VISUAL REPRESENTATIONS OF QUERIED DATABASE RESULTS USING GOOGLE'S MAPPING TECHNOLOGIES

By

STEVEN MICHAEL KILBERT

A Senior Project submitted

In partial fulfillment

Of the requirements for the degree of

Bachelor of Science in Industrial Engineering

California Polytechnic State University

San Luis Obispo

Graded By: _____ Date of Submission: _____

Abstract

Database queries have been used in business applications throughout the world to filter out specific information which is often used as a basis for making a decision. Despite the broad use of queries, there are only a limited number of ways to represent results, the most common being a dynaset, or results table. This paper describes the investigation of methods to create more visual representations of results using maps to show queried geospatial data. From these methods, applications were successfully developed to create customized maps in Google Earth and Google Maps based on queried geographical data from Microsoft Access databases. These applications were designed and developed by Steven Kilbert and mentored by Tao Yang, Ph.D. (California Polytechnic State University- San Luis Obispo) to satisfy the requirements for an undergraduate Senior Design Project.

Keywords: ASP.NET, Database Queries, Decision Making, Geospatial Data, Google Earth, Google Maps, Microsoft Access, VB.NET

Acknowledgements

- Special thanks to Brian Oppenheim (B.S./M.S. Computer Science ’09) and Paul Case (B.S. Computer Engineering ’12) for providing programming assistance on this project
- Special thanks to Dr. Tao Yang (Professor, Department of Industrial & Manufacturing Engineering) for his assistance and direction while in the advisory role for the entire duration of the project
- Special thanks to Nick Sweeney (Lecturer, Department of Industrial & Manufacturing Engineering) for his efforts in updating department computers to Visual Studio 2008 and in troubleshooting web server problems

Your time, insight, and generosity are greatly appreciated!

Table of Contents

List of Tables

Table of Figures

Common Terms

ADO (ActiveX Data Objects): a technology used to connect programming applications with databases or recordsets; a component of Microsoft's .NET technology

ASP (Active Server Pages): a web programming technology within Microsoft's .NET technology that allows for the development of dynamic web pages

Batch File: a file use to launch or reference other files using MS-DOS prompts and Microsoft Windows

.NET Technology: an integral Windows framework developed by Microsoft in 2003 as an object oriented development environment for the development of computer programming applications

HTML (Hyper Text Markup Language): a programming language that enables web browsers to read and display data

KML (Keyhole Markup Language): the programming language of Google Earth; a subset of XML. A KML file has the extension .kml

SQL (Structured Query Language): the standard query language for relational databases

VB (Visual Basic): an object-oriented programming language that is part of Microsoft's .NET technology which serves as the basis for this project

XML (Extensible Markup Language): a programming language for documents containing structured information; many document specific languages (e.g. KML) are subsets of XML

Introduction

Problem Definition

Everywhere and anywhere people may go, they must make decisions. Decision making can be defined as the process of using cognitive methods (e.g. memories, knowledge, intuitions) to choose between alternative courses of action. Tools such as computer programs, colleagues, books, databases, and media often can aid the decision making process by providing added information. With more information, people can make decisions with less uncertainty than before, which results in an increased likelihood of choosing the best course of action.

Currently, poor decisions are often made as a result of insufficient geospatial knowledge. Imagine a storage container company that leases storage containers throughout Southern California. The company maintains a database with the location of their containers leased by different clients, but when querying information they are unable to fully understand the geospatial results. They wonder where their storage containers are exactly (the **geo** part) and where their containers are relative to each other (the **spatial** part). When sending out delivery drivers, the company must locate all these points on map and then decide the best way to create a delivery route to drop-off and pick-up containers. Looking up these locations and planning a route involves increased time and effort, which results in higher operation costs for the company, effectively decreasing their overall profit.

Idea

Microsoft Access databases are commonly used in business for storing data regarding the location of products, assets, customers, historical events, suppliers, and facilities. Although this data exists, these businesses have no way of easily displaying this information on maps. This problem could be resolved if there was a tool to visually display the results of a database query. In the case of geospatial data, a tool in the form of a map could aid in decision making by showing a user both 1) where entities are and 2) where they are relative to each other. This information would increase the user's geospatial knowledge which could result in a better decision. A working application of this tool would have two-fold benefits; it would result in 1) a savings of time, effort, and money normally used on researching and investigating the meaning of geospatial data, and 2) a savings of time, effort, and money in implementing the improved course of action resulting from the better decision.

Objectives

In incorporating this idea to solve the problem, the objectives of this project were stated as follows:

- Design an application linking Microsoft Access databases to Google Earth or Google Maps utilizing Microsoft's .NET technology
- Design a user interface which would enable a user (decision-maker) to build a query for geospatial data of interest

- Create a database containing geospatial data that can be read via the user interface
- Enable the user to view a customized map of their queried results in Google Earth or Google Maps

Scope

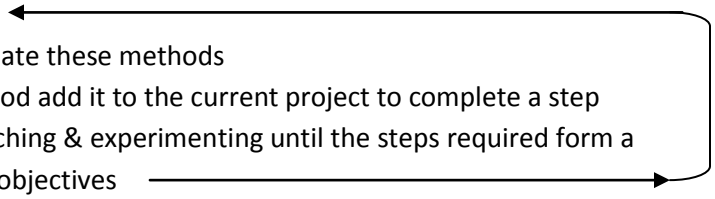
Deliverables for this senior project were defined to include this report in its entirety, a presentation to the Industrial Advisory Board (which oversees the Department of Industrial & Manufacturing Engineering at Cal Poly), and a functional application linking a Microsoft Access Databases with Google Maps or Google Earth. The application was planned to only be able to interface with Google Maps or Google Earth mapping technologies. Similar technologies such as MapQuest, Bing, and Yahoo Maps would not be compatible. The application could be based either locally or on the web, depending on the results of research and experimentation.

The application was specified as being influential to people whose decisions are influenced by the locations and spatial relations of their entities. The application would best interface best with databases that contained geospatial data which was static or dynamic over time, but not necessarily real-time. An example of static data could be the location of utility power poles, whereas a dynamic application could be the movement of products each day through a supply chain. Real-time data would be difficult to capture with the application unless queries were constantly refreshed.

Solution Approach

Given little information and few methods for designing an application, it was difficult to conceptualize the full approach to a solution. Therefore, our approach was simple and closely paralleled the scientific method. It was stated as follows:

1. Identify: Decide which programming languages and platforms to use
2. Research: Discover methods
3. Experiment: Test and manipulate these methods
4. Build: Choose a working method add it to the current project to complete a step
5. Repeat: Continue with researching & experimenting until the steps required form a solution and meet the stated objectives
6. Test & Revise: Test the application for consistency, accuracy, and performance. Make improvements as necessary



The actual process would require multiple iterations, both successful and unsuccessful. There would be difficulties in acquiring software, using and understanding the software interfaces, accessing lab and server facilities, and programming the desired functionality.

Motivation & Timeline

Motivation for this project came from matching my personal interests to research proposed by Dr. Yang. Google Maps and Google Earth contain limitless features such as the ability to view any area of the world, the ability to “fly” or follow a certain course, and the ability to virtually

dive into the ocean. Each day, people use Google Earth and Google Maps to explore the world. In cases like my own, exploring the world via Google Earth or Google Maps can provide substantial entertainment. In Fall 2008, I approached Dr. Yang with the interest of expanding the knowledge I obtained from his IME 312 course on data management and system design. Our discussions resulted in pairing my interest of Google Earth and Google Maps with his proposed research to create an IME 400 independent study course the following quarter, Winter 2009. In IME 400, I investigated methods to bridge Microsoft Access Databases and Google Earth. Unknown to me at the time, the initial research and development from this course formed the basis of my senior project. Provided this, the senior project described in this paper accounts for work in stages as follows in Table 1, beginning with IME 400 and concluding with IME 482:

Time Period	Estimated Time	Course/Topics
Fall 2008	8hrs	Foundations for project, playing with Google Earth, determining objectives for IME 400
Winter 2009	80 hrs (8 hrs/week)	Research; Development of VB Form Application
Spring 2009	10hrs	Carryover from IME 400; Automation, Improvements of VB Application
Spring 2009	20hrs	IME 481: Literature Review
Summer 2009	10hrs	Additional Research
Fall 2009	120hrs	IME 482: Research, Development of ASP.NET Web Application, Presentation, Report

Table 1: Project Timeline

This project draws from the knowledge I have obtained from the courses listed below in Table 2:

Course ID	**Course Title**	**Instructor**	**Topics**
IME 400	Applications for Visual Basic & Google Earth	Dr. Tao Yang	Google Earth, Visual Basic 2008, Batch Files, KML, SQL
IME 312	Data Management & System Design	Dr. Tao Yang	Access, database systems design, queries, SQL, VBA w/

			Access
IME 319	Human Factors Engineering	Lee McFarland	User interaction and psychology via computer interfaces
IME 405	Operations Research II	Dr. Kent Butler/Dr. Tali Freed	Decision Making Analysis
IME 314	Engineering Economics	Lee McFarland	Economic Justification
CSC 232	Programming in Visual Basic	Wagner (CSC Grad Student)	Visual Basic Programming
IME 303/556	Project Management	Dr. Roya Javadpour	Project Management skills for the project's duration

Table 2: Project Basis

Background

This project was not completed for a particular sponsoring company or organization. Instead, it was more research-based. The application was developed for intended use by decision makers across a variety of industries as a tool to assist them in making better decisions based off greater geospatial knowledge. The application could also have implications for personal use.

Currently, there are no applications on the market that create customized Google Earth maps from databases for decision-making purposes. There are several instances of Google Maps and Google Earth that use databases, but these instances have been developed either for personal use or are developed using other technologies such as JavaScript. Many of these rely on pre-built queries or SQL statements, whereas our proposed application allows users the flexibility to build their own queries. Much of the ideas for the project were drawn from the existing instances in combination with .NET code from scratch. The ideas were customized to suit the needs of my application and the relevant sources are cited in the Works Referenced section of this report (pg 71).

The developed application could be marketed and sold online or could be customized and sold for specific use by a small business. The application could also be further developed to work with other larger databases such as Microsoft's SQL Server or Oracle's Oracle.

For additional information regarding specific background concepts for this project, please refer to the Literature Review.

Technical Background & Literature Review

The following section is intended to give the reader a basic understanding of the technologies and ideas mentioned later in the Design & Methodology and Results sections.

Google's Mapping Technologies

Google Inc. is a leader in the development and indexing of websites and other online content with the mission of "organizing the world's information to make it universally accessible and useful" (Google 2009). Google has created and maintained two mapping technologies used by many people around the world—Google Earth and Google Maps. In 2005, Google launched Google Earth, developed from the acquisition of Keyhole Inc. and its Earth Viewer software. Also in 2005, Google Maps was launched after the acquisition of the company Where2 and its mapping software. Together, Google Maps and Google Earth have essentially revolutionized the way many people see the world.

Google Earth is a freeware (free software) program for Windows, Mac, Linux, and mobile phone operating systems that maps the earth and everything on it via overlays from satellite imagery. Google Earth lets users fly anywhere around the earth to explore its many features such as terrain, buildings, roads, highways, oceans, canyons, and landmarks. Google Earth features a search, directions, and multiple layers such as 3D Buildings, Traffic, Weather, and Pictures.

Google Earth was initially designed as an application to entertain curious users who wanted to learn more about their planet. However, given its utility, Google Earth has expanded and is now being used for a variety of business applications. Google Earth can be connected with global positioning systems (GPS) and radio frequency identification (RFID) tags to track the movements of goods and resources for an array of industries. Dell uses Google Earth to track business customers' equipment in Asia as part of its inventory management system (Claburn 2006). The alternative band, Nine Inch Nails, has used Google Earth to represent the number and location of downloads for their latest albums as tool to determine key concert venues (Taylor 2007). Roofing and pool service contractors have used Google Earth for target marketing of residences that may demand their services (Taylor 2007). As the number of applications and capabilities for Google Earth continue to grow, so do the business opportunities.

Google Maps is a basic web mapping service that is run and maintained by Google. According to a Where2 creator of Google Maps, Lars Rasmussen, the web mapping service "is a way of organizing the world's information geographically," which parallels Google's mission statement. By visiting http://maps.google.com people around the world can search for addresses, GPS coordinates, or landmarks and view them on a map.

Google Earth and Google Maps contain many of the same features but there are a few small differences. The most fundamental difference is that Google Earth is three-dimensional and

resembles a globe, while Google Maps is two-dimensional and resembles a map. Both provide the same satellite imagery and similar layers, but Google Earth's functionality surpasses that of Google Maps (users can fly, tilt, pan, or rotate wherever and can connect with a greater variety of applications with Google Earth). Google Maps, however, is a more scalable application, in that more can be done with it being hosted on the web. Google Maps features an Application Programming Interface (API) which enables third parties to use the map in their websites. Developers can write code so users of their site can interact with the map. Both Google Earth and Google Maps contain a street view, but the imagery is more comprehensive with Google Maps. Street views are obtained by company vehicles using video cameras and driving down streets. When displayed, street views can tell users what the intersection will look like where they must make their critical turn. Google Maps also provides turn-by-turn directions for driving, taking public transit, or walking from place to place, whereas Google Earth only does simple driving directions. Google Maps has become even more scalable with recent Google Maps for Mobile technology, which brings the web based application to mobile devices such as the iPhone and Android phones. Other mapping technologies similar to Google Earth and Google Maps include Yahoo Maps, MapQuest, and Bing Maps (formerly Microsoft Virtual Earth), but Google primarily captures the market with their products.

Languages: KML, JavaScript, SQL, VB, ASP, HTML

Keyhole markup language (KML) is the programming language of Google Earth in the data format of extensible markup language (XML). In fact, KML is a subset of XML. Just as web browsers read and display hyper text markup language (HTML), Earth Browsers such as Google Earth, Google Maps, and Google Maps for Mobile read and display KML (Wernecke 2009). KML was originally created in 2001 by Keyhole Inc., a company acquired by Google in 2004. It is an easily read tag-based language composed of text and punctuation. KML can be viewed and edited within a text editor such as Notepad or WordPad and saved with the .kml file extension. KML has a structure as follows:

- *Complex or Parent Elements*: words starting with a capital letter contained within angled brackets <>. Object Elements are also Parent Elements and follow the same syntax.
- *Simple or Children Elements*: words starting with a lowercase letter contained within angled brackets <>.
- *Object Elements*: words that represent the key objects of the KML file which can contain both complex (parent) and simple (children) elements (e.g. *Placemarks*, Ground *Overlays*, *Paths*, *Polygons*, *Screen Overlays* are parent elements; *name*, *description*, *type*, and *color* are child elements).

Code statements for elements are ended or canceled with an angled bracket, a backslash, the element name, and a closing angle bracket (</Element Name Here>). Sample KML code can be viewed for any entity in Google Earth (placemarks, polygons, directions, etc.) by simply right clicking "Copy" and then pasting into any text editor. All KML files begin with a standard XML Header and a KML Namespace Declaration. KML code can be compressed into an archived zip format with the file extension .kmz. The most commonly used entity in Google Earth and

Google Maps is the placemark, which is denoted by coordinates for longitude, latitude, and altitude. This project uses placemarks heavily.

In the case of Google Maps, JavaScript is a powerful programming language because it easily connects with how Google Maps is hosted on the web. In order to bring the ability of programming the map to others users, however, there are many wrappers which take the JavaScript functions for the map used by Google and make them accessible though other programming languages and interfaces. This project will use a JavaScript wrapper designed for ASP.NET web development technology.

Structured query language (SQL) is the standard relational database query language (Silberschatz, Korth, and Sudarshan 2005). SQL was pioneered by IBM in early 1970s under the penname of Sequel, as a part of project called System R. SQL standards are kept and issued by the American National Standards Institute (ANSI) and the International Organization for Standardization (ISO). SQL is the language of queries and recordsets; it's how you retrieve, update, insert, and delete records in your database tables (Cardoza 2004). When creating queries in Microsoft Access's *Design View*, SQL code is actually created in the background to run the query. SQL can be written manually to execute queries in almost any database and can even be used within other languages such as Visual Basic. With special knowledge of SQL, more complex and diverse queries can be made beyond those built within a design view or wizard.

SQL can be read and written in text format—most commands and elements referenced are separated usually only by simple punctuation such as quotes, double quotes, commas, ampersands, exclamation points, and brackets. Like other languages, the practice of concatenation is recommended in SQL. Concatenation is the breaking up lines of code into shorter segments to enhance readability. Common commands used in SQL include SELECT, SELECT DISTINCT, UNION, WHERE, UPDATE, DELETE, JOIN, AND & OR, and GROUP BY.

Visual Basic (VB) is an object-oriented programming language that is an integral part of Microsoft's .NET Technology. An object is a programming structure that encapsulates data and functionality as a single unit and for which the only public access is through the programming structure's interfaces (properties, methods, and events) (Foxall 2008). If we suppose an object is a *pet dog*, we would say it has properties (attributes) of *name, sex, weight, hair color, and number of leg*, and methods (actions an object can perform) of *barking, tail wagging, digging, chewing, and sleeping*. Events are essentially a special type of methods. Dogs are not an integral component of Visual Basic but the ideas and relationships are the same for the properties and methods of Visual Basic. Objects also have classes, which can contain subclasses. Classes are essentially libraries that group objects by category or type.

VB is similar to many other languages in that there are subroutines, functions, constants, variables, special syntaxes, loops, and records. It is a quick, efficient way to design applications for the web, Windows Applications, or mobile devices. Sometimes it is called a rapid application development (RAD) system. VB can run either by itself in an express version or as a component of Visual Studio. Visual Basic for Applications (VBA) differs from VB in that it runs in the

background of Microsoft Office programs like Access, Excel, PowerPoint, and Word, while VB runs on its own. My senior project uses the latest version of VB, Visual Basic 2008 contained within Visual Studio 2008. Visual Studio 2008 has a complete integrated development environment (IDE) which is a design framework created for developers to build applications; every tool needed to create your Visual Basic projects is accessed from within the Visual Basic IDE (Foxall 2008).

ASP is a web programming technology pioneered by Microsoft that allows developers to create dynamic web pages (Mitchell 2008). ASP stands for Active Server Pages which corresponds to web pages being dynamically regenerated each time they are requested. ASP.NET is part of Microsoft's .NET Technology and is usually run within Visual Web Developer or Visual Studio. ASP is a tab based language that is often integrated with HTML to render a page online in a user's web browser. ASP is usually based off source code written in either Visual Basic or Visual C#. More about ASP.NET technology is mentioned below in File Formats & Framework: .NET Technology, Text Files, ADO, Batch Files.

