The Geocode Service Software

1 June 2008
(Revised 30 November 2008)

This document describes installation and operation of the geocode service software. It discusses the contents of the distribution, other software needed, how to install, how the software works, and the apis of the service request and response.

Contents

Author

The author of this document is Walter Sinclair.

License

Sources

This software is open-source, written in commented, unobfuscated ANSI C. All sources are included in the distribution **geocode-dist.tar.gz**. This includes all the C sources and headers. Included are the configuration and makefiles generated by autoconf, automake and libtool, and the template files from which they were generated. Not included are the sources of certain third party packages that may be necessary for the software to successfully compile or operate. All of these packages, however, are also open-source and readily available.

Supported Systems

This software is specifically intended for Windows and Linux systems running the Apache webserver. However, there is nothing to preclude its use on other systems, such as Mac OSX, or its operation with other kinds of webservers.

- **Windows**. Note: Windows 3.1 is not supported. The software will run on Windows 9x and ME if the **PAGC** library is configured at compile-time with the --enable-dbprivate flag. This allows the software to run but places constraints that may not be acceptable.

Geocode Service: Provided Files

The software for the Geocode Service is provided in two files:

geocode-dist.tar.gz

This archive contains the archives and files necessary to build and install the software from source.

gcrespond.exe

This installer contains the pre-compiled binaries and files for a Microsoft Windows installation.

geocode-dist.tar.gz

Contents of geocode-dist.tar.gz

Included in **geocode-dist.tar.gz** are the following archives:

pagc-0.2.0.tar.gz

This archive contains the sources and files necessary for the **PAGC** geocoding library to be built from source. This archive also contains base versions of the standardization files **rules.txt**, **lexicon.csv** and **gazeteer.csv.**

geocode_response-1.0.1.tar.gz

This archive contains the sources and files necessary for the **geocode_response** responder to be built from source.

pagc_build_schema-1.0.1.tar.gz

This archive contains the sources and files necessary to build the **pagc_build_schema** command-line utility that creates the database indices used **by geocode_response**.

pagc_dump-1.0.1.tar.gz

This archive contains the sources and files necessary to build the **pagc_dump** command-line utility from source.

pagc_stand-1.0.1.tar.gz

This archive contains the source and files necessary to build the **pagc_stand** command-line utility from source.

Also included in **geocode-dist.tar.gz** are the following files:

build_streets.sh

This simple shell script is used, in the preferred directory configuration, to couple **pagc_build_schema** with the streets shapeset data and schema file

build_parcels.sh

This simple shell script is used, in the preferred directory configuration, to couple **pagc_build_schema** with the parcels shapeset data and schema file

streets.dbf

This xbase file is used to provide **pagc_build_schema** with the schema information for the streets shapeset

parcels.dbf

This xbase file is used to provide **pagc_build_schema** with the schema information for the parcels shapeset

gcrespond.exe

The Windows installer contains the libraries **libpagc-2.dll, pthreadGC2.dll,** and **fcgi.dll.** It contains the executables **geocode_response.exe**, **pagc_build_schema.exe**, **pagc_dump.exe** and **pagc_stand.exe**. It contains the standardization files **rules.txt**, **lexicon.csv**, and **gazeteer.csv**. It contains two short batch scripts **build_streets.bat** and **build_parcels.bat**.

Contents of gc_respond.exe

The Windows installer **gc_respond.exe** contains four components:

The First Component

libpagc-2.dll

This is the **PAGC** geocoding library compiled as a Windows dynamic link library.

The standardization files

These are the base versions of the standardization files **rules.txt**, **lexicon.csv** and **gazeteer.csv.**

pagc_build_schema.exe

This is the program that transforms the shapesets into the form used by the responder

build_streets.bat

This simple batch file is used, in the preferred directory configuration, to couple **pagc_build_schema** with the streets shapeset data and schema file

build_parcels.bat

This simple shell file is used, in the preferred directory configuration, to couple **pagc_build_schema** with the parcels shapeset data and schema file

streets.dbf

This xbase file is used to provide **pagc_build_schema** with the schema information for the streets shapeset

parcels.dbf

This xbase file is used to provide **pagc_build_schema** with the schema information for the parcels shapeset

The Second Component

The second component in the installer is a choice of two variants of the responder.

FastCGI

This is **geocode_response.exe** compiled and configured to operate with the FastCGI protocol

CGI

This is **geocode_response.exe** compiled and configured to run as an ordinary CGI program

The Third Component

The third component contains the two utilities **pagc_dump.exe** and **pagc_stand.exe**

The Fourth Component

The fourth component contains documentation.

Software Required, but not provided

- The data. The shapesets are not provided. There will be two shapesets.
- The gcc C compiler, autotools, and utilities they use are not provided. These should be readily available for your system if not already installed.
- MINGW versions for Windows are available from http://www.mingw.org.
- Apache httpd webserver.
- Berkeley db.
- Fastcgi Apache module. Note : The software also will function as a CGI program. In this case Fastcgi is not needed.
- Fastcgi library (included in the Fastcgi developer's kit). This too is not needed if the responder is deployed as a CGI program.

Compiler

All configuration, compilation and installation from source assumes the use of the gnu c compiler and standard c library. For Windows this is done with Mingw. Cygwin is not supported. Microsoft and other proprietary compilers are also not supported. However, the software is written in standard c. In order to use, for instance, Microsoft Visual C++ or Visual Studio, you will need to generate your own project.

Berkeley DB

Berkeley downloads are available from Oracle's Berkeley DB download site, located at http://www.oracle.com/technology/software/products/berkeley-db/db/index.html. Choose a version in the range 4.1-4.4, without encryption. For example : **db-4.3.29.NC.tar.gz**. Do **not** use, for example, **db-4.6.21.NC.tar.gz.**

The **PAGC** library needs to link with a Berkeley DB version in the range 4.1-4.4. Many Linux systems will have Berkeley DB already installed, but it may be older version. Ensure that you have a version within the range stated. The **PAGC** configure script will check this. Note: if you alter the **PAGC** configure script, it will not find versions greater than 4.0 unless the pthreads library has already been found. If you edit - make sure the pthreads m4 macro is run before the berkeley m4 macro.

Different versions of Berkeley is available from the Oracle website. You can have several different versions installed. **PAGC** will link with one within range, if it exists in one of the usual locations.

```
extract tar -xzf db-4.3.29.NC.tar.gz
cd db-4.3.29.NC/build_unix
../dist/configure
make
```

```
(as root or su : )
make install
```

Mingw. The Berkeley DB database is built from source as if unix (in the Berkeley distribution's build_unix directory) with the --enable-mingw flag to configure. See the Berkeley documentation.

Windows. The executables provided have Berkeley built-in. A separate installation is not required.

Apache httpd

Apache is available for both Linux and Windows. An installer-packaged executable is available for Windows. The software was tested with Apache 2.0, but the version is important primarily for the fastcgi module. If using fastcgi you will want the module appropriate for the Apache version. It is assumed here that modules are dynamically loaded.

The Fastcgi Apache Module

The fastcgi module is not needed if the program is to run as a CGI program.

The downloads for fastcgi for Apache 2.2 are available from
http://www.fastcgi.com/dist/
For Apache 2.0 and 1.3 mod_fastcgi downloads:
http://www.fastcgi.com/dist/old/

Select the mod_fastcgi that corresponds to the version of Apache it will be running under. Consult the Readme file that comes with the mod_fastcgi distribution on how to compile and install.

Windows

Pre-compiled dlls of mod_fastcgi are available for each version of Apache. The names of the distributions are: **mod_fastcgi-2.4.2-AP13.dll** (for Apache 1.3) , **mod_fastcgi-2.4.2-AP20.dll** (for Apache 2.0) and **mod_fastcgi-2.4.6-AP22.dll** (for Apache 2.2).

Configure Apache for the Module

Place the dll or so in the modules directory (usually a child of the ServerRoot directory). Add the module file name to the `LoadModule` list in the file **`httpd.conf`** (in the conf directory, a child of the ServerRoot directory). For example:

```
LoadModule fastcgi_module $APACHE_MODULES/mod_fastcgi.so
```

$APACHE_MODULES should be replaced with the name of your Apache modules directory.

A Windows example :

```
LoadModule fastcgi_module modules/mod_fastcgi-2.4.2-AP20.dll.
```

Note the forward slashes (instead of Windows back-slashes) in the pathnames.

mod_fastcgi From Source

To build mod_fastcgi from source, follow the directions in the **INSTALL** file in the mod_fastcgi archive. The most recent distribution is **mod_fastcgi-2.4.6.tar.gz.**

The documentation does not state it, but you must also supply the httpd root both to make and make install if the default, /usr/local/apache, doesn't work. You need **http-devel**, **apr-devel** and **apr-devel-util** packages to be installed for the mod_fastcgi Makefile to work.

```
tar -xzvf mod_fastcgi-2.4.6.tar.gz
cd mod_fastcgi-2.4.6
```

If you are the Apache major version number is 2, then :

```
cp Makefile.AP2 Makefile
```

Continue with the following sequence, where $HTTP_ROOT should be replaced by the Apache ServerRoot.

```
make top_dir=$HTTP_ROOT
make install top_dir=$HTTP_ROOT
```

Windows Binary.

