
Ecole Pratique des Hautes Etudes


Université Paris 8

**Rapport de Stage :**

# Architecture SIG pour la mise en ligne de la carte de la répartition mondiale des régions arides

par Rina Ralison

Tuteurs :
Prof. Marie-Françoise COUREL et Dr. Hala BAYOUMI

Responsable pédagogique :
Prof. Marc BUI

Master 2 Professionnel
Géomatique, Géomarketing et Multimédia (G2M)

Paris, juin 2014

**Remerciements** :

Nous tenons à remercier Madame Marie-Françoise COUREL (EPHE), Madame Hala BAYOUMI (CEDEJ-EG) et Monsieur Marc BUI (Laboratoire CHArt) pour la confiance sans réserve qu'ils nous ont accordé.

## Introduction

Ce rapport de stage est réalisé dans le cadre d'une convention de stage entre d'une part l'EPHE–CEDEJ-EG et d'autre part l'Université Paris 8 pour la participation à la mise en place d'une couche cartographique, provenant de la carte de la répartition mondiale des régions arides (Unesco 1977), sur une page web.

Cette participation du stagiaire de Paris 8 comporte trois composants principaux. Les deux premiers sont la revue des technologies de programmation les plus efficaces pour la production de cartes et l'étude d'adaptabilité d'un serveur utilisant l'une de ces technologies pour la réalisation de l'objectif final visé par ce stage. Le résultat de ces deux premiers objectifs sont décrits dans la première partie de ce rapport. La méthodologie utilisée pour la réalisation de cette étude est basée sur de la recherche et de l'étude documentaire.

Le troisième objectif est la mise en œuvre de la publication de la couche géographique vectorielle fournie par l'EPHE–CEDEJ-EG sur une page web. Cet objectif se divise en deux sous-objectifs, respectivement documentés dans la deuxième et troisième partie de ce document. Ainsi la deuxième partie de ce document détaille les procédures d'installation des serveurs tandis que la dernière partie est essentiellement orientée vers l'analyse des besoins, l'ingénierie de l'implémentation et la rédaction des scripts.

# 1. Etat de l'art sur les technologies existantes

La publication online de la carte de la répartition mondiale des régions arides (Unesco 1977) nécessite la mise à plat des technologies existantes susceptibles d'être utilisées de façon efficace et pérenne. Pour cela il faut s'assurer d'une part que l'infrastructure à mettre en place puisse répondre aux exigences fonctionnelles immédiates de l'application, d'autre part que l'infrastructure puisse permettre à l'application d'offrir des fonctionnalités futurs pas encore définies dans le cahier des charges. Ces deux points justifient ce chapitre qui fait un tour d'horizon des outils de programmation socles en SIG.

## 1.1. *Les API de manipulation des images et des vecteurs*

Les API (Application Programming Interfaces) sont des briques de fonctions qui permettent de concevoir des applications de haut niveau. Ces bibliothèques de fonctions sont conçues pour être réutilisables sur différents chantiers logiciels. Dans le cadre de la mise en place de l'infrastructure de diffusion de la carte de la répartition mondiale des régions arides (Unesco 1977), il est important de prendre en compte la qualité des composants principaux sur lesquels s'appuieront les logiciels et middleware de l'architecture.

### 1.1.1. GDAL / OGR / GEOS

L'API GDAL (Geospatial Data Abstraction Library) regroupe des fonctions d'écriture, de manipulation et de lecture de cartes raster. En l'état, GDAL ne permet pas la création de styles complexes comme le remplissage avec hachure, de plus pour lire des fichiers vectoriels, GDAL a besoin de la librairie OGR.

OGR est l'ancien acronyme de "OpenGIS Simple Features Reference Implementation", mais comme la librairie ne remplit pas les exigences de la spécification la bibliothèque il porte le titre officiel de "OGR Simple Features Library". L'API OGR est destiné à lire, écrire et manipuler des fichiers vectoriels.

Bien qu'on puisse utiliser GDAL sans faire appel aux fonctionnalités de OGR, et inversement, les cas d'utilisation les plus courants nécessitent généralement l'utilisation des deux blocs logiciels.

Le tableau ci-dessous représente quelques exemples de cas d'utilisation où les deux composants doivent être utilisés simultanément.

| | GDAL | OGR |
|---|---|---|
| Vectorisation d'un canal de raster issu de télédétection | Lecture du raster en amont. | Écriture des vecteurs en aval |

| | | |
|---|---|---|
| Assemblage d'un vecteur et d'une image en une seule image | Lecture d'une image en amont<br>Rasterisation de l'image et du vecteur. | Lecture du vecteur en amont. |
| Rasterisation de Shapefile | Écriture du fichier image final. | Lecture du Shapefile en amont. |

GDAL et OGR ont été écrits en C/C++, néanmoins de nombreuses interfaces natives sont disponibles en différents autres langages de programmation permettant à ces derniers de bénéficier des fonctionnalités de GDAL et de OGR. Ainsi les développeurs en Python, Perl ou Java peuvent utiliser les interfaces natives de GDAL et de OGR.

Les fonctionnalités de GDAL et de OGR étant limitées à la lecture et à l'écriture, d'autres librairies doivent venir à la rescousses pour d'autres fonctionnalités. On peut citer l'exemple de calcul de surface d'un polygone : OGR est capable de lire les sources de données vectoriel ainsi que d'y ajouter des éléments sans pour autant permettre de faire des calcul sur ces élément. Heureusement la librairie GEOS vient combler ce vide : cette librairie permet la réalisation d'opérations et la vérification de prédicats sur des objets géographiques. GEOS est l'acronyme de Geometry Engine - Open Source.

L'ensemble GDAL/OGR/GEOS offre tellement de fonctionnalités, que des développeurs en Python ont créé un module nommé Shapely permettant, entre autres, de faire des requêtes SQL spatial sans recourir à une base de données spatiale.

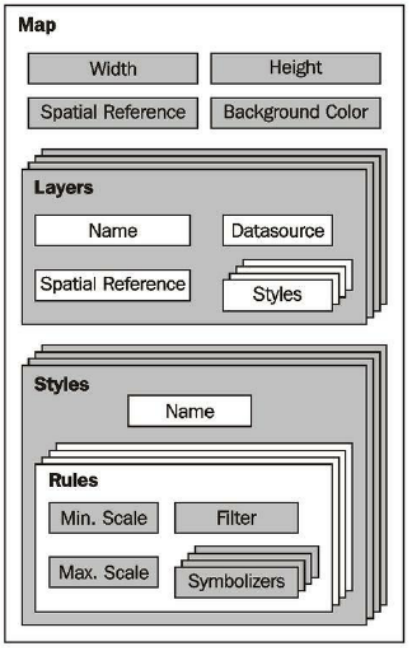
### 1.1.2. Mapnik

L'objectif premier de Mapnik est de "faire de belles cartes". Mapnik lit et écrit des fichiers grâce à un mécanisme d'utilisation de plugin externes, certains plugin comme les chargeurs de Shapefile d'ESRI sont livré et installés en standard.

Bien qu'écrit dans les mêmes langages C/C++ que GDAL/OGR, Mapnik est entièrement indépendant des librairies précédemment mentionnées, néanmoins il peut utiliser leurs fonctionnalités et inversement. Initialement cette librairie est donc conçue pour être utilisée dans des programmes écrits en C/C++, mais des interfaces natives complètes existent permettant aux développeurs en Python de l'exploiter pleinement.

Le principe de fonctionnement de Mapnik reste assez simple, les données géographiques son chargées programmatiquement de façon transparentes via un plugin. Ces données peuvent être modifiées localement avant d'être rasterisé en image. Jusque là on retrouve une certaine similitude de fonctionnement avec GDAL/OGR, mais la différence fondamentale découle du fait que Mapnik permet d'adjoindre aux données à rasteriser en image, *des stylisations avancées* sur les contours, les surfaces et les labels.

Les styles Mapnik sont écrits en XML, le tableau ci-dessous résume les éléments composants une carte selon Mapnik :




D'après ce schéma une carte Mapnik est composée de plusieurs couches, chaque couche peut se voir attribué plusieurs styles. La force de Mapnik réside dans le fait que son architecture permet de styliser un "feature" d'une couche au cas par cas, en fonction de règles, et non pas l'application d'un style globale.

Par exemple, chaque label d'un nom de pays appartenant à un même Shapefile peut être stylisé en fonction d'une règle pré-défini. Ce mécanisme a inspiré l'élaboration du SLD (acronyme de Stylesheet layer descriptor) dont les codes pour ce projet seront commentés plus loin (3.2.1).

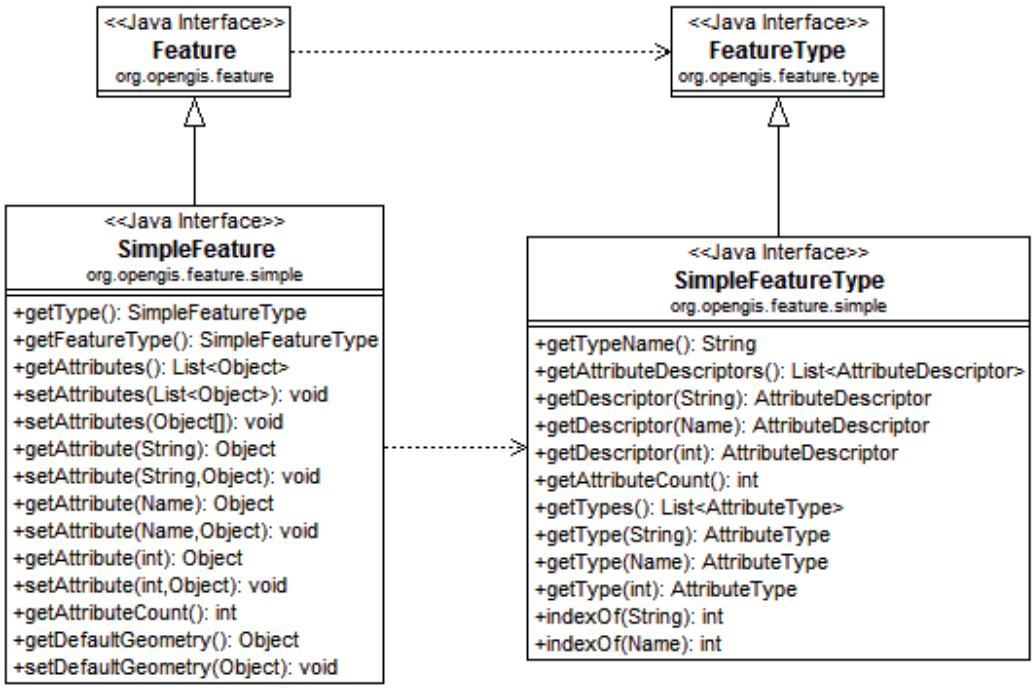
### 1.1.3. Geotools

Geotools est un API de programmation SIG entièrement écrit en Java. L'objectif de ses créateurs est de développer une couche logicielle SIG solide pouvant tourner sur n'importe quelle plate-forme hébergeant une machine virtuelle Java édition standard. Geotools est donc entièrement autonome et ne fait aucun appel à un quelconque interface native.

Comme tout autre programme interprété, les distribuables sont les mêmes d'un plate-forme à l'autre. Cela évite la re-compilation et les risques d'incompatibilité lors d'un changement de plate-forme.

