UbuntuGIS (<https://wiki.ubuntu.com/UbuntuGIS>) projects for additional GIS packages.

When all dependencies have been met and the *configure* script has been run successfully you can compile MapServer by running:

```
$ ./make
```

After MapServer has been compiled you still need to compile the Python Mapscript module that will be used by PCL. Use the following steps to accomplish this.

```
$ cd $MAPSERVER/mapsript/python
$ cp modern/* .
$ python setup.py install
```

**Note to SVN users.** Both MapServer and PCL use SWIG [20] to generate the python bindings and it is imperative that the same version of SWIG is used in both. PCL releases come with generated wrappers that ensure compatibility with MapServer, but SVN versions require that you generate the wrappers yourself. For this reason it is better to use your own version of SWIG to generate the MapServer wrappers also. In this case use the following steps instead.

```
$ cd $MAPSERVER/mapsript/python
$ swig -python -modern -o mapsript_wrap.c
../mapsript.i
$ python setup.py install
```

After installing the MapScript module you can check your installation using the Python interpreter. If you have multiple python versions installed on your system remember to make sure that you are using the same one that Zope is using.

```
$ python

Python 2.3.5 (#2, May 4 2005, 08:51:39)
GCC 3.3.5 (Debian 1:3.3.5-12)] on linux2
Type 'help', 'copyright', 'credits' or 'license'
for more information.

>>> import mapsript
>>> help(mapsript)
```

If the previous commands did not give any errors and the path to the mapsript module shown by the *help* command looks correct, then the MapServer installation was successful.

### Creating a fontset file

When annotating your features you will most likely wish to use TrueType fonts. For this you will need to create a fontset file, which is a simple text file containing

information about the fonts that are available on your system. You can also have only a subset of the available fonts listed in the fontset file, thus limiting PCL's access to only those fonts. Each line in a fontset file contains two parts: the font identifier and the path to the font file on your file system. Below is an example of a fontset file.

```
times_new_roman_bold_italic /fonts/Times_New_Roman_Bold_Italic.ttf
times_new_roman_bold /fonts/Times_New_Roman_Bold.ttf
times_new_roman_italic /fonts/Times_New_Roman_Italic.ttf
times_new_roman /fonts/Times_New_Roman.ttf
trebuchet_ms_bold_italic /fonts/Trebuchet_MS_Bold_Italic.ttf
trebuchet_ms_bold /fonts/Trebuchet_MS_Bold.ttf
trebuchet_ms_italic /fonts/Trebuchet_MS_Italic.ttf
trebuchet_ms /fonts/Trebuchet_MS.ttf
```

On a Debian system you can install the *msttcorefonts* package to get a set of usable TrueType fonts. Name the fontset file as *fontset.txt* and save it somewhere on your file system that is accessible by PCL.

### 4.3.2 PCL

Installing PCL is a simpler task than installing MapServer. First download the latest PCL release and unpack it on your file system. We will refer to the location where the PCL source tree is unpacked as *\$PCL*. Pay attention to the *\$PCL/DEPENDENCIES.txt* file and make sure that you have everything available on your system.

First edit the *\$PCL/PCL-Cartography/setup.py* file and modify the *ms.home* variable to point to the location of the MapServer source tree. After this you can install PCL by running

```
$ cd $PCL/PCL-Cartography
$ python setup.py install
```

Alternately, you can specify extra include paths to the setup.py *build\_ext* target

```
$ cd $PCL/PCL-Cartography
$ python setup.py build_ext -I $MAPSERVER install
```

This will install both the cartography and the referencing modules. You can verify that PCL was successfully installed by running the python interpreter and trying to import the cartography module.

```
$ python
```

```
Python 2.3.5 (#2, May 4 2005, 08:51:39)
```

```
GCC 3.3.5 (Debian 1:3.3.5-12) on linux2
Type 'help', 'copyright', 'credits' or 'license'
for more information.
```

```
>>> import cartography
>>> help(cartography)
```

If the previous commands did not give any errors and the path to the cartography module shown by the *help* command looks correct, then the PCL installation was successful. If you are upgrading from a previous installation please make sure that you do not have any **.pyc** or **.pyo** files lying around from older versions.