Windows dlls may be downloaded from the fastcgi site from each version.

The Fastcgi Library

The fastcgi library is not needed if the program is to run as a CGI program.

The fastcgi developer's kit (**fcgi-2.4.0.tar.gz**) will build the current version of the fastcgi library. The fastcgi library is needed to enable the responder for fastcgi and should be

made before **geocode_response** is made. It can be downloaded from the fastcgi website http:://www.fastcgi.com/dist/.

Installation takes the usual course. A number of demonstration files are also compiled at the same time.

```
tar -xzvf fcgi-2.4.0.tar.gz
cd fcgi-2.4.0
./configure
make
make install
```

Windows Fastcgi Library

For Microsoft Visual C++: The fastcgi developer's kit contains a **Makefile.nt**.

Mingw Fastcgi Library

A Windows binary of the fastcgi library is included in the installer. It is from the devpak library **libfcgi-2.4.0-1cm.DevPak**, which can be downloaded from devpaks.org <http://devpaks.org/details.php?devpak=63>

The contents page of www.devpaks.org contains this advice for using with Mingw :

> A typical devpak will work with any MinGW distribution (with any IDE for MinGW). Simply rename the file from something.devpak to something.tar.bz2 and open it with an archiver (e.g. 7-zip). You will see one file with some meta information (name, version, author etc.) and a directory. Simply unpack the contents of the directory to your MinGW directory tree.

Thus, download **libfcgi-2.4.0-1cm.DevPak,** and rename **libfcgi-2.4.0-1cm.tar.bz2.** This library can be unpacked with tar and bzip2, but you may need (as I did) to move the headers and library files into the /usr/local subtree in order for libtool to link them when **geocode_response** is made. libfcgi.a goes into PREFIX/lib directory. fcgi_config_x86.h and fcgiapp.h go into the PREFIX/include directory.

INSTALLATION OF THE PROVIDED SOFTWARE

Overview.

The basic procedure in installing the software is to

Step 1. Choose a Base Directory for the responder.
Step 2. Choose a Directory Layout for the other components.
Step 3. Create the Directories.
Step 4. Install the software components
Step 5. Add the data.
Step 6. Build the data
Step 7. Configure Apache
Step 8. Start Apache.

You should be wearing the persona of root or administrator in installing the software.

The two principal issues in this process are (1) whether you running the program as CGI or FASTCGI and (2) how you want to integrate the directories required into your system. The second issue is explored in depth in Step 2. The first issue is discussed here.

FastCGI or CGI?

It is anticipated that FastCGI will give quicker response time on queries due to the fact that it is not necessary to relaunch the responder for each request. However, it is easier and possibly more portable to deploy the service as CGI. As a FastCGI server the program is loaded once and stays running to handle requests. As a CGI program **geocode_response** is loaded into memory and executed for each new request.

The decision to use FastCGI is implemented, first of all, in the installation of Third Party Software above. mod_fastcgi and the fastcgi library are not required for CGI. Second, if using FastCGI, **geocode_response** should be configure with the –enable-fastcgi switch, discussed in the installation of **geocode_response** in Step 4. Third, Apache will need to be configured differently for the different protocols. This is discussed in Step 7.

Step 1. Choose a base directory for the service.

A directory for running the geocoding service should be selected and/or created prior to installing the software. This is the directory into which the **geocode_response** executable will go. If the program is running as a CGI program it can be placed in whatever directory is already set up in **httpd.conf** as the CGI directory. The Apache default, out of the box,

is /cgi-bin/. The directory can be configured (in step 7) with an alias, so it does not really matter what the name or location is, if you wish to use an existing directory.

Step 2. Choose a directory layout.

The choice of the location and structure of the layout of directories for various components of the software requires an understanding of how the running program locates the files it needs.

On startup the geocoder needs to know where its database environments are located. It expects two, one for PRECISE matching, which we will call 'parcels', and the other, which we will call 'streets', for INTERPOLATION and INTERSECTION matching. The first is a POINT shapeset and the second is an ARC shapeset. It also needs to know where the **PAGC** standardization files are located.

To reduce the amount of work it needs to do to find these locations, the filenames (the names of the files without the directory path) are compiled in with the executable. Currently those names are (see the section on **Customization** for instructions on how to change, substitute or add datasets) :

- **parcels_all7_points_lat83_point** for the parcels environment
- **tlg_roads_lat83** for the streets environment.

It expects the parcels database environment to exist in a directory named 'parcels', the streets environment to exist in a directory named 'streets', and the standardization files to exist in a directory named 'standard'. It doesn't know, however, where these directories are located.

The problem the geocoder confronts is this : because it is launched by Apache as a CGI or FASTCGI it does not have commandline arguments passed to it. There are two options available to it: (1) to read the locations from a file in a known location or (2) to pull the locations off of the environment (a block of variable-value pairs) provided to it by Apache.

Reading a file is a feasible strategy, but if the file is in a known location relative to the geocoder, why not put all the files it needs in known locations?

The geocoder is not necessarily installed with the same pathname on all systems. It can, in fact, be installed anywhere.

Another consideration is the fact that the environments may need to be updated periodically. The geocoder would like to be able to find the new versions with as little fuss and as little downtime as possible.

The locations can be put on the environment that the geocoder gets from Apache. The values can be expressly assigned in a SetEnv directive in the httpd.conf configuration file for Apache, or they can be passed through from the system environment with a PassEnv directive in httpd.conf.

Thus, the geocoder will try three ways of getting those locations and will go with the first one that works. These three ways correspond to three directory layouts for the geocoder, the standardization files and the data environments.

I will name these three layouts : (1) Unified (2) Decapitated and (3) Dispersed.

Layout 1. Unified Layout.

In the Unified layout the program expects those three directories to be children of its current working directory. So, if the program is launched from (for example) /var/www/geocode, it will look for /var/www/geocode/streets , /var/www/geocode/parcels and /var/www/geocode/standard.
In this case the geocoder doesn't need to be told anything about where the data is. It looks to see if what it wants is where it would like it.

Layout 2. Decapitated Layout

In the Decapitated layout the parent of these directories is some directory other than the geocoder's own. That is, all three of the desired directories are siblings, but the geocoder is not located in their (common) parent. In this case the geocoder needs to know the name of that parent. The geocoder looks in its environment for a value for the variable **PAGC_DATA_PATH**. This is the name of the parent for the three siblings.

Layout 3. Dispersed Layout

In the Dispersed layout, each of those directories are not even siblings, but are dispersed in different locations. In this case the geocoder needs to know the names of all three. Thus the geocoder looks in its environment for values for the three variables **PAGC_STAND_PATH**, **PAGC_STREETS_PATH** , and **PAGC_PARCELS_PATH**. Each of these is the name of a parent for one of the three dispersed directories.

How the build utility builds and rebuilds the environments.

The build utility, **pagc_build_schema**, also needs to know where the data is. The main product of its efforts is a Berkeley DB environment which holds the indices, memory pool and data it creates.

Berkeley DB has a number of virtues - which is why it was chosen for this - but one of them is not the moveability of its environments. They can be moved, but the move

actually takes as much or more time than having **pagc_build_schema** rebuild the environment in a different location.

It is therefore desirable to build the environments in situ -- to build them in the location where they will be accessed.

Rebuilding the datasets

The following strategy is suggested to minimize downtime for building new versions of the data. While Apache is still in service, install the software into a new service directory. Build the data as before but in the new directory subsystem. Produce a new httpd.conf with the directives altered to configure the new location. When ready, stop Apache. Replace the old httpd.conf file with the new one and restart. If the aliasing is done properly, the url will not need to change.

Thus it proposed that the three siblings, streets, parcels and standard, be joined by a fourth, build for the Unified and Decapitated directory model. **pagc_build_schema**, located in the build directory, builds the data in its siblings, streets and parcels.

Step 3. Create the Directories.

The Windows Installer will do this in the Unified Layout. If choosing one of the other options, you will have to move the directories around manually.

If installing from source, create four directories :

PARENT/streets -- the streets.dbf schema file and the streets shapeset is placed and built here.
PARENT/parcels -- the parcels.dbf schema file and the parcels shapeset is placed and built here.
PARENT/standard -- the standardization files rules.txt, lexicon.csv and gazeteer.csv are placed here.
PARENT/build – the build executable, **pagc_build_schema**, will be placed here.

Create your directories as with the following templates.

```
mkdir --parents --mode=644 --verbose ${PARENT}/standard
mkdir --parents --mode=755 --verbose ${PARENT}/streets
mkdir --parents --mode=755 --verbose ${PARENT}/parcels
mkdir --parents --mode=755 --verbose ${PARENT}/build
```

For the Unified Layout, substitute in the above the name of the directory where **geocode_response** will dwell for ${CGIDEST}. For the Decapitated Layout replace the

name of the common parent for ${CGIDEST}. For the Dispersed Layout replace each instance with its parent.

Step 4. Install the Software Components.

Windows Installer

`gcrespond.exe` is the name of the installer.

- click on it
- agree to the license
- choose the components to install. You must make a choice between the cgi and the fastcgi versions.
- navigate to the directory you have selected in Step 1.
- install

Installing from Source with MINGW

The geocode service software is produced for both Linux and Windows. Both versions may be produced from source - however, a Windows source installation will require that the MSYS/MINGW posix-emulator be set up on the Windows machine. Cygwin is not directly supported. For that reason the Windows version of the software is provided pre-compiled and packaged in an installer. The installer is setup for the Unified Layout.

