Tclshp: a Tcl API for shapelib

Devin Eyre, ImpactWeather, Inc., Houston, TX

Abstract: This paper describesTclshp, a Tcl API for shapelib, and my experiences creating it . It contains an interface to all the main shapelib C API functions, and includes TCL interface procedures to make it easier to work with. Shapelib is an open-source library of functions used to read and write shapefiles and the associated xBase database files, which are most often used in Geographical Information Systems applications.

Background: Shapefiles are used to store geometric or geographic shapes, along with data about any of the shapes contained in the shapefile. They can contain point, line, multiline, polygon, multi-polygon, or even three-dimensional shapes. The Shapefile format was created by Environmental Systems Research Institute (ESRI), and is used extensively in Geographical Information Systems (GIS) applications. Shapelib, also known as the Shapefile C library, was first released in 2001. The latest version, 1.2.10, was released in 2003.

Shapefiles normally have three different types of files for one feature:

1) *file*.shp The shapefile
2) *file*.shx Index file
3) *file*.dbf Database file

Shapefiles are organized as records. For each shape in the .shp file, there is a database record in the .dbf file, and a file offset record in the .shx file to the shape's location within the .shp file. Shapefiles can contain the following types of shapes:

Numeric Value	***Shape Type***	***Description***	***Supported***
0	Null shape	Not used	No
1	Point	One point per record	Yes
3	PolyLine	One or more lines per record	Yes
5	Polygon	One or more polygons per record	Yes
8	MultiPoint	More than one point per record	Yes
11	PointZ	3-D point	No
13	PolyLineZ	3-D version of PolyLine	No
15	PolygonZ	3-D version of Polygon	No
18	MultiPointZ	3-D version of MultiPoint	No
21	PointM	Like PointZ, except the Z-coordinate is replaced with a measured value.	No
23	PolyLineM	Like PolyLineZ except the Z-coordinate is replaced with a measured value.	No

Numeric Value	*Shape Type*	*Description*	*Supported*
25	PolygonM	Like PolygonZ, except the Z-coordinate is replaced with a measured value.	No
28	MultiPointM	Like MultiPointZ, except the Z-coordinate is replaced with a measured value	No
31	MultiPatch	Composed of contiguous triangles	No

I have never seen a shapefile for any of the unsupported shape types, but shapelib itself does claim to support all of them.

Prior Art: Five years ago I completed a TCL-only package (called Shp, never released) to read shapefiles and their associated dbf files, but my employer's business plan now calls for providing data in shapefiles to our clientele. Since 2002, there has existed a Perl module which uses Shapelib, and is able to write shape files, called Geo::Shapefile[1], which I used for a couple of years. Miguel Filgueiras made a partial implementation of shapelib[2] for TCL as a part of GPSMan[3], and there are also bindings for Delphi, Python, Perl, .NET, Visual Basic, and Java.

Implementation Strategy: After I had compiled shapelib for the first time, I tried out the sample programs that came with it, and considered just exec'ing them from my scripts, but decided against it due to the increased process overhead that would cause. Instead, I used them as the starting point when deciding which parts of the API to implement for TCL. For both the xBase and Shapefiles, there were programs to create, add to, and dump the file contents. The implementation of creating and adding records to the files was pretty straightforward, but the dumping needed further refinement I split that into two commands: `info` and `get`. The `info` command can work on the whole file, or on a single record for the shapefile. The `get` command can work on the whole file or a single record for both types of files.

In the C file, I created the following new commands: `dbfcreate`, `dbfadd`, `dbfinfo`, `dbfget`, `shpcreate`, `shpadd`, `shpinfo`, and `shpget`. In the TCL file, I create the ::dbf and ::shp namespaces, and renamed the commands into the appropriate namespace. For example, `dbfadd` became `::dbf::add`.

