



2014-06-11

Investigating Virtual Globes for a Prototype Community Archive of 3D Subsurface Data

Derek C. Whitman

Brigham Young University - Provo

Follow this and additional works at: <https://scholarsarchive.byu.edu/etd>

 Part of the [Civil and Environmental Engineering Commons](#)

BYU ScholarsArchive Citation

Whitman, Derek C., "Investigating Virtual Globes for a Prototype Community Archive of 3D Subsurface Data" (2014). *All Theses and Dissertations*. 4105.

<https://scholarsarchive.byu.edu/etd/4105>

This Thesis is brought to you for free and open access by BYU ScholarsArchive. It has been accepted for inclusion in All Theses and Dissertations by an authorized administrator of BYU ScholarsArchive. For more information, please contact scholarsarchive@byu.edu, ellen_amatangelo@byu.edu.

Investigating Virtual Globes for a Prototype Community Archive of
3D Subsurface Data

Derek C. Whitman

A thesis submitted to the faculty of
Brigham Young University
in partial fulfillment of the requirements for the degree of
Master of Science

Norman L. Jones, Chair
Daniel P. Ames
E. James Nelson

Department of Civil & Environmental Engineering

Brigham Young University

June 2014

Copyright © 2014 Derek C. Whitman

All Rights Reserved

ABSTRACT

Investigating Virtual Globes for a Prototype Community Archive of 3D Subsurface Data

Derek C. Whitman

Department of Civil & Environmental Engineering, BYU
Master of Science

Geoscience data sharing and processing is very advanced in terms of surface data. Subsurface data sharing has not received the attention that surface data sharing has received and so there are fewer applications or software packages which focus on it. This research is funded by the NSF EarthCube GEO Domain program in an effort to develop a continental-scale repository of 3D subsurface data to facilitate the sharing of complex 3D data and to enable the development of geoprocessing tools and workflows that operate on that data. The work in this thesis is a small part of the EarthCube project with two parts. The first part is to research current tools for 3D subsurface data visualization, specifically virtual globes, and to recommend one for use in the development of the EarthCube project. The second part is to develop an online prototype visualization platform for the EarthCube project referred to as the “Digital Crust” using the recommended virtual globe. Additional work was done with the Digital Crust to develop geoprocessing tools to show the ability for the Digital Crust to work with a data repository. These tools convert geoscience data file types, and interpolate soil cross-sections from borehole log data.

Keywords: AHGW, virtual globe, geoprocessing, cloud, Digital Crust, EarthCube, CUAHSL, GIS, ArcGIS

ACKNOWLEDGEMENTS

I would like to thank all those who helped me the most with this project and the organizations CUAHSI and NSF for funding my research and having grand dreams.

Thank you Dr. Norman L. Jones for hiring me for this project and for having faith in my ability and for providing me with constructive feedback. Thank you also for introducing me to the wonderful world of programming through your “Computational Methods” class and later allowing me to work as a TA for that class.

Thank you Dr. E. James Nelson for pushing the Civil Engineering Department to introduce GIS software to civil engineering students early on. I may never have known what the world of GIS was if not for you.

Thank you Dr. Daniel P. Ames who introduced to me this research opportunity and to new ways to combine my interests for GIS and programming. I would never have put the two together if not for you.

Thank you to Kelvin Smith for working with me on this project and exchanging ideas with me.

Thank you to the other students who were involved in GIS research and happily gave me input when I was stuck, especially Nathan Swain, Scott Christensen, Ken Clark, and Carlos Osorio.

Most of all, thank you to my wife for her unending patience with me finishing school while she was essentially living the life of an engineering student’s widow. And thank you to my two sons, Jackson and James, who’s tears when I left home and shouts of joy and hugs when I returned reminded me of what is most important in life and where I need to be.