There is a helper shell script available from <http://trac.gispython.org/projects/PCL/wiki/QuickInstallScript> that automates the process of (re)installing Mapsript and PCL. This script is only for advanced users.

### 4.3.3 ZCO

Download the latest release and unpack it in your Zope instance's Products directory. Please make sure that the ZCO directory is named just "ZCO" and not something like "ZCO-0.7.1". Restart Zope and verify that Zope recognized ZCO in **Control\_Panel/Products** using ZMI.

In order to get projection support for ZCO you also need to define a **PROJ\_LIB** environment variable that points to the directory containing the EPSG<sup>8</sup> database file. On Debian systems this is normally */usr/share/proj*. You can modify the **zopectl** and **runzope** to include the variable to make sure that Zope is able to see it. Below is an example of a modified **zopectl** script.

```
#!/bin/sh

PYTHON="/usr/bin/python2.3"
ZOPE_HOME="/usr/lib/zope2.7"
INSTANCE_HOME="/var/lib/zope2.7/instance/primagis"
CONFIG_FILE="/var/lib/zope2.7/instance/primagis/etc/zope.conf"
SOFTWARE_HOME="/usr/lib/zope2.7/lib/python"
PYTHONPATH="$SOFTWARE_HOME"
PROJ_LIB="/usr/share/proj"

export PYTHONPATH INSTANCE_HOME SOFTWARE_HOME PROJ_LIB

ZDCTL="$SOFTWARE_HOME/Zope/Startup/zopectl.py" exec
"$PYTHON" "$ZDCTL" -C "$CONFIG_FILE" "$@"
```

In order to avoid problems with case sensitive file names you should also symlink the **epsg** file to **EPSG** using the following command.

<sup>8</sup>European Petrol Survey Group. <http://www.epsg.org/>.

```
$ ln -s $PROJ_LIB/epsg $PROJ_LIB/EPSG
```

ZCO also has a CMF compatible part that can be installed using the **portal.quickinstaller** tool available in your Plone site. The PrimaGIS installer will also install the ZCO tool if it is not yet installed.

#### 4.3.4 PrimaGIS

Download the latest release and unpack it in your Zope instance's Products directory. Restart Zope and verify that Zope recognized PrimaGIS in **Control Panel/Products** using ZMI.

You should now see PrimaGIS also in your Plone site's **portal.quickinstaller** tool. Use the **portal.quickinstaller** to install PrimaGIS and check the installation log for any errors.

### 4.4 Upgrading from previous versions

PrimaGIS installer comes with migration code that will automatically migrate old versions of PrimaGIS and ZCO objects to the current version. It is therefore important that you re-install PrimaGIS using the **portal.quickinstaller** tool in your Plone site after restarting Zope. The installation log will give details on which objects were modified during the migration process.

## 5 Getting started

After successfully installing PrimaGIS to your Plone site you are ready to start building your maps. You can do this by either creating everything from scratch or using the *createPrimaGISDemo.py* script that is included with PrimaGIS.

The script will create a fully working PrimaGIS map including the ZCO object hierarchy and is a great way to get started. You can learn more about PrimaGIS and start creating your own maps by modifying the demo map. It is also possible to create multiple maps within a single Plone site using the script.

Before running the script you need to configure your **portal\_gis** tool within ZMI. This needs to be done only once. Go to the *Properties* tab within the **portal\_gis** tool and set the following properties:

- **fontset**. Absolute path to the fontset file defined in section 4.3.1, e.g. */usr/local/mapdata/fontset.txt*.
- **incoming**. Absolute path to a directory where PrimaGIS will store temporary files, e.g. */tmp*. Needs to be writable by the Zope process.

### 5.1 Creating a demo map

The *createPrimaGISDemo.py* script is located under *PrimaGIS/skins/primagis/* on your file system, which means that you can find it in your Zope Management

Interface (ZMI) under `portal_skins/primagis/createPrimaGISDemo` within your Plone site object.