Dans la mesure où il utilise des librairies fiables comme Java Topology Suite (équivalent Java de GEOS) et Imageio, Geotools peut à la fois *transformer des coordonnées en différentes SRS, modifier un WKT en WKB et inversement, calculer des distances et même rasteriser des images (exemple en Annexe 1.3). Grâce à l'adjonction de SLD, Geotools est capable de faire de belles cartes comme Mapnik.*

Geotools est resté conforme à la norme GeoAPI 2.2 [1] de l'OGC, de ce fait sa manipulation est aisée pour ceux qui sont familiers avec ces normes ISO et OGC.
Ainsi, à titre d'exemple, le standard ISO 19107 relatif aux features est défini par les interfaces du package org.opengis.feature.simple de GeoAPI dont l'interface principale est SimpleFeature. Ci-dessous le diagramme UML de la classe SimpleFeature selon les standards OGC ():




Dans la pratique, des instances de SimpleFeature sont fournis par des implémentations de FeatureReader, ce dernier est une interface et implémentation propre à Geotools[2].

A travers ces exemples, on découvre que la puissance de Geotools réside dans sa facilité d'utilisation étant donné qu'il respecte les schémas de normalisation bien établi et reconnu par le monde du SIG.

### 1.2. *Les systèmes de persistance de données*

Afin de choisir le système de persistance le plus adapté au projet de publication en ligne de la carte de la répartition mondiale des régions arides (Unesco 1977), il convient de passer en revue les principaux choix existants.

[1] Depuis sa sortie en 2011, GeoAPI 3.0 n'a pas été modifié.
[2] Le code d'exemple suivant montre le lien entre Geotools, GeoAPI et le standard ISO19107 reconnu par l'OGC : http://docs.codehaus.org/display/GEOTOOLS/Data+Reading

### 1.2.1. Fichier Shapefile d'ESRI

Un fichier Shapefile est en réalité composé d'un minimum de trois fichiers avec le même préfixe. Ainsi la différence entre les fichiers qui composent un shapefile sont :
- le premier fichier porte le suffixe shp, il contient les données géométriques
- le deuxième fichier porte le suffixe dbf, il contient les données attributaires, ce fichier est compatible avec la base de données DBaseIII+
- le troisième fichier porte le suffixe shx, il fait le lien entre le fichier shp et dbf et stocke l'index de la géométrie diminuant ainsi les temps de recherche (ou de jointure) spatiale

D'autres fichiers optionnels peuvent s'adjoindre aux précédents. Entre autres on peut citer :
- le fichier avec le suffixe prj, définit la référence géodésique (plus précisément le système de référence spatial) du shapfile dont il porte le nom.
- le fichier avec le suffixe mxd, qui définit un projet ArcGIS. Un projet ArcGIS peut contenir plusieurs couches (shapefile ou autres). Ce fichier définit aussi la projection de la carte formée par les couches.
- le fichier avec le suffixe lyr, est un fichier de style pouvant être attaché à une couche se trouvant dans le projet.

Le format shapefile est reconnu par l'OGC (le sigle pour Open Geospatial Consortium), comme format de données vectorielles et utilisé entre autre par la plupart des SIG bureautique comme ArcGIS et QGIS en tant que format de travail par défaut.

Le format shapefile est déjà utilisé dans le cadre de ce projet de mise en ligne des cartes des aridités, mais juste comme format de sortie et d'échange de données numérisées par l'équipe égyptienne. Ci-dessous le schéma de la couche :

Vue Schéma de la table: aridity

| Nom du champ | Type de données |
|---|---|
| GEOMETRY | Geometry |
| OBJECTID_1 | Integer |
| Shape_Leng | Double |
| Shape_Area | Double |
| area_name | String |
| ARIDITE1 | String |
| d_TYPE | String |
| type | String |

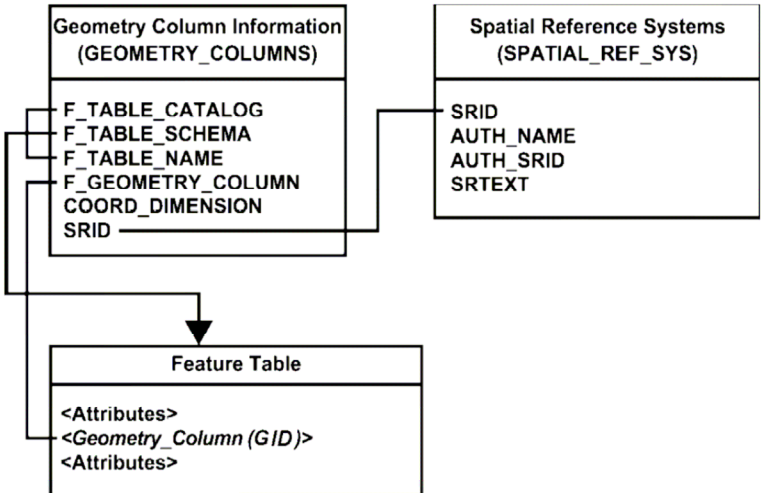
Cette table contient uniquement la couche des différentes zones arides définie par les climatologues comme l'indice de Penman / Évapotranspiration. Les couches sur la température hiver-été, le nombre de mois sec et le régime de pluie hiver-été n'entre pas dans le cadre de ce stage.

### 1.2.2. Base de données Spatiale

De manière schématique, une base de données spatiale permet de stocker et de retrouver des informations géographiques. Théoriquement, n'importe quelle base de données SQL peut répondre à cette exigence sauf qu'en réalité les standards imposent certaines contraintes fonctionnelles un peu plus poussées.

Selon l'OGC, une base de données spatiale doit entre autres proposer deux tables obligatoires, en plus des tables définies par l'utilisateur :
la table GEOMETRY_COLUMNS répertorie toutes les colonnes géographiques créées par l'utilisateur
la table SPATIAL_REF_SYS contient la totalité des systèmes de coordonnées et des transformations géométriques.

Comme le montre le diagramme ci-dessous, chaque colonne géométrique dans la table créée par l'utilisateur (Feature Table) doit être référencé dans GEOMETRY_COLUMNS, chaque référence est liée à la table SPATIAL_REF_SYS par la clé SRID.




Ainsi le SRID des éléments de chaque colonne géographique ainsi que le type d'objet qu'il contient doivent être connu et défini à l'avance.

Un certain nombre d'implémentation respectant cette exigence existent comme Oracle Spatial ou SQL Server mais parmi les logiciels OpenSource, seul PostGresql avec PostGIS possède les meilleurs rapports qualités prix.

La base de données spatiales est donc le nœud fédérateur des échanges entre SI et SIG de l'entreprise.

### 1.2.3. Autres formats populaires : GeoJson et KML

Si l'utilisation de base de données spatiales est acquise dans la mise en ligne la carte de la répartition mondiale des régions arides (Unesco 1977), les fonctionnalités à venir nécessitent la bonne compréhension des flux de données standards.

Se plaçant au dessous des API et en dessus du niveau 7 de la couche ISO OSI (acronyme de International Standardisation Organisation - Open Systems Interconnection ), GeoJson

est un format entièrement indépendant des langages de programmations et des protocoles réseaux.

Bien que de nombreux langages de programmation dispose de parser capable de décortiques un flux GeoJSON, ce dernier est surtout consommé par les navigateurs web. En effet la classe javascript JSON faillite la lecture de ce type de flux par les navigateurs web. D'autres facteurs ont accéléré l'adoption de GeoJSON par les programmeurs, entre autres :

- la rapidité de transfert du flux GeoJSON, comme tout autre format JSON
- la vulgarisation d'utilisation de l'objet Javascript XMLHttpRequest et des framework AJAX
- l'accroissement de l'utilisation de SLD par les framework Javascript de webmapping.

Exemple de GeoJson :

Le flux GeoJson ci-dessous, obtenu sur le site sytadin.fr, est un exemple typique de l'utilisation de ce format de donnée :

Requete :
http://www.sytadin.fr/gp/SegmentTooltipProvider?zoomLevel=0&clicX=609220.0012207&clicY=2425260.0006104

Reponse :


```
{
 "type": "Feature",
 "geometry": {
  "type": "Point",
  "coordinates": [608970.52, 2424879.18]
 },
 "properties": {
  "type": "NA",
  "info": "Travaux sur A4 sens Province-Paris<br />1 voie neutralis&eacute;e &agrave; droite<br />Circulation sur 4 voies"
 },
 "crs": {
  "type": "EPSG",
  "properties": {
   "code": "27572"
  }
 },
 "bbox": [534000, 691000, 2346000, 2472000]
}
```


Le flux ci-dessus a été transféré en 230ms. et est destiné à être affiché dans une infobulle sur un point bien précis de la carte.

Le format KML ( acronyme de Keyhole Markup Langage), un dialecte de XML, a été popularisé par le logiciel Google Earth. KML n'est exprimé qu'en WGS84 (plus connu sous le SRS EPSG:4326), une reprojection est donc obligatoire pour l'afficher dans une carte. Le format KML encapsule aussi les informations de stylisation en plus des informations géographiques et attributaires.

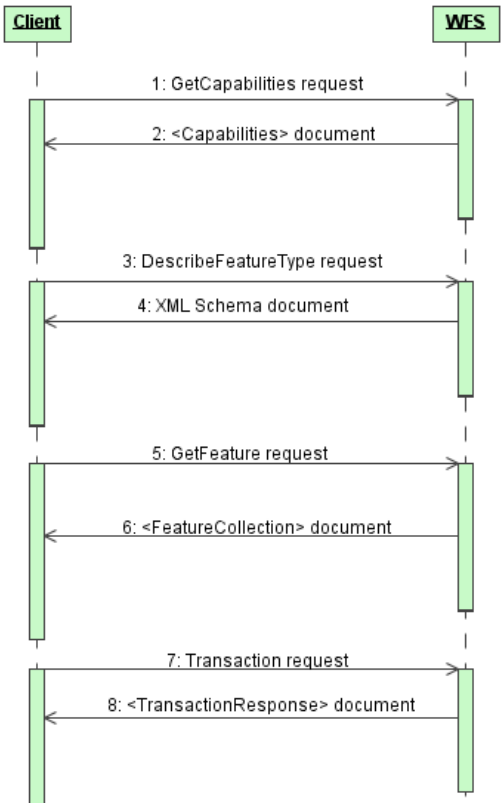
Contrairement à GeoJSON, le format KML est approuvé par l'OGC.

### 1.3. *Les standards autours de la génération d'images ou de vecteurs*

Cette avant dernière partie du chapitre 1 aborde les principaux normes et outils disponibles et dont la connaissance est indispensable dans la récupération et la manipulation des images et des vecteurs géographiques.

#### 1.3.1. WFS et WMS

Les standards WMS (web map service) et WFS (web feature service) sont des mécanismes permettant la récupération de cartes (respectivement image et vectoriel) à partir de requête http.
Chaque service peut être accessible en SOAP ou en simple HTTP avec des paramètres KVP préalablement définis par l'OGC. Ci-dessous le diagramme de séquence des quatre services WFS prévus par le norme OGC :




Pour WFS la reponse à GetFeature peut être un flux dans les formats suivants :

- GML1
- GML2
- KML
- GeoJson

Un serveur WMS n'a que deux opérations SOAP obligatoires :

- GetCapabilities

- GetMap

La réponse à cette dernière opération peut être un flux dans les formats suivants :

- Geotiff
- SVG
- PNG

### 1.3.2. SLD

Un Stylesheet layer descriptor (abrégé en SLD) est un document XML qui contient la description des styles applicables aux couches et aux features. Un document SLD peut être utilisé par un serveur WMS pour styliser une carte lors de la rasterisation d'image ou sur un client WFS pour mettre en forme des vecteurs.

