Walter Sinclair
3715 West Tenth Avenue
Vancouver, BC
V6R 2G5
Telephone: 604-228-8223
E-mail: whitedwarf@deadwrite.com

Proposal for Restructuring the PAGC Geocoding Library

Submitted to: The PAGC Developer's Group
Date: March 15, 2009

OVERVIEW

Various parties have expressed an interest in increasing the flexibility of the current PAGC library by allowing an interface with a wider array of address data sources and data stores.

This goal can be accomplished by restructuring the existing library. Thus, the intent of this project is to remodularize the library's code to remove the restrictions that currently prevent such an interface and to give the library a greater independence from specific data structures and providers. The project proposed will remodularize the library and implement a geocoder that adds specific support for Navteq for the US and Canada, with a plug-in driver for the SQLite database.

This restructuring has, as a longer-term objective, the addition of support for other SQL engines, such as PostgreSQL/PostGIS, Ingres, Oracle, etc. The modules created will serve too to provide a necessary foundation for the future integration of the code -- as a stored procedure, for example – into an SQL database.

This document will outline the nature of the restructuring, propose an approach, and set forth the associated requirements.

PROPOSAL

It is proposed that the library software be remodularized in a manner that isolates the major components and the major interactions with external data sources and stores. These components and interactions correspond to the major functional elements. These elements are, briefly:

1) Configuration,
2) Standardization,
3) Building the standardized data from the unstandardized address and positional data sources,
4) Storing the standardized data,
5) Retrieving the standardized data as candidates for matching, scoring, formatting and geocoding against one or more target addresses,
6) Returning the results.

A number of abstraction layers will be interposed between the relevant modules and external datasets so that the library's input and output are mapped by the layer to handlers specific for those datasets. As part of this restructuring, an implementation will be made of one of those handlers, which is anticipated at this time to be SQLite. A target schema will be constructed, which at this time is anticipated to be Navteq address data for Canada and the US.

The reconstructed library, it should be noted, will also support all other North American road network and address data schemata. An early implementation of a driver for the new Tiger file format can be expected. All current functionality will be retained, including support for BerkeleyDB and shapesets. PAGC's present ability to handle all address and road network data will not be degraded – and will be enhanced by the modularization of configuration.

The PAGC library's interface will change as a result of this restructuring, and new documentation will be necessary. Programs that depend on the interface and which will make use of the new version will need re-writing.

The revised modules, when complete, will need to be tested and debugged.

This work will require a major reorganization of the existing code and is anticipated to take approximately 100-120 hours of programming spread over a 6 month period. The work itself will require funding to complete, but an outline of budget requirements is not part of this proposal.

MOTIVATION FOR RESTRUCTURING

A number of individuals using and involved with the PAGC project have expressed a desire for improvements in the flexibility of the library. It has been observed that the PAGC geocoding library would benefit if it ended its current exclusive reliance on the low-level but somewhat unwieldy BerkeleyDB. This package is very fast, but its files are not particularly transportable.

Also, inherent limits on file size have recently thrown into question the library's exclusive use of shapesets for the input of the unstandardized reference data. Shapesets, moreover, although ubiquitous, are not the native data form for many applications, and often necessitate an irksome pre-processing or post-processing procedure.

Requests have also been received to modify the library to support GML, PostGIS and various other relational databases. The linkage of address and position has an increasingly important role to play for many applications – and this, of course, is PAGC's raison d'etre.

An array of 'plug-in' back-ends would, perhaps, provide a way of simultaneously stilling these various complaints and requests. It seems generally accepted that the path forward requires greater adaptability. Additional motivation derives from the knowledge that PAGC, remodularized, would become a more welcome participant in the on-going development of powerful, open-source geospatial tools.

Our enterprise is to increase the use, dissemination, and development of the software. It is clear that these goals, both singly and in a group, are best served by enhancing the library's ability to integrate with other software. This project is meant to address that enhancement.

REORGANIZATION DETAILS

The restructuring proposed here will divide the library into a constant set of front-end modules and a variable set of back-end modules.

IMPLEMENTATION OF THE FRONT-END

The front-end of the library will consist of a constant set of files, organized by the functionality outlined above, in which the actual address geocoding algorithms are applied. These will be created by splitting away the code that interacts with external input-output from the code that actually performs the work. In many cases this will require only the substitution of a generic

function for a data-specific function. For instance, if there is a call to function that reads an arc as a set of points from a shapefile, it will be replaced by a generic function that simply returns an arc as a set of points from whatever back-end it is that is performing that role.

