The SurveyOS KML Toolkit: Starting With Simple Placemarks

By Landon Blake

Introduction

This is the first installment of a new column in the OSGeo Journal. This column will follow the development of a brand new open source geospatial software project: The SurveyOS KML Toolkit. The software project will develop a library and front-end GUI application to create and manage KML entities.

This column has 3 goals:

1) Teach the basic concepts of Ruby object-oriented programming.

2) Teach the concepts of KML.

3) Develop an open source KML toolkit suitable for a candidate as an OSGeo Labs Project.

Do You Want To Learn More?

- You can learn more about Ruby programming here: http://www.ruby-lang.org/en/
- You can learn more about object-oriented programming here: http://www.codeproject.com/Articles/22769/Introduction-to-Object-Oriented-Programming-Concep

I've been writing open source software in Java for several years, but I'm new to Ruby programming. I'm also not an expert at Google's KML. In this column I will be learning along with my readers, and I look forward to getting constructive feedback from readers with experience in these two (2) areas of technology.

The first component of the library will be a Ruby programming library. The second component will be a front-end GUI program.

Before we discuss the basic concepts and goals behind the SurveyOS KML Toolkit, I thought it would be helpful to talk a little bit about Ruby, KML, and the SurveyOS Project. This discussion will provide some helpful background for the rest of the article.

You should be familiar with object-oriented programming basics to benefit from this article. A bit of knowledge about Ruby programming is also helpful, but isn't critical if you've programmed in other object-oriented programming languages before.

A Little Bit About the Ruby Programming Language

Ruby was developed in the mid-1990s by Yukihiro "Matz" Matsumoto in Japan. It is a dynamic, reflective, and interpreted language. It supports different programming styles, such as object-oriented programming and functional programming.

There are a number of interpreters that can execute Ruby code, but YARV is bundled in the standard 1.9 distribution of Ruby.

Ruby is used as a scripting language in one of my favorite proprietary programs, SketchUp.

A Little Bit About KML

KML (Keyhole Markup Language) is an XML language that can be used to represent and visualize geospatial information. It was initially developed for use in the Keyhole Earth Viewer, but has been adopted by the OGC as a standard.

A Little Bit About the SurveyOS Project

The SurveyOS Project is focused on the creation and management of open source software to increase the ability of land surveyors to create and work with GIS data. The SurveyOS Project includes a number of sub-projects. The sub-projects include software libraries and applications written in Java, Python, Visual Basic .Net, and AutoLISP.

The Basic Concepts and Goals of the KML Toolkit

Before we begin to dissect the first bit of source code for the KML Toolkit, we'll

Do You Want To Learn Participate?

The source code for the SurveyOS Project KML Toolkit can be found here: http://surveyos.svn.sourceforge.net/viewvc/surveyos/ruby/kmltools/

Contact Landon Blake by e-mail at sunburned.surveyor@gmail.com if you have questions, want to contribute code, documentation, or serve as a beta tester.

discuss briefly the basic concepts behind the toolkit. We'll also talk about some of the design goals for the toolkit.

The toolkit will be divided into two (2) main components. The first component will be a Ruby programming library that can be easily integrated into other Ruby applications. The second component will be a front-end GUI program that exposes the functionality of the library.

The typical work-flow of a user or client application with the KML Toolkit will involve these three (3) steps:

1) Build a collection of KML entities.

KML entities will be added to the collection using one (1) of two (2) methods.

In the first method the user will use the toolkit to import existing spatial data which will be converted to KML entities. (For example: Import of an existing ESRI shapefile or text delimited file that stores data about point features that are then converted to KML placemark entities.)

In the second method KML entities will be automatically created based on a set of input criteria. (For example: Creating KML placemark entities on a regular grid, at every intersection of linear features, or at regular intervals along a linear feature.)

2) Organize and manage the collection of KML entities.

Once KML entities have been added to a collection, they can be organized. Two (2) primary ways to organize the data will be available. One way will be with layers. A layer holds KML entities of the same type and with the same attributes. (For example: One layer might hold placemarks for industrial buildings, while another holds placemarks for commercial buildings.)

A second way the collection of KML entities can be organized is with groups. A group can be made up of KML entities of different types that share some high level relationship. (For example: You might group paths and placemarks representing a railroad network.)

The user will also be able to query and manage selections of KML entities based on their identity, properties, and descriptions. (For example: Select all of the paths of type "highway" that are over 10 miles in length.)

3) Export KML documents.

After a collection of KML entities is complete, the user (or client application) can export actual KML documents based on the collection of KML entities. (For example: This would allow the user to export KML documents for different audiences based on the same collection of KML entities.) The user will be able to style entities in the exported KML document using style templates. Style templates can be applied to KML layers or groups.

Design Goals

The SurveyOS KML Toolkit has the following high-level design goals:

1) Support of a plug-in framework for easy extension by third party programmers.

2) Support for an undo/redo framework.

3) Support for clean separation between GUI and core program code.

Getting Started with the SimplePlacemark Class

In my object-oriented programming projects I find it helpful to start the design of a library or program with the simple core data objects. I find that most of my software projects will only have a handful of core data objects. (Many of my projects have only a single core data object.)

After the core data objects are designed, I work on adding a “program” structure to my software project. This program structure usually hosts a structure to contain instances of the core data object and a framework for tools that can create, manipulate and manage these core data objects.