### 5.1.1 Obtaining and configuring the spatial data

The demo map uses a shapefile containing the world political borders that is freely available from [http://www.mappinghacks.com/data/world\\_borders.zip](http://www.mappinghacks.com/data/world_borders.zip). Download the zip file and unpack in a suitable location on your file system. The world borders data set is unfortunately unreferenced, meaning that it does not contain any information about the coordinate system used with it. For this reason we need to create an XML description for it that contains this information — this is called an OVF<sup>9</sup> file. Below is an example for the .ovf file that describes the coordinate system for the world borders data set. You should place the .ovf file in the same directory as the shape file and name it `world_borders.ovf`.

```
<OGRVRTDataSource>
  <OGRVRTLayer name="world_borders">
    <SrcDataSource relativeToVRT="1">world_borders.shp</SrcDataSource>
    <SrcLayer>world_borders</SrcLayer>
    <LayerSRS>EPSG:4326</LayerSRS>
  </OGRVRTLayer>
</OGRVRTDataSource>
```

For details on the file system backend configuration refer to [http://zcolgia.org/cartography/filesystem\\_backend.html](http://zcolgia.org/cartography/filesystem_backend.html).

### 5.1.2 Customizing the script

There is a user configurable section at the beginning of the script and in most cases you need to customize the configuration variables before running the script. The user configurable section contains the following options:

- **PROXY\_SUBJECT\_TYPE.** The script creates some *GeoArchetype-Proxy* objects and corresponding content objects. This determines the type of content objects (proxy subjects) that will be created. Any Archetypes based content type will do. Plone 2.1 users may simply use *'Document'* here. If you are using Plone 2.0.x and do not have any Archetypes based objects available you can use *DummyContent*<sup>10</sup>, which is a trivial Archetypes content type created for testing purposes. Note! You need to install *DummyContent* before it can be used.
- **WORLD\_BORDERS\_OVF.** This should be the path to your `world_borders.ovf` file created in Section 5.1.1.

<sup>9</sup>OGR Virtual File. See [http://ogr.maptools.org/drv\\_vrt.html](http://ogr.maptools.org/drv_vrt.html) for details.

<sup>10</sup>Download from <http://www.primagis.fi/download/DummyContent-0.1.tar.gz>

To customize the script go to *portal.skins/primagis/createPrimaGISDemo* within ZMI and click the **Customize** button. This will make a copy of the script in *portal.skins/custom/createPrimaGISDemo* which you can edit.

### 5.1.3 Running the script in the right context

After the script has been successfully customized you need to call it through your Plone site by accessing it directly with your browser using an URL similar to <http://your.site/ploneroot/createPrimaGISDemo>. You should not try running it with the **Test** tab through ZMI.

The script will create a Plone Folder called *demo* under the path that you specified in the URL. You can easily create multiple demo maps within your Plone site by simply calling the script in different contexts (locations). Under the *demo* folder you will find three folderish objects:

- **europedata**. A Plone Folder containing some dummy content used in the demo.
- **primagis**. The PrimaGIS map hierarchy.
- **zco**. The ZCO object hierarchy, which is visible only from within ZMI.

Viewing the 'primagis' object should give a working world map with some optional layers.

### 5.1.4 Demo map layers

The demo map contains five layers which demonstrate the use of different data sources. The layers are:

- **World political borders**. A shapefile based layer depicting the world political boundaries.
- **JPL Landsat Global Mosaic**. A WMS layer showing satellite imagery from the NASA Jet Propulsion Laboratory<sup>11</sup>.
- **MapServer users**. A WFS layer showing the locations of MapServer users. The MapServer website (which uses Plone) at <http://ms.gis.umn.edu/> allows members to specify their location in their member data and this is published as a WFS service. See Sean Gillies' blog entry for more details about the WFS service at <http://zcologia.com/news/33>.
- **European capitals**. A *PrimaGISDataLayer* providing POINT data of the European capitals through the use of GeoArchetypesProxy objects. The data for the pop-ups comes from the dummy content objects located in the **europedata** folder while the spatial data (latitude, longitude) is stored within the proxy objects.