Exemple de SLD:

```
<StyledLayerDescriptor version="1.1.0">
  <NamedLayer>
    <Name>Rivers</Name>
    <NamedStyle>
      <Name>CenterLine</Name>
    </NamedStyle>
  </NamedLayer>
  <NamedLayer>
    <Name>Roads</Name>
    <NamedStyle>
      <Name>CenterLine</Name>
    </NamedStyle>
  </NamedLayer>
  <NamedLayer>
    <Name>Houses</Name>
    <NamedStyle>
      <Name>Outline</Name>
    </NamedStyle>
  </NamedLayer>
</StyledLayerDescriptor>
```

Sur un serveur WMS il est possible de définir pour chaque couche de données géographiques un SLD par défaut, mais cela n'empêche pas le client d'adjoindre à la requête SOAP GetMap d'une couche, un document SLD spécifique ou l'URL de celui-ci.

Le SLD permet donc une réelle séparation entre d'un côté les informations géographiques et les données attributaires et de l'autre côté les informations de présentation et de mise en page.

### 1.3.3. Les choix possibles parmi les serveurs WMS les plus populaires

Plusieurs choix d'implémentation se sont présentées pour le déploiement en ligne des cartes des aridités de l'Unesco de 1977. Les produits ci-dessous ont tous été candidat.

### 1.3.3.1. GeoServer

GeoServer a été retenu pour la diffusion des cartes des aridités pour plusieurs raisons. En premier lieu ce produit est capable de prendre en considération plusieurs formats de données aussi bien en entrée qu'en sortie comme le montre le schéma ci dessous :


En second lieu, il est entièrement écrit en Java et s'appuie en très grande partie sur la libraire Geotools mentionnée précédemment.

La conséquence de cette dépendance est non négligeable car cela implique que d'une part son installation ne nécessite aucune compilation ni installation de librairie native, ce qui facilite son déploiement sur des systèmes vendus sans compilateur comme Windows, d'autre part la pérennité de l'outil est relativement assurée grâce à la collaboration des communautés de développeurs Geotools et GeoServer.

GeoServer a aussi l'avantage d'être très intuitif, aussi bien à l'installation qu'à l'utilisation, les fonctionnalités de base du logiciel ne nécessitent aucune configuration par script car elles sont accessibles depuis une interface graphique.

GeoServer a l'avantage d'être livré avec une distribution d'OpenLayers prêt à l'emploi.

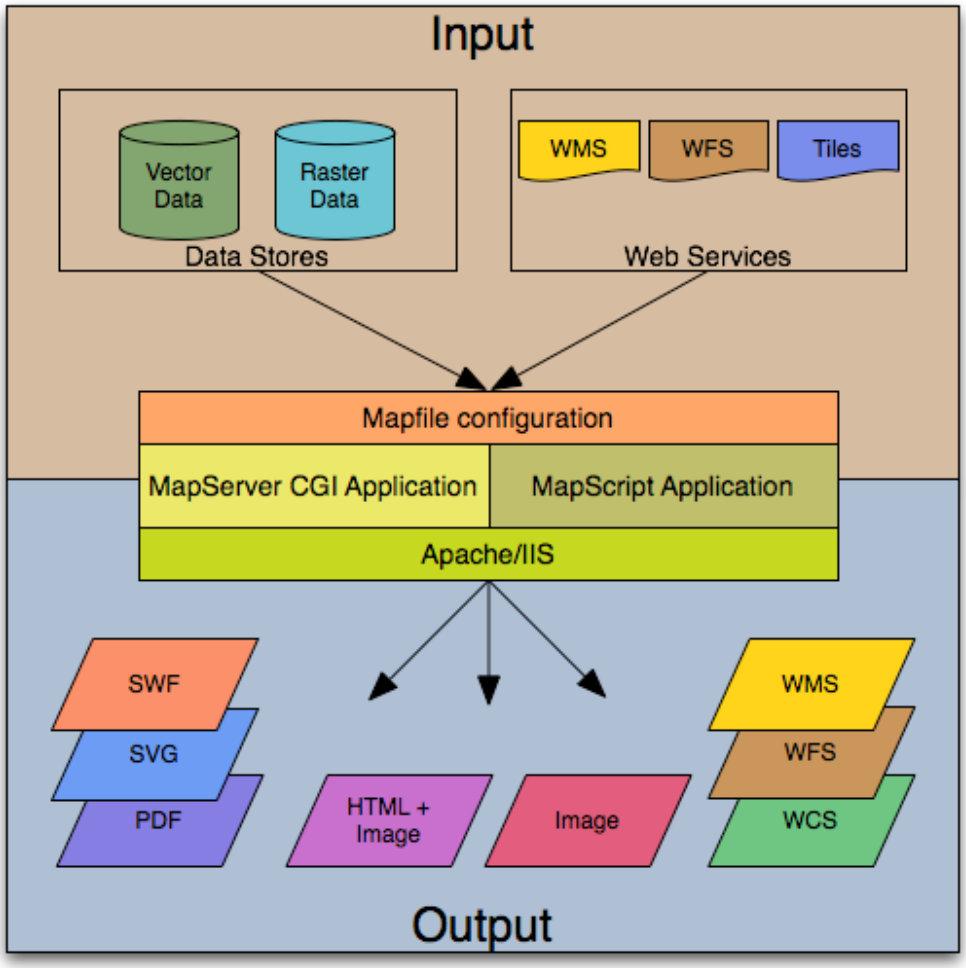
### 1.3.3.2. MapServer

MapServer est une implémentation gratuite des normes WMS/WFS, il a la réputation d'être rapide. En effet MapServer est entièrement écrit en C/C++ par une équipe de chercheurs de l'université de Minnesota.
Pour ne pas réinventer la roue, MapServer utilise les librairies classiques :

- GDAL/OGR
- GEOS
- Proj4

La présence de ces dépendances sur la machine où se trouve l'instance MapServer est indispensable.

Ci-dessous le schéma d'architecture globale d'une installation MapServer standard :




Comme le montre le schéma ci-dessus, MapServer est un CGI exécutable appelé par un serveur Apache ou IIS. A chaque requête, en fonction des variables transmis en http et des configurations internes de MapServer, l'application choisit un fichier de description Mapfile pour confectionner une carte.

Le fichier Mapfile est le cœur de MapServer. Il définit les relations entre les objets et signale à MapServer où sont localisées les données et définit comment les choses doivent être dessinées. Un exemple de Mapefile est donné à l'annexe 1.4

Le fichier Mapfile contient donc la localisation des sources de données à utiliser pour la confection de la carte et éventuellement il inclut directement le style à appliquer aux objets vectoriels. Cela étant MapServer est aussi capable d'utiliser les SLD.

MapServer n'offre aucune interface graphique, la configuration doit se faire par :

- Positionnement de variable d'environnement système
- Positionnement de variable dans la configuration d'Apache httpd et répercutée dans le CGI Mapserver
- Envoi de variable http, réinjecté comme variable d'environnement dans Mapserver par le processus httpd
- Utilisation de Mapfile

#### 1.3.3.3. Ogcserver

Ogcserver est un serveur WMS entièrement écrit avec Python mais utilise l'interface Python de Mapnik et d'autres interfaces de dépendances écrits en C/C++. Ce dernier élément rends l'installation de Ogcserver relativement difficile sur des plate-formes ne disposant pas de compilateur C/C++.

Même si Ogcserver ne dispose pas d'interface graphique, la structure de son système de fichier est relativement simple pour permettre la modification des fichiers de configuration ainsi que l'ajout d'un fichier de style XML Mapnik.

# 2. Mise en œuvre de PostGIS et de GeoServer

GeoServer a donc été sélectionné pour produire les couches WMS de la carte des aridités de 1977 éditées par l'UNESCO. En parallèle avec GeoServer, une instance PostGIS doit être installée.

## 2.1. *Installation de PostGIS et chargement des données*

PostGresql, avec le module PostGIS, a été choisi comme mécanisme de persistance des données pour GeoServer.

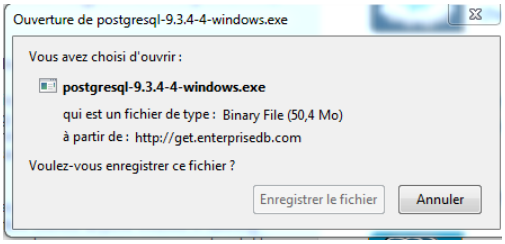
### 2.1.1. Installation de PostGIS

La procédure décrite dans cette sous-partie est propre à Windows, pour Unix et Mac OSX des packages pré-compilés prêts à l'installation existent.

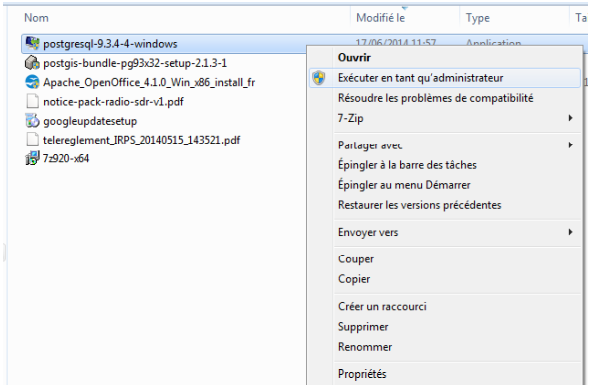
Le numéro de version la plus récente de PostGIS doit être recherche sur le site http://PostGIS.net ainsi que le numéro de version de PostGresql compatible avec ce module.

| |
|---|
| Cette précaution est importante surtout sur des pleteformes nécessitent une re-compilation des sources car une version donnée de PostGIS ne peut fonctionner qu'avec une version spécifique de PostGresql. |

Une distribution compilée de la version de Postgres trouvée ci-dessus est disponible à l'URL suivant : http://www.enterprisedb.com/products-services-training/pgdownload , c'est le seul fichier à télécharger manuellement:



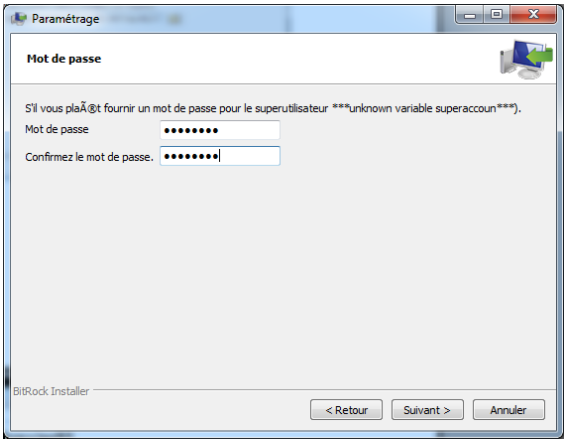

Une fois l'exécutable téléchargée il faut l'exécuter en tant qu'administrateur, ouvrant ainsi la fenêtre marquant le début de l'installation :

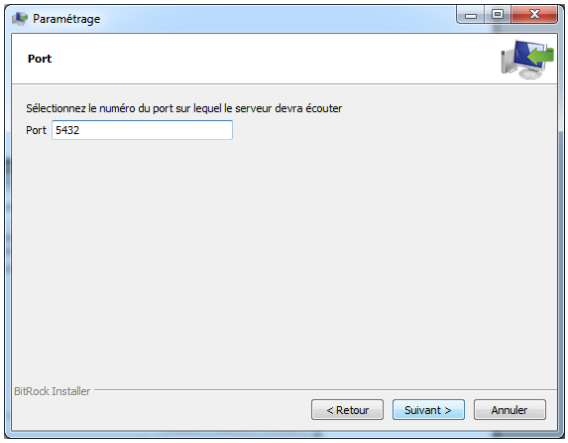




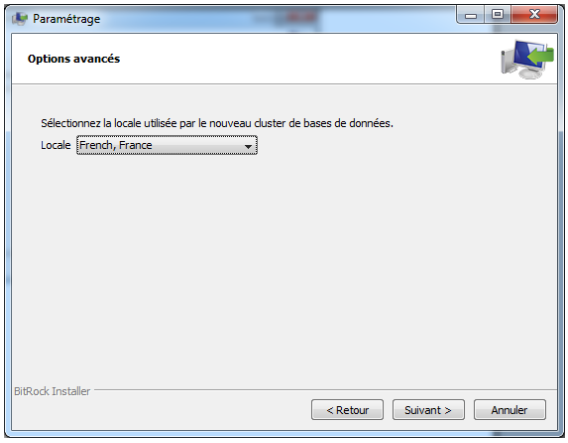
A partir de l'apparition de l'écran ci-dessus il suffit de cliquer sur le bouton suivant de toutes les autres fenêtres qui se succèdent jusqu'à l'apparition de l'écran ci-dessous.