IMPLEMENTATION OF THE BACK-ENDS

There are various options available to implement the linking of back-ends to the PAGC front-end. Because we are using C rather than a higher-level object-oriented language, we are not limited by the constraints that accompany such concepts as polymorphism. Instead we are governed only by the two factors. The first is the choice we are given (by the OS, principally) between dynamic (run-time) and static (compile-time) linkage. The second is the degree of displacement between the definition and the application of a function.

I will assume that the first factor is relatively transparent and well-known - and not in need of explanation. It should be noted here that a dynamic linkage is less-efficient computationally, but has the flexibility of being able to react to conditions unknown and unforeseeable when the program is compiled.

The second factor, the displacement, refers first to the fact that, in C, we may have:

1) A pointer to a function, and/or
2) A preprocessor re-definition.

In other words, we can have a variable function, to which we may then assign as a value any one of a number of actual, defined, functions. It is the assignment of the actual value to the variable, an operation termed 'binding', which constitutes the capacity of a front-end to interact with any number of back-ends. Thus we can either define the function itself, immediately, at compile-time, or delay the definition until run-time, using the pointer to a function, or simply a dynamic redefinition of the function name (i.e. loading different modules in which the same function name is defined differently).

In the model proposed we will be using a thin (as thin as possible) manager that maps the calling functions from the front end to the actual functions in the back-end. The binding, early or late, will take place within this intermediate manager. Therefore it is proposed that back-ends be implemented as libraries capable of being linked either statically or dynamically. By making them libraries we can move dependencies (the BDB dependency for one manager, for instance, or the SQLite dependency for another) to the back-end library and free PAGC itself of that dependency.

A. REFERENCE DATA

Two principal parts of the remodularization relate to the interactions with the reference data.

1. Reading Unstandardized Data.

Currently unstandardized reference data is read from a shapeset, consisting of an xbase (.dbf) table of attribute data and a shapefile (.shp/.shx). Shapelib is used to read the data. The reference data is read row by row (shape by shape), transformed, and stored for later retrieval. The functions used to access the shapeset are those that open it, close it, retrieve the number of rows, identify field names and other attributes, and similar administrative functions. The payload functionality is the field-by-field, row-by-row reads of the attribute, and the marshalling of the shape, point-by-point, into the transformative mapping algorithms.

In the proposed model, the current shapelib functions that do these tasks will be replaced by generic calls to a data source manager. The calls will correspond bijectively to the current calls and return data of the same kind (integer, string, etc) currently returned. For example (and this is only an example), when it comes time to read the base street name of an address or address range, a call will be issued to a char *read_reference_string, which returns a pointer to a buffer containing the requested string. This function, read_reference_string, will be defined in a file that treats it as a wrapper to the real function call, to DBFReadStringAttribute for shapesets, or to a function that takes a cursor-selected database row and returns a field.

The reorganization will substitute calls to shapelib with calls to the reference data manager. The reference data manager will unite the calls from the front end with the actual back-end functions that provide the data. It is anticipated, in SQL back-ends, that there will be two commands, expressed as queries. It will be the job of those two queries to bring the data in as tables, one as a table with the address attributes, the other with the actual coordinates. This data will then be stepped through and doled out to the front end as the front end requests it.

2. Writing to Data Stores.

Currently standardized data, both attribute and positional, is stored in Berkeley DB B-trees and in the Berkeley DB memory pool. Currently the program manages the indexing of this data directly. In the proposed model the data store manager will marshal a standardized record, which will mainly consist of parsed address attributes and coordinates, into storage. The manager will manage the indexing, and thus (in retrieval) the creation of blocks of candidates. This is a relatively critical point in the proposed model. Because it is not feasible to manage indexing directly in many data stores, this part of the process must go into the manager. That is, the manager itself will need to supply this capacity when necessary, when (as with Berkeley) the external agent does not. Most relational databases, as a means of managing optimization, are relatively opaque when it comes to directly manipulating table indices.

It is the process of building in which the interfaces (1) and (2) are central. The files most involved here are ***init.c, index.c, indexput.c, build.c, alpharef.c, makebeta.c*** and ***shapeset.c***. These files will need to be reorganized.

3. Retrieval from Data Stores.

Currently the target address or intersection is taken through a series of standardizations, and each presented to a series of index lookups, using the memory pool for the approximate search, or the Berkeley B-trees to get a row/shape id. Each standardized record indicated by that id is retrieved (if not already retrieved) and scored on the basis of its similarity to the target, and collected onto a candidate list based on the score.