<sup>11</sup>Read more about JPL at <http://www.jpl.nasa.gov/>

- **Areas of Interest.** A *PrimaGISDataLayer* that contains the selectable views (Areas of Interest) for the demo in the form of *PrimaGISView* objects. The *PrimaGISView* objects implement the *IGeoAware* interface and putting them inside a *PrimaGISDataLayer* makes it possible to render them on the map.

## 6 Creating a map

A PrimaGIS map is built by first creating a *PrimaGISMap* object. Use the Plone’s “Add new item” pull-down menu to do this. The *PrimaGISMap* object contains several configuration fields which are explained below.

- **Title.** This is the default Dublin Core Title field available in all Plone content types.
- **ZCO hierarchy container.** PrimaGIS needs to be made aware of the associated ZCO object hierarchy that contains the data stores, layers, etc. It is possible to have multiple ZCO hierarchies in your ZODB and this allows you to use a specific one. The field expects a traversable (in acquisition sense) path to the top container of the ZCO hierarchy relative to the *PrimaGISMap* object.  
**Example values:** `"zco"`, `"../complex/path"`
- **Map background color.** The background color for the rendered map images. Use a hex coded RGB value or leave the field blank for transparent background.  
**Example values:** `"#c0ffee"`, `"#ffffff"`, `""`
- **Map image format.** PrimaGIS is able to produce the map image in different formats and you can choose the most appropriate one for your application. Please note that not all browsers are able to render TIFF formats although PrimaGIS supports them.
- **Spatial reference system.** The spatial reference system (SRS) defines the default projection for the map. Your spatial data may be in other projections and PrimaGIS will automatically reproject your data to the projection defined in this field. Reprojecting on-the-fly may hurt your performance and you should consider keeping everything in the same projection. The value for this field may either be an EPSG code or whitespace separated PROJ.4 parameters.  
**Example values:** `"EPSG:4326"`, `"proj=latlong datum=WGS84"`
- **Map units.** The units used for the map. This depends on the selected reference system.
- **Default map extent.** The default extent (view of the map) that users will see before they modify their map session by navigating the map. The

“Set default view” navigation button will also reset the map to this extent, unless users have overridden this in their sessions. The value for this field consists of four whitespace separated values – *minx miny maxx maxy* – given in the projection defined above in “Spatial reference system”.

**Example values:** ”-180 -90 180 90”

- **Default size.** The default map image size in pixels given as a “[width]x[height]” string.  
**Example values:** ”500x500”, ”1024x768”
- **User selectable sizes.** PrimaGIS allows you to provide user selectable sizes for the map image. You can also provide a description for the size which will be used in the map UI. Each selectable size must be on its own line and given as “[width]x[height] description”. Note! You should include the default size in the selectable sizes or otherwise users are unable to return to it.  
**Example values:** ”800x400 Wide map”
- **Geo-aware types.** It is possible to restrict the content types are allowed to be used as data for this map. The select box allows you to choose the types that can be used with the map (within `PrimaGISDataLayers`). You have to make sure that all the types that you allow implement the `IGeoAware` interface.

## 7 Adding new layers

Map features can be divided into logical groups called layers. The final map image is then rendered by overlaying the layers in a specified order to create the final composite image.

Both `PrimaGISLayer` and `PrimaGISDataLayer` objects depend on existing ZCO Layer objects. `PrimaGISLayer` defines a simple one-way relationship with a ZCO Layer object which uses the spatial data from a ZCO Data Store object. `PrimaGISDataLayer` defines a two-way relationship with a ZCO Layer object by acting also as a Data Store for the ZCO Layer. Figure 4 shows the differences between the two.