Difficulties: I wanted to make the commands more flexible, so that they could take the data as either a list or as individual arguments, but I couldn't figure out how to do it using the Tcl C API, so I did that part in TCL. Overall, compiling shapelib on Solaris-X86 was the most difficult part of this project due to some difficulties with libtool not putting out valid options for linking. Recently, when I compiled it on a new workstation running a slightly later version of Solaris-X86, I discovered that it is much easier to link the shapelib object files (*.o) directly into the tclshp shared libray. On Linux, for which there is a shapelib RPM available, linking in the shared library is simple.

Usage: We use the package at ImpactWeather, Inc. to write shapefiles which show our hurricane forecast track, wind extents, and other data. We also use it to read data

from shapefiles and convert it into other data formats, such as vector maps for GEMPAK[4] and McIDAS[5] (open-source meteorological applications), and to store as lists in SQLite databases for quick searching and subsequent drawing in a canvas.

You can see approximately what these files look like in ArcGIS by viewing the tclet embedded in http://www.impactweather.com/tclplugin/RA.html. The setting of display options for the shapefiles has to be done from within ArcGIS, and any scripting is done using Vbscript. I haven't been able to figure out how to do animations in ArcGIS such as is shown in the tclet, which is one reason we use Tcl/Tk instead of ArcGIS ourselves. Also, I could find no easy way to draw the wind arrows in ArcGIS like those shown when you zoom in within the tclet.

Example:

```
package require Tclshp
#Creating a shapefile

# shape types:
#    1 = Point
#    3 = PolyLine
#    5 = Polygon
#    8 = MultiPoint

# dbf_field_definition_list is a simple list.  Format:
#    for numeric: -n field_name length decimals precision
#    for string:        -s field_name length
#   (note:  the first field is normally a record ID,
#           sequential, starting at 1)
set dbf_field_definition_list {-n id 5 0 -s name 10 -s state
2 -n fips 5 0 -s time_zone 20}
set shptype 5
set shapefilename /tmp/testing

# Create a shapefile
::shp::Create $shapefilename $shptype
$dbf_field_definition_list

# Add a shape to the shapefile
set geometry {-88.27 41.99 -87.94  41.997 -87.93 41.998
-87.926 41.96 -87.92 41.72 -87.98 41.695 -88.03 41.69 -88.03 41.73
-88.27 41.72 -88.27 41.99}
set values [list 1 DuPage IL 17043 US/Central]

::shp::Add $shapefilename $geometry $values

#See what's inside the  shapefile:
::shp::info $shapefilename
5 1 -88.27 41.69 -87.92 41.998
#results are shapeType, record count, minimum X, minimum Y,
# maximum X, maximum Y.

#Retrieve all the shapes from the shapefile.
::shp::get $shapefilename
{{-88.27 41.99 -87.94 41.997 -87.93 41.998 -87.926 41.96 -87.92 41.72
```

-87.98 41.695 -88.03 41.69 -88.03 41.73 -88.27 41.72 -88.27 41.99}}

```
#Retrieve just the first record of shapefile:
::shp::get $shapefilename 1
```
{-88.27 41.99 -87.94 41.997 -87.93 41.998 -87.926 41.96 -87.92 41.72 -87.98 41.695 -88.03 41.69 -88.03 41.73 -88.27 41.72 -88.27 41.99}

```
#See what's inside the db file:
::dbf::info $shapefilename
```
1 {id Integer 5 0} {name String 10 0} {state String 2 0} {fips Integer 5 0} {time_zone String 20 0}

```
#results are record count, field definitions

#Retrieve all records from the database file.
::dbf::get $shapefilename
```
{1 DuPage IL 17043 US/Central}

```
#Retrieve first record from the database file:
::dbf::get $shapefilename 1
```
1 DuPage IL 17043 US/Central

Lessons learned:	Some things are a lot easier to do in TCL than they are in C.
Version supported:	shapelib version 1.2.10
Where to get it:	http://sourceforge.net/projects/tclshp/

1 http://search.cpan.org/~jasonk/Geo-ShapeFile-2.51/ShapeFile.pm
2 http://shapelib.maptools.org/
3 http://www.ncc.up.pt/gpsman/gpsman.html
4 http://www.unidata.ucar.edu/software/gempak/
5 http://www.unidata.ucar.edu/software/mcidas/