TABLE OF CONTENTS

LIST OF TABLES	viii
LIST OF FIGURES	ix
1 INTRODUCTION	1
1.1 Review of Current Subsurface Data Applications and Repositories	1
1.1.1 Canadian Groundwater Information Network	2
1.1.2 British Government 3D Geology Project.....	3
1.1.3 Unidata.....	3
1.1.4 GEON	4
1.2 Project Objective.....	5
2 VIRTUAL GLOBE ENVIRONMENTS	7
2.1 Google Earth	8
2.2 Cesium Globe	9
2.3 World Wind	11
2.4 Selection of Globe for Prototype	12
3 DEVELOPING A DIGITAL CRUST PROTOTYPE	13
3.1 Developing a World Wind Virtual Globe.....	13
3.1.1 Obtaining the Source Code	14
3.1.2 Selecting an IDE	15
3.1.3 Java Code Organization	16

3.1.4	Application or Applet	16
3.1.5	Launching Through Java Web Start	20
3.2	Geoprocessing Using ArcGIS and AHGW	22
3.2.1	Setting Up the ArcGIS Environment to Use AHGW Tools in Python Scripts.....	23
3.2.2	Initial Problems with Arc 10.0.....	24
3.2.3	Developments in Arc 10.1	24
3.2.4	Using Arc 10.2	25
3.2.5	ArcGIS/AHGW Summary of Conflicts	27
3.3	Website Development.....	28
3.3.1	World Wind Applications	29
3.3.2	World Wind Applet.....	35
4	CONCLUSIONS.....	43
4.1	Selection of a Virtual Globe	43
4.2	Development of World Wind Applets and Applications.....	44
4.3	Using ArcGIS and AHGW for Web Processing.....	44
4.4	Prototype Website Developed	45
REFERENCES	46
APPENDIX A	SETTING UP SOURCE CODE LIBRARIES IN ECLIPSE.....	48
APPENDIX B	CREATING APPLICATIONS AND APPLETS	50
B.1	Application Code Examples	50

B.1.a	KMLViewer	50
B.1.b	ReadingFiles	57
B.1.c	LineBuilder	60
B.1.d	CoordsUrl.....	69
B.1.e	WMSLayerManager	70
B.2	Applet Code Example.....	72
APPENDIX C	Putting Applications and Applets Online	82
C.1	Creating a Keystore in the Command Line	82
C.2	Signing a JAR file in the Command Line.....	83
C.3	Example of JNLP File for Application	84
C.4	Launching Application Through Web Start in HTML	85
C.5	Example of JNLP File for Applet.....	85
C.6	Launching Applet Through Web Start in HTML	86
APPENDIX D	Create a Custom Toolbox with AHGW Tools in ArcGIS	87
APPENDIX E	Cloud-Based Processing	88
E.1	Converting Shapefile	88
E.1.a	HTML/JavaScript – index.html (homepage of website).....	88
E.1.b	JavaScript - MainJavascript.js (JavaScript page referenced in the web pages)	90
E.1.c	PHP - checkText.php (run by checkText() function in JavaScript).....	90
E.1.d	PHP - upload_shp.php (first visible PHP page with loading gif)	90

E.1.e	PHP - SaveToLyrToKMZ.php (second visible PHP page)	92
E.1.f	Python - SaveToLyrToKMZ.py (conversion process for applet)	93
E.2	Creating a Geosection	94
E.2.a	PHP - lonlatvar.php (sends processing request from application to server)	95
E.2.b	Python - PyToCsv.py (process for creating geosections using AHGW)	95

LIST OF TABLES

Table 1: Summary of how virtual globe criteria were met with the X mark indicating a criteria which was not met and the check mark indicating a criteria which was met.....	12
Table 2: Summary of problems with using different versions of ArcGIS with a PHP Web Service for AHGW	28

LIST OF FIGURES

Figure 1: Error message given when trying to run RastersToGeosections tool from a web request	26
Figure 2: Application menu option which will load a file showing the extent of borehole log data on the server which may be used to generate soil cross sections.....	30
Figure 3: A polygon loaded from selecting the menu option in Fig2. This polygon represents the extent of available borehole log data on the server	31
Figure 4: Buttons used for polyline drawing controls in application.....	32
Figure 5: General processing workflow for application. The blue boxes represent elements of the workflow and the red ovals represent events within the workflow between elements. The orange oval represents the last workflow to be accomplished.....	34
Figure 6: Example of collapsed controls menu. Top - Menu is fully compressed. Bottom - Menu is partially expanded.....	36
Figure 7: Image of the expanded map controls for the applet	37
Figure 8: Image of the HTML controls to upload shapefiles into the applet globe.....	38
Figure 9: General processing workflow for applet. The blue boxes represent elements of the workflow and the red ovals represent events within the workflow between elements. The orange oval represents the last workflow to be accomplished	42
Figure 10: Example of command line inputs and outputs after creating and self-certifying a keystore	83
Figure 11: Example of command line window inputs and outupts after signing a JAR file	84

1 INTRODUCTION

Geospatial data is a crucial element for understanding our Earth and making important decisions dealing with the environment and the development of society. Three things need to be done with the data to make these decisions. These will be referred to as “data tasks” for the purposes of this thesis.