HTML (Hyper Text Markup Language) is a markup language that specifies how content should be displayed in a web browser (Mitchell 2008). HTML serves as the basis for the internet and is evidenced by the URL of many web pages ending in *.html* or *.htm*. It is a tag-based language with elements that specify parameters for displaying data. HTML is used within Visual Studio 2008 in both ASP files and web configuration files.

Graphical User Interface: Forms and Controls Used in Visual Basic

A graphical user interface (GUI) is a human-computer interface (i.e. a way for humans to interact with computers) that uses windows, icons and menus that can be manipulated by a mouse or a keyboard (The Linux Information Project 2004). Most VB programs have GUIs that use forms and controls to interact with users. *Forms*, also called *user forms*, are essentially windows that contain controls (see Figure 1). *Controls* are the various widgets and doodads with which the users interact (Foxall 2008). All forms and controls have properties such as color, size, name, and title that can be customized either in their design or with code. Controls are often contained within a Toolbox as shown by the screenshot in Figure 2 from the Visual Studio development environment. Below is a brief description of several common controls used within forms and shown in Figure 2:

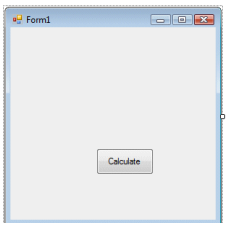
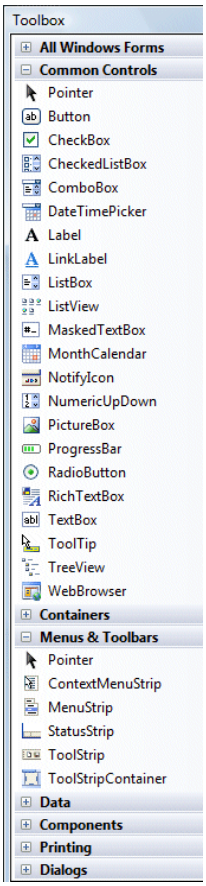


Figure 1: User Form



- **Buttons:** boxes that the user can click to invoke a function or operation (e.g. "OK," "Cancel," "Accept," "Quit," "Go")
- **Combo Box/Drop-Down List:** a drop-down list of items; users can select items or type their own
- **Check Box:** enables a user to select true/false or yes/no values for certain criteria

- **Group Box:** houses several controls of similar type in a frame
- **Input Box:** a form of a text box which requests text input from a user
- **Label:** contains text information that a user cannot change
- **List Box:** a box that presents a list of items to a user; users can select one or more items
- **Message Box:** a pop-up box that informs a user of information
- **Radio Box:** enables a user to select a mutually exclusive option for certain criteria
- **Scrollbar:** enables the user to scroll vertically or horizontally if information is not viewable in the provided area
- **SqlDataSource:** creates a connection so the user can access and query a database as well as update, insert, and delete data from it
- **Text Box:** contains text information that can be entered, edited, or displayed
- **Text Box (Multi-Line):** same as text box but can contain multiple lines of text

Figure 2: Common Controls

Another control used specifically in this application is the ASP.NET Google Maps web control, developed and downloaded from Subgurim.NET for non-commercial use. This control combines the power of the Google Maps API with only a single line of JavaScript code (Subgurim.NET 2009). The control enables a fully-functional instance of a Google Map in an ASP.NET webpage which is programmable with source code in either Visual Basic or Visual C#. This senior project uses the latest version of the control v3.2.1.0 in ASP.NET.

Microsoft Access Databases

A database system is a combination of software and hardware that makes it possible and convenient to store and manipulate large amounts of information (Yang 2008). Microsoft Access, first launched in 1992, is a relational database management system (RDBMS), which means you can store data that's related in multiple ways (Cardoza 2004). Other relational database management systems include IBM's DB2, Tandem's Non-Stop SQL/MP, Oracle's Oracle, Computer Associates' CA-OpenIngres, and Sybase's SQL Server. Access is a Microsoft Jet database that works very well for single-user applications. For multiple users, Microsoft offers larger database systems such as SQL Server or Microsoft Development Environment (MSDE).

As a relational database, Access consists of a collection of tables. Each unique piece of data in a database is assigned a unique name via a central item or column called the primary key. Entities (rows of a table) are objects that exist and are distinguishable from other objects (e.g. a specific person, company, event, or plan). Each of these entities has attributes (e.g. people have names and addresses) which exist within a certain range of values, or domain. Attributes can also have attributes, which are called component attributes. When a set of entities of the same type have similar properties, an entity set exists. Two tables of entity sets can be joined by a relations table, which is where the entity-relation model for database comes about. Databases can be modeled and planned ahead through the use of Entity-Relationship Diagrams.

Given large bundles of information, it can often be very difficult to obtain information we want or need to know. Fortunately, Microsoft Access has queries, which enable users to filter out the

information they want to know. This can be done either with provided query wizards, creating a query in design view by selecting criteria of interest and not of interest, or by writing SQL code to explicitly specify what records we are interested in. Queries can be run and executed via user interfaces (forms are the most common) or by referenced code. The specific result of a query is referred to by Microsoft as a dynaset, which is a table of results that is typically viewed within a form or report.

Visual Basic can connect with a Microsoft Access database via ADO.NET, which is the .NET technology of ActiveX Data Objects (ADO). Just like a form within Access, a form in Visual Basic can also link to a database to manipulate or retrieve data from certain recordsets. ADO is a technology which is more specifically designed to connect to a wide variety of external data sources (Cardoza 2004), compared to it's predecessor, Data Access Objects (DAO). ADO uses the Object Linking and Embedding Databases (OleDB) interface to manage the connection to an Access database. ADO writes SQL statements in order to retrieve the appropriate data specified as a query.

ASP can connect with a Microsoft Access database using either a SqlDataSource or an AccessDataSource web control. These controls also use the Object Linking and Embedding Databases (OleDB) interface to manage their connections. Likewise, these connections also can retrieve and manipulate data from certain recordsets by writing SQL statements.

Framework & Technology

.NET Technology

Having previously mentioned VB.NET, ASP.NET and ADO.NET, Microsoft's .NET technology should also be defined. Microsoft's .NET technology (or framework) is an integral Windows component that supports building and running the next generation of applications and XML web services (MSDN 2008). Microsoft developed the .NET framework in 2003 to provide an object-oriented programming environment that was easy to use, consistent, current, safe, and able to span a wide variety applications and users. The latest version is the .NET Framework 3.5. .NET Technology is composed to two parts: a) a common language runtime (an agent that manages code at execution time) and b) a class library (a comprehensive, object-oriented collection of reusable types that can be used to develop applications) (MSDN 2008). VB. NET and ADO.NET are contained within this class library as a way to develop a GUI for an application and as a command to connect with a database, respectively. ASP.NET is a class library component that enables the development of dynamic web applications through web forms and XML Web Services. ASP.NET can be used to create anything from small, personal websites through to large, enterprise-class web applications (ASP.NET 2009). Several companies such as Costco, Dell, Match.com, and even MySpace currently use ASP.NET for their websites.

Client/Server

When an application opens a connection to any database management system (DBMS) data source, it is considered to be a "client." A "client" application can be a Windows form application, a middle tier XML Web Service, an ASP application, or an application on a handheld

PC connected to a milking machine. "Client/server" applications are considered to be those programs that open (sometimes persistent) connections and provide interactive user interfaces. A "server" is the system that hosts a shared DBMS (Vaughn 2007). Both parts of this senior project design involve "client-server" applications.

Like the development of any website, an ASP web application, requires the use of a web server. A web server is a software application that continually waits for incoming web requests, which are requests for a particular URL (Mitchell 2008). Upon receipt of a request assuming a static web page, the web server examines the request, locates the appropriate file, and then sends this file back to the client that made the web request. Since ASP serves dynamic web pages however, the model is slightly different. An ASP web application contains source code written in either VB or Visual C# that is executed when the page is requested. When the code is executed, the web server produces an HTML output which forms a page which is returns a page to a user's web browser. The web browser receives this HTML file and renders the markup graphically, so the user can view the page properly on the computer screen. This dynamic model allows for dynamic content because the content isn't actually created until the web page is requested. Using a static web page, data would need to be edited as it changes so the proper page could be viewed, which is very infeasible when data changes rapidly over time. The client portion of an ASP application refers to the web server, the ASP based application, and its source code, whereas the server portion of the application refers to what happens between the web server and the user's browser window.

StreamWriter

A significant part of this project concerns creating a KML file from Visual Basic that can be read by Google Earth. As mentioned previously, a KML file is simply a text file in the XML data format (see Languages: KML, SQL, VB), that can be viewed in a text editor such as Notepad or WordPad. In order to create a KML file from VB, we must first create a text file of the KML text in ASCII format and then change the file extension to .kml which is accomplished through VB's StreamWriter class. (ASCII is a plain format of text that only contains the letters a-z, numbers, carriage returns, and punctuation marks while a class in VB is a template that defines an object—the StreamWriter object is in a class because it is a common object used within VB.) StreamWriter will take text contained within code or a text box in VB and save it as a file in a specified directory. Since the extension of a standard text file (.txt) can easily be changed to .kml, StreamWriter can directly write to a KML file.

ADO (ActiveX Data Objects)

Another important part of this project is connecting VB with the .mdb Access database in order to pull geographical data of interest and insert it into KML code. As mentioned above, ADO.NET technology enables this to happen. First, a connection must be established using the OleDB connection object contained within VB as part of ADO. This connection object is declared in a modular level variable. OleDB's connection string property then allows parameters to be set such as the provider, data source, user ID, password, driver, or server for proper connection. An open and close method of the OLE DB object can enable the connection to be open or closed.

After establishing a connection to a database, a data adapter is created to populate a data table in VB with values from a table or query in Access. SQL code can be written here in VB, to pull certain values from the data table and insert them into our text-based KML code. The data table is read line-by-line with a row position counter until all rows are read. The text extraction operations are performed by VB code which loops through the same set of operations for each line of the data table. Through this same ADO.NET connection, Access data can be viewed in forms, edited, or even appended to.

Batch Files

Batch files are also an integral part of this project. Batch files allow MS-DOS and Microsoft Windows users to create a list of commands to run in sequence once the batch file has been executed (ComputerHope.com 2009). Batch files have the file extension .bat and can be used to automatically launch a program and open a file. In our case, a batch file will automatically launch Google Earth and open the KML file outputted by VB.

Design & Methodology

Design Part 1

The initial plan for a database application interfacing with Google mapping technologies was developed while setting learning objectives for IME 400, an independent study course advised by Dr. Tao Yang. Objectives included investigating and developing a better understanding of both Microsoft Access databases and Google Earth with the end goal of creating an application in Visual Basic to link them together, as shown in Figure 3 below. As mentioned in the Solution

Figure 3: Design Part 1

Approach (pg 11), the design would evolve with further research based on the idea of this layout. Experimentation using trial and error techniques would be used until either possibilities were exhausted or solutions were found and built into the application.

Methodology Part 1

Research

We began our research by investigating Google Earth. This involved learning about all the basic features such as placemarks, paths, polygons and ground overlays as well as new layers such as 3D Buildings, Traffic, Weather and Ocean. Next, we studied the code structure of KML. Discussed earlier, KML is a programming language based off XML (see Languages: KML, SQL, VB in the Literature Review). We discovered that all KML files had common headers and declarations followed by a tag-based structure of different elements, much like HTML or XML. After more research, we found that KML code could be viewed for any feature or group of features in Google Earth, by a simple copy/paste operation into a text editor. To explore and understand the KML, several code examples were printed, compared, and analyzed. These included features self-created in Google Earth, pre-existing Google Earth features, as well as features others had created in Google Earth. This research led to a greater understanding for both how maps are rendered in Google Earth from KML code and also how Google Earth maps are translated to code.

Next, we tried creating new KML files by writing code within a text editor (i.e. NotePad), saving the file, manually changing the file extension from .txt to .kml, and finally opening the KML file with Google Earth. Two main KML files were created- the first with one placemark and the second with three placemarks. We discovered that when Google Earth read the files, it would automatically zoom to the appropriate level needed to view all placemarks or features within

the same window. Since Google Earth automatically did this, the need to develop a method to view the points of interest on the same screen was eliminated.

Additional research occurred in refreshing my fundamental knowledge of Visual Basic. This fundamental knowledge came from my VB exposure through VBA, or Visual Basic for applications, most commonly with Microsoft Access and Excel back in 2005. To best suit the needs of our research and development, Dr. Yang suggested working with a Visual Basic editor, such as Visual Basic 6 or Visual Studio 2008. A Visual Basic editor would provide the ability to write code and create a Visual Basic application without being limited to the constraints of staying within a single, existing application. After some research, Visual Basic 2008 was the language of choice for this project. Visual Basic 2008 was the most up-to-date and could be of future benefit or use. The only compromise to using Visual Studio 2008 was that files could not be sent and exchanged with Dr. Yang or IME Department computers because theirs were not the most current version. This issue was soon resolved in Summer 2009 however, when Dr. Yang had Visual Studio 2008 installed on his web server and in the one of the labs. After obtaining a student copy of Visual Studio 2008 and purchasing a book on Visual Basic 2008, my Visual Basic knowledge was sufficient to begin expanding my development skills in finding a way to incorporate Google Earth.

Create a KML File using Visual Basic

The next goal was to figure out how to create a KML file using Visual Basic. With successful trials of writing and editing KML code within a text editor, the only missing link was creating a text file from Visual Basic. After some research, a solution was found in the StreamWriter class. The StreamWriter class (see Framework & Technology in the Literature Review) is a tool that can write data inputted by the user to a storage device such as a hard drive (VB Tutor 2009). By following examples online, code was written in the Stream Writer class to take the contents of a textbox entered by a user from a form and create a text file, saved locally on the computer. Expanding off this example, KML code was then inserted into the textbox. A text file was generated and placed on the desktop. By changing the file's extension and double clicking the KML file, Google Earth was launched to display the code. Everything worked well, including the maintenance of an easy to read cascading style of code used to separate tags in both the text and KML file. Within Visual Basic, a *SaveFileDialog* user control was added to enable the user to specify a location to place their file. Another upgrade included VB code to write directly to a KML file, instead of later changing the file extension. These upgrades helped make the control more flexible and scalable if it were to be shared with others.

Design the Database

In the case of this project, a database with some types of geospatial elements was required. With any database, there were also considerations in the structure and layout of tables, simplicity, and ease of data retrieval. The result was a database of mountain biking trails, based on both personal interest and a possible need. There are several websites such as http://socalmtb.com/ and http://mountainbikeslocounty.com/ that contain information about various trails throughout Southern California which are regularly updated by mountain bike

enthusiasts. Information includes directions to the trailheads, current conditions, trail types, trail photos, and key things to look for. The sites are a great resource for mountain bikers, but hand-typed directions for trailheads and lists of key turns are often insufficient. With a visual representation of this data, mountain bikers can find their trailheads and the relative location of other nearby rides to possibly connect to. Since many riders have some type of GPS, there would also be potential for downloading data to their devices.

For the ease of testing and building, the database was kept to a single table with a limited number of records. The table (see Figure 4) contained information about a variety of trails in the Santa Clarita and San Luis Obispo areas with the following fields: *ID*, *Trail_Name*, *Trail_Type*, *Trail_City*, *Trail_State*, *Trailhead_Longitude*, and *Trailhead_Latitude*. In addition, simple queries were created to find trails of a certain type or trails near a certain geographical locale (see Figure 5).

Tables

Trails

Trails

ID	Trail_Name	Trail_Type	Trail_Locale	Trail_State	Trailhead_Longitude	Trailhead_Latitude
1	Madonna Mountain	Singletrack	San Luis Obispo	CA	-120.672333	35.274886
10	Wilson Canyon	Fire Road	Sylmar	CA	-118.452408	34.356918
11	Viper	Singletrack	Santa Clarita	CA	-118.45259	34.361153
12	May Canyon	Fire Road	Sylmar	CA	-118.432541	34.35579
13	Los Pinetos	Singletrack	Santa Clarita	CA	-118.432594	34.358548
14	East Canyon	Fire Road	Santa Clarita	CA	-118.546013	34.35038
15	Weldon Canyon	Doubletrack	Santa Clarita	CA	-118.525955	34.340907
16	Three Trees	Singletrack	Granada Hills	CA	-118.517461	34.304307
2	Irish Hills	Singletrack	San Luis Obispo	CA	-120.695935	35.253897
3	Shooters	Singletrack	San Luis Obispo	CA	-120.643039	35.342791
4	Roller Coasters	Doubletrack	San Luis Obispo	CA	-120.643039	35.342471
5	Morning Glory	Singletrack	San Luis Obispo	CA	-120.657945	35.361662
6	Cerro Alto	Singletrack	Atascadero	CA	-120.74036	35.424428
7	Cabrillo Peaks	Singletrack	Morro Bay	CA	-120.824981	35.346757
8	Manzanita Loop	Singletrack	Los Osos	CA	-120.853337	35.292096
9	San Fernando Rd.	Fire Road	Santa Clarita	CA	-118.500331	34.364751

Figure 4: Database Table

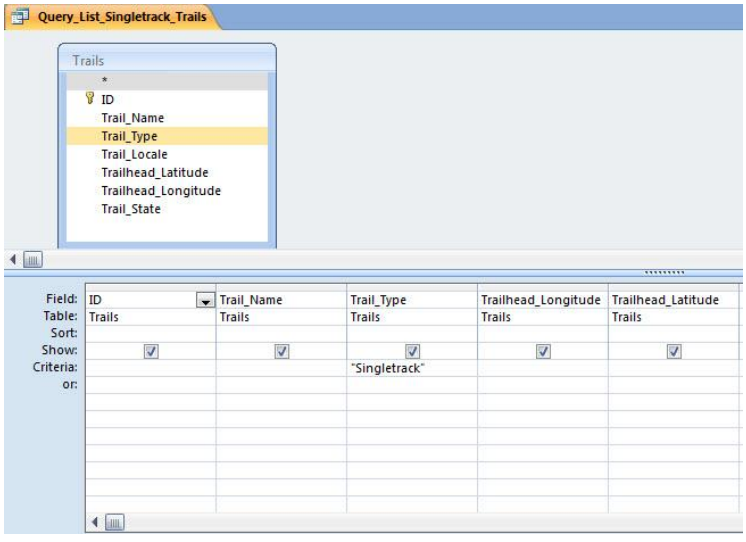


Figure 5: Simple Database Query

Establish the Database Connection

After creating the database, a connection was needed to enable Visual Basic to read and access data from the database. Using a past knowledge of VBA for Microsoft Access, we made an attempt to use the ADODB Recordset class. This attempt was unsuccessful though, because Visual Basic 2008 contained a different set of classes to work with than VBA for Microsoft Access did. After some research, we discovered the OleDB Connection class as a solution to make the connection. This connection consisted of several parts:

- Creating the connection
- Creating a data adapter to populate a data table (A Data Table is a behind-the-scene essence of a database within Visual Basic)
- Creating a command builder to handle the tasks of updating, inserting, and deleting data from the data table
- Creating the data table variable at the modular level (meaning it can be used anywhere throughout my project and code file)
- Creating an integer variable to keep track of position in the data table

We produced code under the form's load event (see Form Load Code in Appendix A) so that upon launch of the form, connection to the database would occur automatically. We added specific lines of code to specify the data source (the directory where the database was located), open the connection, connect the data adapter, and populate the data table.