Cette fenêtre permet la saisie et la première affectation du mot de passe du super utilisateur "postgres", ce dernier à tout les droits sur l'instance du serveur. La fenêtre d'après permet de saisir le numéro de port sur lequel le serveur est accessible. Le numéro de port par défaut est 5432 mais il peut être modifié après l'installation :



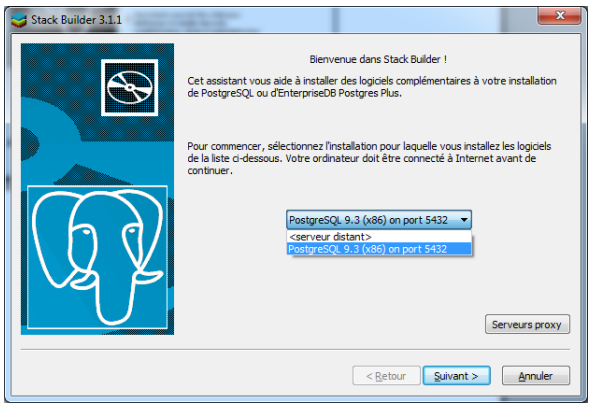



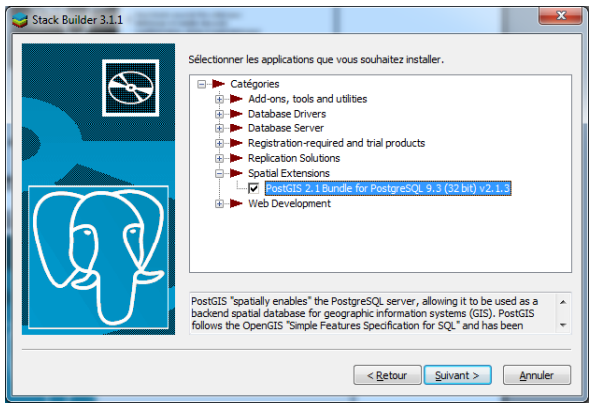
La fenêtre d'après ci-dessous permet de choisir la localisation (langue, date ...), suivie de la fenêtre de fin de l'installation de PostGresql :





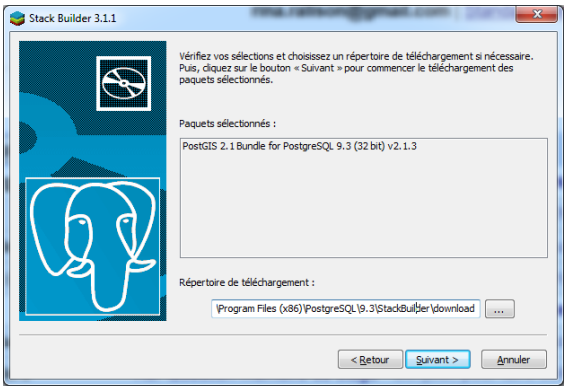
Cette dernière fenêtre marque le succès de l'installation de PostGresql et le début de la sélection, rechargement et installation des autres modules.
L'installation des modules (PostGIS...) commence par la sélection de l'instance du

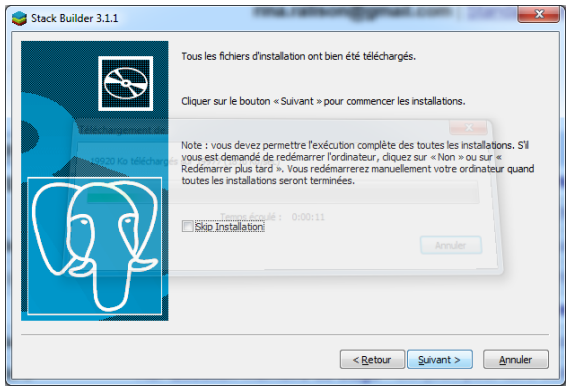


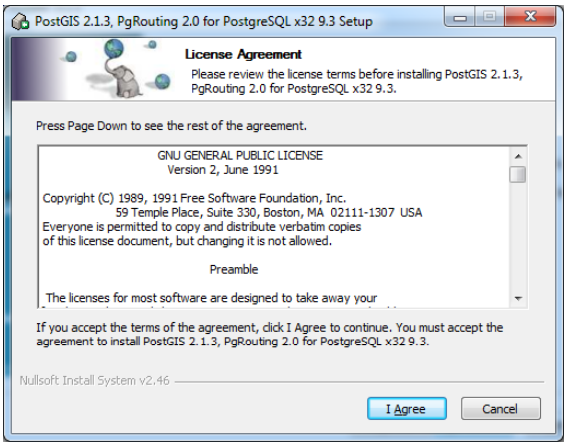

serveur sur lequel il va s'installer et des modules à télécharger, notamment PostGIS:

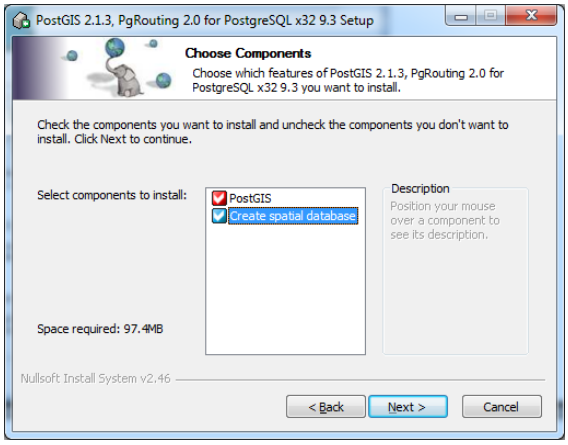
La fenêtre ci-dessous indique la localisation du répertoire de stockage local du module téléchargé suivi de la fenêtre de notification de la fin du téléchargement :



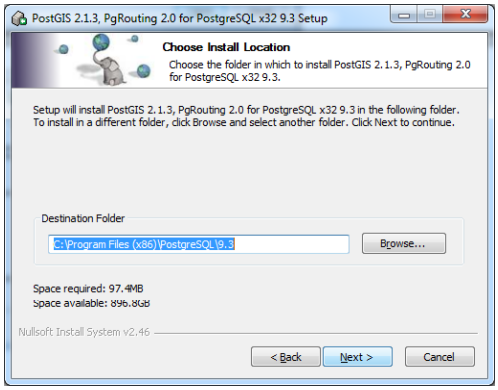



La première fenêtre ci-dessous affiche la licence d'utilisation et de distribution de PostGIS suivi de la fenetre indiquant le souhait de création d'une base de données spatiale:

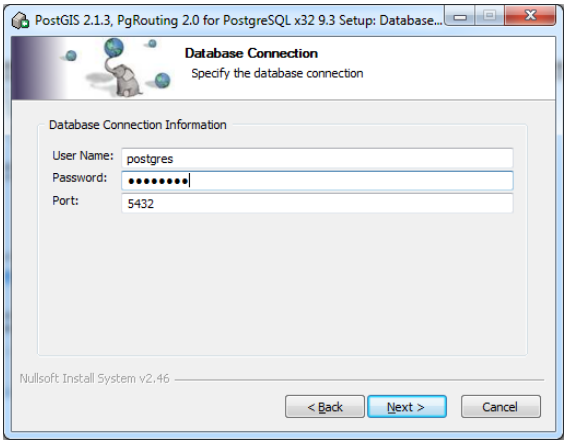


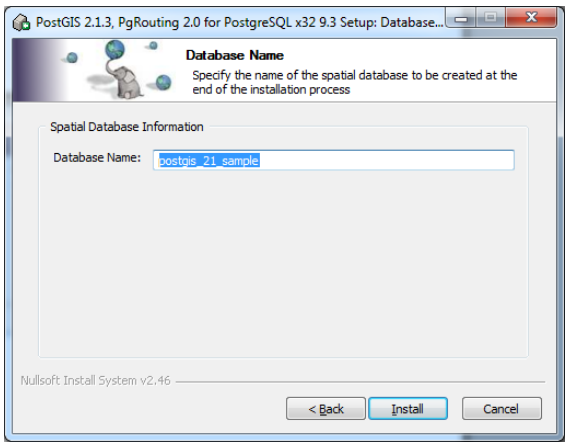




Le module PostGIS téléchargé sera décompressé et installé dans le répertoire indiqué ci-dessous :

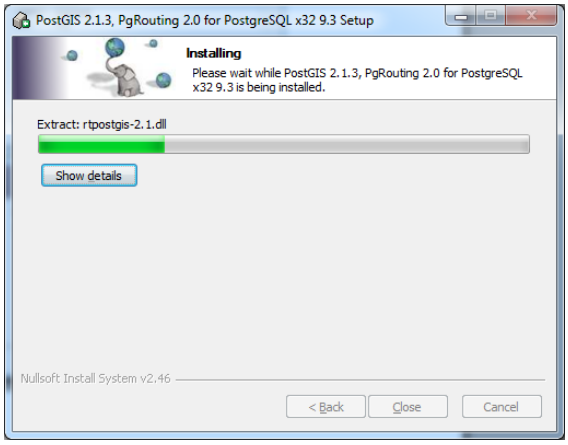


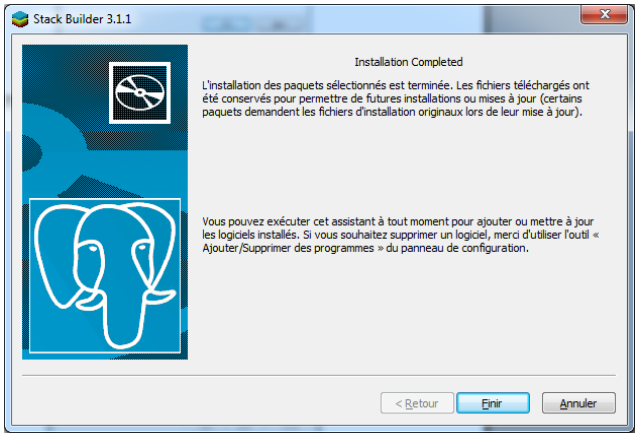

A la suite de cela le StackBuilder collecte les informations de connexion à PostGresql pour la création d'une base de données spatiale :