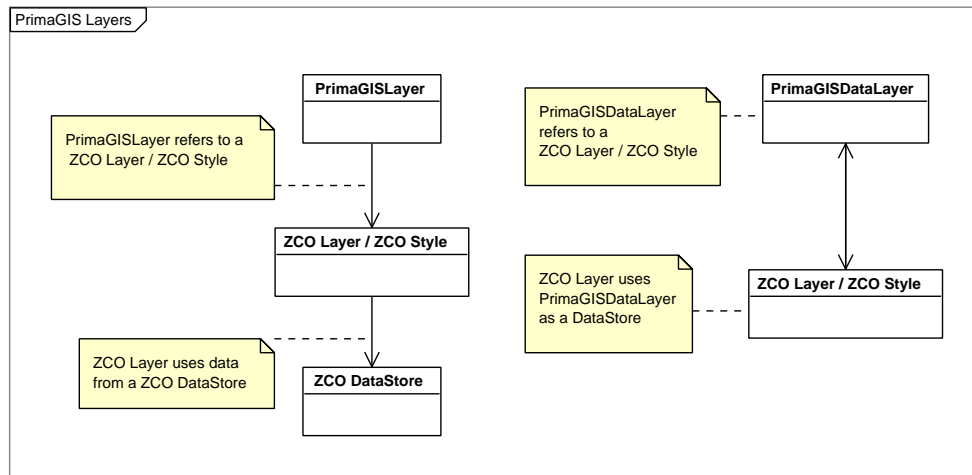
Before creating new `PrimaGISLayer` or `PrimaGISDataLayer` objects you need to configure ZCO Layers. Please refer to the ZCO documentation for details.

TODO: Shall we include the instructions for creating ZCO DataStores/Layers/Styles/Symbolizers here?

### 7.1 PrimaGISLayer

To create a `PrimaGISLayer` object, go to a `PrimaGISMap` object in your Plone site and use the “Add new item” pull-down menu to create a new `PrimaGISLayer` object. `PrimaGISLayer` has the following attributes you can use to configure the layer.

Figure 4: Differences between PrimaGISLayer and PrimaGISDataLayer



Created with Poseidon for UML Community Edition. Not for Commercial Use.

- **Title.** This is the default Dublin Core Title field available in all Plone content types.
- **Styled Map Layer.** This allows you to select which ZCO Layer the *PrimaGISLayer* refers to. The list of available ZCO Layers is automatically generated from the available ZCO Layer / ZCO Style combinations in the ZCO hierarchy associated with the current *PrimaGISMap* object. Already used ZCO Layers are also filtered out so you will only see layers that are not yet used within the map.
- **Display by default.** This lets you decide whether the layer should be rendered by default. When disabled the user must explicitly ask the layer to be rendered. It is recommended to disable this for slow layers (e.g. layers requiring lots of processing or additional network requests such as WMS and WFS layers) so the initial map will render quickly.
- **Show icons in legend.** This lets you decide whether the map legend should contain icons for the features within this layer.
- **Hidable.** This lets you decide whether the user should be allowed to show/hide this layer.

The view template for the *PrimaGISLayer* object will show a preview of the layer that you can use to make sure it is properly configured.

## 7.2 PrimaGISDataLayer

To create a *PrimaGISDataLayer* object, go to a *PrimaGISMap* object in your Plone site and use the “Add new item” pull-down menu to create a new *PrimaGISDataLayer* object. In addition to the attributes available in *PrimaGISLayer*, *PrimaGISDataLayer* has the following additional attributes you can use to configure the layer.

- **Path to the data container.** Since *PrimaGISDataLayer* acts also as a Data Store for the associated ZCO Layer it needs to store the spatial features somewhere. This field allows you to define the location of the spatial data. *PrimaGISDataLayer* is a containerish object so you can store the spatial data within it and indicate this by using “.” (dot without the quotes) as the path. Alternatively you can use any other location by providing an acquirable path to the external container containing the spatial data.  
**Example values:** “zco”, “../path/to/somewhere”
- **Spatial Reference System (SRS).** The spatial reference system for this *PrimaGISDataLayer*. For efficiency you should have all the data in the same projection as the *PrimaGISMap*. PrimaGIS will reproject the data automatically so it is possible to use any projection supported by the underlying PCL library. The value for the SRS can be either an EPSG code or whitespace separated PROJ.4 parameters.  
**Example values:** “EPSG:4326”, “proj=latlong datum=WGS84”
- **Spatial data type.** The type of spatial data for this *PrimaGISDataLayer*. This can be one of *Point*, *Line* or *Polygon*.
- **Default content type.** PrimaGIS allows administrators to create new spatial features (IGeoAware objects) through the map by selecting the appropriate layer and clicking at a specific location on the map. This field determines the type of the IGeoAware content object that will be created when a new feature is added through the map for this layer. It is possible to use other IGeoAware content types as data for this layer (as long as they are of the same spatial data type as defined above), but these need to be created by hand.