We also followed an example from a reference text and created a form which would navigate between records from the database, as well as add, edit, or delete records from it. The form contained several text fields displaying the records as well as several buttons to navigate and edit the records. We used some If/Then structured code to make sure each record could be viewed and message boxes to help with debugging and tracing any connection errors that arose.

Design the User Interface

Designing the interface was a relatively easy process, given the drag-and-drop capabilities of Visual Basic. We began by creating a user form (see Figure 6) and then added a label control, combo box, and two buttons. The label control provided instructions for the user on how to use the control. The combo box provided a drop-down list of items for the user to select. The two buttons were labeled "Create KML File" and "Close." "Create KML File" would execute my code to create a KML file for viewing with Google Earth, while the "Close" button would safely exit the application and close any open database connections. Finally a pop-up message box (see Figure 7) would inform the user that their file was created.

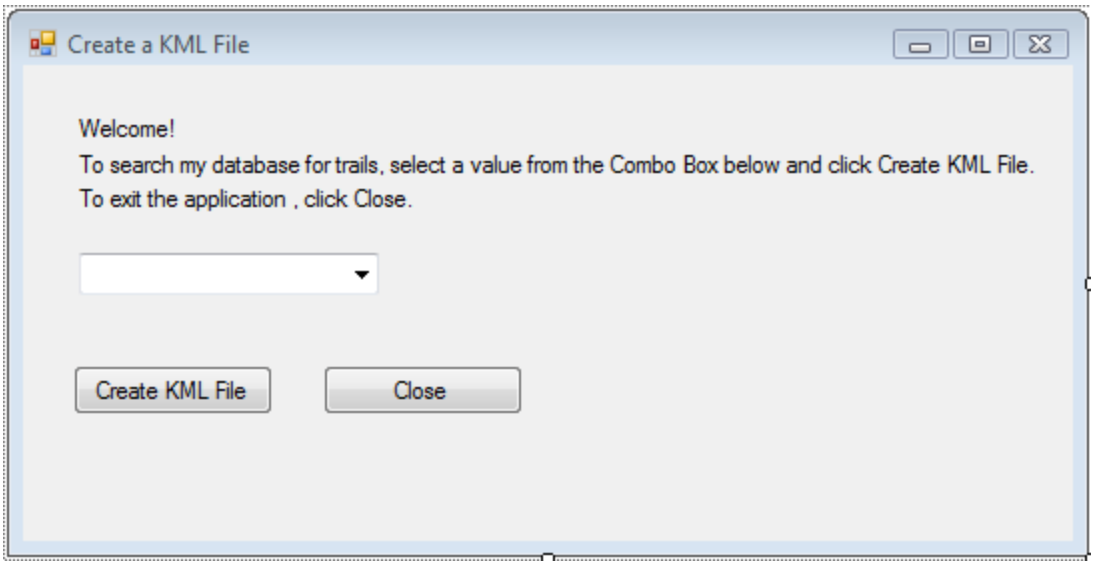


Figure 6: User Interface

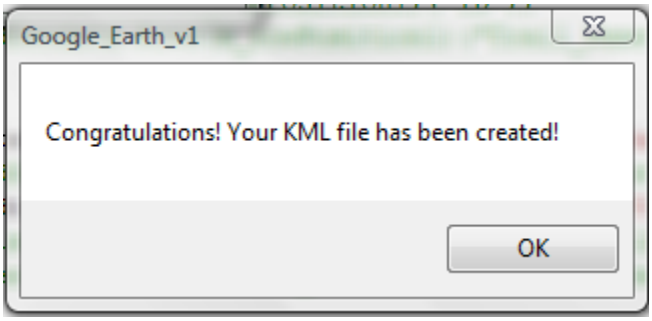


Figure 7: Pop-Up Message Box

Write Query Results to KML File

With knowledge of writing a KML file, connecting to a database, and designing a user interface, the application could now be assembled by writing sources code to piece together the desired actions. We created a design for user interaction with the application as follows:

1. Application is launched
2. User reads instructions
3. User selects a value (type of mountain bike trail) they are interested in
4. User clicks the “Create KML Button”
5. Visual Basic code runs, executes query, writes results to a KML file
6. User receives a pop-up notification that a KML file has been created.
7. User clicks the “Close” button to exit the application
8. User locates the file on the desktop
9. User double clicks the icon for the KML file
10. Google Earth is launched and displays the user’s query results visually in the form of labeled placemarks

Following this design, we began to strategically write our code, which is referenced in its entirety in Appendix A. We began code initialization code which declared all our variables globally for use throughout the application (see Initialization). Upon loading the user form, we inserted code to create the connection to the database (see Form Load Code). We also added code to fill the combo box with selectable options from the database by querying unique trail types contained within the database (see Code to fill Combo Box with Options).

Next, we programmed the *Close* button to properly exit the application (see Close Button Code in Appendix A). This code called a function (see Form Close Code) which properly closed the database connection and stopped the use of required resources.

Finally, the *Create KML File* button was programmed to have Visual Basic read the value selected by the user from the combo box upon being clicked (see Create KML Button Code in Appendix A). In our case, a type of mountain bike trail, whose value was either "Singletrack," "Doubletrack," or "Fire Road." This value from the combo box then was matched with the appropriate query in Access, to populate a new data table. We then wrote code to read through all records in the new data table (see Code to Create KML File & Launch Google Earth). Creating a *For* Loop basically told Visual Basic to read through each record in the table until it the last record was reached, and print data regarding the name, latitude, and longitude fields associated with each row to a KML file. This code can be viewed in Figures 24-26 under Code to Create KML File & Launch Google Earth within Appendix A.

Test Application

Following the completion of the application we tested the application for usability, reliability, and accuracy. We needed to ensure the forms on the user control operated as designed, that the same actions could be performed multiple times to yield the same results, and that the outputs were accurate. Using Visual Basic's debugging tool, problems that prevented the program from running properly were located and identified. We also utilized multiple message boxes to trace through steps in the process and confirm that actions were performed in the proper sequence according to our design.

One of the main errors encountered was in how Visual Basic read the data table. We noticed it would count the total amount of records or rows, but when running through the *For* Loop, it would sometimes skip a record. This error likely stemmed in a difference between how the data table stores its rows and how the database displays its rows. To fix this we needed to rewrite several formulas with the addition of a +1 or a -1 element to ensure the tables were read correctly and fully. These changes can be seen in the code in Figures 22, 25 & 26 under the Code to fill Combo Box with Options and Code to Create KML File & Launch Google Earth sections of Appendix A.

Revise Application

In addition to testing and fixing errors in the application, we also determined the need to make some changes, primarily to make the application more user-friendly. Instead of using *If* statements to run existing queries pre-designed in Access, it would be better to write an SQL

statement to run the queries. After some short studying of SQL, we were able to draft SQL statements that ran the queries and generated the data tables. These results of these revisions are shown in the code in Figures 22, 25 & 26 under the Code to fill Combo Box with Options and Code to Create KML File & Launch Google Earth sections of Appendix A.

We also decided to eliminate the hardcoded values in the combo box and link the combo box's drop-down options to those values contained in the database. This would enable the combo box to be up-to-date should trail types be added or removed from the database. Next, we reconfigured our SQL statements to select the appropriate records from the database whose *Trail_Type* field corresponded to the trail type field selected by the user from the combo box. These changes are reflected in Figure 22 within Code to fill Combo Box with Options (Appendix A).

Automate Application

Finally, another desired feature of the application was automation. Automating the application would provide the benefit of automatically launching Google Earth to view the query results, instead of requiring the user to find and open the file on their Desktop. Dr. Yang suggested we research batch files, which are files used to launch applications, perform file operations, or open files.

After a few hours of research, we knew the basics of batch files and were able to write one to launch Google Earth and open the KML file located on the desktop. However, this batch file wasn't sufficient because it did not connect to Visual Basic. After some research, we were able to create a function within Visual Basic and copy some code syntax to create the same batch file within Visual Basic. Upon the click of the same "Create KML File" button, the function was evoked to open a newly created KML file and launch Google Earth (see Figure 26 within Code to Create KML File & Launch Google Earth and Figure 27 within Code to Launch Batch File, all within Appendix A). After some further testing, we now had a fully developed application, complete with an automated launch of Google Earth.

Design Part 2: Web-Based Application

Part 1 was somewhat limited in scope because the application could only be run locally on a computer. In order for a user to run the application, they would have to have both Google Earth and a recent Visual Basic editor/reader (Visual Studio 2008 or Visual Basic 2008 Express Edition) installed on their machine. Additionally, they would have to edit code to specify the directory of the database they would have to import and save on their hard drive.

Even though Part 1 was successful, its scalability was somewhat limited. The application would be more beneficial if the features were accessible to a larger audience. Instead of a VB.NET form application, an ASP.NET web application would be an even more suitable solution. By building an ASP.NET web application using the same ideas from the VB.NET form application, an ASP.NET Web Application would offer the same functionality but be available to anyone via the internet. Users would be able to design a query online, run it, and view the results in a Google Map all within the same browser window or tab. A new project goal was developed to create an ASP.NET web application hosted on Dr. Yang's web server, IME5. For more information regarding ASP.NET technology please refer to the .NET Technology section of the literature review.

The proposed layout of this application was created as shown below in Figure 8. An ASP.NET Web Application would serve as the central component of the solution. The application would interact with the Access Database and also house a control to display a Google Map. Source code would be written behind the ASP pages in Visual Basic to specify the actions to be performed. Upon execution of the VB Code, the IME5 server would produce an HTML output to

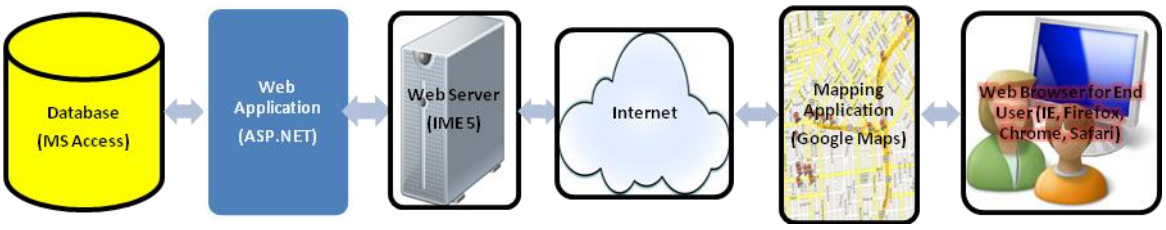


Figure 8: Design Part 2

form a page which is read and displayed graphically in the user's web browser. The Google Map is shown toward the right of Figure 8 because during its request the web server would have to use the internet to request permission from Google for use its map. This would be done by contacting the server that hosts Google Maps and validating a key-code provided initially by Google for use of the map at a certain URL. Like Part 1, the design of the application evolved with even further research based on the idea of the layout. Experimentation using trial and error techniques was used until possibilities were exhausted or solutions were found and built into the application.

Methodology Part 2

Research

Our research for Part 2 began with investigating Google Maps and how to embed a Google Map in a web page. We explored fully how Google Maps worked and discovered the Google Maps Application Programming Interface (API). An API is basically a collection of libraries that contains routines, data structures, object classes, and protocols created to support the development of applications.

The Google Maps API is a tool which provides developers with the ability to embed a Google Map in a web page within their own web application. Developers can write code to manipulate the map and link it to other components of their application such as a database or user interface. We found that the Google Maps API is available for free from Google and can be activated by using a code known as the API Key, given the developer specifies the directory or specific URL where the web page containing a map would be published. The only exceptions to this are websites that are only available to paying customers or sites that are only accessible within a specific company or intranet. In order to create our ASP.NET web application, we applied for a Google Key using the URL *http://ime5.ime.calpoly.edu/ime312labuseb/ime312_stu313/HomePage/*. This would enable a Google Map to be embedded on any website published within the HomePage folder. We also applied for a second key for a local host directory (*http://localhost:49486/Default.aspx*) where we could conduct preliminary builds and tests of the web application. Since this was a local host, this directory was only accessible from the computer developing the application, not the web.

We did additional research in learning about ASP.NET technology and how to use the integrated development environment (IDE) within Visual Studio 2008. We found that a project can be created to contain multiple websites. Each website contained 3 core files: the ASPX file, the source code file, and the source code designer file. The ASPX file brings together ASP and HTML code to specify the location, size, and position of all web controls and elements forming the page. The source code file contains all the code which pertains to the functions the application website will perform and can be written using Visual Basic, Visual C#, or JavaScript programming languages. The designer source code file is an auto-generated file that contains the field declarations for controls in the ASPX file. In addition to these three code files, the web configuration file is a file contained within the project that specifies the connection strings to databases referenced in any of the websites.

We found the Visual Web Developer environment very similar to the Visual Basic development environment used in Part 1. Both contained a solution explorer to view the files within the current project, a properties window to view the properties of a particular element or control, toolbars containing all the standard file operations and edit functions, and a toolbox containing all the controls that could be added to a project. There were however, a few differences. The set of controls for use in ASP.NET web applications differed in that a combo box was replaced by a drop-down-list and several other web-only controls were added. Another change in terms of

toolbars was the addition of a *Build* menu. This menu contained options for building the project, rebuilding the project, and publishing the project. With some simple testing, we found it both necessary and practical to build and edit the project from one location or folder and publish all project files to another location of folder, separate from the one used for editing.

Another difference was the views available within the ASPX page. These views were the *Design View*, *Source View*, and *Split View*. The design view enabled the dragging and dropping of web controls from the toolbox and was made to create the page visually, while the source view contained the code used to create these objects. The split view showed the screen split into half being a design view and half being a source view. The ASPX page could be designed using either code in source view or the drag-and-drop capabilities of design view. Either way, the opposite view would generate the missing components. Now having a good grasp of the ASP.NET web development environment, we changed the focus of our research to investigating ways to include a Google Map in the project.

Google Maps Web Control

An essential part of the project would be figuring out a way to embed a Google Map in our ASP.NET web application. We had permission to utilize Google Maps, but no way of inserting it in our application. In reading Google's documentation pertaining to the API, we found that maps are traditionally inserted and manipulated using the JavaScript programming language. In order to create the ASP.NET web application we would either have to learn JavaScript or figure out a way to utilize Google Maps with a familiar language such as Visual Basic. A good solution would ideally be a web control, which could be added to ASP.NET's toolbox and used in the application through a simple drag-and-drop operation.

Further research resulted in finding wrappers for the Google Maps web control. A wrapper for a control or feature is essentially a complied version of the control translated for use in another language. We found wrappers for the Google Maps web control which enabled development in ASP.NET, eliminating the need to write JavaScript code. This would enable us to develop in ASP while all the JavaScript conversions were taken care of in the background. After trying and testing many of these wrappers, Google Maps could be inserted into ASP.NET web pages. However, we later discovered that we could not manipulate our maps because the source code used to access the control was written in Visual C# instead of Visual Basic. We made an attempt to translate the code files written in Visual C# to Visual Basic and then redevelop the control using Visual Basic files. This resulted in numerous errors in terms of the files interacting with each other. We fixed several configurations but were unable to figure out solutions to many of the other errors and ended up abandoning that approach.

Going back to research, we attempted to find a wrapper for the Google Maps web control for ASP.NET that was written with source code in Visual Basic. Results from these searches were minimal so we began to seek assistance by posting to forums and developer blogs online. We also solicited help from fellow students with programming knowledge in both the languages we encountered during research and in Visual Basic, with which we were familiar.

One of these students was a friend named Paul Case, who helped devise our solution. In several collaboration sessions, we communicated our needs to Paul and determined how his knowledge could help solve our problem. We then used our combined knowledge to research and experiment with methods for manipulating a Google Maps web control with Visual Basic source code. We elaborated on previous methods where Paul wrote code in Visual C# and JavaScript to call upon Visual Basic source code, but those attempts were unsuccessful. After further research, we discovered a control developed by a Spanish company, Subgurim. Subgurim had made a Google Map web control for ASP.NET, but unlike previous wrappers theirs was compiled as a dynamic link library (DLL). A DLL is a library of executable functions or data that can be used by a Windows Application (). Subgurim's DLL could be imported into Window's applications such as an ASP.NET web application and also be downloaded for free from the web, given it was for non-commercial use. Additionally, the Subgurim control could pass a developer's key code for the Google Maps API directly to Google to authenticate use.

We downloaded Subgurim's DLL file and followed their step-by-step instructions on how to import the DLL and reference it. By using one line of JavaScript code and adjusting a few configurations (a reference declaration and a web configuration string containing our Google Maps API key), we were able to create a simple Google Map in a webpage. By following some sample source code from Subgurim, we learned how to manipulate the map and some basic properties. Like the previous wrappers, their source code was also written in Visual C#. However, when translating Subgurim's code to its Visual Basic equivalent online, we could manipulate the map since both code in Visual C# and Visual Basic referenced the same library from the DLL. Previous wrappers only included a way to program using Visual C# because the lack of an existing DLL. From here, we played with code derived from both the drop-down Intellisense menus and past VB code syntax to manipulate the control. The figures contained within the Page Load Code and Query & Display Markers Button Code sections in Appendix C are the primary sections where Visual Basic source code is used to manipulate the Google Maps web control.

Design the Database

In Part 2, there was nothing new to consider in designing the database. The same database created in Part 1 was utilized in Part 2 (recall Figures 4 & 5) with the exception of a different type of connection (see Establish the Database Connection). For easy reference, the database was located in the *App_Data* folder within the ASP.NET web application.

Design the User Interface

Designing the interface was a relatively easy process given the drag-and-drop capabilities of ASP.NET, which closely paralleled the design process from Part 1. The only difference between drag-and-drop in Visual Basic and drag-and-drop in ASP.NET is that with ASP, code is rendered in HTML with ASP tags as opposed to Visual Basic where it is not. The code is rendered as part of the *.aspx* page within the web application. For our project, *GoogleMap107,* our ASP code was rendered within *GoogleMap107.aspx*, which can be seen in ASP.NET Web Controls in Appendix B. We began by creating a web form and then added a label control, drop-down list box, and a

button. The label control provided instructions for the user on how to use the control. Instead of a combo-box like Part 1, we used the ASP.NET equivalent of a drop-down list box. The drop-down list box provided a drop-down list of items for the user to select. The button was labeled *Query & Display Markers* which would execute code to read through appropriate database records and place markers accordingly on the Google Map. It would also be used to evoke code that cleared previously displayed markers, so the unique query solution could be shown. Finally we inserted the Google Map by following the procedure mentioned earlier to complete our interface seen below in Figure 9.