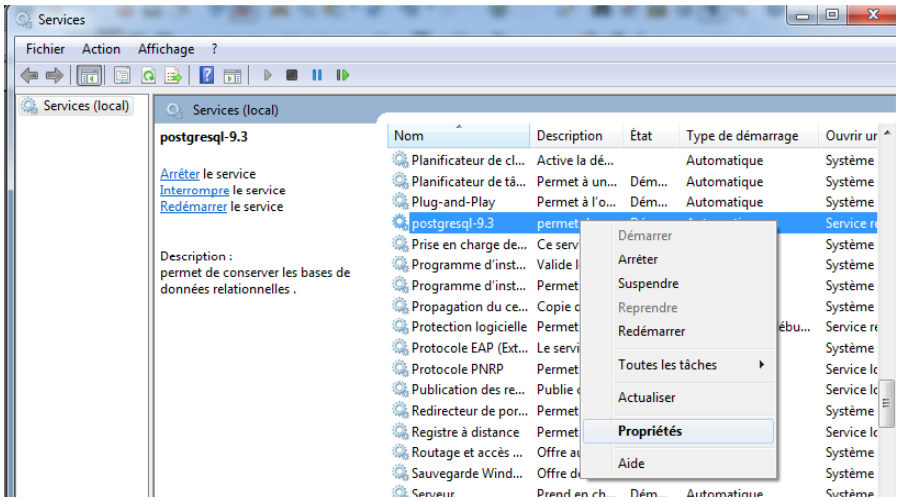



Plusieurs fenêtres contextuelles vont apparaître jusqu'à la fin de l'installation :



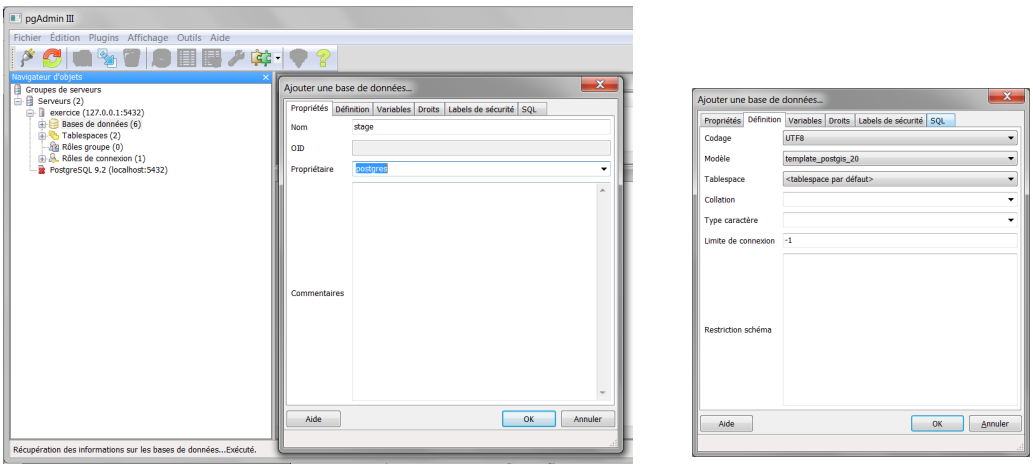




A ce stade PostGresql et PostGIS sont installés et le service doit être visible dans la liste des services :

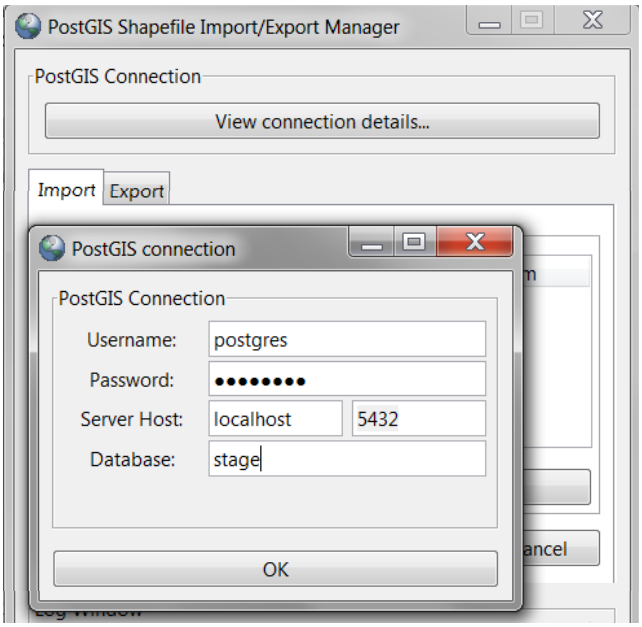



### 2.1.2. Chargement des données

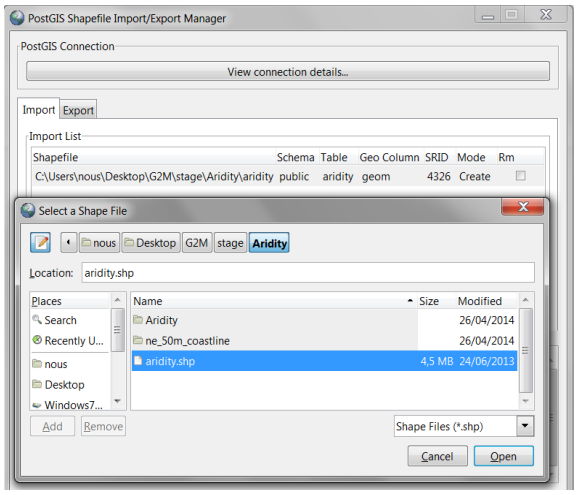
La première étape concerne la création d'une base de données spatiales nommée « stage », cela peut se faire avec commande shell createdb ou avec PgAdminIII outil installé par défaut. C'est cette dernière méthode qui est adoptée ici.




La deuxième phase consiste à charger des données dans PostGresql, plus précisément elle consiste à injecter dans une table « aridity » (même nom que le shapefile) la totalité du fichier des aridités mentionnées dans la partie 1.2.1. Pour cela il faut utiliser la commande shell shp2pgsql ou le programme graphique shp2pgsql-gui, c'est ce dernier qui est montré ici :

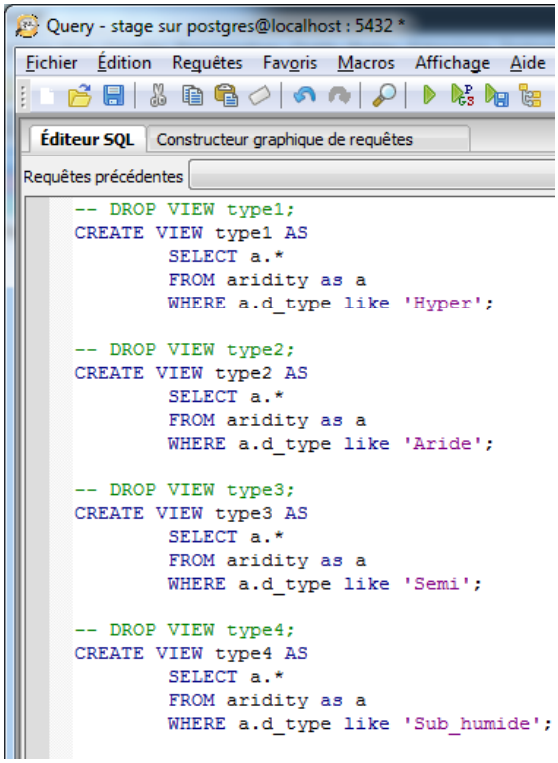


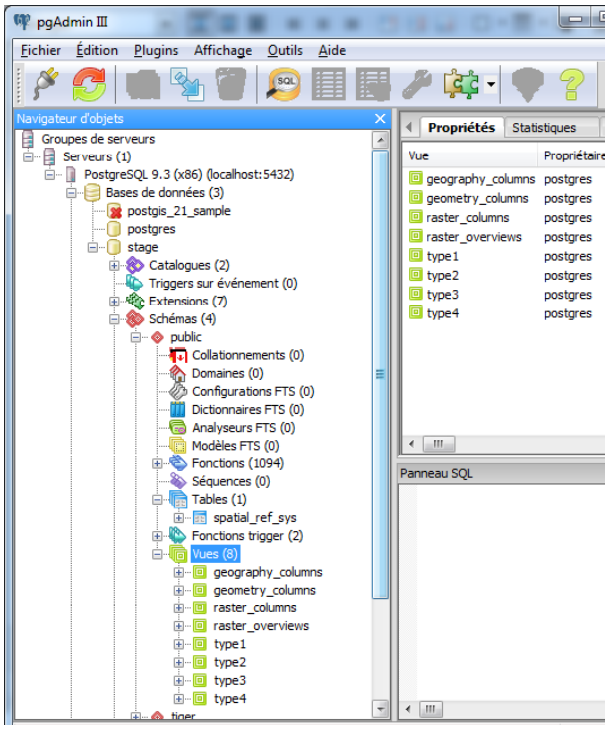

Conformément à l'écran ci-dessous, il est impératif de renseigner le SRID du Shapefile à charger, en l'occurrence 4326. Par défaut, la nouvelle table est placée dans le schéma « public ».




Cette étape est aussi une occasion pour charger dans la base de données le shapefile de la carte du monde téléchargeable sur le site Natural Earth Data[3]. Hormis le nom de fichier, la procédure à suivre est tout à fait similaire au capture d'écran ci-dessus. Cette couche servira de contours aux continents.

La deuxième étape consiste à la mise en place de quatre vues à partir des données se trouvant dans la table « aridity » précédemment créée.




[3] Le lien est : http://www.naturalearthdata.com/http//www.naturalearthdata.com/download/50m/physical/ne_50m_coastline.zip

La nomenclature des vues est :

- Hyper aride : type1
- Aride : type2
- Semi aride : type3
- Sub humide : type4

L'intérêt de la création de vue réside dans le fait que la modification de la table « aridity » est immédiatement reporté sur la vue, sans exécution d'un trigger ou de mécanisme programmatique externe.

Toutes les manipulations réalisées dans cette sous-partie peuvent être entièrement réalisées en ligne de commande.

## 2.2. Procédure d'installation de GeoServer

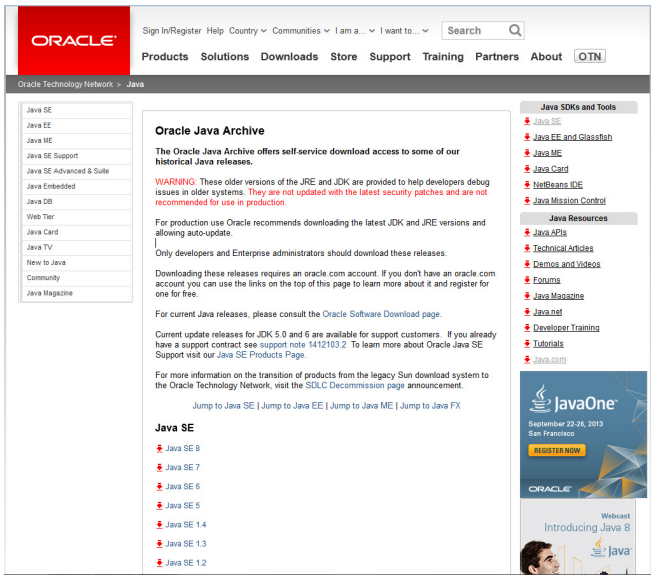
Une fois les données insérées dans la base, la phase suivante consiste à installer GeoServer, le pré-configurer et le démarrer.