If you have MSYS/MINGW setup on your system you can use the Linux source. You will need recent autotools (automake, autoconf, libtool in particular), the gnu c compiler, and a version of the Berkeley DB database.

The Windows binaries of this software was prepared on MSYS/MINGW using the same sources as the Linux version. The software can be built from source if you have installed MSYS and MINGW. Get the latest of everything from www.mingw.net. Make sure you get the latest stable versions of the gcc compiler, autotools and libtool. The main thing to be aware of in doing this is that the software is compiled with MINGW in MSYS but it meant to work on Windows without MSYS. Apache does not have an MSYS version. The **PAGC** dll (unless configured with --enable-msys) will not work properly on MSYS. It is installed to the usr/local/bin directory, but must go into one of the directories in which Windows looks for it to work outside of MSYS. The executables (*.exe) will not handle the MSYS / directory dividers properly. It is really a cross-compilation that is taking place. The compilation process under MSYS is therefore the same as Linux. It should be possible (although I haven't tried it) to do the cross-compilation for Windows under Linux using a mingw target for autoconf. The object of all this is to use the same set of tools (open-source tools) to generate the software. However, some of the elements required only support compilation. For these elements (Apache, FastCgi), binaries are available. Or, of course, they could be compiled with Visual Studio.

Installing the PAGC library from the archive.

You are in the directory containing the archive. Enter this sequence of commands on the terminal commandline:

```
tar -xzvf pagc-0.2.0.tar.gz
cd pagc-0.2.0
chown ${APACHE_OWNER}:${APACHE_GROUP} *
./configure
make
make install
```

Substitute for the variables ${APACHE_OWNER} and %{APACHE_GROUP} the values these have **in httpd.conf**.

This will install the **PAGC** library into PREFIX/lib and PREFIX/include directories. The default value for PREFIX is /usr/local, but this value may vary. This value can also be set to a different value by means of a commandline argument. Type ./configure –help for all the options. The switches specific to this package are `–enable-dbprivate` (a flag required to limit Berkeley access to a single process – a necessary condition for the library to operate under Windows 9.x and ME). `--enable-msys` configures the library to run on MSYS rather than ordinary Windows. `--disable-approximate` disables the approximate index facility. If the approximate index is not needed, the library's performance can be speeded up. –enable-threading is self-explanatory. The library uses a small number of pthread mutexes or Windows critical sections to lock access to the approximate trie. `--enable-defaultcache` is used to force the use of Berkeley's default cache of 256 k. It is not anticipated that any of these switches will need to be used except under exceptional circumstances.

Necessary pre-requisites for installation of **PAGC** are, principally, an acceptable version of Berkeley DB. The configure script will conduct a search for this and other needed components and abort if any are absent. The Berkeley version must be not greater than 4.4 and not less than 4.1.

Installing geocode_response from the archive

You are in the directory containing the archive. Enter this sequence of commands on the terminal commandline:

```
tar -xzvf geocode_response-1.0.1.tar.gz
cd geocode_response-1.0.1
chown ${APACHE_OWNER}:${APACHE_GROUP} *
./configure
make
```

Substitute for $APACHE_OWNER and $APACHE_GROUP, the owner and group for Apache as specifed in httpd.conf.

The **PAGC** libary is required for this installation. If the software is to be used with Fastcgi, then the Fastcgi library must also be installed before geocode_response. If the software runs threaded, then the pthreads library must all be present.

In addition to the usual configure switches (./configure --help will list them), the following have also been provided :

`--with-pagc=DIR`

If the **PAGC** library is installed in a location other than the default, then you will need to supply an argument to configure (--with-pagc=DIR). The software is built with libtool, so it will be able to reconstruct the dependency library for Berkeley from the libpagc.la file.

`--enable-fastcgi`

This switch enables the USE_FCGX define in main.c and configures the software for using the fastcgi library. Without this switch the software functions as a CGI program.

If this switch is used, the fastcgi library should be installed first. Ensure that the fastcgi library's header files and libraries are installed in the subdirectories of the PREFIX.

`--enable-threads`

This switch requires pthreads.

Mingw : To use Windows pthreads get the package from www.devpak.org. Change the name of the file extension to bz2. Use bunzip2 to decompress the package and then use tar to install. Read the file Readme in /docs on Mingw linking. configure expects the GC2 version. semaphore.h, sched.h , pthread.h go into PREFIX/include, pthreadGC2.dll goes into PREFIX/bin and libpthreadGC2.a goes into PREFIX/lib.

After the software is configured, type :

```
make.
```

Do not install (i.e. don't type : make install). Instead, install the program into the directory where it will be running as either a CGI or FASTCGI program – the directory you selected in Step 1.

Use the following as a template for installation of the executable :

```
install -c -m 755 $APACHE_OWNER -g $APACHE_GROUP ./geocode_response
${CGIIDEST}/geocode_response
```

The values for APACHE_OWNER and APACHE_GROUP can be found in the Apache configuration file httpd.conf. CGIDEST represents the pathname that will be aliased in the httpd.conf, the directory selected in Step 1.

Installing pagc_build_schema from the archive

You are in the directory containing the archive. Enter this sequence of commands on the terminal commandline:

```
tar -xzvf pagc_build_schema-1.0.1.tar.gz
cd pagc_build_schema-1.0.1
chown ${APACHE_OWNER}:${APACHE_GROUP} *
./configure
make
```

The values for APACHE_OWNER and APACHE_GROUP can be found in the Apache configuration file httpd.conf.

The PAGC libary is required for this installation. The getopt facility must also be available.

In addition to the usual configure switches (./configure --help will list them), the following have also been provided :

`--with-pagc=DIR`

If the pagc library is installed in a location other than the default, then you will need to supply an argument to configure (--with-pagc=DIR). The software is built with libtool, so it will be able to reconstruct the dependency library for Berkeley from the libpagc.la file.

After the program has been configured and compiled, it should be copied into the directory in which it will be used.

Below the value of INSTALLDIR is the build directory created in Step 3, ${PARENT}/build.

```
install -c -m 755 -o $APACHE_OWNER -g $APACHE_GROUP ./pagc_build_schema
$INSTALLDIR/pagc_build_schema
```

If you are using the Unified Layout, or will be editing the scripts to correspond to another layout, install the shell scripts in the same directory as **pagc_build_schema**.

```
install -c -m 755 -o $APACHE_OWNER -g $APACHE_GROUP ./build_streets.sh
$INSTALLDIR/build_streets.sh
install -c -m 755 -o $APACHE_OWNER -g $APACHE_GROUP ./build_streets.sh
$INSTALLDIR/ build_parcels.sh
```

For Windows the batch files build_streets.bat and build_parcels.bat will be substituted for the shell scripts.

INSTALLING THE STANDARDIZATION FILES

The Windows Installer will do this in the Unified Layout. If choosing one of the other options, you will have to move the standardization files around manually.

Install the standardization files in the /standard directory created in Step 3. For each of :

```
FILENAME=rules.txt
FILENAME=lexicon.csv
FILENAME=gazeteer.csv
```

and `INSTALLDIR=${PARENT}/standard,` and APACHE_OWNER and APACHE_GROUP are as before, copy using the command template :

```
install -c -m 644 -o $APACHE_OWNER -g $APACHE_GROUP ./${FILENAME}
$INSTALLDIR/$FILENAME
```

Copy them also into the /build directory, the directory containing the executable **pagc_build_schema**. That is, copy each of the three as in the previous set, but substitute `INSTALLDIR=${PARENT}/build`.

Note: the installation of the **PAGC** library also installs them into a directory under **/usr/local/share** (if gnu standards are observed).

See the discussion in topics.html on the function of the standardization files.

INSTALLING THE PAGC SCHEMA TABLES

The Windows Installer will do this in the Unified Layout. If choosing one of the other options, you will have to move the schema tables around manually. The **PAGC** schema tables for each of the two shapesets, streets.dbf and parcels.dbf, go into their respective directories.

For each of `FILENAME=streets.dbf` and `INSTALLDIR=${PARENT}/streets` and `FILENAME=parcels.dbf` and `INSTALLDIR=${PARENT}/parcels,` use this template:

```
install -c -m 644 -o $APACHE_OWNER -g $APACHE_GROUP ./${FILENAME}
$INSTALLDIR/$FILENAME
```

See the discussion in topics.html on the function of the schema tables.

Step 5 : Copy the data to the directories

The streets shapeset is copied to the directory ${CGIDEST}/streets, in which the streets.dbf schema has been placed. The parcels, similarly, goes to the /parcels directory to join the parcels.dbf schema table. It is assumed (see Step 2) that these shapesets are named:

- **parcels_all7_points_lat83_point** for the parcels environment
- **tlg_roads_lat83** for the streets environment.

Step 6 : Build the data

Once the directories are set up and the software is installed, build the data by executing the scripts in the build directory from the command line or by invoking **pagc_build_schema** directly. It is assumed in this discussion that the Unified Layout is used. If not the shell scripts will have to be edited, or **pagc_build_schema** used directly.