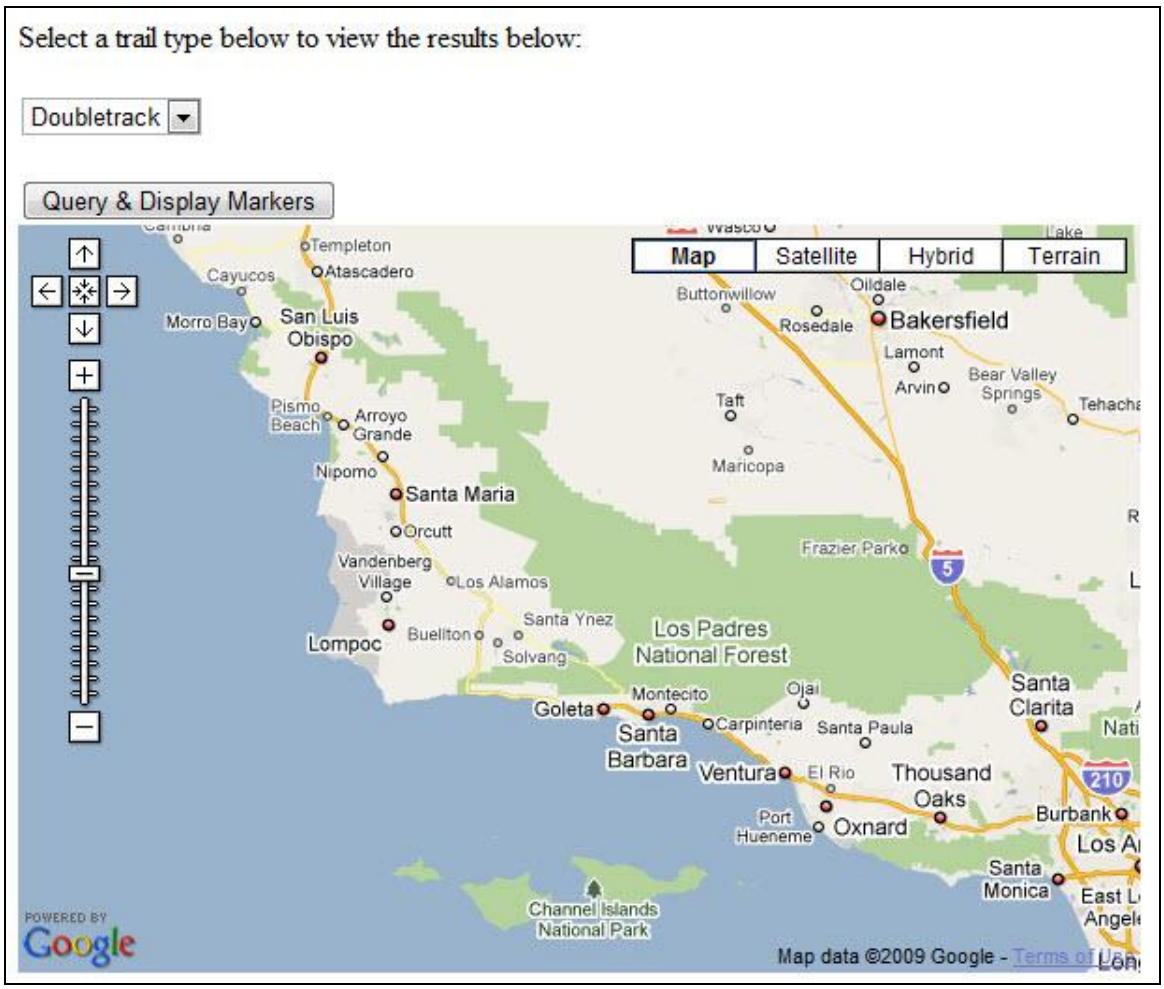


Figure 9: User Interface for Web Application

Establish the Database Connection

In our ASP.NET web application, we used the SqlDataSource web control which differs slightly from the OleDB Connection procedure used in Part 1. The SqlDataSource web control is a web control designed to access data from a database such as Microsoft's Access, Microsoft's SQL Server, and Oracle's Oracle using an OleDB type connection on the .NET framework. Like all web controls, the SqlDataSource was inserted with a simple drag-and-drop operation where HTML codes with ASP tags are rendered (see ASP.NET SqlDataSource Control in Appendix B). In design

view, we clicked the control's smart tag which enabled us to configure the data source by using a pre-created design wizard. Following the wizard we completed the following steps:

1. *Choose the data connection*: To choose the data connection, we selected the database of mountain biking trails located in the *App_Data* folder of our project as shown in Figure 10 below. We also checked a box with an option to save our database connection string in the web configuration file (see Appendix D), so the connection could easily be found and edited globally in future reference.

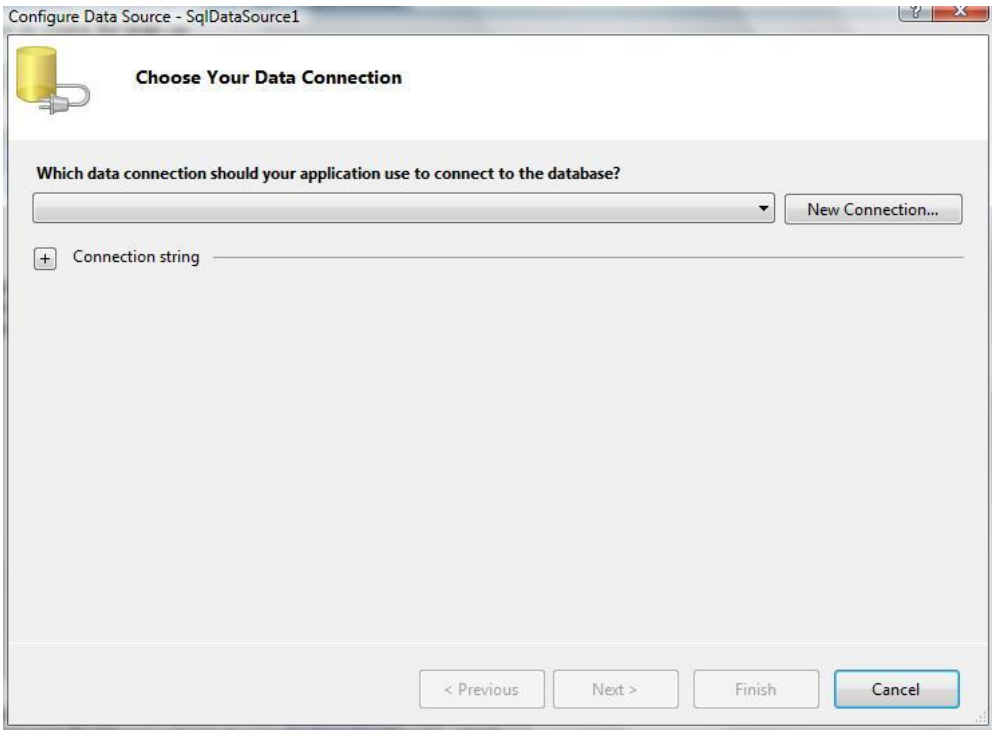


Figure 10: SqlDataSource Data Connection Prompt

2. *Configure the Select Statement*: By configuring a Select statement with the window seen in Figure 11, we specified what data to retrieve from the database. In our case, we chose to retrieve all fields (columns) from the table labeled "Trails" (see Figure 5), since we planned to create our queries from this datasource later in the process. The wizard automatically generated an SQL Select statement that read: "SELECT * FROM [Trails]". Other options available for this statement included selecting only particular fields, or only selecting data fitting a certain criteria.

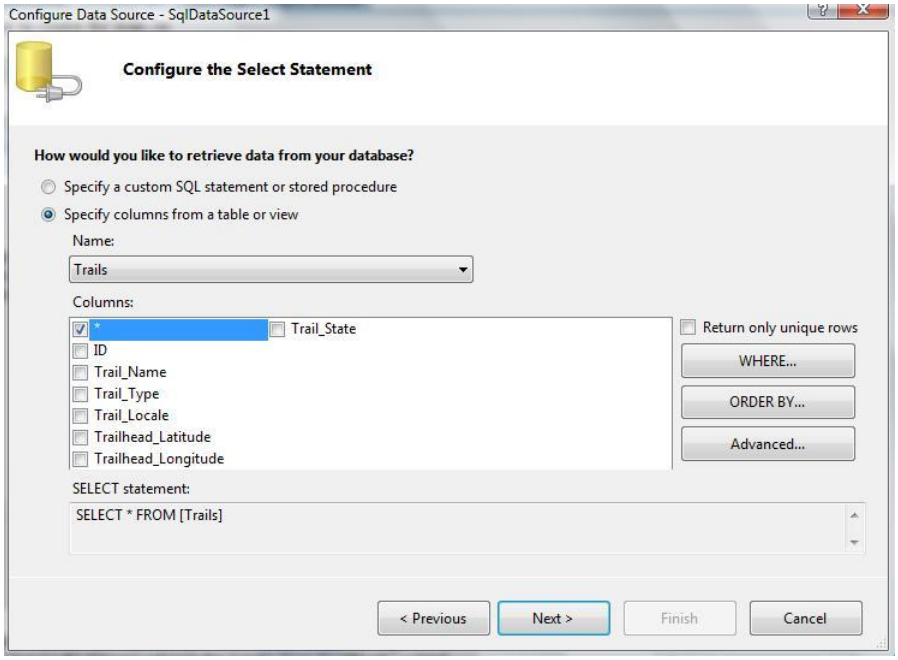


Figure 11: SqlDataSource Configure Select Statement Prompt

3. *Test the query*: By clicking a button within the wizard as shown in Figure 12 below, we tested our connection and confirmed that our Select statement retrieved the proper data.

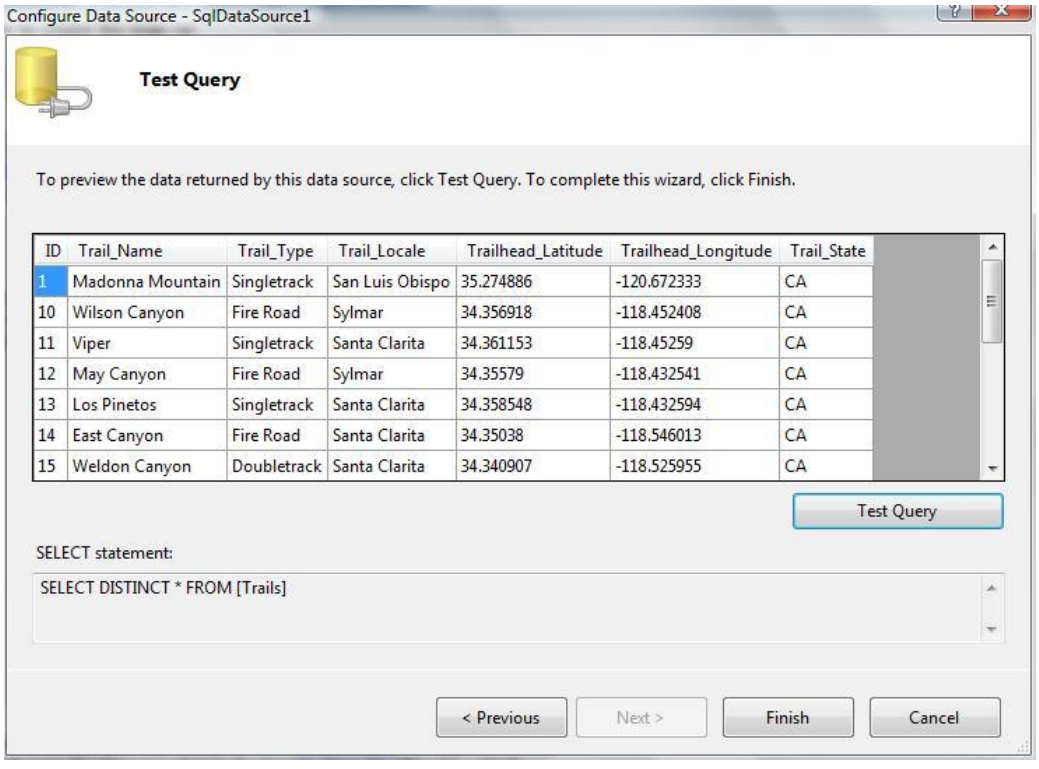


Figure 12: SqlDataSource Test Query Window

Unlike Visual Basic where we had to add specific lines of code to the form's load event to specify the connection string (directory where the database is located), open the connection, and connect to a data table, the SqlDataSource control and its wizard in ASP.NET took care of these actions automatically by writing code to our ASP file (see ASP.NET SqlDataSource Controls in Appendix B).

Next, we began figuring out code to programmatically access the contents of the SqlDataSource control we had created. We did this by creating a data view, or an instance of our database table that runs in the background of the ASP.NET application (see Figure 28 under Query & Display Markers Button Code in Appendix C). To link our data view to our database connection, we ensured the DataSource mode specified in the ASP code was set to *Data Set* and then linked the data view to our SqlDataSource.

Next, we created another SqlDataSource for the drop-down list box to draw from (see ASP.NET SqlDataSource Controls in Appendix B). We made another connection to the database of mountain bike trails (see Figure 13 below) with a select statement that read, "SELECT DISTINCT [Trail_Type] FROM [Trails]". This statement brought in unique values from the *Trail_Type* database field as selectable options from the drop-down list box. With this select statement, any changes to the database concerning trail types would be automatically reflected in the options of the drop-down list box seen by the user.

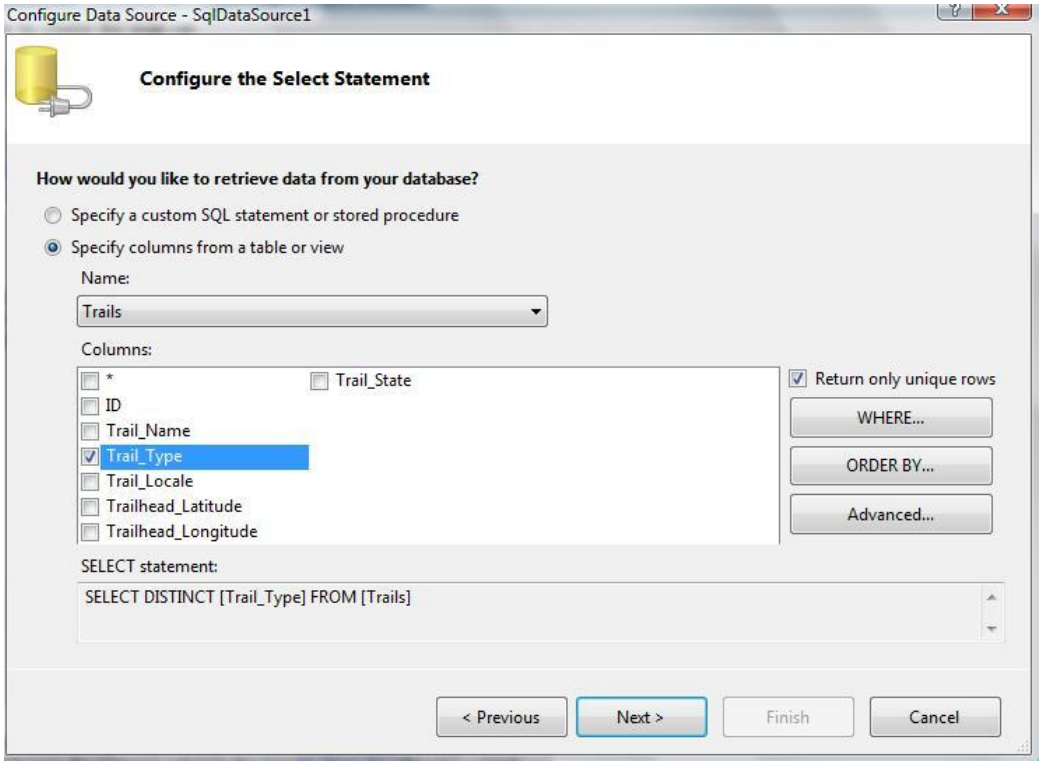


Figure 13: SqlDataSource Configure Select Distinct Statement

Configure Google Maps Web Control

Next, we configured the Google Maps web control with several features for the user. To allow the user to view and move the map in different ways, we enabled a control to allow dragging of the map with the mouse, a control to allow movement of the map via keyboard arrows, and a control to use the scroll wheel on a mouse to zoom in or zoom out. These controls would enable users to explore the locations and find more information about their queried entities. We also added the large map control, which is a pre-built control from Google designed to add "+" and "-" buttons for zooming in and out, a scaled bar to show the current zoom level, four navigational hours for panning north, east, south or west, and a centralized button to return to the previous position before panning. For a view of these features, please consult Figure 9 in Design the User Interface. For applicable code, please consult Page Load Code in Appendix C.

Write Query Results to Google Map

By figuring out how to manipulate a Google Map, create a connection to the database, and design the user interface, we had the essential components for our solution. To assemble our application, we took these components and combined them with source code written in Visual Basic. A design for user interaction was created as follows:

1. Web Site is Visited
2. User reads instructions
3. User selects a value (type of mountain bike trail) they are interested in
4. User clicks the "Query & Select Markers" button
5. Visual Basic code runs, executes query, places markers to the Google Map
6. User views the results of their query visually in the form of a Google Map within their web browser without ever leaving the page. They also view text containing information on the trails shown.

An important note was that this design was a much simpler process than Part 1 from both the user's and designer's points of view. Behind the scenes, however, the design was much more complicated. Hundreds of operations would actually take place with the user's browser sending requests to the server and the server returning requested page content back to the browser. There would also be many operations occurring with the Subgurim control connecting to Google's mapping servers. For more information on these processes please consult the Literature Review and Design Part 2.

Following this design for our ASP.NET application, we began to strategically write our source code. Taking the code used to create a dataview (which would read through the recordset specified by the SqlDataSource), we created a *For* Loop, which read through each record in the table until it the last record was reached. Within this loop, we followed an *If/Then/Else* syntax which operated as follows: If the current data row being read contained a trail type corresponding to the trail type selected by the user in the drop-down list box, a marker was created for that trail by referencing the values from its latitude and longitude fields (see Query & Display Markers Button Code and Figure 28). Additionally, the trail name, locale, and trail type were written to a string and displayed in the label above the map. This procedure

continued until all records in the database were read and tested with the *If/Then/Else* syntax. Below, Figure 14 shows the resulted outcome for a query specifying trails of type, "Singletrack."