It is helpfull to start the design of a library or program with the simple core data objects.

I started the design of the SurveyOS KML Toolkit with a Ruby class representing a single core data object. This class is used to represent the simplest sort of placemark KML entity. Before we look at the design of the class itself, let’s take a look at how such a placemark would actually look in a KML document. You can see this in Source Code Listing #1 shown at the end of this article.

We can see that our class needs to store a name, description, and coordinate for the placemark. That’s

exactly what we do in the SimplePlacemark class.

Source Code Listing #2 contains the source code for the current version of the SimplePlacemark class. On the next page is a graphical overview of the SimplePlacemark class member variables and methods.

All of the member variables, or data for our SimplePlacemark class are simple Ruby primitives, except for the Coordinate class. The Coordinate class data is defined entirely with Ruby data primitives. I call this type of class a terminal data class. (A terminal data class doesn't reference any external class definitions, only data primitives.) They are the simplest type of class. The Coordinate class simply bundles up three floating point number values that represent a latitude, longitude and elevation.

Whenever I have a set of simple data values that will often be passed around my class as a set, I consider making a terminal data class to clarify my source code. The Coordinate class is just this type of class.

You can see the methods of our class can also be organized into three (3) groups:

1) Accessor methods that allow access to the member variables.

2) Methods to compare the equality of member data.

3) Common utility methods that should be implemented on most core data classes for a program, including a clone method and a method to represent the core data object as a string.

What's Next

In the next article we will take a look at the implementation for a couple of the methods of the SimplePlacemarkClass. We will also look at the unit test we designed for the class. Then we can start poking around the collection class that will hold our SimplePlacemark objects and that will form the core of our Program class.

Conclusion

In this article we've accomplished the following tasks:

1) We've looked at the concepts and design goals for our toolkit.

2) We've examined the design of our first core data object, which is defined

SimplePlacemark Class

Member Data		
Variable Name	Data Type	Description
@name	String	Stores the name of the placemark.
@description	String	Stores the description of the placemark.
@latitude	Float	Stores the latitude of the placemark location.
@longitude	Float	Stores the longitude of the placemark location.
@coordinate	Coordinate	Stores the location of the placemark as a Coordinate.

Accessor Methods	
Method Name	Return Value Data Type
get_name	Float
get_description	String
get_latitude	Float
get_longitude	Float
get_coordinate	Coordinate

Comparison Methods	
Method Name	Return Value Data Type
has_same_name	Boolean
has_same_description	Boolean
has_same_location	Boolean

Utility Methods	
Method Name	Return Value Data Type
clone	Simple Placemark
to_string	String

by the SimplePlacemark class.

There are other classes (including a unit test for the SimplePlacemark class) in the SurveyOS SVN repository folder for this software project. If you are interested in what you've read in this article, you might look further at the source code found there.

Source Code Listing #1

```
<Placemark>
  <name>Landon’s House</name>
  <description>A placemark representing Landon’s house.</description>
  <Point>
    <coordinates>-121.10233356,37.9255487,0</coordinates>
  </Point>
</Placemark>
```

Source Code Listing #2

```
# Represents a KML placemark. This simple version of a placemark stores and allows
# access to the placemark name, the latitude and longitude of the placemark, and
# a simple placemark description with no embedded HTML. This class is immutable.
class SimplePlacemark
  include KMLEntity

  # Creates a new placemark.
  #
  # name        = The name of the palcemark as a string.
  # latitude    = The latitude of the placemark in decimal desgrees as a double.
  # longitude   = The longitude of the placemark in decimal degrees as a double.
  # description = The description of the placemark as a string.
  def initialize( name, coordinate, description)
    @name = name
    @coordinate = coordinate
    @latitude = coordinate.get_latitude();
    @longitude = coordinate.get_longitude();
    @description = description
  end # End constructor.


  def get_name()
    return @name
  end # End method.

  def get_description()
    return @description
  end # End method.
```

```
  def get_coordinate()
    return @coordinate
  end

  def get_latitude()
    return @latitude
  end

  def get_longitude()
    return @longitude
  end

  def to_string()

    # Convert the latitude and longitude to stirngs.
    latitudeAsString = @latitude.to_s
    longitudeAsString = @longitude.to_s

    placemarkAsString = "Placemark{Name: \"" + @name +"\", " + "Latitude: " +
latitudeAsString + ", " + "Longitude: " + longitudeAsString + ", " + "Description:
\"" + @description + "\"}"
    return placemarkAsString

  end # End method.

  def has_same_name(placemark)
    if
      @name == placemark.get_name()
      return true
    else
      return false
    end # End if.
  end # End method.

  def has_same_description(placemark)
    if
      @description == placemark.get_description()
      return true
    else
      return false
    end # End if.
  end # End method.
```

```
  def has_same_location(placemark)
    equality_counter = 0

    if
      @coordinate.get_latitude() == placemark.get_latitude()
      equality_counter = 1
    end # End if.

    if
      @coordinate.get_longitude() == placemark.get_longitude()
      equality_counter = equality_counter + 1
    end # End if.

    if
      equality_counter == 2
      return true
    else
      return false
    end #End if.

  end # End method.

  def clone()
    clone = SimplePlacemark.new(@name, @coordinate, @description)
    return clone
  end # End method.

end # End class.
```