Using the Scripts

./build_streets.sh
The build streets script executes the **pagc_build_schema** program to build the indices and normalized records of the streets dataset (used for intersections and interpolation by the geocoder. **pagc_build_schema** takes an -r flag, telling it the name and location of the shapeset that it will build, and an -s flag, telling it the name and location of the schema it will use to do it. Edit the build_streets.sh script to correct the shapeset name if it is different from the name used in the script.

./build_parcels.sh
This operates in a similar fashion to build_streets. It builds the dataset for the parcels (used for precise site matching by the geocoder).

Windows
build_streets.bat and build_parcels.bat operate in a similar manner to their Linux analogues.

Using pagc_build_schema

pagc_build_schema is a command-line utility that may be used to convert a reference shapeset into a form suitable for matching and geocoding.

Required runtime files

Required at runtime are:

- The three standardization files
- The source shapeset
- A **PAGC** schema file
- The presence of the **PAGC** library (if linking dynamically)

Productions
Productions of **pagc_build_schema** will be placed in the same directory as the source shapeset. The main product is the normalized matching reference. See the discussion of indices in the pagc topics document.

Build the data

You may examine the scripts to see how the program should be used – or type

```
pagc_build_schema –h
```

for a brief usage screen. **pagc_build_schema** takes an -r flag, telling it the name and location of the shapeset that it will build, and an -s flag, telling it the name and location of the schema it will use to do it. Open a terminal and navigate to the `build` directory. In the `build` directory you will find the `pagc_build_schema (*.exe)` program. This is the program that builds the data.

pagc_build_schema is a command-line utility. The arguments that direct it are passed as command-line flags. It is invoked as

```
pagc_build_schema [-rREFERENCE_PATHNAME -sSCHEMA_PATHNAME [-p] [-
l] [-z]
```

If the -p flag is included the program will print, at 10% intervals, how far along it has proceeded. This is sometimes reassuring.
If the -z flag is included the program will create, along with the database, a statistics file `REFERENCE_PATHNAME.sts` that contains information on how what standardization rules were used in normalizing the data. See the documentation on how to read it.
If the -l flag is included then a log is produced, `build_log.err`, that will document the schema created, flags used, and what errors were encountered in normalizing and storing the data.
The -rREFERENCE_PATHNAME is the location and name of the reference data. The installer has created the parcels and streets directories in a known location for the responder and build program to find easily.
The -sSCHEMA_PATHNAME is the location and name of the schema files used for building the data. The installer has copied the proper schemas into the directories. Look for `streets.dbf` in the `streets` directory and `parcels.dbf` in the parcels directory.

See the discussion in Topics.html on the file produced by pagc_build_schema.

Permissions on the Data

After **pagc_build_schema finishes**, check each dataset for owner, group and mode. If set as root or as your username, change to the apache user and apache group. Set the mode for 644.

The utilities pagc_dump and pagc_stand

These utilities are not directly involved in the operation of the geocoder and do not need to be installed. They can be installed as needed to test standardizations and check the contents of the database files.

pagc_dump is used to extract the contents of a pagc file to stdout or to examine an individual record. Use > to redirect to a file. The -r flag gives pagc_dump the pathname of the reference to operate on. The other flags (type pagc_dump without arguments to get usage) tell it which indices. See topics.html for a discussion of which files correspond to which indices.

pagc_stand takes addresses in two lines and gives details on the standardization of that address. There are no arguments.

Both utilities require the PAGC library to be installed.

pagc_dump may be installed in the usual fashion : That is,

```
PACKAGE_NAME=pagc_dump-1.0.1
tar -xzvf pagc_dump-1.0.1.tar.gz
cd pagc_dump-1.0.1
./configure
make
make install
```

pagc_stand

```
tar -xzvf pagc_stand-1.0.1.tar.gz
cd pagc_stand-1.0.1
./configure
make
make install
```

Step 7 : Configure Apache

Apache must be configured for the base directory selected or created in Step 1 and where the **geocode_response** executable was installed in Step 4. If the url for the service is to be expressed as HOSTNAME/geocode/geocode_response (for example), then you need to alias the realname of the directory . If you are running other programs in the same directory, programs that cannot use that alias, you will need to use the alias already set.

For example, this is the directory I created on a Windows system:

```
C:\Program Files\Apache Group\Apache2\fcgi-bin
```

This is on a system where the Apache ServerRoot is

```
C:\Program Files\Apache Group\Apache2
```

Configure the directory for cgi

If you choose to run the program as CGI, the configuration consists of changing httpd.conf so that your directory is substituted for the cgi-bin. In httpd.conf you will find a script alias directive that enables the default cgi directory. You are looking for a line that looks like:

```
     ScriptAlias /cgi-bin/ "C:/Program Files/Apache Group/Apache2/cgi-
bin/"
```

It is ordinarily cgi-bin. Edit the directive so that it looks something like this :

```
ScriptAlias /geocode/ "C:/Program Files/Apache Group/Apache2/fcgi-bin/"

<Directory "C:/Program Files/Apache Group/Apache2/fcgi-bin">
    AllowOverride None
    Options None
    Order allow,deny
    Allow from all
</Directory>
```

Note : I am assuming here that you do not have other CGI programs running. You will have to add a new configuration to use a different alias from that already set.

Configure the directory for fastcgi

If you instead wish to use fastcgi, you will have installed the fastcgi module and configured Apache for it (see Third Party installations above).

Add the following directive to the **`httpd.conf`** , substituting your directory name for mine as the value of `Directory`. **Windows**: Note the use of "/" instead of "\" and the "\" at the end.

```
<IfModule mod_fastcgi.c>
   Alias /geocode/ "C:/Program Files/Apache Group/Apache2/fcgi-bin/"
   <Directory "C:/Program Files/Apache Group/Apache2/fcgi-bin">
      SetHandler fastcgi-script
      Options +ExecCGI
   </Directory>
   FastCgiServer "C:/Program Files/ApacheGroup/Apache2/fcgi-
bin/geocode_response.exe"
</IfModule>
```

Consult the mod_fastcgi documentation for other options. On some systems it will be necessary to add a line assigning an IPC directory. This is where mod_fastcgi and its process manager store their socket file descriptors. If that directory is inaccessible to fastcgi then another directory is needed. If the default directory is symlinked and you cannot change the permissions, you need a real directory with permissions that allow access by all. For more information on how to configure Apache, consult mod_fastcgi.html. This document is available on the fastcgi website http://www.fastcgi.com/mod_fastcgi/docs/mod_fastcgi.html and is also included in the mod_fastcgi distribultions.

Notes : If you get Permission Denied errors, try first to change the settings on the logs/httpd directory (the real one, not the symlink) to o+x.

```
chown $APACHE_USER:$APACHE_GROUP
```

where $APACHE_USER and $APACHE_GROUP are the values assigned to the apache user and group as defined in httpd.conf.

Decapitated and Dispersed Directory Layouts

For layout options other than the Unified Directory Layout, you will also need to add the `PassEnv` or `SetEnv` directives for CGI or `-initial-env` directive for FastCGI. See Step 2 for the variables that will need to be passed to the program.

Step 8 : Start Apache

Once the data is built and Apache is configured, the geocode service is started by starting Apache with the new http.conf file.

How the Geocoder Works

The following is a brief description of how the Geocoder does its work. For more details consult the c source code.

Initialization

In CGI mode the responder is re-launched by the Apache httpd webserver as each new request is received. In FastCGI mode it is launched by Apache when Apache starts and stays running, waiting for requests. Requests are relayed to **geocode_response** by the fastcgi module, mod_fastcgi, as they are received.

The program initializes by establishing where it is (the current working directory) and where its data files are located. Once it has possession of this information it opens the **PAGC** library and creates a **PAGC** schema record for each of the parcels and the streets data sets, opening the database files and indices. In CGI the responder will then create a **PAGC** matching context. In FastCGI it creates a number of matching contexts, each of which awaits a request.

Request

When a request arrives, the variable-value pairs are retrieved. Each is checked against the appropriate constraints to ensure that this value is a valid value for this variable. Then the address data is concatenated into a form suitable for standardization and the results dispatched, bound to the matching context, to **PAGC**.

PAGC standardizes the query address strings, producing up to 5 different standardization. Each standardization, starting with the most likely (according to weights assigned to the rules), is used to produce index lookup keys for the query.

Scoring and Matching

The kinds and number of keys produced will vary depending on whether this is an intersection or site address query.

A site address query will first look for an exact match on the complete streetname. Then it looks for an exact match on the base string name (ie without directionals, type or modifieers). Then it looks for approximate matches, within an edit distance of 2 (within 2

deletions, insertions or transpositions) of the base street name. Finally it looks up a key created from the soundex keys of each (non-numeric) word in the base street name.

With each index lookup a standardized address record is retrieved that serves as a candidate for matching. For each candidate, the query address and candidate address are compared, part by part, for a match. Each part of the address, if it matches, contributes a positive weighted value. If it doesn't match it contribues a negative weighted value.The sum of these values constitutes the candidate's score.

Candidates

The scored candidate is then placed in score order on the matching context's candidate list. If the list is full the candidate will displace another candidate if its score is greater or equal to the score of the last candidate on the list. Otherwise it is chucked. If at any point in the candidate generation process, a candidate is found that has the maximum possible score, the search is terminated and that candidate is returned. However, except in that circumstance, the search continues until it has a list of the top 100 candidates for the matching context. At this point control returns from **PAGC** to the geocoder.