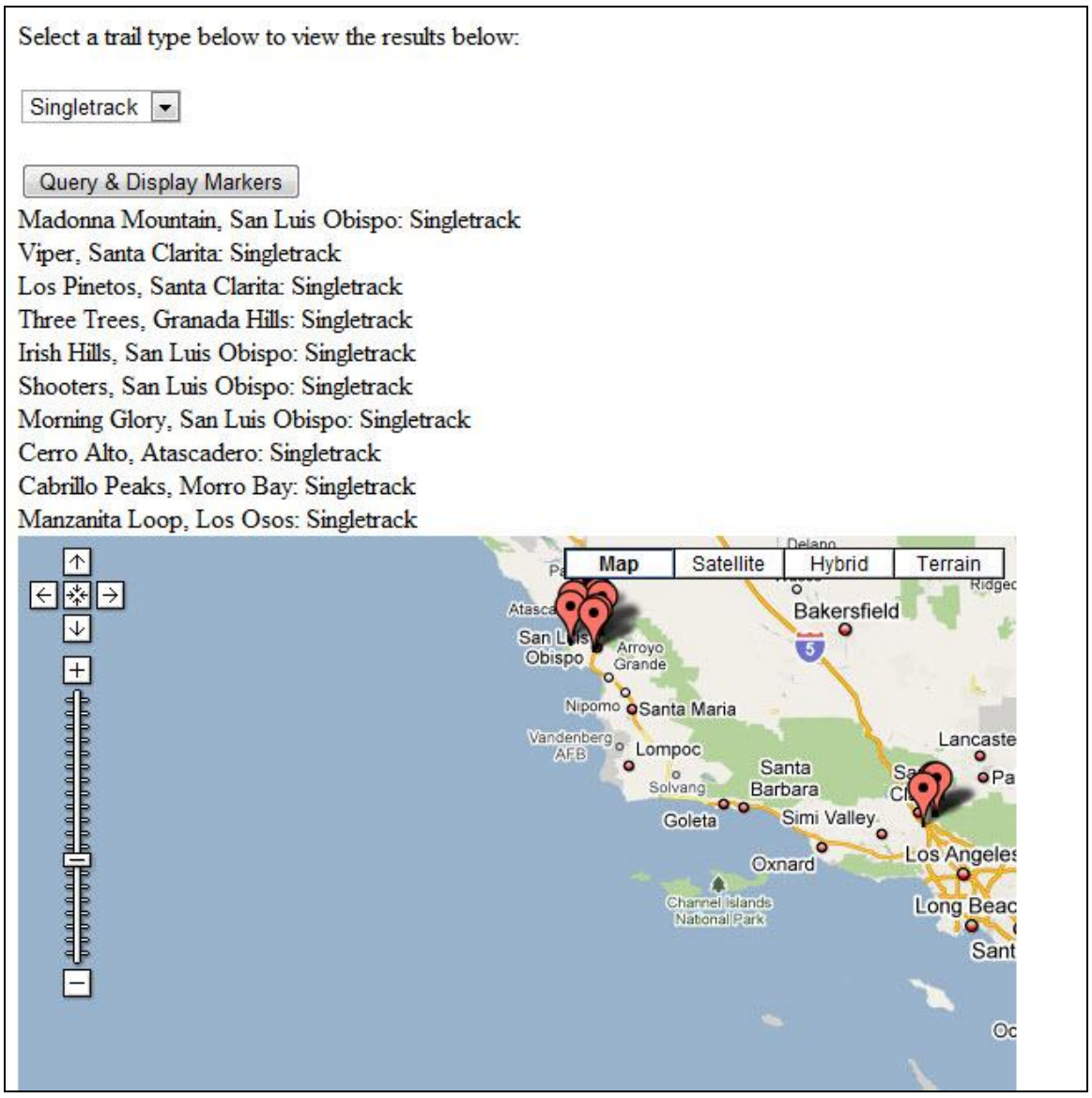


Figure 14: Screenshot for Operational Web Application

Building & Publishing

Finally, the last step was to publish to the web using the IME5 web server. This involved both a building operation and a publishing operation. In the build operation, the application was assembled. Throughout the process, we had built and rebuild applications and tested them on a local host. When a build operation succeeded, the application would be ready for publishing. When the build failed, there would be a list of errors, warnings, and messages which would aid in troubleshooting the problem. On our final build, we encountered errors pertaining to how the database was referenced. With some investigation, we found that the file path for the

database could not be read by the IME 5 server because the server did not recognize the drive reference. This occurred because the database was referenced from a mapped network drive in the lab at the location *Z:\ime312_stu313\TestPage1\App_Data\ Google Earth Project_v2.mdb*. To solve this problem we changed our web configuration file (see Figures 32 and 33 in Appendix D) to reference the server's version of this network drive by changing the location to *C:\InetPub\wwwroot\ime312labuseb\ime312_stu313\TestPage1\App_Data\Google Earth Project_v2.mdb*.

To publish our web application, we created and specified a folder called *HomePage* on the IME5 web server for publishing and selected an option to publish all data files (see Figure 15 below). The final result was posted and is accessible by visiting *http://ime5.ime.calpoly.edu/ime312labuseb/ime312_stu313/HomePage/googlemap107.aspx*.

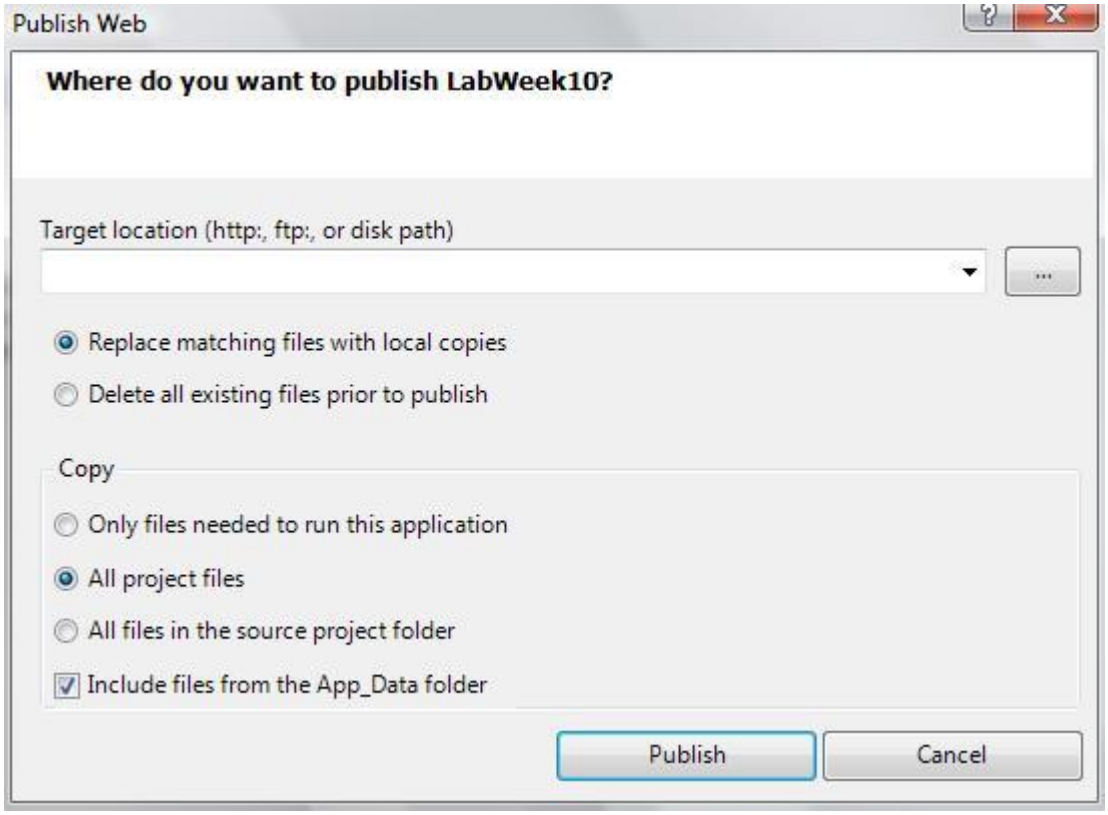


Figure 15: ASP.NET Publish Prompt

Results

Key Findings

Our investigation of methods to create a tool to visually display the results of a database query proved very successful. We were able to design two applications using the methods we investigated. A locally-based application was developed to successfully display the results of a user-built query in Google Earth, while a web-based application was developed to successfully display the results of a user-built query in a Google Map embedded in an ASP.NET web page. Both applications utilized Microsoft's .NET technology as a development environment and Microsoft's Visual Basic as programming language. Each of these applications could also be expanded for business use to locate products, assets, customers, historical events, suppliers, and facilities of interest. Such expansion would result in substantial savings in time, effort, and money for a business in both making a decision and experiencing the results of a decision. The code used for the locally-based Google Earth application is available in Appendix A. For the Google Maps web application, code is available in Appendices B, C, D, and E.

The locally based application was developed on a laptop computer with Visual Basic 2008, a component of Visual Studio 2008. The application references a locally stored Microsoft Access Database and published query results to placemarks in Google Earth. Expansion of this application would require configuration with a database, revising the file paths, and the installation of Google Earth and Visual Studio, if not present. A locally based web application would be best for a business using it in-house to locate entities which change little over time.

The web based application was developed using the ASP.NET web development technology within Visual Studio 2008. The web pages comprising this application were based off Visual Basic source code and are accessible on the web from any browser by connecting to pages on the IME Department's IME5 Web Server. The web application enables users to dynamically interact with a database hosted on the web server and show the results of queries in a Google Map embedded in the website being viewed within their browser. Expansion of this application would require rebuilding the same application on the web server of a particular business or organization since the university system follows a not-for-profit model. With a web application though, once an initial database design is created the system would require little maintenance since the database could be edited and appended to by users. Such a site could be hosted either within a company intranet for exclusive company use or on the internet for use by customers and the public. An enterprise license would also be required for commercial use of Subgurim's Google Maps Web Control for ASP.NET.

Opportunities for Improvement

There are many opportunities for improvement for these applications, the greatest being expansion. The basic ideas of these applications can be expanded to suit a variety of business applications, depending on the databases available. Several potential applications are mentioned in the following section, Potential Applications. For expansion, one possible improvement is creating the ability to work with databases other than Microsoft Access such as Microsoft's SQL server or Oracle's Oracle. This would be beneficial in that many of these databases can store much larger amounts of information for the larger business purposes, thus expanding the scalability of our application. As mentioned previously, expansion of the application would also require a commercial license of Subgurim's Google Maps Web Control.

The application can also be expanded to include the use of other features of Google mapping technology such as polygons or paths. Paths in particular can be constructed with an algorithm that finds the best method of travel between multiple placemark points contained in a database.

Finally, the application can be expanded to work with technologies such as global positioning systems and real-time traffic management systems. Integration of these technologies could improve business applications associated with planning routes for pickup and delivery.

Potential Applications

There are many potential applications that can be built using the framework and the ideas of the applications developed in this project. Below are marketable suggestions for expansion of the application:

1. *Storage Container Industry (e.g. PODS, PackRat):* As mentioned previously, storage container companies could use the applications to locate their storage containers and gauge their locations relative to each other. From this, they can make better decisions in efficiently placing and removing customers' containers each day and determining the proper amount and placement of unused storage containers in warehouses.
2. *Grocery Industry (e.g. Albertsons, Safeway):* The grocery industry could use the applications to locate consumers who had purchased recalled products. Given product recalls of meat and peanut butter, the grocery supply chain could identify which stores the affected products were sold at and then locate and contact the affected customers using information from their club cards. By notifying customers of the recalls, grocery stores can prevent their customers from experiencing the consequences of the bad products. Additionally, it can help improve their customer service by easily allowing and verifying an exchange of the recalled product for one of the same or greater value.
3. *Car Dealers (e.g. Ford, Toyota):* Car dealers could use the applications to locate vehicles in their lots and showrooms by connecting it with a database of available vehicles. By easily locating the vehicles fitting a customer's needs, the dealer can easily show the customers to new and used vehicles without wasting time searching through hundreds of vehicles looking for the desired particular set of features.
4. *Medical Research:* The applications could be combined with patient databases in the medical industry for data mining research. Such research may draw conclusions such as patients having cancer because the proximity of their residence to power lines, patients having hearing problems as a result of their proximity to the railroad tracks, or patients having breathing problems as a result of poor air or water quality where they live or work. Also, medical research may be able to draw conclusions from data regarding the transmission of widespread diseases geographically.
5. *Historical Research:* The applications would also work well with historical research in finding events that have happened at a particular location, within a particular region, or by a particular person or group of persons over time. Seismologists may study the movement of the earth, rocks, or hills over time. Meteorologists may study past trends in weather and investigate which geographical locations are more susceptible to different weather types. Biologists can use the applications to help track the geographic movement of animals and plant species over time. Local governments may study the trends of population and development growth in a particular area.
6. *Business Planning & Marketing (e.g. Los Angeles Dodgers, recording artists, Cisco):* The applications could work for a variety of business planning and marketing purposes.

Sports teams such as the Los Angeles Dodgers could use the database to view the locations of their season ticket holders and fans. They could use this information to market differently between represented and underrepresented areas, as well as create more effective promotional schemes and event nights. Recording artists can use the applications to help locate their fans and album purchasers to plan tours and events which will capture the best audiences. They can also use the information to locate areas to further promote their works. Finally, companies such as the tech-giant Cisco can use the application to access a database of supply chain parties. This could help them decide which markets are most appropriate for manufacturing and distributing their goods and services with their continued globalization.

7. *Delivery Services (e.g. Dominos Pizza, United States Postal Service):* The application could be used in business for a variety of delivery service providers. A company like Dominos Pizza could use the software to locate customers currently requiring delivery service and help drivers plan routes to efficiently deliver warm pizzas, without wasting time locating each address on a map and delivering to these addresses in an inefficient order. The United States Postal Service or local garbage collectors could use the application to locate customers requiring the pickup and delivery of mail or garbage. From this, they can efficiently plan routes. With further research and technology, the application could be expanded to plan routes based on real-time traffic conditions or alert customers when their mail has arrived or their garbage has been picked up.
8. *Oil& Mining Industries (e.g. Chevron, Shell):* Oil and mining companies could use the application to link to a database to locate certain excavation areas or deposits and their proximity to each other. This knowledge would help plans for pickup and potential future excavation sites.
9. *Public Safety Planning (e.g. city and county governments):* Local governments could also use the application to pair with databases for public safety planning. City officials can investigate the locations and amount of certain traffic incidents to investigate possible problems with roadways, signage, or traffic control devices. The application may also be expanded from containing placemarks to containing paths, so it could be used for applications such as emergency evacuation plans for a building or region. Local governments have databases containing large amounts of geographical information.
10. *Tracking/Tracing:* The application could also be used in conjunction with a variety of tracking/tracing procedures. The application of RFID (Radio Frequency Identification) is used by businesses such as Dell and Wal Mart to track the movement of goods and parts across their supply chains. Since RFID data is often stored in databases, the databases could easily be queried with these applications to show the movement of certain items on a map over time.
11. *Emergency Services Personnel:* Local emergency service personnel could use the applications to find the best way of accessing a region given current emergency conditions. The involved agencies such as police, fire, and emergency medical responders would be routed to the area of need the quickest way possible, though the use of placemarks or paths. One particular example parallels the California Department

of Forestry's response guide for Fire Station 12 in the County of San Luis Obispo. The guide shows turn by turn directions on how to best serve the outlying areas of the city (county jurisdiction). The example below in Figure 16 shows an emergency response plan for responding to calls on Watson Dr. just north of the San Luis Obispo city limits.

From: Station 12 **To: Watson Dr.**

Turn	Road	Mileage
Left	Santa Rosa (Hwy 1)	5.5
Right	Education	0.1
Right	Watson	

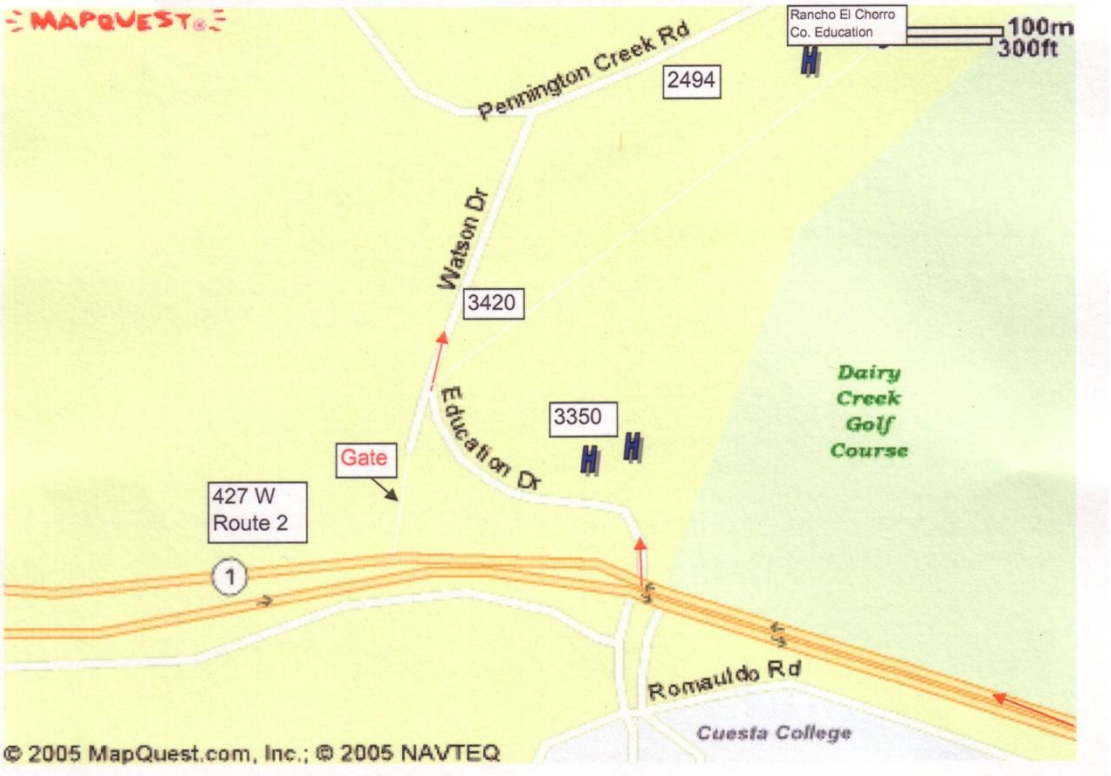


Figure 16: CDF Response Guide for Watson Dr - County of San Luis Obispo

This information could be combined with the Google Maps applications to visually show the path required for service on an as-needed basis as opposed to thumbing through pages in the response guide.