## 8 Adding spatial content

One of the ideas behind PrimaGIS is the ability to add Plone content to the map. In order to do this we need to somehow add spatial information (coordinates) to the content objects. PrimaGIS provides two approaches for accomplishing this which are introduced below.

## 8.1 IGeoAware objects

PrimaGIS provides an interface called *IGeoAware* that content types may implement to become usable with PrimaGIS. The *IGeoAware* interface defines methods for accessing the feature geometry and meta-data (that is visible in pop-up windows).

This is an easy and straight-forward approach if you are developing your own content types or you are willing to modify existing, possibly 3rd party, code. It is also possible to inherit an existing content type and extend it by adding the *IGeoAware* methods to it. The *GeoAwareObjects*[5] product provides many concrete examples on how to either extend existing content types or to create your own spatially enabled types.

## 8.2 GeoArchetypesProxy

It is not always possible or feasible to modify or extend existing content types and in these cases it is better to store the spatial data external to the content objects. PrimaGIS comes with an Archetypes based content type called *GeoArchetypesProxy* which implements *IGeoAware* and stores POINT geometry information. It uses the Archetypes reference mechanism to connect the existing content objects to the spatial data. The “European capitals” layer in the demo map is an example of *GeoArchetypesProxy* use.

# 9 Exporting data

*PrimaGISDataLayers* are able to serialize their data into two different XML based formats: GML [6] and KML [10]. GML is a generic markup language for describing spatial information and KML is a GML-like markup language used by Google Earth [8].

The serialization process is implemented using Zope Page Templates and you can access the serialized form by simple appending either “data.gml” or “data.kml” to the URL of a *PrimaGISDataLayer* object to get a GML or KML document. The *PrimaGISDataLayer* object has also tabs for accessing the serialized forms that are visible to Plone administrators.

## 9.1 Geography Markup Language (GML)

The GML export feature creates a simple GML document of the *PrimaGISDataLayer* data. This can be used with tools like **ogr2ogr** [14] to convert the data to other formats. Below is an example how to convert a *PrimaGISDataLayer* to a shapefile.

```
# Get the PrimaGISDataLayer as a GML document
$ wget http://server/.../primagisdatalayer/data.gml

# Convert the data to ESRI Shapefile
```

```
$ ogr2ogr -f "ESRI Shapefile" data.shp data.gml
```

You can use any appropriate method to save the GML document on your file system before running **ogr2ogr**. If you wish to modify the GML output you can customize the Page Template at *PrimaGIS/skins/primagis/data.gml.pt*.

Currently it is not possible to use PrimaGIS as a WFS feature source but this will be addressed in future versions.

## 9.2 Keyhole Markup Language (KML)

Currently *PrimaGISDataLayers* are able to produce simple KML documents which make it possible to use the spatial features defined in your Plone site with Google Earth. You can add the *PrimaGISDataLayers* to your Google Earth client by creating a new Network Link in GE and pointing it to the URL of your *PrimaGISDataLayer* appended with “/data.kml”, similar to the one below

```
http://server/.../primagisdatalayer/data.kml
```

If you wish to modify the KML output you can customize the Page Template at *PrimaGIS/skins/primagis/data.kml.pt*.

## 10 Customizing the PrimaGIS User Interface

Like the rest of Plone, the PrimaGIS UI is customizable using two approaches: Cascading Style Sheets (CSS) and the Zope Page Templates.

### 10.1 Cascading Style Sheets (CSS)

PrimaGIS uses an external CSS file – *primagis.css.dtml* – to apply styling to the map view. This can be customized to apply your own styles or to disable certain UI features.