The responder now creates its own candidate list summoning **PAGC** on each to geocode the address or intersection. The candidates are scored and stored purely on the basis of the correspodence between their addresses and the query address. The streets database consists of blockrange records. It may be the fact that a candidate address, representing a blockrange, scores well enough on other attributes to make it onto the candidate list, but the query address number does not, in fact, fall into the interval given by the blockrange. An address that is non-geocodable in this manner is not added to the responder's list. The score of each candidate that is kept is normalized to a value between 0.00 (least likely) and 1.00 (most likely) and is formatted according to the format specified by the request. The responder retains the top 30 candidates.

The responder performs the above procedure to a site address. In this case the **PAGC** matching context is first bound to the parcels schema and looks for a "precise" match. After this procedure, if the top candidate fails to reach or exceed a certain score, the responder rebinds the context to the streets schema record and looks for an "interpolated" match, once again summoning **PAGC** to produce candidates. The products of this are sorted into the geocoder's candidate list.

Intersection Address

The procedure for an intersection address is performed in a like manner. **PAGC** however employs a different kind of index search for intersections. The same sequence of name, approximate name and soundex key searches is conducted. However, because the streets

environment contains indices formed from a concatenation of the base street name of the record and the base street name of the cross-street, it searches these indices first. Then, if it is disappointed by the results, it retrieves records matching one street name and those matching the other and joins them based on their coordinates.

Response

When the responder has candidate list in its possession each candidate will have been formatted, geocoded and scored. The list is then combined into the appropriate geocode list format. That list is combined with the other elements of the response – the original requested address, the response header and a list of faults, if any – and the response is returned (via Apache and mod_cgi or mod_fastcgi) to the user-agent that generated the request.

Customization of the Geocoder Software

Customization of the geocode service software for different datasets is accomplished by editing the `data_cap.h` header prior to compilation. The dataset names specified in that header should be built by pagc_build_schema in a similar manner to that described in this document, according to schema tables that may or may not differ from those provided (see the topics document on the schema tables on how to construct them).
The `data_cap.h` includes a definition of a PAGC_DATA_CAP record. Do not modify this. There follow 4 lines of definitions:

```
#define NUM_LANDMARK_SCHEMAS 1
#define NUM_MATCH_SCHEMAS 1
#define HAVE_SITE_INTERPOLATED
#define HAVE_SITE_ADDRESS_PRECISE
```

Do not delete or modify the first two lines if you have 1 or fewer match schemas (a dataset that matches precisely on the coordinates). If you have more than 1, change the 1 at the end of line 2 to the number of datasets you will have. If you have no match schemas, delete line 4. To specify your datasets, add corresponding records to the array `match_data_cap_table and` provide the strings for each of the 5 fields defined in the PAGC_DATA_CAP record:

```
pagc_schema_name , local_directory_name , environment_variable ,
reported_source_name , reported_geocode_method .
```

The **pagc_schema_name** is the file name of the data set. The **local_directory_name** is the relative path name of the directory in which the data set is located so that the geocoder can locate it by either the unified or decapitated layout described above. The **environment_variable** is the associated variable used in the dispersed layout described above. The **reported_source_name** is the name used to report a source in the source field, and the **reported_geocode_method** is the method reported. As an example, the current match_cap_table is given as :

```
PAGC_DATA_CAP match_data_cap_table[ NUM_MATCH_SCHEMAS ] = {
   { "parcels_all7_points_lat83_point" , "parcels" , "PAGC_PARCELS_PATH"
, "parcels_all7_points_lat83_point" , "PRECISE" }
} ;
```

To add new schemas, modify the NUM_MATCH_SCHEMA value and add a new record to the match_data_cap_table array. Schema records should appear in the order you wished them searched. Add new records with quote marks enclosing strings and commas following between fields and records (but not at the end of a record or the table). The beginning of the table and each record has a left parenthesis. The end of the table and each record has a right parenthesis.

It is assumed that you will have precisely one interpolation schema (a streets dataset that uses interpolation to locate addresses and is used for intersection searches). If you do not have an interpolation database, delete line 3 above (`#define HAVE_SITE_INTERPOLATED`). Change the PAGC_DATA_CAP record to customize its values. As an example, the current interpolation record is defined as :

```
PAGC_DATA_CAP interp_data_cap = {
   "tlg_roads_lat83" ,
   "streets" ,
   "PAGC_STREETS_PATH" ,
   "tlg_roads_lat83" ,
   "INTERPOLATED"
} ;
```

The ability of your schemas to provide values for all the fields described in the api will depend on the data in the dataset itself and the values specified in its schema table.

In addition to the above, you may change the local directory name of the standardization directory (as used by the unified, decapitated and dispersed layouts), which is currently defined as :

```
static const char *stand_directory_string = "standard" ;
```

You may also modify the score which triggers a new dataset search. It is currently defined as :

```
#define ACCEPTABLE_SCORE .9
```

And, finally, you may also modify the header that is included in the XML version of the response. Note that it should correspond to the xsd employed.

Geocode Service API

The Geocode Service API consists of two parts. The first part, the GEOCODE REQUEST API, describes the protocol for making a geocode request. The second part, the GEOCODE RESPONSE API describes the format and content of the response that is returned.

GEOCODE REQUEST API

The geocoder is invoked by submitting a PARAMETER_STRING as an HTTP POST or GET request to:

```
HOST_URL/geocode_response.exe
```

The request, in the PARAMETER_STRING form described below, is sent via the HTTP POST method with content-type set to `application/x-www-form-urlencoded`. The characters are expected to be UTF-8 encoded and the entire request must be less than 4 kilobytes (4096 bytes) in length. The request may also be sent via the HTTP GET method in the following format:

```
HOST_URL/geocode_response.exe?PARAMETER_STRING
```

PARAMETER_STRING

The PARAMETER_STRING is composed of variable-value pairs concatenated together with ampersands (&). Each variable-value pair consists of a permissable variable name bound with an equal sign (=) to a urlencoded client-assigned value. The parameter string will be composed of general parameters which may appear in any request, and of request-specific parameters which will be present or absent, depending on the kind of request. Notwithstanding this distinction, variable-value pairs may appear in the string in any order. The values assigned may be submitted either in upper or lower case letters.

Urlencoded

A value is urlencoded by (a) substituting a plus for a space and (b) subsituting a three-character code for any character not permissable in an http url query string context. This three-character code is composed of a percentage sign (%) followed by the 2 character hexadecimal representation of the character. For example, a forward slash (in a fraction) or ampersand (in a street name such as Joseph & Mary) will need to be urlencoded. Note: some urlencoders substitute %20 (the hexadecimal representation for the space character)

for a plus. This should decode correctly. Even so, a plus that is not used as a substitution for a space must be urlencoded.

PARAMETER_STRING example

```
methodName=GeocodeRequest&Version=1.1&CompleteAddressNumber=1234&C
ompleteStreetName=W+Main+St&PlaceName=Anywhere
```

General Request Parameters

The General Request Parameters must or may appear in any request, regardless of the kind of request.

methodName

Required. All requests must include a value for the `methodName` variable. This value describes the type of request being made. Currently accepted values are:

- `methodName=GeocodeRequest`

The method GeocodeRequest is a request to take an unnormalized address and produce a normalized address bound to standard coordinates. The methodName must be a valid query method that the responder can accept. In version 1.1 the only valid methodName is `GeocodeRequest`. This method is a request for the latitude and longitude of an address for either (a) A site address - the address of a site on a thoroughfare identified by a number, street name, place name and/or postal code - or (b) An intersection address - the intersection of two thoroughfares identified by two street names with place name and/or zip code.

Version

Required. All requests must include a value for the Version. This is a decimal value stating the method Version. This will allow the requester to expect a predictable response. Currently accepted values are:

- `Version=1.1`

CountryCode

Required. All requests must include a value for the `CountryCode` variable.

A string of 2 characters that gives the CountryCode for which the request is applicable. It must be present but the value is ignored in this version.

An example is

- `CountryCode=US`

.

RequestID

Optional All requests may include a client-assigned value for `RequestID`. This value, if included, must be a string not less than 1 character and not greater than 255 characters in length. An example is

- `RequestID=12345ABC`

maximumResponses

Optional All requests may include a value for the variable `maximumResponses`. This value, if provided, must be a positive integer not less than 1 and not greater than 30. If not given, the default of 30 is used. It controls the number of candidates returned, in the event of an imperfect match. An example of its use is

- `maximumResponses=3`

ResponseFormat

Optional All requests may include a value for the parameter `ResponseFormat`. Currently accepted values:

- `ResponseFormat=XML`
- `ResponseFormat=JSON`
- `ResponseFormat=CSV`

Request Specific Parameters

There are two types of GeocodeRequest supported by this responder:

- **Site Address**
- **Intersection Address**

Site Address

A site address is a location denoted by a locale-specific thoroughfare name and an identifier (usually numeric) that positions the location relative to the extent of the thoroughfare.

Intersection Address

An intersection address is the location of the intersection or junction of two thoroughfare and is denoted by the pairing of the locale-specific thoroughfare names.