Conclusions

In our investigation of methods for using maps to show queried geospatial data, we successfully developed two applications- one that worked locally to show the results of an database query in an instance of the Google Earth application and the other that worked through a web server to show the results of a database query on a Google Map within a user's web browser. The framework and ideas of these applications can be expanded to business applications to help decision makers make better decisions based on the greater knowledge achieved in promptly knowing the location of entities and the spatial relations between them. These better decisions, in turn, can result in improved operations and cost savings for a company. All objectives were accomplished in the development of each application. Each application:

- Could successfully be opened, ran, and closed by the user
- Was based off Microsoft's .NET Technology
- Had a user interface that enabled the user (decision-maker) to build queries for geospatial data of interest
- Linked to a created database containing geospatial data that could be read or manipulated via the user interface
- Enabled the user to view of a customized map of their queried results using one of Google's mapping technologies

The approach of identifying, researching, experimenting, and building worked for the entire development of the applications, but the anticipated time durations were largely underestimated due to the difficulties encountered in the process.

From this project came several learning outcomes:

- Greater knowledge of .NET Technology including Microsoft Access databases
- Greater knowledge of the Visual Basic and SQL programming languages
- A basic understanding of dynamic web sites, ASP.NET, web servers and batch files
- A basic grasp at other programming languages (JavaScript, Visual C#, HTML, XML, KML)
- Familiarity with creating and manipulating Google Earth and Google Maps within each applications and programmatically
- A greater understanding with troubleshooting, researching, and diagnosing problems with code

This project served as a great opportunity to apply the skills and techniques I have developed in my undergraduate education as an Industrial Engineer, as well as foster some new skills that could be of use in future research or a future career.

Appendices

Appendix A: Google Earth Windows Form Application Code

The following is a step-by-step breakdown of the Visual Basic code used to create the initial Windows Form application which connected an Access Database with Google Earth on a local computer. This form application was built in Visual Basic 2008 Express Edition as a Windows Form Application.

Initialization

The following code seen in Figure 17 was used to define the class for the windows form application named *frmCreateKML* and declare the necessary variables to establish the database connection. These variables included the database connection, the data adapter, the command builder, two data tables and two integer variables. The *m_dtSelectedGeoData DataTable* was the datatable created based on the query while the *m_dtcboSelectData DataTable* was the list of selected items that would be referenced by and appear in the combo box. The first row position variable, *m_rowPosition*, helped track the position in the datatable pertaining to the referenced combo box while the second variable, *m_rowPosition1*, helped track the position in the datatable pertaining to the query.

```
Public Class frmCreateKML

    Private m_cnADONetConnection As New OleDb.OleDbConnection()
'Creates new connection

    Private m_daDataAdapter As OleDb.OleDbDataAdapter 'Create a data
adapter to populate a DataTable

    Private m_cbCommandBuilder As OleDb.OleDbCommandBuilder 'Creates a
command builder to handle the tasks

    'of updating, inserting, and deleting data behind the scenes.

    Private m_dtSelectedGeodata As New DataTable 'create data table
variable at the module level

    Private m_dtcboSelectData As New DataTable 'create another data
table variable at the module level

    Private m_rowPosition As Integer = 0 'Integer variable to keep
track of user's position in datatable

    Private m_rowPosition1 As Integer = 0 'Integer variable to keep
track of user's position in datatable
```

Figure 17: Google Earth App- Form Initialization code

Create KML Button Code

The code in Figure 18 was used in the event that the button *Create KML* was clicked. It simply calls the private subroutine, *Create KML*, which contains the code necessary to execute the query, write the KML file and trigger the launch of Google Earth (see Code to Create KML File & Launch Google Earth).

```
    Private Sub btnCreateKML_Click(ByVal sender As System.Object, ByVal
e As System.EventArgs) Handles btnCreateKML.Click

        Me.CreateKML() 'Calls the Create KML Procedure

    End Sub
```

Figure 18: Google Earth App- Create KML Button Click Event code

Close Button Code

Figure 19 shows the code that was used in the event that the button *Close* was clicked. It simply closes the form safely and properly, which ends the running of the form application. When it does this, it calls the *FormClosed* event (see Form Close Code below).

```
    Private Sub btnClose_Click(ByVal sender As System.Object, ByVal e
As System.EventArgs) Handles btnClose.Click

        Me.Close()

    End Sub
```

Figure 19: Google Earth App- Close Button Click Event code

Form Close Code

The following code seen in Figure 20 was executed upon the close of the user form. It safely and properly disconnects the database connection with the close method and frees any resources the connection may be holding with the dispose method.

```
    Private Sub frmCreateKML_FormClosed(ByVal sender As Object, ByVal e
As System.Windows.Forms.FormClosedEventArgs) Handles Me.FormClosed

        m_cnADONetConnection.Close() 'Explicitly closes the connection

        m_cnADONetConnection.Dispose() 'Frees any resources the
connection may be holding

    End Sub
```

Figure 20: Google Earth App- Form Close code

Form Load Code

Figure 21 shows code that was executed upon the user form's load event. It opens a connection to a datasource and creates a data table, and calls a procedure to put the data in the combo box for the user to be able to select items from. The connection string uses the provider *Microsoft.Jet.OLEDB.4.0*, the standard for connecting to an Access Database. The connection string is basically the directory where the database is located which is *C:\Users\Steven\Documents\MS Office Files\Access\IME 400 Google Earth Project\Google Earth Project.mdb*, a directory contained on the laptop where the application was developed. The open method opens the connection and the data adapter specifies the SQL statement which selects trail types from the *Trail_Type* field in the database table labeled *Trails*. The keyword DISTINCT selects unique trail type values from the field, so that types are not listed more than once in our combo box. The fill method of the data adapter populates the data table. Finally, the private subprocedure, *FillcboSelectData* (see Code to fill Combo Box with Options), is called to fill the combo box with the selectable options meeting the criteria from the query.

```
    Private Sub frmCreateKML_Load(ByVal sender As System.Object, ByVal
e As System.EventArgs) Handles MyBase.Load

        m_cnADONetConnection.ConnectionString = _

                "Provider=Microsoft.Jet.OLEDB.4.0; Data
Source=C:\Users\Steven\Documents\MS Office Files\Access\IME 400 Google
Earth Project\Google Earth Project.mdb"

        'Specifies the data source to which you want to connect


        m_cnADONetConnection.Open() 'Opens the desired data source


        m_daDataAdapter = New OleDb.OleDbDataAdapter("Select DISTINCT
Trail_Type From Trails", m_cnADONetConnection) 'Connects data adapter


        m_cbCommandBuilder = New
OleDb.OleDbCommandBuilder(m_daDataAdapter) 'Attaches the command
builder

        'to the data adapter

        m_daDataAdapter.Fill(m_dtcboSelectData) 'populates the
DataTable with the Fill() Method

        Me.FillcboSelectData() 'Calls the FillcboSelectData procedure


    End Sub
```

Figure 21: Google Earth App- Form Load code

Code to fill Combo Box with Options

The following code (see Figure 22) fills the combo box with selectable options upon execution of code in the form's load event (mentioned above in Form Load Code). A variable named *AddItem* was declared as a string to store the value of text being read and added to the combo box's list of selectable options. A *For/Next Loop* then read through each option in the data table until all rows were read. Within the loop the trail type value was assigned to the *AddItem* variable, whose value was then added to the combo box using the combo box's *Items.Add* method. Finally the code set the row position integer to the next value for the loop to repeat.

```
    Private Sub FillcboSelectData()

        Dim AddItem As String


        For i = 1 To m_dtcboSelectData.Rows.Count

            AddItem =
m_dtcboSelectData.Rows(m_rowPosition)("Trail_Type")

            'MsgBox("Adding item " & AddItem & " to Combo Box.")

            cmbSelectType.Items.Add(AddItem)

            'MsgBox("Added item " & AddItem & " to Combo Box.")

            m_rowPosition = m_rowPosition + 1

        Next i

        m_rowPosition = m_rowPosition - 1

    End Sub
```

Figure 22: Google Earth App- Code to fill Combo Box with Selectable Options

Code to Create KML File & Launch Google Earth

The following code (see Figure 23) writes the KML file and launches Google Earth to open the KML file and show the results to the user. This code is the heart of the Google Earth application and is read upon execution of code in the *Create KML* button's click event (see Create KML Button Code).

Similar to the code for filling the combo box with selectable options (see Code to fill Combo Box with Options), we connect by using a data adapter to pull the proper information from the database. This data adapter is referenced in the command builder and fills the datatable, *m_dtSelectedGeodata*, using the fill method. In the data adapter, our SQL is statement is set to select all values from the *Trails* table where the *Trail_Type* field matches the value selected by the user from the combo box. A key element to this SQL statement is the unique use of

quotation marks. Since a reference to a combo box value resides within an SQL statement, special quotes are necessary to differentiate from referencing just a normal string of text.

```
Private Sub CreateKML()

        m_daDataAdapter = New OleDb.OleDbDataAdapter("Select * From
Trails where Trail_Type = '" & cmbSelectType.Text & "'",
m_cnADONetConnection) 'Connects data adapter

        m_cbCommandBuilder = New
OleDb.OleDbCommandBuilder(m_daDataAdapter) 'Attaches the command
builder to the data adapter

        m_daDataAdapter.Fill(m_dtSelectedGeodata) 'populates the
DataTable with the Fill() Method
```

Figure 23: Google Earth App- Code to set data adapter to query

Next, we declare an integer variable, *i*, which is required for a *For/Next* Loop (see Figure 24). The integer variable keeps track of which iteration the loop is on until the stopping rule is met. Another variable of the StreamWriter type, *objFile*, is declared. As mentioned in Framework & Technology within the Literature Review, this variable enables us to write streams of text to a file. In this case, *objFile* writes a KML file to be named *test.kml* located within the directory *C:\Users\Steven\Desktop*, a directory on the local computer.

Writing the KML file begins with two essential lines referred to as the XML Header and the KML Namespace Declaration. Each line written by the objFile variable begins with the syntax *objFile. WriteLine*. The XML Header declares that version 1 of XML be used and specifies how it is encoded. Special quotes (shown as *Chr(34)*) had to be used within the XML Header for the version number and encoding method because these are quoted within the KML file and we were already using quotes to specify the text to write. The KML Namespace Declaration specifies we are using keyhole markup language (KML) as a subset of XML. We are using version 2.2 which is obtained from Google at the URL, *http://earth.google.com/kml/2.2* . Again we use the special quotes denoted by *Chr(34)* since we have quotes within quotes. The next line opens the document element which will contain all the code for the KML file. The following line creates a message box which was used to tell us how many records were filled into the datatable as specified from the SQL statement. This line helped to test the program for accuracy and helped to troubleshoot where problems occurred.

The *For/Next* Loop begins with the next line of code, which writes a blank line, followed by specifying a *<placemark>* element. Within the *<placemark>* element, a *<name>* element is specified. The *<name>, <description>,* and *<point>* elements are all nested within the *<placemark>* element. There is a tab (5 spaces) before the *<name>* element which helps to create the cascading styles of our KML file, making it easier to read. The trail's id is used as a unique identifier of the trail and is only used in testing as is contained within code for displaying a message box. It is not written in the KML file. The trail type component is written to the

<description> element of the placemark. All of these nested tags contain an opening and closing tag and refer to the database fields *Trail_Name*, *ID*, and *Trail_Type* respectively. There are several message boxes which are commented out. These were used to confirm the trail id, name, and type were specified correctly and corresponded all to the same data row. They are commented out to simplify the running of the working application.

```
        Dim i As Integer

        Dim objFile As New
System.IO.StreamWriter("C:\Users\Steven\Desktop\test.kml")

        'writes the XML header properly with double quotes

        objFile.WriteLine("<?xml version=" & Chr(34) & "1.0" & Chr(34)
& _

            " encoding=" & Chr(34) & "UTF-8" & Chr(34) & "?>")

        'writes the KML namespace declaration

        objFile.WriteLine("<kml xmlns=" & Chr(34) &
"http://earth.google.com/kml/2.2" & Chr(34) & ">")

        objFile.WriteLine("<Document>")

        MsgBox("Report me how many records: " &
m_dtSelectedGeodata.Rows.Count)

        For i = 1 To m_dtSelectedGeodata.Rows.Count

            'MsgBox("The trail ID is: " &
m_dtSelectedGeodata.Rows(m_RowPosition1)("ID"))

            'MsgBox("The trail name is: " &
m_dtSelectedGeodata.Rows(m_rowPosition1)("Trail_Name"))

            objFile.WriteLine("")

            objFile.WriteLine("<Placemark>")

            objFile.WriteLine("     <name>" &
m_dtSelectedGeodata.Rows(m_rowPosition1)("Trail_Name") & "</name>")

            'MsgBox("The trail type is: " &
m_dtSelectedGeodata.Rows(m_RowPosition1)("Trail_Type"))
```

Figure 24: Google Earth App- Code to write KML file (Part 1)

The code continues similarly (see Figure 25) with writing the remaining *<point>* element. The *<coordinates>* element is nested within the *<point>* element, which is still nested within the *<placemark>* element, nested within the *<document>* element. According to Google's KML syntax, the coordinates are written to match the *longitude, latitude* format. After the specification of the coordinates, the *<point>* and *<placemark>* elements are closed and the integer for the row position is increased to read the next row of the data table. The commented

out message boxes help to verify the correct longitude and latitude is pulled and written to the code.

```
            objFile.WriteLine("     <description>" &
m_dtSelectedGeodata.Rows(m_rowPosition1)("Trail_Type") &
"</description>")

            'MsgBox("The trail longitude is: " &
m_dtSelectedGeodata.Rows(m_RowPosition1)("Trailhead_Longitude"))

            'MsgBox("The trail latitude is: " &
m_dtSelectedGeodata.Rows(m_RowPosition1)("Trailhead_Latitude"))

            objFile.WriteLine("     <Point>")

            objFile.WriteLine("         <coordinates>" &
m_dtSelectedGeodata.Rows(m_rowPosition1)("Trailhead_Longitude") & _

                "," &
m_dtSelectedGeodata.Rows(m_rowPosition1)("Trailhead_Latitude") &
",0</coordinates>")

            objFile.WriteLine("     </Point>")

            objFile.WriteLine("</Placemark>")

            m_rowPosition1 = m_rowPosition1 + 1
```

Figure 25: Google Earth App- Code to write KML file (Part 2)

Next, the integer, *i,* is increased to run the next iteration of the loop. The loop is continued until the entire datatable is read and written to the KML file (see Figure 26). The KML code is completed by writing a blank line, followed by the close of the document element and the KML element. The last line of code, *objFile.Close*, tells the *objFile* variable of the StreamWriter type that its work is done and no more writing to KML file is required.

The next segment of code (also shown in Figure 26) is used to write a batch file which will be named *test6.bat* and located within the directory *C:\Users\Steven\Desktop*, a directory on the local computer. Similar to writing the KML file, a variable of the StreamWriter class called *objFile2* is declared. The batch file is written in four lines. The first line is a standard line used in creating batch files. The second line, *echo I am launching Google Earth*, is code to display the text *I am launching Google Earth* in the MS-DOS type window that appears when the batch file is executed. It helps confirm that the application is running properly. The third line specifies the key purpose of the batch file which is opening the newly created KML file located at *C:\Users\Steven\Desktop*. The fourth line ends the batch file and closes the MS-DOS type window. Next, the *objFile2* variable is closed to end the StreamWriter operation for the batch file. A message box is shown to the user notifying them that their KML file has been created. This message box confirms the write operations are complete. Finally, the private sub procedure, *LaunchBatchFile* (see Code to Launch Batch File), is called within the user form.

```
    Next i

        m_rowPosition1 = m_rowPosition1 - 1

        objFile.WriteLine("")

        objFile.WriteLine("</Document>")

        objFile.WriteLine("</kml>")

        objFile.Close()


        Dim objFile2 As New
System.IO.StreamWriter("C:\Users\Steven\Desktop\test6.bat")

        objFile2.WriteLine("@echo off")

        objFile2.WriteLine("echo I am launching Google Earth")

        objFile2.WriteLine("Start C:\Users\Steven\Desktop\test.kml")

        objFile2.WriteLine("End")

        objFile2.Close()


        MsgBox("Congratulations! Your KML file has been created!")


        Me.LaunchBatchFile()

    End Sub
```

Figure 26: Google Earth App- Code to finish KML file and write batch file

Code to Launch Batch File

```
Private Sub LaunchBatchFile()


System.Diagnostics.Process.Start("C:\Users\Steven\Desktop\test6.bat")

End Sub

End Class
```

Figure 27: Google Earth App- Code to launch batch file

The launch batch file subroutine runs the newly created batch file located at *C:\Users\Steven\Desktop\test6.bat* as show in Figure 27 above. Mentioned previously in Code to Create KML File & Launch Google Earth, this batch file specifies the command to open the KML file, using the default application of Google Earth. Thus, when this code is executed the

user will see Google Earth being launched automatically and zooming in to a view of all the placemarks corresponding to the queried database data. The entire code for the form application is then closed with the *End* Class statement.

Appendix B: ASP.NET Web Form Application Code- ASPX Page

The following is a step-by-step breakdown of the ASP and HTML code used to create the ASP.NET Web Application which connected an Access Database contained on a web server to a Google Map embedded in a webpage. This code occurs within the *GoogleMap107.aspx* page and specifies the elements that are rendered in HTML within a web browser.

ASP.NET Page Declarations

The first lines of the ASPX page are declarations in HTML as shown in Figure 28. The *@Page directive* indicates that Visual Basic is the programming language and that the source code is provided in the *GoogleMap107.aspx.vb* file. The application is published within the *TestPage1* folder. The next line is the *DOCTYPE html line* which is a standard auto-generated line in all aspx pages. Next, the *@Register directive* indicates the use of the assembly *GMaps*, the namespace *Subgurim.Controles*, and a defined tag prefix (ours was referred to as *x*). This directive is essential in ensuring the Google Maps web control works properly.

The next html line is the standard auto-generated namespace for XML contained within the head element. A blank title element exists because no page title was chosen.

```
<%@ Page Language="vb" AutoEventWireup="false"
CodeBehind="GoogleMap107.aspx.vb" Inherits="TestPage1.GoogleMap107" %>

<!DOCTYPE html PUBLIC "-//W3C//DTD XHTML 1.0 Transitional//EN"
"http://www.w3.org/TR/xhtml1/DTD/xhtml1-transitional.dtd">
<%@ Register Assembly="GMaps" Namespace="Subgurim.Controles"
TagPrefix="x" %>
<html xmlns="http://www.w3.org/1999/xhtml" >
<head runat="server">
    <title></title>
</head>
```

Figure 28: Google Maps App- ASPX Page Declarations

ASP.NET Web Controls

Within the <body> and elements of the ASPX page shown in Figure 29 is the ASP code that specifies the web controls. Before the element, an HTML line declares a form of id *form1* with a runat=”server” attribute. After the element opening, we see a line that contains instructions for the user followed by two page breaks (
 in HTML).