### 10.2 Zope Page Templates (ZPT)

The PrimaGIS UI is modularized using METAL [13] macros which make it easy to rearrange the UI components in a different layout. The main layout is defined in the *primagis-view.pt* page template which simply calls the available macros. Customizing this file will allow you to rearrange the UI components by simply changing the order they appear in the template. The *map\_templates.pt* page template contains the actual macros and can be modified to do more detailed customizations.

## 11 Developing IGeoAware content types

One of the great advantages of PrimaGIS over simple web mapping applications is the possibility to integrate normal Plone content easily with the maps. The glue between PrimaGIS maps and Plone content is the *IGeoAware* interface that acts as a contract between the two. The interface aims to be unobtrusive and not get in the way of the normal functionality of your content type.

The following subsections introduce the *IGeoAware* interface and two approaches you may take when implementing it in your own projects.

### 11.1 IGeoAware interface

The *IGeoAware* interface consists of four methods which are described below.

- **getGeometry()**. This is the most important method and it is responsible for returning a PCL geometry object containing the spatial data for the implementing object. The geometry object should be an instance of either *Point*, *LineString* or *Polygon* from *cartography.spatial*. Please refer to the PCL documentation for more details. This is the only method that deals with spatial data and the other three methods handle only related meta-data.
- **getGeoTitle()**. Returns the title of the spatial feature in the implementing object. This is rendered as the title of the Javascript pop-up window when a user hovers above this feature.
- **getGeoDescription()**. Returns the description of the spatial feature in the implementing object. The description may include rich-content (images, hyperlinks, videos, etc) using arbitrary XHTML markup. The description is shown in the body of the Javascript pop-up window when a user hovers above this feature.
- **getGeoURL()**. Returns an URL to a related resource. The URL can be of any form understood by web browsers and it will be linked into the imagemap hotspot (<area /> tag). To disable the hotspot link the method should return *None*<sup>12</sup>.

### 11.2 Inheriting from GeoAwareMixin

PrimaGIS provides a mixin module, *GeoAwareMixin*, to help with implementing the *IGeoAware* interface. The module contains three mixin classes and Archetypes schemas, one for each each spatial type.

The procedure for using the mixin classes and schemas is the same for all types. You should inherit from the mixin class in your own class and append the mixin schema to your own schema as shown in the pseudo code excerpt below.

---

<sup>12</sup>This is sometimes useful when there is no single URL available that represents the feature adequately. You can also provide (possibly multiple) links in the Javascript pop-up body returned by the *getGeoDescription()* method

```
class YourOwnType(PrimaGISMixin):
    schema = YourOwnSchema + PrimaGISMixinSchema
    ...
```

### 11.2.1 PointMixin and PointMixinSchema

The *PointMixinSchema* adds two new FloatFields (*geoX* and *geoY*) to your content type that will hold the X and Y coordinates for the object. The *PointMixin* class implements the interface methods in the following manner.

- **getGeometry()**. Returns a *cartography.spatial.Point* object initialized with the values from *geoX* and *geoY*.
- **getGeoTitle()**. Returns the Dublin Core Title of your object.
- **getGeoDescription()**. Returns the Dublin Core Description of your object.
- **getGeoTitle()**. Returns the absolute URL to your object.

### 11.2.2 LinestringMixin and LinestringMixinSchema

The *LinestringMixinSchema* adds a new LinesField (called “linestring”) to your content type that will hold a list of (X,Y) coordinate pairs for the object. The *LinestringMixin* class implements the interface methods in the following manner.

- **getGeometry()**. Returns a *cartography.spatial.Linestring* object initialized with the values from the *linestring* field.
- **getGeoTitle()**. Returns the Dublin Core Title of your object.
- **getGeoDescription()**. Returns the Dublin Core Description of your object.
- **getGeoTitle()**. Returns the absolute URL to your object.

### 11.2.3 SimplePolygonMixin and SimplePolygonMixinSchema

The *SimplePolygonMixinSchema* adds a new LinesField (called “simplePolygon”) to your content type that will hold a list of (X,Y) coordinate pairs for the object. The coordinate pairs must form a closed polygon by having the same value as the first and last item in the list. The *SimplePolygonMixin* class implements the interface methods in the following manner.