1. The data need to be shared or made discoverable.
2. The data need to be processed to predict outcomes.
3. The data need to be visualized so that it can be interpreted.

These data are often shared through database repositories that are made accessible through online requests to websites created by the data holders. Software packages exist for processing the data into results used to make decisions. 2D virtual maps and 3D virtual globes make data visualization more dynamic and easier to interpret. Unfortunately, most software applications focus on the sharing, processing, and viewing of surface data, not subsurface data.

1.1 Review of Current Subsurface Data Applications and Repositories

Subsurface data sharing has not received the same attention that surface data sharing has received. There are few applications which have developed tools for accomplishing the three data tasks for subsurface data and none of them accomplish all three. These applications are often separate from online data repositories for subsurface data. This separation of the processing tool from the repository requires the data to be downloaded without being viewed first then

moved to the processing application for viewing and processing. Many of these subsurface data applications which allow subsurface data visualization show the data disconnected from basemaps. This makes it difficult to understand the geographic location and extent of the data. Some of these applications include a basemap but will only show the data above the surface where it exists. Only some of the applications that can produce a 3D visualization of subsurface data also maintain the ability to perform geoprocessing calculations on the data.

1.1.1 Canadian Groundwater Information Network

The Groundwater Information Network (GIN) is a project developed by GeoConnections which has done extensive work in collecting and maintaining groundwater data for much of Canada (Brodaric, Sharpe et al. 2009). GIN maintains a database which makes borehole log data within Canada publicly available and accessible through their website. By using the “Analysis of Water Wells of Canada” link on the website, the user is directed to a map view of the well log locations within the database. This website uses the 2D Microsoft Bing maps developed in Silverlight. From the map the user can zoom in to well locations and get information about any single well or a set of wells. Some tools are provided which allow the user to draw a line across the map and view a 3D rendering of the soil profile under the line. Interpolating from the neighboring wells generates the 3D rendering which appears in another window and not in the map itself. To the left of the map is an information window where data to download can be searched. From here the well log data can be downloaded in several different Open Geospatial Consortium (OGC) (Consortium 2010) formats which are translated from the server on the fly.

The GIN is an extremely valuable data repository that makes great use of subsurface groundwater data but still lacks the ability to show its 3D rendering of soil profiles in place with reference to the surface data or a basemap.

1.1.2 British Government 3D Geology Project

The “Geological Surveying and Investigation in 3 Dimensions” (GSI3D) software is supported by the British Geological Survey (BGS) (Mathers and Kessler 2010). Although GSI3D was developed in Britain and focused on the geology of the British Isles it can now be downloaded and used for any location. The software is not free but it is offered at “competitive prices and on a not-for-profit basis” according to the site. GSI3D has the capability to use a terrain model and borehole log data in a 3D environment to build cross sections and geovolumes within the geographic extent of the data provided.

This software does not show the data in a virtual globe with a basemap reference. To view the data in reference to a basemap it must be exported to a file format for showing in Google Earth. Geoprocessing calculations cannot be performed when the data is viewed in Google Earth and the data is shown above the virtual globe surface rather than in place below the surface.

1.1.3 Unidata

An integrated data viewer (IDV) was developed by Unidata with funding by the NSF (Userguide 2009). Unidata is part of the University Corporation for Atmospheric Research (UCAR). The IDV is a virtual globe Java application built on an OpenGL visual platform which allows the user to display and work with satellite imagery, gridded data, and surface, upper air, and radar data within a single interface. The virtual globe can be accessed through the Unidata home page and comes with some preloaded data sets to display. The Unidata IDV was developed to show surface and atmospheric data in a 3D virtual globe environment but not subsurface data. This IDV was used as a base for the development of the GEON IDV by UNAVCO.