The first ASP web control element present is the drop-down list box, the equivalent of Visual Basic’s combo box. Since this element is an ASP web control element, the ending and closing tags differ from standard HTML in that an asp: portion is added. The drop-down list box is named DropDownList1, contains the *runat=”server”* attribute, and is linked to the *Trail_Type* field in the database connected in the *SqlDataSourceDDL* datasource. After two blank lines, we

have a button web control labeled as *btnSubmit*. It contains a *runat="server"* attribute and the text *Query & Display Markers* to be displayed to the user. Next, a label web control is placed named *lblTest* which also contains a *runat="server"* attribute. Finally, the last web control element is the Google Maps web control. The tag prefix *x* matches the tag prefix previously stated in the *@Register directive*. This reference is another key in successfully using the Google Maps web control and its associated dynamic link library. Like other ASP web controls, we define our Google Map with an id of *GMap1* and a *runat="server"* attribute.

```
<body>
    <form id="form1" runat="server">
    <div>
        Select a trail type below to view the results below:<br />
        <br />
        <asp:DropDownList ID="DropDownList1" runat="server"
            DataSourceID="SqlDataSourceDDL" DataTextField="Trail_Type"
            DataValueField="Trail_Type">
        </asp:DropDownList>
        <br />
        <br />
        <asp:Button ID="btnSubmit" runat="server" Text="Query &amp;
Display Markers" />
        <br />
        <asp:Label ID="lblTest" runat="server"></asp:Label>
    <x:GMap id="GMap1" runat="server"></x:GMap>
```

Figure 29: Google Maps App- ASP.NET Web Controls

ASP.NET SqlDataSource Controls

The last several lines within the element specify the data sources referenced in the web application (see Figure 30 below). Our project references two SqlDataSources, *SqlDataSource1* and *SqlDataSourceDDL*. They both contain the common *runat="server"* attributes of ASP.NET web controls, select commands that are SQL statements indicating which data to reference, and connection strings which correspond to the connection names specified in the *web.config* file (see Appendix D). *SqlDataSource1* is the datasource meant to serve as the basis for retrieving all data from the *Trails* table of the mountain biking database. From this data, the user specified query is then executed to filter the data accordingly. *SqlDataSourceDDL* is created to select the distinct trail types from the database, so they can be used as selectable options in the drop-down list box. Finally the , <form>, <body>, and <html> elements are closed, ending the code for the ASPX page.

```
        <asp:SqlDataSource ID="SqlDataSource1" runat="server"
            ConnectionString="<%$ ConnectionStrings:Connection5 %>"
            ProviderName="<%$
ConnectionStrings:Connection5.ProviderName %>"
            SelectCommand="SELECT * FROM [Trails]"></asp:SqlDataSource>
        <br />
        <asp:SqlDataSource ID="SqlDataSourceDDL" runat="server"
            ConnectionString="<%$ ConnectionStrings:Connection6 %>"
            ProviderName="<%$
ConnectionStrings:Connection6.ProviderName %>"
            SelectCommand="SELECT DISTINCT [Trail_Type] FROM [Trails]">
        </asp:SqlDataSource>
    </div>
    </form>
</body>
</html>
```

Figure 30: Google Maps App- ASP Database Connection Elements

Appendix C: ASP.NET Web Form Application Code- VB Source Code

The following is a step-by-step breakdown of the Visual Basic source code, *GoogleMap107.aspx.vb*, used to create the ASP.NET Web Application which connected an Access Database contained on a web server to a Google Map embedded in a webpage. This code is the backbone of the application and contains all the functionality of the web controls, effectively creating the working application.

Initialization

The Visual Basic source code begins as shown in Figure 31 by importing the library for the Google Maps web control, *Subgurim.Controles*. Without this library, Google Maps cannot be manipulated programmatically and the Intellisense feature within Visual Basic would fail to display any methods, events, or properties for objects of the Google Map type.

Our source code is part of the partial public class *GoogleMap107* which is consistent with our other files. It inherits the format of *System.Web.UI.Page* from Visual Studio's class libraries.

```
Imports Subgurim.Controles
Partial Public Class GoogleMap107
    Inherits System.Web.UI.Page
```

Figure 31: Google Maps App- Initial Source Code (VB)

Page Load Code

The following code in Figure 32 was executed upon the web form's load event (i.e. a user visits the website). It specifies initial properties for the Google Map web control which are applied when the webpage is loaded. The map is set to a width of 600 pixels and a height of 400 pixels. The user is also provided with the ability to zoom in or out on the map using the wheel on their mouse, move the map using their keyboard arrows, and pan across the map by dragging their mouse cursor. A pre-built control is placed upon the map which includes a 4 way pan and a zoom bar with a scale. The map type is set to Normal and the additional capability of having a

map of the physical types is added. The map's center is initially set to a latitude of 34.7° N, a longitude of 119.9° W (-119.9°), and a zoom level of 8.

```
    Protected Sub Page_Load(ByVal sender As Object, ByVal e As
System.EventArgs) Handles Me.Load
        GMap1.enableHookMouseWheelToZoom = True
        GMap1.enableDragging = True 'enables user to drag across the
map to move it
        GMap1.Height = 400 'sets height of map in pixels
        GMap1.Width = 600 'sets width of map in pixels
        GMap1.setCenter(New GLatLng(34.7, -119.9), 8) 'sets center of
GMap
        GMap1.mapType = GMapType.GTypes.Normal 'sets default map type
(Hybrid, Satellite, Physical, Normal)
        GMap1.addMapType(GMapType.GTypes.Physical) 'enables physical
terrain view (not normally in control)
        GMap1.addControl(New
GControl(GControl.preBuilt.MapTypeControl)) ' adds buttons to map to
switch maptype
        GMap1.enableGKeyboardHandler = True 'lets user use keyboard
keys to move map
        GMap1.Add(New GControl(GControl.preBuilt.LargeMapControl))
'activate pre-built large control
        'this control includes 4 way pan, zoom bar w/ scale
    End Sub
```

Figure 32: Google Maps App- Web Form Load Event Source Code (VB)

Query & Display Markers Button Code

Most of the functionality of the application is contained under the click event of the submit button labeled *Query & Display Markers*, which is shown in Figure 33 below. The variable *dv* is declared as a dataview which is set to use the data source and connection from *SqlDataSource1*. The following code ensures the contents of the label control are clear and any markers previously existing on the map control are erased.

Like the Google Earth form application, a *For/Next* loop is followed for referencing the recordset and writing the query results. In this case however, the results are written as markers on the Google Map instead of text in a KML file. The loop views each data row in the recordset or dataview until all values are read and uses an *If/Then/Else* code structure to specify the actions to perform based on the user's request and the database contents. If the trail type contained within a data row matches the user-selected value from the drop-down list box, the information from that row is used to place a marker at the appropriate coordinates on the map and is written to a string which will be displayed in the label web control to accompany the map results. If the trail type does not match, no action is performed and the following data row is checked. The code to write text to a label uses concatenation to properly display the text. A break element (
) is inserted between each valid result so that the label results are distinguishable from each other.

Finally, the *End Sub and End Class* lines of code specify the end of the source code portion of the program.

```
    Protected Sub btnSubmit_Click(ByVal sender As Object, ByVal e As
EventArgs) Handles btnSubmit.Click
        Dim dv As DataView =
CType(SqlDataSource1.Select(DataSourceSelectArguments.Empty), DataView)

        lblTest.Text = String.Empty
        GMap1.resetMarkers()

        For Each dr As DataRow In dv.Table.Rows
            If dr("Trail_Type") = DropDownList1.SelectedValue Then
                lblTest.Text &= dr("Trail_Name").ToString() & ", " &
dr("Trail_Locale") & ": " & dr("Trail_Type").ToString() & "<br/>"
                Dim point101 As GLatLng = New
GLatLng(dr("Trailhead_Latitude"), dr("Trailhead_Longitude"))
                GMap1.addGMarker(New GMarker(point101))
                'Dim window As New GInfoWindow(point101, dr("Trail_Name"))
                'GMap1.addInfoWindow(window)

            Else
            End If
        Next

    End Sub
End Class
```

Figure 33: Google Maps App- Query & Display Markers Button Click Event Source Code (VB)

Appendix D: ASP.NET Web Form Application Code- Web.Config File

The following (Figures 34-42) is the code from the web configuration file used in the project. Most of this code is auto-generated as the application is created within the ASPX and source code pages. One important function of the web configuration file is storing the connection strings for different SqlDataSources, seen in Figures 35-38.

```
<?xml version="1.0"?>
<!--
    Note: As an alternative to hand editing this file you can use the
    web admin tool to configure settings for your application. Use
    the Website->Asp.Net Configuration option in Visual Studio.
    A full list of settings and comments can be found in
    machine.config.comments usually located in
    \Windows\Microsoft.Net\Framework\v2.x\Config
-->
<configuration>

  <configSections>
      <sectionGroup name="system.web.extensions"
type="System.Web.Configuration.SystemWebExtensionsSectionGroup,
System.Web.Extensions, Version=3.5.0.0, Culture=neutral,
PublicKeyToken=31BF3856AD364E35">
        <sectionGroup name="scripting"
type="System.Web.Configuration.ScriptingSectionGroup, System.Web.Extensions,
Version=3.5.0.0, Culture=neutral, PublicKeyToken=31BF3856AD364E35">
          <section name="scriptResourceHandler"
type="System.Web.Configuration.ScriptingScriptResourceHandlerSection,
System.Web.Extensions, Version=3.5.0.0, Culture=neutral,
PublicKeyToken=31BF3856AD364E35" requirePermission="false"
allowDefinition="MachineToApplication"/>
          <sectionGroup name="webServices"
type="System.Web.Configuration.ScriptingWebServicesSectionGroup,
System.Web.Extensions, Version=3.5.0.0, Culture=neutral,
PublicKeyToken=31BF3856AD364E35">
            <section name="jsonSerialization"
type="System.Web.Configuration.ScriptingJsonSerializationSection,
System.Web.Extensions, Version=3.5.0.0, Culture=neutral,
PublicKeyToken=31BF3856AD364E35" requirePermission="false"
allowDefinition="Everywhere" />
            <section name="profileService"
type="System.Web.Configuration.ScriptingProfileServiceSection,
System.Web.Extensions, Version=3.5.0.0, Culture=neutral,
PublicKeyToken=31BF3856AD364E35" requirePermission="false"
allowDefinition="MachineToApplication" />
            <section name="authenticationService"
type="System.Web.Configuration.ScriptingAuthenticationServiceSection,
System.Web.Extensions, Version=3.5.0.0, Culture=neutral,
PublicKeyToken=31BF3856AD364E35" requirePermission="false"
allowDefinition="MachineToApplication" />
            <section name="roleService"
type="System.Web.Configuration.ScriptingRoleServiceSection,
System.Web.Extensions, Version=3.5.0.0, Culture=neutral,
PublicKeyToken=31BF3856AD364E35" requirePermission="false"
allowDefinition="MachineToApplication" />
          </sectionGroup>
        </sectionGroup>
      </sectionGroup>
    </configSections>
```

Figure 34: Google Maps App- Web Configuration Code Part 1

The key connection strings used for the *GoogleMap107.aspx* page were Connection20, Connection21, Connection22, and Connection23. Seen twice in the code, these connections are listed once for editing and once for publishing. When being used for editing, the connection

strings reference a database contained on the Z: drive of a lab computer. When being used for publishing, the connection strings reference a database on the C: of the IME5 web server. The database does not change locations but the accessible path is different based on where it is being used and what it is being used for.

Another important function is the <appSettings> element seen in Figure 35. This element contains the API key needed to enable viewing of the Google Map in any webpage within the web application. The key is sent to Subgurim who sends it to Google for authorization and verification.

```
 <appSettings>
    <add key="googlemaps.subgurim.net"
value="ABQIAAAA916xjBnD7JYr91CNMmpDShT_qw3u8Mqd-
sqwbqpvk2m_z1LCyBSMJl1EcSeNecgzKdJ8G2bEvvWplw" />
  </appSettings>

    <connectionStrings>
        <add name="Google Earth Project_v2ConnectionString"
connectionString="Provider=Microsoft.Jet.OLEDB.4.0;Data
Source=C:\InetPub\wwwroot\ime312labuseb\ime312_stu313\TestPage1\App_Data\Goog
le Earth Project_v2.mdb"
            providerName="System.Data.OleDb" />
        <add name="Google Earth Project_v2ConnectionString2"
connectionString="Provider=Microsoft.Jet.OLEDB.4.0;Data
Source=C:\InetPub\wwwroot\ime312labuseb\ime312_stu313\TestPage1\App_Data\Goog
le Earth Project_v2.mdb"
            providerName="System.Data.OleDb" />
        <add name="ConnectionStringv1"
connectionString="Provider=Microsoft.Jet.OLEDB.4.0;Data
Source=C:\InetPub\wwwroot\ime312labuseb\ime312_stu313\TestPage1\App_Data\Goog
le Earth Project_v2.mdb"
            providerName="System.Data.OleDb" />
        <add name="Google Earth Project_v2ConnectionString3"
connectionString="Provider=Microsoft.Jet.OLEDB.4.0;Data
Source=C:\InetPub\wwwroot\ime312labuseb\ime312_stu313\TestPage1\App_Data\Goog
le Earth Project_v2.mdb"
            providerName="System.Data.OleDb" />
        <add name="Connection1"
connectionString="Provider=Microsoft.Jet.OLEDB.4.0;Data
Source=C:\InetPub\wwwroot\ime312labuseb\ime312_stu313\TestPage1\App_Data\Goog
le Earth Project_v2.mdb"
            providerName="System.Data.OleDb" />
        <add name="Connection2"
connectionString="Provider=Microsoft.Jet.OLEDB.4.0;Data
Source=C:\InetPub\wwwroot\ime312labuseb\ime312_stu313\TestPage1\App_Data\Goog
le Earth Project_v2.mdb"
            providerName="System.Data.OleDb" />
        <add name="Connection3"
connectionString="Provider=Microsoft.Jet.OLEDB.4.0;Data
Source=C:\InetPub\wwwroot\ime312labuseb\ime312_stu313\TestPage1\App_Data\Goog
le Earth Project_v2.mdb"
```

Figure 35: Google Maps App- Web Configuration Code Part 2

```
            providerName="System.Data.OleDb" />
        <add name="Connection4"
connectionString="Provider=Microsoft.Jet.OLEDB.4.0;Data
Source=C:\InetPub\wwwroot\ime312labuseb\ime312_stu313\TestPage1\App_Data\Goog
le Earth Project_v2.mdb"
            providerName="System.Data.OleDb" />
        <add name="Connection5"
connectionString="Provider=Microsoft.Jet.OLEDB.4.0;Data
Source=C:\InetPub\wwwroot\ime312labuseb\ime312_stu313\TestPage1\App_Data\Goog
le Earth Project_v2.mdb"
            providerName="System.Data.OleDb" />
        <add name="Connection6"
connectionString="Provider=Microsoft.Jet.OLEDB.4.0;Data
Source=C:\InetPub\wwwroot\ime312labuseb\ime312_stu313\TestPage1\App_Data\Goog
le Earth Project_v2.mdb"
            providerName="System.Data.OleDb" />
        <add name="MyWorldConnectionString"
connectionString="Provider=Microsoft.Jet.OLEDB.4.0;Data
Source=C:\InetPub\wwwroot\ime312labuseb\ime312_stu313\TestPage1\App_Data\MyWo
rld.mdb"
            providerName="System.Data.OleDb" />
        <add name="MyWorld2"
connectionString="Provider=Microsoft.Jet.OLEDB.4.0;Data
Source=C:\InetPub\wwwroot\ime312labuseb\ime312_stu313\TestPage1\App_Data\MyWo
rld.mdb"
            providerName="System.Data.OleDb" />
        <add name="Connection7"
connectionString="Provider=Microsoft.Jet.OLEDB.4.0;Data
Source=C:\InetPub\wwwroot\ime312labuseb\ime312_stu313\TestPage1\App_Data\MyWo
rld.mdb"
            providerName="System.Data.OleDb" />
        <add name="Connection8"
connectionString="Provider=Microsoft.Jet.OLEDB.4.0;Data
Source=C:\InetPub\wwwroot\ime312labuseb\ime312_stu313\TestPage1\App_Data\MyWo
rld.mdb"
            providerName="System.Data.OleDb" />
        <add name="Connection9"
connectionString="Provider=Microsoft.Jet.OLEDB.4.0;Data
Source=C:\InetPub\wwwroot\ime312labuseb\ime312_stu313\TestPage1\App_Data\MyWo
rld.mdb"
```

Figure 36: Google Maps App- Web Configuration Code Part 3

```
providerName="System.Data.OleDb" />
        <add name="Connection11"
connectionString="Provider=Microsoft.Jet.OLEDB.4.0;Data
Source=C:\InetPub\wwwroot\ime312labuseb\ime312_stu313\TestPage1\App_Data\MyWo
rld.mdb"
            providerName="System.Data.OleDb" />
        <add name="Connection12"
connectionString="Provider=Microsoft.Jet.OLEDB.4.0;Data
Source=C:\InetPub\wwwroot\ime312labuseb\ime312_stu313\TestPage1\App_Data\MyWo
rld.mdb"
            providerName="System.Data.OleDb" />
        <add name="Connection13"
connectionString="Provider=Microsoft.Jet.OLEDB.4.0;Data
Source=C:\InetPub\wwwroot\ime312labuseb\ime312_stu313\TestPage1\App_Data\MyWo
rld.mdb"
            providerName="System.Data.OleDb" />
     <!--
      <add name="Connection20"
connectionString="Provider=Microsoft.Jet.OLEDB.4.0;Data
Source=Z:\ime312_stu313\TestPage1\App_Data\MyWorld.mdb"
            providerName="System.Data.OleDb" />
        <add name="Connection21"
connectionString="Provider=Microsoft.Jet.OLEDB.4.0;Data
Source=Z:\ime312_stu313\DOTNET Learning\MyWorld.mdb"
            providerName="System.Data.OleDb" />
        <add name="Connection22"
connectionString="Provider=Microsoft.Jet.OLEDB.4.0;Data
Source=Z:\ime312_stu313\TestPage1\App_Data\MyWorld.mdb"
            providerName="System.Data.OleDb" />
        <add name="Connection23"
connectionString="Provider=Microsoft.Jet.OLEDB.4.0;Data
Source=Z:\ime312_stu313\DOTNET Learning\MyWorld.mdb"
            providerName="System.Data.OleDb" />
      -->
```