- **getGeometry()**. Returns a *cartography.spatial.Polygon* object initialized with the values from the *simplePolygon* field.
- **getGeoTitle()**. Returns the Dublin Core Title of your object.

- **getGeoDescription()**. Returns the Dublin Core Description of your object.
- **getGeoTitle()**. Returns the absolute URL to your object.

### 11.3 Building from scratch

The mixin classes help with creating simple spatially enabled content types, but for more control and better results you will need to implement *IGeoAware* by hand. It is also possible to use both approaches at the same time by inheriting from the mixin classes and overriding the appropriate methods.

Please refer to the GeoAwareObjects [5] product for more hands-on examples on how to implement *IGeoAware* in your own custom content types.

## 12 Resources and additional information

- PrimaGIS web site at <http://www.primagis.fi/>. The site contains a live demo and additional documentation and information on using PrimaGIS. If you register as a member for the site you can also access the IRC logs of the #zco channel which contain answers to many questions.
- IRC channel #zco at [freenode.net](http://freenode.net) used by developers and users of the PrimaGIS stack to discuss project related issues. This is a good place to get immediate help with any problems you have with PrimaGIS.
- ZCO and PrimaGIS project page at <http://trac.gispython.org/projects/zope>. The project page uses the versatile Trac [21] project management software which includes a built in wiki, repository browser and a ticket manager among other things.
- PCL project page at <http://trac.gispython.org/projects/PCL>.
- PCL, ZCO and PrimaGIS mailing lists at <http://lists.gispython.org/>. Currently all the projects at <http://gispython.org> share a common “community” mailing list that may be used to discuss the projects and related issues. There are also lists for SVN commit messages and ticket updates for those interested in the day-to-day development events.
- Josh Livni’s introductory screencasts on using PrimaGIS available at <http://www.livniconsulting.com/primagis.html>.

## References

- [1] Cartographic Objects for Zope. <http://zcologia.org/zco/>. 4
- [2] CMFSin. <http://plone.org/products/cmfsin>. 9
- [3] FWTools. <http://fwtools.maptools.org/>. 13
- [4] GDAL Geospatial Data Abstraction Library. <http://gdal.maptools.org/>. 4
- [5] GeoAwareObjects. <http://trac.gispython.org/projects/zope/wiki/GeoAwareObjects>. 9, 25, 29
- [6] Geography Markup Language. [http://en.wikipedia.org/wiki/Geography\\_Markup\\_Language](http://en.wikipedia.org/wiki/Geography_Markup_Language). 25
- [7] GEOS Geometry Engine Open Source. <http://geos.refrations.net/>. 4
- [8] Google Earth - A 3D interface to the planet. <http://earth.google.com/>. 25
- [9] Javascript Scalebar for MapServer. <http://mapserver.commenspace.org/tools/scalebar/>. 9
- [10] Keyhole Markup Language. [http://www.keyhole.com/kml/kml\\_doc.html](http://www.keyhole.com/kml/kml_doc.html). 25
- [11] Mapserver. <http://ms.gis.umn.edu/>. 4
- [12] Matplotlib – Python 2D plotting library. <http://matplotlib.sourceforge.net/>. 4
- [13] METAL – Macro Expansion for TAL. <http://www.zope.org/Wikis/DevSite/Projects/ZPT/METAL/>. 26
- [14] OGR Simple Feature Library. <http://ogr.maptools.org/>. 4, 25
- [15] Plone: A user-friendly and powerful open source Content Management System. <http://plone.org/>. 4
- [16] PostGIS – Spatial extension to PostgreSQL. <http://postgis.refrations.net/>. 6
- [17] PROJ.4 Cartographic Projections Library. <http://proj.maptools.org/>. 4, 13
- [18] Python Cartographic Library. <http://zcologia.org/cartography/>. 4
- [19] simplejson – Python JSON encoder / decoder. <http://cheeseshop.python.org/pypi/simplejson>. 12

- [20] SWIG – Simplified Wrapper and Interface Generator. <http://www.swig.org/>. 15
- [21] Trac – Integrated SCM & Project Management. <http://www.edgewall.com/trac/>. 29