### 2.2.1. Installation de Java

GeoServer requiert une version spécifique de « Java SE Runtime Environment », il est donc indispensable de rechercher ce numéro de version sur le site http://GeoServer.org

A ce jour, le version requise de Java est 1.6 ou supérieure, il peut être téléchargée sur le site de Oracle.

Il est préférable d'utiliser la version mentionnée sur le site de GeoServer, mais parfois il arrive que Oracle modifie rapidement en profondeur le contenu de ses nouvelles distributions de Java, dans ce cas il est nécessaire de consulter leur rubrique « Oracle Java Archive » pour pouvoir télécharger l'installation la plus adaptée.



A la fin de l'installation du « Java SE Runtime Environment », la commande shell « %JAVA_HOME%/bin/java –version » doit afficher le numéro de version correct :

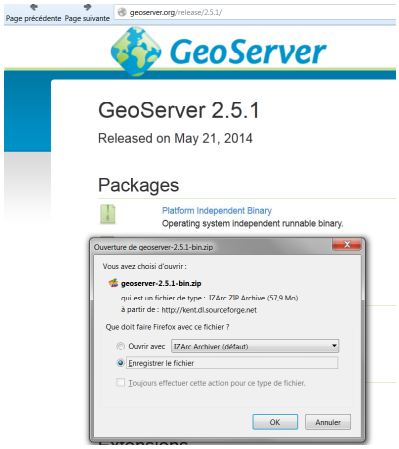
```
C:\Users\nous>%JAVA_HOME%/bin/java -version
java version "1.8.0_05"
Java(TM) SE Runtime Environment (build 1.8.0_05-b13)
Java HotSpot(TM) Client VM (build 25.5-b02, mixed mode, sharing)

C:\Users\nous>
```

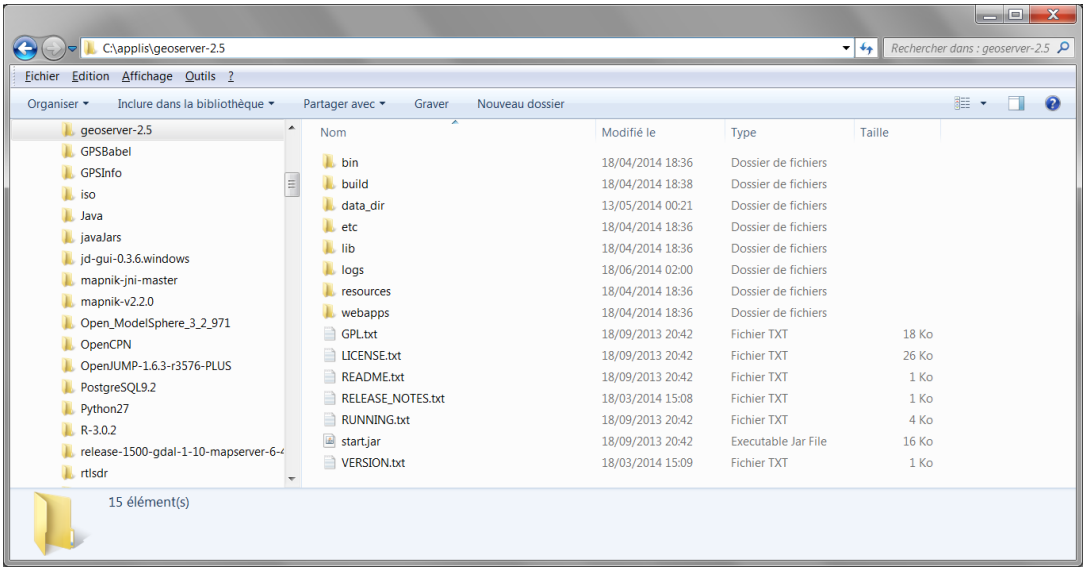
Si tel n'es pas le cas, il serait judicieux de désinstaller et de réinstaller Java.

### 2.2.2. Déploiement de GeoServer standalone

Une fois l'installation de Java correctement effectuée, la prochaine étape consiste à télécharger GeoServer sur le site officiel comme le montre l'écran ci-dessous :




Ensuite il suffit décompresser le fichier téléchargé dans un répertoire quelconque du système de fichier. Il faut simplement veiller à ce que le chemin complet vers ce répertoire ne comporte aucun caractère interdit (ni accent, ni espace…), ni de restriction d'accès en lecture, écriture et exécution.



La capture d'écran ci-dessus montre le contenu de l'archive de GeoServer une fois décompressé dans son répertoire d'instalation.

L'installation de GeoServer ne nécessite aucune édition de script de configuration. Cela étant deux variables d'environnement doivent être positionnés pour assurer son bon fonctionnement : GEOSERVER_HOME et GEOSERVER_DATA_DIR.

| | |
|---|---|
| GEOSERVER_DATA_DIR | %GEOSERVER%\data_dir |
| GEOSERVER_HOME | C:\applis\geoserver-2.5 |

Avant le démarrage de GeoServer, il convient de copier dans le répertoire désigné par GEOSERVER_DATA_DIR les fichiers ci-dessous :

| | |
|---|---|
| innerSubHumideBlanc.png | |
| innerSemiArrideBlanc.png | |
| barbul.svg |  <br>*barbul.svg*<br>751 octets<br>5 x 20 pixels |
| barbul2.svg |  <br>*barbul2.svg*<br>745 octets<br>4 x 8 pixels |

Ces fichiers images seront utilisés pour la stylisation.

Ces images on été créées spécialement pour les besoins de la mise en ligne des cartes des aridités de 1977. La création de ces images avec les outils Gimp et Inkscape ont été consommatrice en temps de travail mais ne sera pas détaillée dans ce document.

Les commandes ci-dessous permettent de démarrer GeoServer :

```
C:\>cd %GEOSERVER_HOME%/bin

C:\applis\geoserver-2.5\bin>startup.bat
```

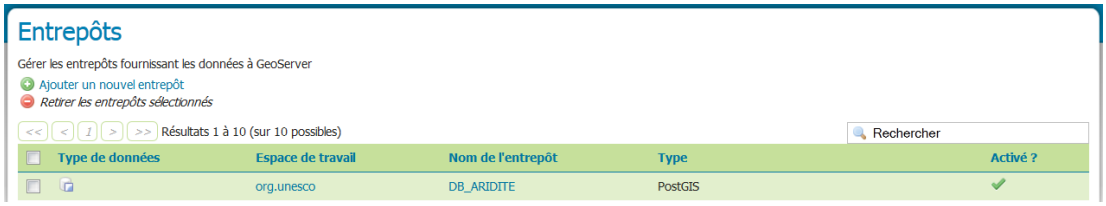
### 2.2.3. Paramétrage de GeoServer et intégration des SLD

Le paramétrage de GeoServer se fait intégralement via l'interface graphique accessible à l'URL : http://localhost:8080/geoserver/web/ accessible sur la même machine que GeoServer.
La première démarche à effectuer est l'authentification. L'identifiant par défaut est admin et le mot de passe associé est geoserver

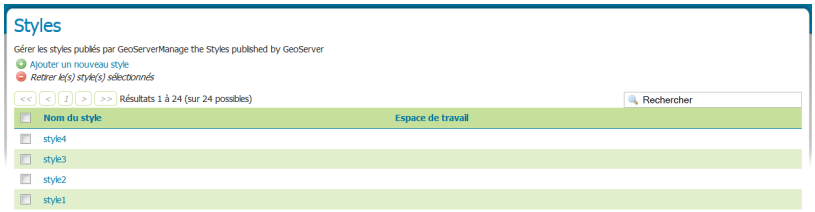
Pour la publication des cartes des aridités, seul le bloc « Données » nécessite quelques manipulations de paramétrage :


Le paramétrage de l'espace de travail s'effectue dans « Données / Espaces de travail ». Créer l'espace de travail « org.unesco »


Le paramétrage des sources de données se fait dans « Données / Entrepôts» : créer l'entrepôt PostGIS «DB_ARIDITE»

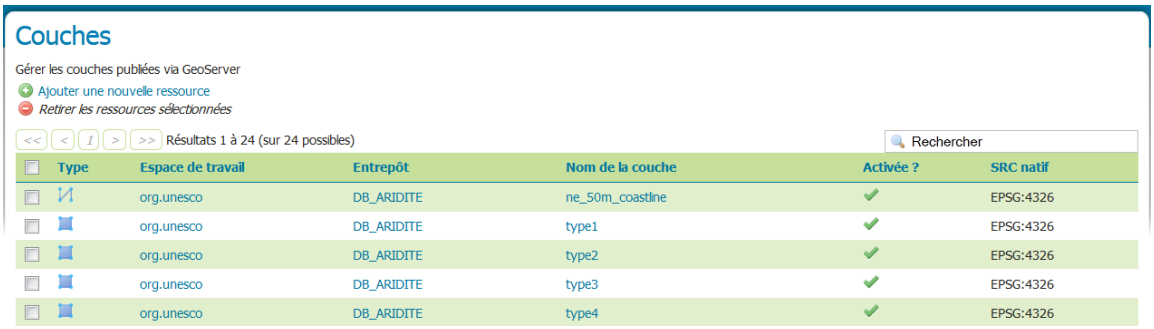



Le paramétrage des styles se fait dans « Données / Styles» :

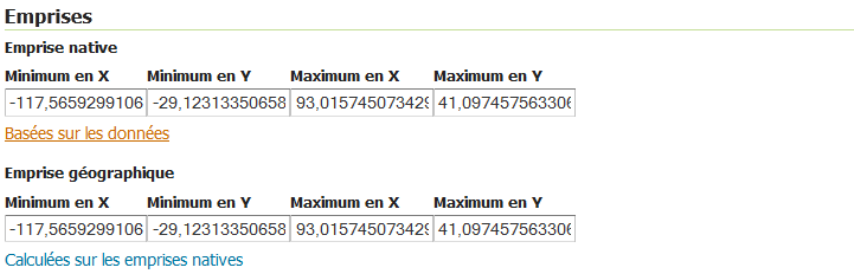



La création de style peut être faite de deux manières : en insérant directement le script du XML dans la fenêtre d'édition ou en chargeant un ficher SLD. Dans le cas présent c'est cette dernière option qui est utilisé puisque les fichiers SLD sont dorénavant disponibles. Il faut donc charger les fichiers style1.sld, style2.sld, style3.sld et style4.sld respectivement pour les styles style1, style2, style3 et style4

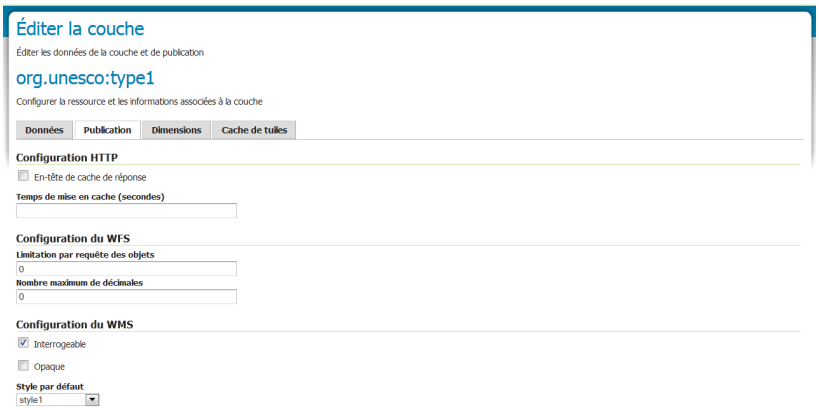
Le paramétrage des couches raster ou vectoriel se fait dans « Données / Couches» : il suffit de charger les cinq couches à partir de l'entrepôt PostGIS «DB_ARIDITE»




Lors de la création des couches ci-dessus il est nécessaire de définir les emprises de la couche. Pour faciliter cette opération il suffit de cliquer sur les liens « Basées sur les données » et « Calculées sur les emprises natives ».