The responder determines the nature of the request from the presence or absence of certain parameters. The presence of the address number, for example, should indicate that a request is for a site address, while the presence of a second street name should indicate that a request is for an intersection. Consequently two (intersecting) sets of parameters are specified: SITE_ADDRESS_PARAMETERS and INTERSECTION_ADDRESS_PARAMETERS.

SITE ADDRESS PARAMETERS

- A Site Address GeocodeRequest **must** include CompleteAddressNumber.
- A Site Address GeocodeRequest **must** include CompleteStreetName.
- A Site Address GeocodeRequest **must not** include CompleteStreetName2.
- A Site Address GeocodeRequest **may** include CompleteOccupancyIdentifier.
- A Site Address GeocodeRequest **may** include Place State Zip Parameters.
- A Site Address GeocodeRequest **may** include a value for the InterpolationOffset variable.
- A Site Address GeocodeRequest **may** include a value for the RequestStrategy variable.

INTERSECTION ADDRESS PARAMETERS

- A Intersection Address GeocodeRequest **must not** include CompleteAddressNumber.
- A Site Address GeocodeRequest **must** include CompleteStreetName.
- A Site Address GeocodeRequest **must** include CompleteStreetName2.
- A Site Address GeocodeRequest **must not** include CompleteOccupancyIdentifier.
- A Site Address GeocodeRequest **may** include Place State Zip Parameters.
- A Site Address GeocodeRequest **may** include a value for the InterpolationRadius variable.

Complete Feature Address Parameters

CompleteAddressNumber

Required for Site Address, **Forbidden** for Intersection Address. This is a string that identifies an address identifier. It is sometimes called the house number or civic number. It should be a series of digits and may be preceded by a series of letters and/or followed by a series of letters. Rural route boxes (and other non-thoroughfare locators, such as latitude-longitude addresses) are not supported by this version. If no alphabetic characters are specified, it may be terminated by a fraction. It must not be present in an intersection request.

CompleteStreetName

Required for Site Address, **Required** for Intersection Address. This is an string that identifies the full name of the thoroughfare, including directionals, types and qualifiers.

CompleteStreetName2

Forbidden for Site Address, **Required** for Intersection Address. This is an string that identifies the full name of the intersecting thoroughfare, including directionals, types and qualifiers.

CompleteOccupancyIdentifier

Optional for Site Address, **Forbidden** for Intersection Address. This is a string that serves as a floor, unit or building identifier within the location identified by the CompleteAddressNumber. This version of the geocoder will accept the string but does not use it.

Place State Zip Parameters

PlaceName

Optional. This is a string that identifies the municipal, town or city name in which the address is located.

StateName

Optional. This is a string that identifies the state, province, or national subdivision in which the place name is located.

ZipCode

Optional. The value for this parameter should be the USPS postal zip code for the address.

ZipPlus4

Optional. This 4 digit number identifies the extension to the zip code and should not be present if the zip code is absent.

Additional Parameters

RequestStrategy

Optional. The default is `RequestStrategy=Both` - The responder first attempts to match with a precise address. If results are not satisfactory, it abandons the precise results and does an interpolated matching. If the value for this variable is `RequestStrategy=Precise` the responder returns only the precise results. If the value is `RequestStrategy=Interpolated` it returns only interpolated results.

InterpolationOffset

Optional. This value must be a decimal number not less than 0.00 and not greater than 100.00. It is interpreted as the number of meters to offset an interpolated address from the street. The default is 5.0. For example, to set the returned coordinates at 10 meters from the road, use `InterpolationOffset=10.0`.

IntersectionRadius

Optional. This value must be a decimal number not less than 0.00 and not greater than 100.00. It is interpreted as the maximum distance within which two points will snap together to form an intersection. It is interpreted as the radial distance from the center of an intersection such that all points. This parameter allows the user to configure for locales which have large intersections or have short distances between intersections. As a special usage, setting the variable to zero will ensure that all points within the default in an intersection are returned (rather than a single representative point). The default is 30.00 meters.

GeocodeService API: The Response

Format of the Response

The format of the Response is governed by the value of the ResponseFormat variable in the client-submitted request. There are, therefore, three possible formats: XML (default), JSON and CSV.

XML

XML is the default format for the response. The content-type of the xml Response is `text/xml`. An xml schema, GeocodeResponse.xsd, specifies the form.

JSON

The content-type of the JSON response is `application/json`. The Response is a JSON object.

CSV

The content-type of the CSV Response is `text`. It will consist of newline-terminated lines of comma-delimited values. For each section that appears in the response the first line will give the element names and will be followed by one or more lines giving the corresponding values.

The Response

The Response is the response received from the GeocodeService. The XML Response structure will have two attributes, the RequestID, the Version, and an element, the GeocodeResponse. The JSON object will have three fields, the RequestID, the Version, and the GeocodeResponse. The The first line of the response is the header field list. The CSV header will have have the four field names "Version", "RequestID", "numberOfGeocodedAddresses", and "numberOfFaults". The values given for the numberOfGeocodedAddresses and numberOfFaults determines the structure of the rest of the response. If the numberOfGeocodedAddresses is non-zero there will be a list of geocoded addresses. If the numberOfFaults is non-zero there will be a list of error reports. If both are non-zero the list of geocoded addresses will precede the list of errors. For each list there will exist, in addition to the values for each item on the list, a header line giving the fieldnames. The final structure is the requested address, which will again consist of a headerline of field names followed by a single line consisting of the corresponding values.

Version

This is the interface version of the response. It should match the Version of the submitted request. The minimum value is 1.1.

RequestID

This returns verbatim the RequestID submitted by the sender. It will be blank if no RequestID was submitted.

GeocodeResponse

The value of the GeocodeResponse field is a GeocodeResponse object

---

GeocodeResponse

The GeocodeResponse object has three elements: The GeocodeResponseList, The ResponseFaultList, and the Requested Address.

GeocodeResponseList

This contains the list of geocoded match candidates. The GeocodeResponseList is empty if no addresses are returned and absent if an error occurs before it is generated. See the GeocodeResponseList object

ResponseFaultList

The ResponseFaultList is a list of error reports and is present only if an error occurs. See the ResponseFaultList object

RequestedAddress

The RequestedAddress is absent only if an error occurs before it is retrieved. See the RequestedAddress object

---

ResponseFaultList

The ResponseFaultList object is a sequence of one or more Faults. The number of items on the list will be given by the numberOfFaults.
In an XML response the ResponseFaultList is an element of the GeocodeResponse object. It will contain a sequence of Faults. The numberOfFaults is an attribute of the ResponseFaultList.
In a JSON response the ResponseFaults field will belong to the Response object and will possess the ResponseFaultList field and the numberOfFaults field.
The CSV response will contain a separate header-initiated section. It will contain a header following by a list, one line each of the corresponding reports. The header will read : "faultcode", "faultstring", "detail". There will be one comma-delimited value for each of these fields and thus three values per line.

Fault

If the ResponseFaultList is present in the response, then there will be one or more Faults. See the Fault object. In JSON this will be an array of faults. In XML it will be a sequence of fault objects

numberOfFaults

In a CSV response this value will be stated in the header section. If it is zero the ResponseFaultList will not be present.

Fault

Each fault object will contain three fields: a faultcode, faultstring and a detail field.

faultcode

The faultcode is one of either "Client" or "Server".

faultstring

The faultstring for "Client" is "Bad client content" and for "Server", "Server process error". It should be noted that the expression of these two fault types is not, as it would seem, an assignation of blame.

detail

The detail field will give a brief diagnostic of the fault that may assist either the client or server in correcting the problem encountered. This diagnostic may be an error message generated by the responder or by a software library linked to the responder.

GeocodeResponseList

The GeocodeResponseList is a list of geocoded addresses. The number of elements on the list is given by the numberOfGeocodedAddresses field.

numberOfGeocodedAddresses

This is the number of GeocodedAddresses that appear on the list. This is an integer value that can range from 0 to the maximum (30). In XML this is an attribute.

GeocodedAddress

See the GeocodedAddress object

GeocodedAddress

A GeocodedAddress consists of three parts.

- The normalized address.
- The position
- The accuracy.
- Address source data

In a JSON response the GeocodedAddress field has as its value an object consisting of the address object, a Point field which has as its value an object with the fields Latitude and Longitude, and a GeocodeMatchCode field. In a CSV response the Address values will be followed by the latitude, longitude, accuracy matchType, note, dataSource and addressIdentifier fields, all on a single line. The CSV field names will thus be the Address field names followed by "Latitude", "Longitude", "accuracy" , "matchType" , “dataSource” and “addressIdentifier”.

Address

See the Address object

gml:Point

This is the latitude and longitude of the position. In JSON and CSV Address objects the latitude and longitude are represented in separate fields. In XML the field is named "gml:Point".

GeocodeMatchCode

See the GeocodeMatchCode object

source

See the source object

Address

The Address element will contain either a SiteAddress or an IntersectionAddress. The Address is the normalized address of the candidate. It will be articulated in a fashion consistent with the Street Address Data Standard. It will be either a SiteAddress or IntersectionAddress, depending on the nature of the request.