Figure 37: Google Maps App- Web Configuration Code Part 4

```
      <add name="Connection20"
connectionString="Provider=Microsoft.Jet.OLEDB.4.0;Data
Source=C:\InetPub\wwwroot\ime312labuseb\ime312_stu313\TestPage1\App_Data\MyWo
rld.mdb"
           providerName="System.Data.OleDb" />
      <add name="Connection21"
connectionString="Provider=Microsoft.Jet.OLEDB.4.0;Data
Source=C:\InetPub\wwwroot\ime312labuseb\ime312_stu313\DOTNET
Learning\MyWorld.mdb"
          providerName="System.Data.OleDb" />
      <add name="Connection22"
connectionString="Provider=Microsoft.Jet.OLEDB.4.0;Data
Source=C:\InetPub\wwwroot\ime312labuseb\ime312_stu313\TestPage1\App_Data\MyWo
rld.mdb"
          providerName="System.Data.OleDb" />
      <add name="Connection23"
connectionString="Provider=Microsoft.Jet.OLEDB.4.0;Data
Source=C:\InetPub\wwwroot\ime312labuseb\ime312_stu313\DOTNET
Learning\MyWorld.mdb"
          providerName="System.Data.OleDb" />
    </connectionStrings>
    <system.web>
        <!--
            Set compilation debug="true" to insert debugging
            symbols into the compiled page. Because this
            affects performance, set this value to true only
            during development.

            Visual Basic options:
            Set strict="true" to disallow all data type conversions
            where data loss can occur.
            Set explicit="true" to force declaration of all variables.
        -->
        <compilation debug="false" strict="false" explicit="true">
```

Figure 38: Google Maps App- Web Configuration Code Part 5

```
        <assemblies>
            <add assembly="System.Core, Version=3.5.0.0, Culture=neutral,
PublicKeyToken=B77A5C561934E089"/>
            <add assembly="System.Web.Extensions, Version=3.5.0.0,
Culture=neutral, PublicKeyToken=31BF3856AD364E35"/>
            <add assembly="System.Data.DataSetExtensions, Version=3.5.0.0,
Culture=neutral, PublicKeyToken=B77A5C561934E089"/>
            <add assembly="System.Xml.Linq, Version=3.5.0.0, Culture=neutral,
PublicKeyToken=B77A5C561934E089"/>
        </assemblies>

        </compilation>
        <pages>
          <namespaces>
            <clear />
            <add namespace="System" />
            <add namespace="System.Collections" />
            <add namespace="System.Collections.Generic" />
            <add namespace="System.Collections.Specialized" />
            <add namespace="System.Configuration" />
            <add namespace="System.Text" />
            <add namespace="System.Text.RegularExpressions" />
            <add namespace="System.Linq" />
            <add namespace="System.Xml.Linq" />
            <add namespace="System.Web" />
            <add namespace="System.Web.Caching" />
            <add namespace="System.Web.SessionState" />
            <add namespace="System.Web.Security" />
            <add namespace="System.Web.Profile" />
            <add namespace="System.Web.UI" />
            <add namespace="System.Web.UI.WebControls" />
            <add namespace="System.Web.UI.WebControls.WebParts" />
            <add namespace="System.Web.UI.HtmlControls" />
          </namespaces>
```

Figure 39: Google Maps App- Web Configuration Code Part 6

```
          <controls>
            <add tagPrefix="asp" namespace="System.Web.UI"
assembly="System.Web.Extensions, Version=3.5.0.0, Culture=neutral,
PublicKeyToken=31BF3856AD364E35"/>
            <add tagPrefix="asp" namespace="System.Web.UI.WebControls"
assembly="System.Web.Extensions, Version=3.5.0.0, Culture=neutral,
PublicKeyToken=31BF3856AD364E35"/>
          </controls>

        </pages>
        <!--
            The <authentication> section enables configuration
            of the security authentication mode used by
            ASP.NET to identify an incoming user.
        -->
        <authentication mode="Windows" />
        <!--
            The <customErrors> section enables configuration
            of what to do if/when an unhandled error occurs
            during the execution of a request. Specifically,
            it enables developers to configure html error pages
            to be displayed in place of a error stack trace.

        <customErrors mode="RemoteOnly"
defaultRedirect="GenericErrorPage.htm">
            <error statusCode="403" redirect="NoAccess.htm" />
            <error statusCode="404" redirect="FileNotFound.htm" />
        </customErrors>
        -->

      <httpHandlers>
        <remove verb="*" path="*.asmx"/>
        <add verb="*" path="*.asmx" validate="false"
type="System.Web.Script.Services.ScriptHandlerFactory, System.Web.Extensions,
Version=3.5.0.0, Culture=neutral, PublicKeyToken=31BF3856AD364E35"/>
        <add verb="*" path="*_AppService.axd" validate="false"
type="System.Web.Script.Services.ScriptHandlerFactory, System.Web.Extensions,
Version=3.5.0.0, Culture=neutral, PublicKeyToken=31BF3856AD364E35"/>
        <add verb="GET,HEAD" path="ScriptResource.axd"
type="System.Web.Handlers.ScriptResourceHandler, System.Web.Extensions,
Version=3.5.0.0, Culture=neutral, PublicKeyToken=31BF3856AD364E35"
validate="false"/>
      </httpHandlers>
```

Figure 40: Google Maps App- Web Configuration Code Part 7

```
      <httpModules>
        <add name="ScriptModule" type="System.Web.Handlers.ScriptModule,
System.Web.Extensions, Version=3.5.0.0, Culture=neutral,
PublicKeyToken=31BF3856AD364E35"/>
      </httpModules>

    </system.web>

    <system.codedom>
      <compilers>
        <compiler language="c#;cs;csharp" extension=".cs" warningLevel="4"
                  type="Microsoft.CSharp.CSharpCodeProvider, System,
Version=2.0.0.0, Culture=neutral, PublicKeyToken=b77a5c561934e089">
          <providerOption name="CompilerVersion" value="v3.5"/>
          <providerOption name="WarnAsError" value="false"/>
        </compiler>
        <compiler language="vb;vbs;visualbasic;vbscript" extension=".vb"
warningLevel="4"
                  type="Microsoft.VisualBasic.VBCodeProvider, System,
Version=2.0.0.0, Culture=neutral, PublicKeyToken=b77a5c561934e089">
          <providerOption name="CompilerVersion" value="v3.5"/>
          <providerOption name="OptionInfer" value="true"/>
          <providerOption name="WarnAsError" value="false"/>
        </compiler>
      </compilers>
    </system.codedom>

    <!--
        The system.webServer section is required for running ASP.NET AJAX
under Internet
        Information Services 7.0.  It is not necessary for previous version
of IIS.
    -->
    <system.webServer>
      <validation validateIntegratedModeConfiguration="false"/>
      <modules>
        <remove name="ScriptModule" />
        <add name="ScriptModule" preCondition="managedHandler"
type="System.Web.Handlers.ScriptModule, System.Web.Extensions,
Version=3.5.0.0, Culture=neutral, PublicKeyToken=31BF3856AD364E35"/>
      </modules>
```

Figure 41: Google Maps App- Web Configuration Code Part 8

```
      <handlers>
         <remove name="WebServiceHandlerFactory-Integrated"/>
         <remove name="ScriptHandlerFactory" />
         <remove name="ScriptHandlerFactoryAppServices" />
         <remove name="ScriptResource" />
         <add name="ScriptHandlerFactory" verb="*" path="*.asmx"
preCondition="integratedMode"
              type="System.Web.Script.Services.ScriptHandlerFactory,
System.Web.Extensions, Version=3.5.0.0, Culture=neutral,
PublicKeyToken=31BF3856AD364E35"/>
         <add name="ScriptHandlerFactoryAppServices" verb="*"
path="*_AppService.axd" preCondition="integratedMode"
              type="System.Web.Script.Services.ScriptHandlerFactory,
System.Web.Extensions, Version=3.5.0.0, Culture=neutral,
PublicKeyToken=31BF3856AD364E35"/>
         <add name="ScriptResource" preCondition="integratedMode"
verb="GET,HEAD" path="ScriptResource.axd"
type="System.Web.Handlers.ScriptResourceHandler, System.Web.Extensions,
Version=3.5.0.0, Culture=neutral, PublicKeyToken=31BF3856AD364E35" />
      </handlers>
    </system.webServer>


    <runtime>
      <assemblyBinding xmlns="urn:schemas-microsoft-com:asm.v1">
        <dependentAssembly>
          <assemblyIdentity name="System.Web.Extensions"
publicKeyToken="31bf3856ad364e35"/>
          <bindingRedirect oldVersion="1.0.0.0-1.1.0.0"
newVersion="3.5.0.0"/>
        </dependentAssembly>
        <dependentAssembly>
          <assemblyIdentity name="System.Web.Extensions.Design"
publicKeyToken="31bf3856ad364e35"/>
          <bindingRedirect oldVersion="1.0.0.0-1.1.0.0"
newVersion="3.5.0.0"/>
        </dependentAssembly>
      </assemblyBinding>
    </runtime>

</configuration>
```

Figure 42: Google Maps App- Web Configuration Code Part 9

Appendix E: ASP.NET Web Form Application Code- VB Designer File

Figures 43 & 44 contain the code from the Visual Basic designer file, *GoogleMap107.aspx.designer.vb*. A key element is ensuring this file's partial public class (GoogleMap107) matches that of the other files in the web application. This code is an important component of the project and is generated automatically as the ASPX and VB pages are created—there was no additional code added to this file.

```
'------------------------------------------------------------------------------
--
' <auto-generated>
'     This code was generated by a tool.
'     Runtime Version:2.0.50727.3603
'
'     Changes to this file may cause incorrect behavior and will be lost if
'     the code is regenerated.
' </auto-generated>
'------------------------------------------------------------------------------
--

Option Strict On
Option Explicit On

Partial Public Class GoogleMap107

    '''<summary>
    '''form1 control.
    '''</summary>
    '''<remarks>
    '''Auto-generated field.
    '''To modify move field declaration from designer file to code-behind
file.
    '''</remarks>
    Protected WithEvents form1 As Global.System.Web.UI.HtmlControls.HtmlForm

    '''<summary>
    '''DropDownList1 control.
    '''</summary>
    '''<remarks>
    '''Auto-generated field.
    '''To modify move field declaration from designer file to code-behind
file.
    '''</remarks>
    Protected WithEvents DropDownList1 As
Global.System.Web.UI.WebControls.DropDownList

    '''<summary>
    '''btnSubmit control.
    '''</summary>
    '''<remarks>
    '''Auto-generated field.
    '''To modify move field declaration from designer file to code-behind
file.
    '''</remarks>
    Protected WithEvents btnSubmit As Global.System.Web.UI.WebControls.Button
```

Figure 43: Google Maps App- VB Designer Code Part 1

```
    '''<summary>
    '''lblTest control.
    '''</summary>
    '''<remarks>
    '''Auto-generated field.
    '''To modify move field declaration from designer file to code-behind
file.
    '''</remarks>
    Protected WithEvents lblTest As Global.System.Web.UI.WebControls.Label

    '''<summary>
    '''GMap1 control.
    '''</summary>
    '''<remarks>
    '''Auto-generated field.
    '''To modify move field declaration from designer file to code-behind
file.
    '''</remarks>
    Protected WithEvents GMap1 As Global.Subgurim.Controles.GMap

    '''<summary>
    '''SqlDataSource1 control.
    '''</summary>
    '''<remarks>
    '''Auto-generated field.
    '''To modify move field declaration from designer file to code-behind
file.
    '''</remarks>
    Protected WithEvents SqlDataSource1 As
Global.System.Web.UI.WebControls.SqlDataSource

    '''<summary>
    '''SqlDataSourceDDL control.
    '''</summary>
    '''<remarks>
    '''Auto-generated field.
    '''To modify move field declaration from designer file to code-behind
file.
    '''</remarks>
    Protected WithEvents SqlDataSourceDDL As
Global.System.Web.UI.WebControls.SqlDataSource
End Class
```

Figure 44: Google Maps App- VB Designer Code Part 2

Works Referenced

1. ASP.NET, "Get Started with ASP.NET." *Microsoft ASP.NET*. 2009. 24 May 2009 <http://www.asp.net/get-started/>.
2. Cardoza, Patricia, Teresa Hennig, Graham Seach, and Armen Stein. *Access 2003 VBA: Programmer's Reference*. Indianapolis: Wiley Publishing Inc., 2004. Print.
3. Case, Paul. Personal Interviews. 04 11 2009.
4. Chapra, Steven. *Power Programming With VBA/Excel*. Upper Saddle River, NJ: Pearson Education Inc., 2003. Print.
5. Claburn, Thomas. "Google Earth Catches On In The Business World." *Information Week* 20 03 2006: 1. Print.
6. ComputerHope.com, "Information on Batch Files." *Computer Hope*. 2009. 24 May 2009 <http://www.computerhope.com/batch.htm>.
7. Foxall, James. *Sams Teach Yourself Visual Basic 2008 in 24 Hours*. Indianapolis: Pearson Education Inc., 2008. Print.
8. Freeman, Adam, and Allen Jones. *Microsoft .NET XML Web Services Step by Step*. Redmond, WA: Microsoft Press, 2003. Print.
9. Ghata, Shabdar. "Google Maps Control for ASP.NET."Shabdar.org. 23 11 2008. Shabdar.org, Web. 19 Sept 2009. <http://www.shabdar.org/google-maps-user-control-for-ASP-Net-part1.html>.
10. Ghata, Shabdar. "Google Maps Control for ASP.NET - Part 1." The Code Project: Your Development Resource. 18 03 2008. Code Project, Web. 19 Sept 2009. <http://www.codeproject.com/KB/custom-controls/Google-Maps-User-Control.aspx?msg=3243412#xx3243412xx>.
11. Ghata, Shabdar. "Google Maps Control for ASP.NET - Part 2." The Code Project: Your Development Resource. 04 06 2008. Code Project, Web. 22 Sept 2009. <http://www.codeproject.com/KB/custom-controls/Google-Map-Control-Part-2.aspx>.
12. Google Inc., "Company Overview." Corporate Information 2009. 23 May 2009 <http://www.google.com/intl/en/corporate/>.
13. Google Inc., "Explore, Search, and Discover." *Google Earth*. 2009. 24 May 2009 <http://earth.google.com/>.
14. "GoogleMaps.Subgurim.Net." ASP.NET Google Maps User Control. 02 10 2009. Subgurim.NET, Web. 24 Nov 2009. <http://en.googlemaps.subgurim.net/>.
15. "Google Maps." Wikipedia. 24 11 2009. Wikimedia Foundation, Inc., Web. 24 Nov 2009. <http://en.wikipedia.org/wiki/Google_Maps>.
16. The Linux Information Project, "GUI Definition." LINFO. 01 10 2004. The Linux Information Project. 23 May 2009 <http://www.linfo.org/gui.html>.
17. Mitchell, Scott. "Accessing and Updating Data in ASP.NET 2.0: Accessing Database Data."4GuysFromRolla.com. 26 02 2006. Web Media Brands Inc., Web. 08 Nov 2009. <http://aspnet.4guysfromrolla.com/articles/022206-1.aspx>.
18. Mitchell, Scott. "Creating a Databound Label Control."4GuysFromRolla.com. 13 08 2008. Web Media Brands Inc., Web. 04 Oct 2009. <http://aspnet.4guysfromrolla.com/articles/032702-1.aspx>.
19. Mitchell, Scott. "Efficiently Iterating Through Results from a Database Query using ADO.NET."4GuysFromRolla.com. 27 03 2002. Web Media Brands Inc., Web. 08 Nov 2009. <http://aspnet.4guysfromrolla.com/articles/032702-1.aspx>.
20. Mitchell, Scott. Sams Teach Yourself ASP.NET 3.5 in 24 Hours. Indianapolis: Pearson Education Inc., 2008. Print.
21. MSDN, ".NET Framework Conceptual Overview." *Visual Studio Development Center*. 2009. Microsoft Developer's Network. 24 May 2009 <http://msdn.microsoft.com/en-us/library/zw4w595w.aspx>.

22. MSDN, "SqlDataSource Web Server Control Overview." *Visual Studio Development Center*. 2009. Microsoft Developer's Network. 24 Sept 2009 <http://msdn.microsoft.com/en-us/library/dz12d98w(VS.80).aspx>.
23. Murkoth, Jeevan. "Google Maps and ASP.NET: Building a custom server control." .NET Developer's Journal. 15 10 2008. SYS-CON Media, Inc., Web. 04 Oct 2009. <http://dotnet.sys-con.com/node/171162>.
24. Newton, Greg. California Department of Forestry, San Luis Obispo County Fire Station 12. (2005). Response guide. San Luis Obispo, CA
25. Pierce, Bill. "Lat Lays Flat - Part 1: A Google Maps .NET Control." The Code Project: Your Development Resource. 03 09 2005. Code Project, Web. 22 Sept 2009. <http://www.codeproject.com/KB/custom-controls/LatLaysFlat-Part1.aspx>.
26. "PG&E Electrical System Outage Map." PG&E: Customer Service. Web. 11 Oct 2009. <http://www.pge.com/myhome/customerservice/energystatus/outagemap/>.
27. "Retrieve Data from SqlDataSource." 23 01 2008. Online Posting to Dani Web- Web Development: ASP.NET. Web. 25 Nov 2009.
28. Silberschatz, Korth, and Sudarshan. Database System Concepts. '5th ed'. Mc-Graw-Hill Primis, 2005. Print.
29. Sur, Abhishek. "Google Maps in HTML, ASP.NET, PHP, JSP etc. with ease." The Code Project: Your Development Resource. 21 09 2007. Code Project, Web. 22 Sept 2009. <http://www.codeproject.com/KB/scripting/Use_of_Google_Map.aspx>.
30. Taylor, Frank. "NIN Downloads." *Google Earth Blog* 30 June 2008 1. Web.23 May 2009. <http://www.gearthblog.com/blog/archives/2008/06/links_outreach_birthday_bullit_chas.html> .
31. Taylor, Frank. "Pool Guy Taps Google Earth." *Google Earth Blog* 10 July 2007 1. Web.23 May 2009. <http://www.gearthblog.com/blog/archives/2007/07/pool_guy_taps_google.html>.
32. Vaughn, William, and Peter Blackburn. *Hitchhiker's Guide to Visual Studio and SQL Server: Best Practice Architectures and Examples*. '7th ed'. Boston: Pearson Education Inc., 2007. Print.
33. VB Tutor, "Lesson 21: Reading and Writing Text Files." *Visual Basic 2008 Tutorial*. 2008. 24 May 2009 <http://www.vbtutor.net/vb2008/vb2008_lesson21.html>.
34. Wernecke, Josie. *The KML Handbook: Geographic Visualization for the Web*. Kendallville IN: Pearson Education Inc., 2009. Print.
35. "What is the Google Maps API?" Google Maps API. 2009. Google Inc., Web. 24 Nov 2009. <http://code.google.com/apis/maps/>.
36. Yang, Tao. “IME 312 Lecture” IME 312: Data Management and System Design. California Polytechnic State University. San Luis Obispo, Spring 2008. Print.