Par ailleurs, dans l'onglet « Publication » de la fenêtre d'édition de la couche, il faut modifier le « Style par défaut » afin de permettre à la couche d'avoir son propre style.




# 3. Déploiement de la page de la carte des zones arides et commentaires des codes

Une fois l'infrastructure sous-jacente installée et les scripts SLD déployés, ce chapitre est consacré au déploiement de la partie IHM front de l'application et aux commentaires des scripts développés.

## 3.1. Création de l'archive embarqué de OpenLayers et test de l'IHM

Avant d'aborder la compilation de OpenLayers en elle même, cette partie revient sur quelques généralités concernant Openlayers.

### 3.1.1. Généralités sur OpenLayers

OpenLayers est une couche logiciel d'affichage cartographique client visualisable sur un navigateur Web écrit en Javascript. Parmi quelques logiciels ayant les même fonctionnalités, OpenLayers est le seul livré avec GeoServer.

Etant donné qu'il doit être pris en charge par un navigateur web, OpenLayers peut être intégré dans une page web par l'une des trois manières ci-dessous (avec des résultats fonctionnels identiques):

- par ajout de l'URL complet du site officiel d'OpenLayers
- par téléchargement et copie d'une version de l'arborescence des répertoires complets d'OpenLayers vers le site hébergeant la carte des zones arides
- par création et copie d'un fichier compact d'OpenLayers vers le site hébergeant la carte des zones arides

La première pratique n'a pas été retenue car elle oblige le navigateur du visiteur du site de la carte des zones arides à se connecter systématiquement sur le site de OpenLayers. Cela

peut conduire à une saturation du réseaux et des serveurs d'OpenLayers en cas d'affluence sur la page. Il en est de même pour la deuxième méthode car cela revient tout simplement à déplacer le problème précédent vers d'autres réseaux et d'autres serveurs.

La dernière technique permet de réduire notablement la charge de bande passante, voir supprimer l'accès réseaux, tout en ayant du code Javascript dans un fichier compact au lieu de multitude de codes Javascript disséminés dans d'innombrables répertoires. Cette technique est donc la plus appropriée.

### 3.1.2. Fabrication de l'archive unique OpenLayers.js

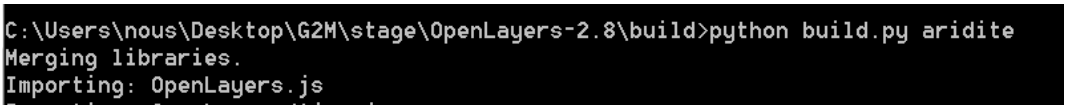
La première étape consiste à installer Python car ce dernier est l'utilitaire utilisé par OpenLayers pour la fabrication de l'archive final. Contrairement à OSX et Unix, l'utilitaire Python n'est pas installé par défaut sur Windows, il faut télécharger l'installateur MSI de la version compilée sur http://python.org/download

Une fois Python installé, il faut procéder au test suivant en commande shell suivant "python --version":


Si le test n'est pas concluant il est judicieux de désinstaller Python et de le réinstaller.

La deuxième étape consiste à télécharger l'archive zip de OpenLayers 2.8 sur le site http://www.openlayers.org et de le décompresser dans un répertoire sans limitation d'écriture ou de lecture.
Le code Javascript pour la carte des zones arides ont été écrits pour la version 2.8 et non la version 3 de OpenLayers, car à ce jour cette dernière version est encore en bêta.

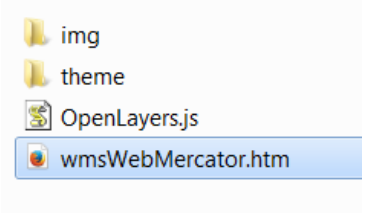
A la racine du répertoire où le fichier téléchargé est décompressé se trouve un répertoire nommé "build", toutes les opérations et manipulations qui suivent doivent se faire à l'intérieur de ce répertoire, notamment la copie de full.cfg vers aridite.cfg
Comme il n'y a pas de contrainte concernant la taille de l'archive à produire, et étant donné qu'aucune modification de la librairie OpenLayers n'a été effectuée, il n'est pas nécessaire de modifier le contenu du nouveau fichier aridite.cfg ni de modifier le contenu du répertoire "lib/OpenLayers". Il suffit donc de lancer la commande shell "python build.py aridite" comme le montre la capture d'écran ci-dessous :



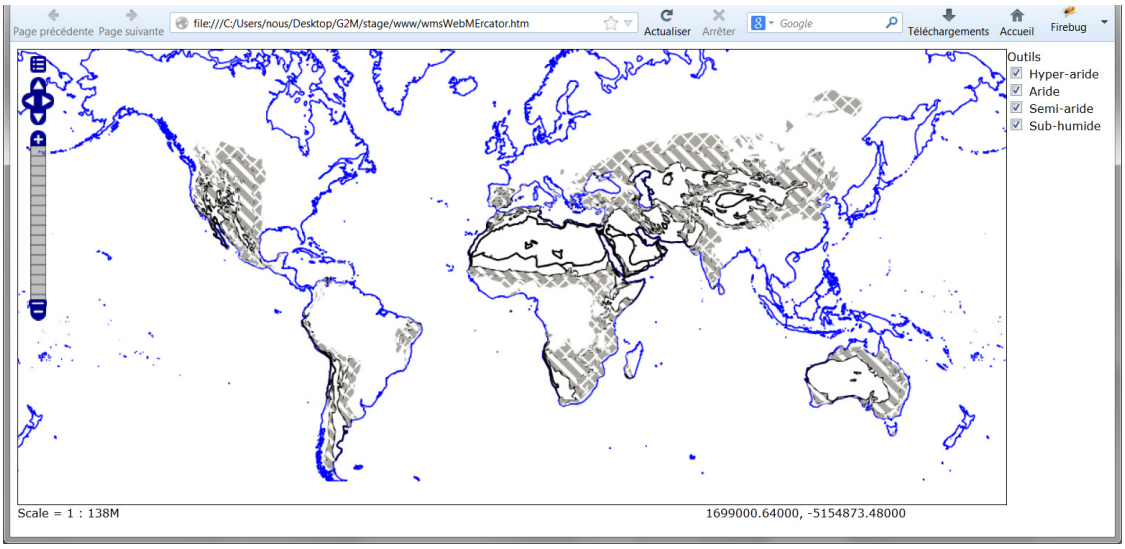

Au terme de ce processus, le fichier OpenLayers.js contiendra la totalité de la librairie.

### 3.1.3. Test de la carte des aridités

Le test de la carte des zones arides nécessite la constitution de l'arborescence ci-dessous dans un répertoire quelconque sur la machine hébergeant GeoServer :




Les répertoires img, thème et le fichier OpenLayers.js proviennent du répertoire de OpenLayers version 2.8. Le fichier wmsWebMercator.htm est la page principale du site de la carte des zones arides de l'UNESCO 1977. Ci-dessous la capture d'écran du produit final à l'ouverture du site :




## 3.2. Commentaires sur les scripts

Avant de conclure ce rapport, ce dernier sous-chapitre apporte des éclaircissements sur les différents scripts développés au cours de ce projet.

### 3.2.1. Les scripts de stylisation des zones

Visuellement, les zones hyper arides sont représentées par une ligne de contours entrecoupée par des barbules orientées vers intérieures du polygone. Cette stylisation appliquée à la carte des zones arides de l'UNESCO de 1977 doit impérativement être

conservée sur la carte en ligne. La première difficulté consiste à «réinventer » ce concept réalisé sur papier au format électronique.

La solution est donc de créer une image de barbule venant entrecouper régulièrement la ligne de contours de la zone. Comme SLD impose que l'équateur de l'image se cale le long la ligne de contours, il est impératif que la moitié du dessin de la barbule soit transparent invisible et que la moitié visible seule soit colorée en noir.

La possibilité de changer de niveau de zoom a soulevé un autre problème inconnu sur la carte papier : la visibilité des barbules en fonction du niveau de zoom. Bien que visible à tout les niveaux de zoom, la densité et la taille des barbules à l'écran doivent être modulées en fonction du niveau de zoom : plus l'échelle augmente, plus la taille et la densité à l'écran des barbules augmentent et inversement.

La réalisation de cette solution a conduit à créer deux images de barbule de tailles différentes. Au sein du SLD la sélection de l'image à afficher se fait à l'aide de la balise <Rule> et <MaxScaleDenominator>, l'image de la barbule affichée dépends donc de la règle applicable à l'échelle courante. Deux niveaux d'échelle sont visibles dans style1.sld.

Le style des zones semi-aride et sub-humide ont la particularité de laisser transparaître en partie la couleur sous-jacente de la carte (exprimant d'autres informations comme le régime de pluie hiver été...) tout en restant blanc opaque sur le restant du polygone. Ces stylisations complexes sont réalisées par la création d'images PNG avec transparence.

Les images utilisées pour la stylisation sont visibles dans le sous chapitre 2.2.2.

### 3.2.2. Commentaire sur les codes Javascript

Cette sous-partie décrit les points importants sur le code Javascript présents dans le fichier wmsWebMercator.htm (Annexe 1.1)

Ce fichier ne comporte qu'une seule fonction « init() » appelée lors du chargement de la page. C'est dans cette fonction que se fait l'instanciation de l'objet Map (cf. tableau ci-dessous), ce qui est intéressant est d'étudier de plus prés les paramètres utilisés pour cette création d'objet.

```
var bounds = new OpenLayers.Bounds( -21237508, -7360000, 21237508, 10217508);
                                        /* METRE left, bottom, right, top */
var options = {
   controls: [],
   maxExtent: bounds.clone(),
   restrictedExtent: bounds.clone(),
   maxResolution: "auto",
   projection: "EPSG:900913",
   units: 'm'
};
map = new OpenLayers.Map('map', options);
```

Le paramètre projection: "EPSG:900913"  indique que l'affichage finale de la carte utilise la projection web Mercator. Cette valeur peut être choisie de façon arbitraire, mais il est bien connue que la projection optimale de la carte du monde (hors pôles) est le web Mercator. Ce choix conduit à exclure de l'emprise maximum de la carte (bounds) les deux pôles et à exprimer les dimensions en mètre.

Par ailleurs, il n'est pas nécessaire de mentionner une quelconque type de projection dans les paramètres d'instanciation des couches WMS : la requête WMS utilisée comportera par défaut la projection de la carte (au cas présent EPSG:900913), à la charge du serveur d'effectuer les transformations géographiques nécessaires et de retourner la carte demandée à sa bonne projection même si la couche d'origine sur le serveur utilise un système de référence géographique différente.

En ce qui concerne la partie HTML de la page, il convient de noter que la dimension du bloc « div #map » de fait en dernier de façon itérative car elle dépend de la taille finale de l'image fournit par le serveur WFS sous forme de plusieurs tuiles.

## Conclusion

Les objectifs attendus par la convention de stage entre d'une par l'EPHE–CEDEJ-EG et d'autre part l'Université Paris 8 sont pleinement atteint.

En effet, après avoir passé en revue les différents API de base les plus populaires en programmation SIG ainsi que les serveurs utilisant ces API, on est arrivé à la conclusion que l'architecture basée sur PostGis, GoeServer et OpenLayers est le plus adaptée pour la partie web de la publication online de la carte de la répartition mondiale des régions arides. Ce choix est basé sur deux critères : la pérennité de la solution et sa disponibilité sur plusieurs systèmes d'exploitation.