SiteAddress

See the SiteAddress object

IntersectionAddress

See the IntersectionAddress object

GeocodeMatchCode

The GeocodeMatchCode field has as its value an object with an accuracy element and a matchType element.

accuracy

The accuracy field will be a decimal value not less than 0.00 and not more than 1.00 and will indicate the degree of correspondence between the requested address and the normalized reference address.

matchType

The matchType field will contain one of the values "Precise" or "Interpolated", indicating whether the position was determined by matching a record that specified that position or whether the position was determined by matching with an arc record and calculating the ratio of its address number with the range between the starting address number and its position and the ending address number and its position.

note

The note field will contain the value “P” if the returned address disagrees in parity with other addresses in its address range. This is used for “Interpolated” addresses and will be empty if the address is not interpolated or if the parity does not disagree.

source

The source field has as its value an object with a dataSource element and an AddressIdentifier element.

dataSource

The dataSource field will indicate the file name or other identifier of the dataset from which the address data is taken.

addressIdentifier

The addressIdentifier field will indicate the identifier for the specific record from which the address data is taken. This element may be empty for Intersection responses.

IntersectionAddress

An Intersection Address expresses the intersection of two thoroughfares.
Note: Only one set of PlaceStateZip elements are given, despite the fact that there could be more than one at intersections that fall upon a civic or postal boundary.

CompleteStreetName

This will be a sequence of precisely two normalized CompleteStreetNames

PlaceName

See PlaceName.

PlaceName_USPS

See PlaceName_USPS.

StateName

See StateName.

ZipCode

See ZipCode.

ZipPlus4

See ZipPlus4.

SiteAddress

This is an address identified by a numeric or quasi-numeric identifier and a streetname. The CompleteAddressNumber and CompleteStreetName fields will always be present.

CompleteAddressNumber

See CompleteAddressNumber

CompleteStreetName

See CompleteStreetName. This will be a normalized, parsed object in a SiteAddress response.

CompleteOccupancyIdentifier

See CompleteOccupancyIdentifier. This will be a normalized, parsed object in a SiteAddress response

PlaceName

See PlaceName.

PlaceName_USPS

See PlaceName_USPS.

StateName

See StateName.

ZipCode

See ZipCode.

ZipPlus4

See ZipPlus4.

CompleteOccupancyIdentifier

Unit

This is generally an internal building subdivider

Building

This is a separate building identifier where a single CompleteAddressNumber identifies more than one building

---

CompleteStreetName

The CompleteStreetName, in the RequestedAddress object, represents the unnormalized, unparsed street name as sent by the client. In the response, the name is normalized and parsed into the below fields. In the normalized CompleteStreetName any one of the fields may be present, but the StreetName is always present

PreModifier

A pre-positioned qualifier to the street name, such as `Old` in `Old Highway 99`

PreDirectional

A directional indicator that precedes the Street name, such as `West` in `West 107th Street`

PreType

This is a street type that precedes the StreetName. For example, `Highway` in `Highway 17`, or `Rue` in `Rue Morgue`.

StreetName

This field is always present. It is the base name for the street. This will be the official (unstandardized) name of the Street, as represented in the record.

PostType

The street type that follows the Street Name. For example, `Street` in `Main Street`

PostDirectional

A directional indicator that follows the street name, such as `Northwest` in `17th Avenue Northwest`

PostModifier

A post-positioned qualifier to the street name

RequestedAddress

This returns the unnormalized address submitted by the client. The fields stated correspond to those included in the request and will differ, depending on the nature of the Request. There will be, for example, a CompleteStreetName and CompleteStreetName2 field if it is an intersection request.

Other Address Attributes

CompleteAddressNumber

This is the identifier for a SiteAddress.

PlaceName

This is the city, town or municipal name of the area in which the address is located. This may occur in a SiteAddress, IntersectionAddress or the RequestedAddress.

PlaceName_USPS

This is the post office name for this address. This may occur in a SiteAddress or IntersectionAddress.

StateName

This is the state in which the address is located. This may occur in a SiteAddress, IntersectionAddress or the RequestedAddress.

ZipCode

This is the 5 digit USPS postal code for the address. This may occur in a SiteAddress, IntersectionAddress or the RequestedAddress.

ZipPlus4

This is the 4 digit extension to the USPS postal code. This may occur in a SiteAddress, IntersectionAddress or the RequestedAddress.