In the proposed model, PAGC's front end will present the target address or intersection to the back-end in a series of standardizations (as done currently), but the back-end will manage the retrieval from the data store. For example, at present we use a target string and a cursor to step through a B-tree, retrieving each entry and scoring it against each candidate's similarity to the target. In the new model we pass the back-end the target string. There, hidden from the front, the back-end would take the steps necessary – a single SQL query, perhaps, to produce a block of candidates. These candidates would then be taken from the block and supplied one by one to the front end. Matching, scoring and geocoding will be done, as now, against the produced candidates and these, thus, will be functions performed by the front end. Initially we will assume that the same back-end can provide us with both attribute and positional data.

It is in matching, scoring and geocoding that (3) is involved. The files ***collect.c , geocode.c***

indexget.c, restore.c and ***score.c*** handle this functionality. These files will need to be reorganized.

B. STANDARDIZATION AND CONFIGURATION

1. Configuration

Configuration of the library to support the various address data formats and structures will be eased by an ability to plug-in different configuration agents.

Currently configuration of the library is done in header files in the source code, in (.in) files for autotools for the compilation, in xbase (.dbf) schema tables for a particular dataset, and on the command line for such executables as pagc_build_schema. It increases the flexibility of the library if many of the compile-time settings could be set or reset at run-time, and if the xbase tables were replaced with a document that is easily editable with a text editor. It is not possible, however, to foresee all the situations in which one kind of document (an xml file, for instance) would be preferred to another. Moreover, there are situations too in which the library might be best configured by a software agent. I should also note that it will become possible in some circumstances to drop in a reconfiguration for a reference that has already been standardized, so that (for example) we can use different scoring weights, without rebuilding.

The files specifically involved in configuration, in addition to header files, are ***init.c, make_sch.c*** and ***restore.c***. Every file, however, makes use of configuration and will need reorganization to interface with a new configuration manager.

2. The Standardizer

The standardizer is effectively modularized now, using a distinct standardization structure. This was done so that we simultaneously standardize in different threads or (as with intersections) produce two standardizers, one for each street name, in the same thread.

However initialization of the standardizer -- which consists mainly of reading in the lexicon, gazetteer and rules -- can be transferred to the proposed configuration module. Instead of a CSV lexicon, for instance, we can then read the same definitions from a database, a spreadsheet, or an XML document. This will also have benefits in terms of internationalization. The standardizer can itself become a separate library. This will permit the later production of standardizers for languages and address expressions that differ radically from those of North America.

The standardizer currently resides in 6 files: ***analyze.c , standard.c , tokenize.c, lexicon.c, gamma.c***, and ***export.c***. Some of the initialization referred to above is driven by code that resides in ***init.c***.

C. OUTPUT

1. Batch Geocoding

Earlier versions of PAGC worked as batch geo-coders. The user supplied addresses (which were input in an xbase table (.dbf)) to the program, which then produced its list of candidates. If there was not a perfect match, the user was presented with a list of candidates to make the choice of which candidate to select for inclusion in the batch. The geo-coded points were associated with the user-provided addresses to create a shapeset. This functionality is still present in the current library, inhabiting the files ***user.c*** and ***select.c***, even though there does not now exist a batch geo-coder that uses the current library directly. To create this kind of output we will require yet

another, perhaps two, interfaces. One is needed to read the user's input and one for the geocoder output. Instead of a shapefile, a single SQL table for example, or perhaps an XML file, may be the product desired. ***Shapeset.c*** is the file that produces the shape points.

2. Single Address Queries

For single address queries, such as those received from the geocoding web service, the list of candidates that the library produces can be formatted in CSV, JSON, or XML (in a draft Street Address Data Standard structure or in OpenLS). This, I believe, should be retained as a front end function. ***Format.c*** and ***candform.c*** are the relevant files here.

D. LIBRARY INTERFACE

The library interface, which will be handled by the front end, will need to be restructured to incorporate some of these changes. The file ***pagc_client.c*** maps interface calls to internals.

E. APPROXIMATE STRING MATCHING

The current approximate string matching in the PAGC library is accomplished through the use of the Berkeley memory pool facility. We will be providing a separate interface to replace this. The reason is that it will not necessarily be the case that the agent which stores and provides the standardized reference data can also perform the approximate string function.

As an aside we should note that the same may also be true of positional data. However, the initial candidates to supply standardized reference data are, in fact, capable of also providing the positional data.

CONCLUSION

The proposed changes to the PAGC geocoding library are extensive and will take a great deal of effort to realize. The benefits, however, should far outweigh the cost.