Au cours du développement de l'implémentation, plusieurs problèmes liés aux exigences de stylisation cartographiques imposée par le modèle papier de la carte de 1977 ont vu le jour. Mais grâce à la flexibilité du modèle SLD ces difficultés ont toutes été résolues.

En l'état actuel de choses la diffusion online de la carte de la répartition mondiale des régions arides n'attends plus que l'intégration des autres couches géographiques en cours de vectorisation. Mais à moyen terme le plate-forme répondra à des cas d'utilisation innovants et de nouveaux besoins qui ne fera que renforcer la justification de l'architecture SIG client serveur adoptée dans le cadre de cette étude.

## Bibliographie


- A. Santiago Perez, *OpenLayers Cookbook,* Packt Publishing, 2012
- E. Hazzard, *OpenLayers 2.10 Beginner's Guide,* Packt Publishing, 2011
- S. Iacovella, B. Youngblood, *GeoServer Beginner's Guide,* Packt Publishing, 2013
- "GDAL API Tutorial" http://www.gdal.org/gdal_tutorial.html
- "GDAL: OGR API Tutorial" http://www.gdal.org/ogr_apitut.html
- "GEOS — OSGeo-Live 7.9 Documentation" http://live.osgeo.org/fr/overview/geos_overview.html
- "Shapely 1.3.2 Geometric objects, predicates, and operations" https://pypi.python.org/pypi/Shapely
- "GeoAPI alignment with standards" http://www.geoapi.org/2.1/
- "State of GeoTools" http://fr.slideshare.net/jgarnett/state-of-geo-tools-2012
- "GeoAPI interfaces 2.2-M2 API" http://www.geoapi.org/2.2/javadoc/index.html
- "Overview (Geotools modules 12-SNAPSHOT API)" http://docs.geotools.org/latest/javadocs/
- "ESRI Shapefile Technical Description" http://www.esri.com/library/whitepapers/pdfs/shapefile.pdf

# Annexes

## 1.1. Le code JavaScript

```
<script defer="defer" type="text/javascript">
    var map;
    var tiled;
    var pureCoverage = false;
    // pink tile avoidance
    OpenLayers.IMAGE_RELOAD_ATTEMPTS = 5;
    // make OL compute scale according to WMS spec
    OpenLayers.DOTS_PER_INCH = 25.4 / 0.28;

    function init(){
        // if this is just a coverage or a group of them, disable a few items,
        // and default to jpeg format
        format = 'image/png';

        var bounds = new OpenLayers.Bounds( -21237508, -7360000, 21237508, 10217508); /* METRE left, bottom, right, top */

        var options = {
            controls: [],
            maxExtent: bounds.clone(),
            restrictedExtent: bounds.clone(),
            maxResolution: "auto",
            projection: "EPSG:900913",
            units: 'm'
        };
        map = new OpenLayers.Map('map', options);

        // setup tiled layer
        tiled1 = new OpenLayers.Layer.WMS(
            "org.unesco:type1 - Tiled", "http://127.0.0.1:8080/geoserver/org.unesco/wms",
            {
                LAYERS: 'org.unesco:type1',
                STYLES: '',
                    transparent:true,
                format: format
            },
            {
                buffer: 0,
                displayOutsideMaxExtent: true
            }
        );
        tiled2 = new OpenLayers.Layer.WMS(
            "org.unesco:type2 - Tiled", "http://127.0.0.1:8080/geoserver/org.unesco/wms",
            {
                LAYERS: 'org.unesco:type2',
                STYLES: '',
                    transparent:true,
                format: format
            },
            {
                buffer: 0,
                displayOutsideMaxExtent: true
            }
        );
        tiled3 = new OpenLayers.Layer.WMS(
            "org.unesco:type3 - Tiled", "http://127.0.0.1:8080/geoserver/org.unesco/wms",
            {
                LAYERS: 'org.unesco:type3',
                STYLES: '',
                    transparent:true,
                format: format
            },
            {
```

```
            buffer: 0,
            displayOutsideMaxExtent: true
        }
    );
    tiled4 = new OpenLayers.Layer.WMS(
        "org.unesco:type4 - Tiled", "http://127.0.0.1:8080/geoserver/org.unesco/wms",
        {
            LAYERS: 'org.unesco:type4',
            STYLES: '',
                transparent:true,
            format: format
        },
        {
            buffer: 0,
            displayOutsideMaxExtent: true
        }
    );

    tiled5 = new OpenLayers.Layer.WMS(
        "org.unesco:ne_50m_coastline - Tiled", "http://127.0.0.1:8080/geoserver/org.unesco/wms",
        {
            LAYERS: 'org.unesco:ne_50m_coastline',
            STYLES: '',
                transparent:true,
            format: format,
            tiled: true,
            tilesOrigin : map.maxExtent.left + ',' + map.maxExtent.bottom
        },
        {
            //buffer: 0,
            //displayOutsideMaxExtent: true,
            isBaseLayer: true,
            yx : {'EPSG:4326' : true}
        }
    );

    map.addLayers([tiled1, tiled2, tiled3, tiled4, tiled5]);
    //map.addLayers([tiled5]);

    // build up all controls
    map.addControl(new OpenLayers.Control.PanZoomBar({
        position: new OpenLayers.Pixel(2, 15)
    }));
    map.addControl(new OpenLayers.Control.Navigation());
    map.addControl(new OpenLayers.Control.Scale($('scale')));
    map.addControl(new OpenLayers.Control.MousePosition({element: $('location')}));
    map.zoomToExtent(bounds);
  }
</script>
```

## 1.2. exemple de SLD des zones Arides

```
<?xml version="1.0" encoding="ISO-8859-1"?>
<StyledLayerDescriptor version="1.0.0"
 xsi:schemaLocation="http://www.opengis.net/sld StyledLayerDescriptor.xsd"
 xmlns="http://www.opengis.net/sld"
 xmlns:ogc="http://www.opengis.net/ogc"
 xmlns:xlink="http://www.w3.org/1999/xlink"
 xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance">
  <!-- a Named Layer is the basic building block of an SLD document -->
  <NamedLayer>
    <Name>style2</Name>
    <UserStyle>
    <!-- Styles can have names, titles and abstracts -->
      <Title>Default Polygon</Title>
      <Abstract>A sample style that draws a polygon</Abstract>
      <!-- FeatureTypeStyles describe how to render different features -->
      <!-- A FeatureTypeStyle for rendering polygons -->
      <FeatureTypeStyle>
        <Rule>
          <Name>rule1</Name>
          <Title>Gray Polygon with Black Outline</Title>
          <Abstract>A polygon with a gray fill and a 1 pixel black outline</Abstract>
          <PolygonSymbolizer>
            <Fill>
              <!-- <CssParameter name="fill">#e29f66</CssParameter> -->
              <CssParameter name="opacity">0</CssParameter>
            </Fill>
            <Stroke>
              <CssParameter name="stroke">#000000</CssParameter>
              <CssParameter name="stroke-width">1</CssParameter>
            </Stroke>
          </PolygonSymbolizer>
        </Rule>
      </FeatureTypeStyle>
    </UserStyle>
  </NamedLayer>
</StyledLayerDescriptor>
```

## 1.3. Exemple de rasterisation de Shapefile en Java avec Geotools

```
import java.awt.Dimension;
import java.io.File;
import java.io.IOException;

import org.geotools.coverage.grid.GridCoverage2D;
import org.geotools.data.shapefile.ShapefileDataStore;
import org.geotools.data.simple.SimpleFeatureCollection;
import org.geotools.data.simple.SimpleFeatureSource;
import org.geotools.gce.geotiff.GeoTiffWriter;
import org.geotools.geometry.jts.ReferencedEnvelope;
import org.geotools.process.raster.VectorToRasterProcess;
import org.opengis.util.ProgressListener;

public class LucaTest {

   public static void main(String[] args) throws Exception {
      ShapefileDataStore ds = new ShapefileDataStore(new File("/tmp/states.shp").toURI().toURL());
      SimpleFeatureSource source = ds.getFeatureSource();
      SimpleFeatureCollection features = source.getFeatures();

      ReferencedEnvelope bounds = source.getBounds();

      Dimension gridDim = new Dimension(500, 500);

      String covName = ds.getTypeNames()[0];
      ProgressListener monitor = null;

      GridCoverage2D cov = VectorToRasterProcess.process(features, "FAMILIES", gridDim, bounds, covName, monitor);
      System.out.println(cov.toString());

      File tmpTiff = new File("/tmp/testcov.tif");
      GeoTiffWriter writer;
      try {
         writer = new GeoTiffWriter(tmpTiff);

         writer.write(cov, null);
      } catch (IOException e) {
         // TODO Auto-generated catch block
         e.printStackTrace();
      }
   }
}
```

## 1.4. Exemple de MapFile

```
MAP
  NAME "COUCHE_FOND"
  STATUS ON
  IMAGETYPE PNG24
  SIZE 800 800
  EXTENT 534000 2346000 691000 2472000
  UNITS METERS
  RESOLUTION 254 #DPI
  MAXSIZE 4096

  PROJECTION
    "init=epsg:27572"
  END

  OUTPUTFORMAT
   NAME PNG24
   DRIVER "AGG/PNG"
   MIMETYPE "image/png"
   IMAGEMODE RGBA
   EXTENSION "png"
   TRANSPARENT ON
   FORMATOPTION "QUANTIZE_FORCE=ON"
   FORMATOPTION "QUANTIZE_DITHER=OFF"
   FORMATOPTION "QUANTIZE_COLORS=256"
  END



    LAYER
        GROUP "fond"
        NAME "zoom0"
        TYPE RASTER
        DATA "../fdp/FDP_sytadin_zoom1.tif"
        MINSCALEDENOM 1000000

        PROCESSING "RESAMPLE=BILINEAR"

        METADATA
            "wms_title"           "FONDZoom1"
            "wms_srs"             "EPSG:27572"
        END

        PROJECTION
            "init=epsg:27572"
        END
    END

    LAYER
        GROUP "fond"
        NAME "zoom1"
        TYPE RASTER
        DATA "../fdp/FDP_sytadin_zoom2.tif"
        MAXSCALEDENOM 1000000
        MINSCALEDENOM 500000

        PROCESSING "RESAMPLE=BILINEAR"

        METADATA
            "wms_title"           "FONDZoom2"
            "wms_srs"             "EPSG:27572"
        END
```

```
        PROJECTION
            "init=epsg:27572"
        END
    END

    LAYER
        GROUP "fond"
        NAME "zoom2"
        TYPE RASTER
        DATA "../fdp/FDP_sytadin_zoom3.tif"
        MAXSCALEDENOM 500000
        MINSCALEDENOM 250000

        PROCESSING "RESAMPLE=BILINEAR"

        METADATA
            "wms_title"           "FONDZoom3"
            "wms_srs"             "EPSG:27572"
        END

        PROJECTION
            "init=epsg:27572"
        END
    END


    LAYER
        GROUP "fond"
        NAME "zoom3"
        TYPE RASTER
        DATA "../fdp/FDP_sytadin_zoom4.tif"
        MAXSCALEDENOM 250000

        PROCESSING "RESAMPLE=BILINEAR"

        METADATA
            "wms_title"           "FONDZoom4"
            "wms_srs"             "EPSG:27572"
        END

        PROJECTION
            "init=epsg:27572"
        END
    END
END
```