An XSD for the Geocode Service

```
<?xml version="1.0" encoding="UTF-8" ?>
- <xsd:schema targetNamespace="http://www.metrogis.org/geocode"
      xmlns:xsd="http://www.w3.org/2001/XMLSchema"
      xmlns:gml="http://www.opengis.net/gml"
      xmlns="http://www.metrogis.org/geocode">
     <xsd:import namespace="http://www.opengis.net/gml"
        schemaLocation="http://dp.schemas.opengis.net/05-
        029r4/gml/3.1.1/profiles/point/0.4.0/gml311PointProfile.xsd
        " />
        - <!--

        ************************************************************

     -->
        - <!--

        **  The Top-Level Element.                          **

     -->
        - <!--

        ************************************************************

     -->
   - <xsd:element name="GeocodeService">
     - <xsd:annotation>
          <xsd:documentation xml:lang="en">The global
             element.</xsd:documentation>
       </xsd:annotation>
     - <xsd:complexType>
       - <xsd:all>
            <xsd:element name="Response" type="ResponseType"
               />
       </xsd:all>
     </xsd:complexType>
   </xsd:element>
   - <xsd:simpleType name="AddressAttributeString_type">
       <xsd:restriction base="xsd:normalizedString" />
   </xsd:simpleType>
   - <xsd:simpleType name="faultcode_type">
     - <xsd:restriction base="xsd:token">
          <xsd:enumeration value="client" />
          <xsd:enumeration value="server" />
       </xsd:restriction>
   </xsd:simpleType>
   - <xsd:simpleType name="faultstring_type">
     - <xsd:restriction base="xsd:string">
```

```
        <xsd:enumeration value="Bad client content" />
        <xsd:enumeration value="Server process error" />
      </xsd:restriction>
    </xsd:simpleType>
- <xsd:simpleType name="ZipCode_type">
  - <xsd:restriction base="xsd:string">
      <xsd:pattern value="[0-9]{5}" />
    </xsd:restriction>
  </xsd:simpleType>
- <xsd:simpleType name="ZipPlus4_type">
  - <xsd:restriction base="xsd:string">
      <xsd:pattern value="[0-9]{4}" />
    </xsd:restriction>
  </xsd:simpleType>
- <xsd:simpleType name="GeocodeNote_type">
    <xsd:restriction base="xsd:string" />
  </xsd:simpleType>
- <xsd:simpleType name="GeocodeAccuracy_type">
  - <xsd:annotation>
      <xsd:documentation xml:lang="en">0.0 is least probable
        and 1.0 is most probable</xsd:documentation>
    </xsd:annotation>
  - <xsd:restriction base="xsd:decimal">
      <xsd:minInclusive value="0.0" />
      <xsd:maxInclusive value="1.0" />
    </xsd:restriction>
  </xsd:simpleType>
- <xsd:simpleType name="ListEnumerator_type">
    <xsd:restriction base="xsd:nonNegativeInteger" />
  </xsd:simpleType>
- <xsd:simpleType name="GeocoderVersion_type">
  - <xsd:annotation>
      <xsd:documentation xml:lang="en">The Version of the
        response should match that of the
        request.</xsd:documentation>
    </xsd:annotation>
  - <xsd:restriction base="xsd:decimal">
      <xsd:minInclusive value="1.1" />
    </xsd:restriction>
  </xsd:simpleType>
- <xsd:simpleType name="GeocodeMatchType_type">
  - <xsd:annotation>
      <xsd:documentation xml:lang="en">A position calculated
        by interpolating on a block range is specified as
        interpolated - otherwise it is
        precise.</xsd:documentation>
    </xsd:annotation>
  - <xsd:restriction base="xsd:token">
      <xsd:enumeration value="precise" />
      <xsd:enumeration value="interpolated" />
```

```
    </xsd:restriction>
  </xsd:simpleType>
- <xsd:simpleType name="RequestID_type">
  - <xsd:annotation>
      <xsd:documentation xml:lang="en">The RequestID is
        verbatim as submitted by the
        client.</xsd:documentation>
    </xsd:annotation>
    <xsd:restriction base="xsd:string" />
  </xsd:simpleType>
- <xsd:simpleType name="ErrorReport_type">
    <xsd:restriction base="xsd:string" />
  </xsd:simpleType>
- <xsd:simpleType name="DataSourceID_type">
    <xsd:restriction base="xsd:string" />
  </xsd:simpleType>
- <xsd:complexType name="RequestedAddress_type">
  - <xsd:annotation>
      <xsd:documentation xml:lang="en">This is the address
        submitted by the client, parsed as
        submitted.</xsd:documentation>
    </xsd:annotation>
  - <xsd:all>
      <xsd:element name="CompleteAddressNumber"
        type="AddressAttributeString_type" minOccurs="0"
        maxOccurs="1" />
      <xsd:element name="CompleteStreetName"
        type="AddressAttributeString_type" minOccurs="1"
        maxOccurs="1" />
      <xsd:element name="CompleteStreetName2"
        type="AddressAttributeString_type" minOccurs="0"
        maxOccurs="1" />
      <xsd:element name="CompleteOccupancyIdentifier"
        type="AddressAttributeString_type" minOccurs="0"
        maxOccurs="1" />
      <xsd:element name="PlaceName"
        type="AddressAttributeString_type" minOccurs="0"
        maxOccurs="1" />
      <xsd:element name="StateName"
        type="AddressAttributeString_type" minOccurs="0"
        maxOccurs="1" />
      <xsd:element name="ZipCode"
        type="AddressAttributeString_type" minOccurs="0"
        maxOccurs="1" />
      <xsd:element name="ZipPlus4"
        type="AddressAttributeString_type" minOccurs="0"
        maxOccurs="1" />
    </xsd:all>
  </xsd:complexType>
- <xsd:complexType name="CompleteStreetName_type">
  - <xsd:all>
```

```
        <xsd:element name="PreModifier"
          type="AddressAttributeString_type" minOccurs="0"
          maxOccurs="1" />
        <xsd:element name="PreDirectional"
          type="AddressAttributeString_type" minOccurs="0"
          maxOccurs="1" />
        <xsd:element name="PreType"
          type="AddressAttributeString_type" minOccurs="0"
          maxOccurs="1" />
        <xsd:element name="StreetName"
          type="AddressAttributeString_type" minOccurs="1"
          maxOccurs="1" />
        <xsd:element name="PostType"
          type="AddressAttributeString_type" minOccurs="0"
          maxOccurs="1" />
        <xsd:element name="PostDirectional"
          type="AddressAttributeString_type" minOccurs="0"
          maxOccurs="1" />
        <xsd:element name="PostModifier"
          type="AddressAttributeString_type" minOccurs="0"
          maxOccurs="1" />
      </xsd:all>
    </xsd:complexType>
- <xsd:complexType name="CompleteOccupancyIdentifier_type">
  - <xsd:all>
        <xsd:element name="Unit"
          type="AddressAttributeString_type" minOccurs="0"
          maxOccurs="1" />
        <xsd:element name="Building"
          type="AddressAttributeString_type" minOccurs="0"
          maxOccurs="1" />
      </xsd:all>
    </xsd:complexType>
- <xsd:complexType name="SiteAddress_type">
  - <xsd:all>
        <xsd:element name="CompleteAddressNumber"
          type="AddressAttributeString_type" minOccurs="1"
          maxOccurs="1" />
        <xsd:element name="CompleteStreetName"
          type="CompleteStreetName_type" minOccurs="1"
          maxOccurs="1" />
        <xsd:element name="CompleteOccupancyIdentifier"
          type="CompleteOccupancyIdentifier_type"
          minOccurs="0" maxOccurs="1" />
        <xsd:element name="PlaceName"
          type="AddressAttributeString_type" minOccurs="0"
          maxOccurs="1" />
        <xsd:element name="PlaceName_USPS"
          type="AddressAttributeString_type" minOccurs="0"
          maxOccurs="1" />
```

```
        <xsd:element name="StateName"
          type="AddressAttributeString_type" minOccurs="0"
          maxOccurs="1" />
        <xsd:element name="ZipCode" type="ZipCode_type"
          minOccurs="0" maxOccurs="1" />
        <xsd:element name="ZipPlus4" type="ZipPlus4_type"
          minOccurs="0" maxOccurs="1" />
      </xsd:all>
    </xsd:complexType>
  - <xsd:complexType name="IntersectionAddress_type">
    - <xsd:annotation>
        <xsd:documentation xml:lang="en">Only one set of
          PlaceStateZip elements are given, despite the fact
          that there could be more than one at intersections
          that fall upon a civic or postal
          boundary.</xsd:documentation>
      </xsd:annotation>
    - <xsd:sequence>
        <xsd:element name="CompleteStreetName"
          type="CompleteStreetName_type" minOccurs="2"
          maxOccurs="2" />
        <xsd:element name="PlaceName"
          type="AddressAttributeString_type" minOccurs="0"
          maxOccurs="1" />
        <xsd:element name="PlaceName_USPS"
          type="AddressAttributeString_type" minOccurs="0"
          maxOccurs="1" />
        <xsd:element name="StateName"
          type="AddressAttributeString_type" minOccurs="0"
          maxOccurs="1" />
        <xsd:element name="ZipCode" type="ZipCode_type"
          minOccurs="0" maxOccurs="1" />
        <xsd:element name="ZipPlus4" type="ZipPlus4_type"
          minOccurs="0" maxOccurs="1" />
      </xsd:sequence>
    </xsd:complexType>
  - <xsd:complexType name="GeocodeMatchCode_type">
    - <xsd:annotation>
        <xsd:documentation xml:lang="en">This characterizes the
          matching of the requested address to this normalized
          address and its position.</xsd:documentation>
      </xsd:annotation>
      <xsd:attribute name="accuracy" type="GeocodeAccuracy_type"
        />
      <xsd:attribute name="matchType"
        type="GeocodeMatchType_type" />
      <xsd:attribute name="note" type="GeocodeNote_type" />
    </xsd:complexType>
  - <xsd:complexType name="source_type">
    - <xsd:all>
```

```
        <xsd:element name="dataSource" type="dataSource_type"
          />
        <xsd:element name="addressIdentifier"
          type="dataSource_type" />
      </xsd:all>
    </xsd:complexType>
  - <xsd:complexType name="Address_type">
    - <xsd:annotation>
        <xsd:documentation xml:lang="en">The Address element
          will contain either a SiteAddress or an
          IntersectionAddress.</xsd:documentation>
      </xsd:annotation>
    - <xsd:choice>
        <xsd:element name="SiteAddress"
          type="SiteAddress_type" />
        <xsd:element name="IntersectionAddress"
          type="IntersectionAddress_type" minOccurs="0" />
      </xsd:choice>
    </xsd:complexType>
  - <xsd:complexType name="GeocodedAddress_type">
    - <xsd:annotation>
        <xsd:documentation xml:lang="en">This is a candidate
          matched to the client's requested
          address.</xsd:documentation>
      </xsd:annotation>
    - <xsd:sequence>
        <xsd:element name="Address" type="Address_type"
          minOccurs="1" maxOccurs="1" />
        <xsd:element ref="gml:Point" minOccurs="1"
          maxOccurs="1" />
        <xsd:element name="GeocodeMatchCode"
          type="GeocodeMatchCode_type" minOccurs="1"
          maxOccurs="1" />
      </xsd:sequence>
    </xsd:complexType>
  - <xsd:complexType name="GeocodeResponseList_type">
    - <xsd:annotation>
        <xsd:documentation xml:lang="en">This is the list of
          candidates matched to the requested address. The
          numberOfGeocodedAddresses attribute enumerates
          the number of items on the list</xsd:documentation>
      </xsd:annotation>
    - <xsd:sequence>
        <xsd:element name="GeocodedAddress"
          type="GeocodedAddress_type" maxOccurs="100" />
      </xsd:sequence>
      <xsd:attribute name="numberOfGeocodedAddresses"
        type="ListEnumerator_type" use="required" />
    </xsd:complexType>
  - <xsd:complexType name="Fault_type">
    - <xsd:all>
```

```
      <xsd:element name="faultcode" type="faultcode_type" />
      <xsd:element name="faultstring" type="faultstring_type"
        />
      <xsd:element name="detail" type="ErrorReport_type" />
    </xsd:all>
  </xsd:complexType>
- <xsd:complexType name="ResponseFaultList_type">
  - <xsd:annotation>
      <xsd:documentation xml:lang="en">This is a list of error
        reports. The numberOfFaults attribute enumerates the
        number of items on the list</xsd:documentation>
    </xsd:annotation>
  - <xsd:sequence>
      <xsd:element name="Fault" type="Fault_type"
        maxOccurs="unbounded" />
    </xsd:sequence>
    <xsd:attribute name="numberOfFaults"
      type="ListEnumerator_type" use="required" />
  </xsd:complexType>
- <xsd:complexType name="GeocodeResponse_type">
  - <xsd:annotation>
      <xsd:documentation xml:lang="en">The
        GeocodeResponseList is empty if no addresses are
        returned and absent if an error occurs before it is
        generated. The RequestedAddress is absent only if an
        error occurs before it is
        retrieved.</xsd:documentation>
    </xsd:annotation>
  - <xsd:all>
      <xsd:element name="RequestedAddress"
        type="RequestedAddress_type" minOccurs="0" />
      <xsd:element name="GeocodeResponseList"
        type="GeocodeResponseList_type" minOccurs="0" />
      <xsd:element name="ResponseFaultList"
        type="ResponseFaultList_type" minOccurs="0" />
    </xsd:all>
  </xsd:complexType>
- <xsd:complexType name="ResponseType">
  - <xsd:annotation>
      <xsd:documentation xml:lang="en">The Response will
        contain the GeocodeResponse. Other responses would
        be added here.</xsd:documentation>
    </xsd:annotation>
  - <xsd:choice>
      <xsd:element name="GeocodeResponse"
        type="GeocodeResponse_type" minOccurs="0" />
    </xsd:choice>
    <xsd:attribute name="Version" type="GeocoderVersion_type"
      use="required" />
    <xsd:attribute name="RequestID" type="RequestID_type" />
  </xsd:complexType>
```

```
</xsd:schema>
```