1.1.4 GEON

GEON is a project which began in 2002 through the San Diego Super Computing Center (SDSC) and was funded under the NSF Information Technology Research (ITR) program (Workflows and Portal). The objective of the GEON project includes hosting vast amounts of surface and subsurface data in a publicly and easily accessible manner that allows the data to be visualized and analyzed. At the GEON website under the “myGEON” tab there is a Google map where a user can select an area of interest and access data within the geographic extent of that area. This map is 2D and is not for visualizing and analyzing the underlying data. The visualization of the data happens after it has been downloaded from the website and loaded into the virtual globe application.

GEON has funded other projects to further the work of visualization and analysis such as the GEON Integrated Data Viewer (IDV) and the Open Earth Framework (OEF).

1.1.4.1 GEON IDV

GEON has funded the development of an integrated data viewer through the UNAVCO using the Unidata IDV (Meertens, Wier et al. 2006). The GEON IDV is a plugin that takes the Unidata IDV a step further in terms of visualization to show subsurface data as well as surface and atmospheric data. The GEON IDV is designed for the exploration and analysis of any data from the Earth’s inner core through the atmosphere. This plugin can be downloaded online and added to the Unidata IDV using instructions provided on the UNAVCO website.

While this software provides the ability to visualize and process subsurface data in a virtual globe, it requires data to be downloaded from a data repository elsewhere rather than providing the ability to access the data from the globe directly.

1.1.4.2 Open Earth Framework

The Open Earth Framework (OEF) is another virtual globe designed to incorporate the viewing of surface and subsurface data with a temporal component so that it can show 2-, 3-, and 4-dimensional earth science data (Baru, Keller et al. 2008). It is based on the World Wind virtual globe developed by NASA (Bell, Kuehnel et al. 2007). It is written in Java and built on an OpenGL visual platform similar to the Unidata IDV. The OEF contains software libraries for processing 2D and 3D data and provides access to common Earth science file formats.

An available version of this software could not be found because many of the web links to information about the OEF have been broken or ill maintained. Efforts to contact the administrators of the project at the San Diego Super Computing Center were unsuccessful.

1.2 Project Objective

Given the variety of tools that accomplish one or two of the required subsurface data tasks (e.g. the tools listed in the previous section) there must certainly be a demand and need for a single application which can accomplish all three data tasks. As part of a recent NSF EarthCube GEO Domain program, CUAHSI and the USGS-Powell-Synthesis Center hosted a workshop in Boulder, Colorado from January 29, 2013 to January 30, 2013 to explore this need. Academics, scientists, and practitioners were invited to attend the workshop and discuss ideas and suggestions for developing a continental-scale repository of 3D subsurface data. The objective of the repository is to facilitate sharing of complex 3D surface and subsurface data and to enable the development of geoprocessing tools and workflows that operate on the hosted data.

The objective of this research project is to accomplish a small part of the EarthCube program by doing two things.

1. Researching current virtual globe technology in terms of subsurface data visualization for the EarthCube project
2. Developing an online prototype called the “Digital Crust” which allows subsurface data to be visualized and made discoverable.

In addition to this objective some geoprocessing routines were built for the Digital Crust prototype to demonstrate an ability to accomplish all three data tasks in one place with a basemap. The actual building of the community data model portion is left to future research.

This two-part thesis presents the findings from research of virtual globes available as online applications (Chapter 2) as well as the design and development of a prototype Digital Crust web based application which allows users to find surface and subsurface data online, visualize subsurface data in a virtual globe, and perform geoprocessing calculations on the data (Chapter 3).

2 VIRTUAL GLOBE ENVIRONMENTS

For the first part of this research project, I investigated current virtual globes because different virtual globes offer different features for visualizing and processing data. The suitability of any virtual globe to be used in this project was determined based on its ability to fulfill the following three main criteria:

1. The virtual globe must allow the visualization of subsurface geospatial data as it exists below the Earth's surface.
2. The virtual globe must be hosted in a publicly accessible website with no requirement for special licensing or payments.
3. The virtual globe must be able to support file types with the Keyhole Markup Language such as KML and KMZ.

The first criterion assures that the prototype will be able to maintain its focus on subsurface data visualization to keep in purpose with the overall EarthCube program. The second criterion is in place to allow subsurface data to be made discoverable on line in place with a basemap on the virtual globe. The third criterion was established to make sure that the globe would be able to use a standard data type for geospatial data. KML and KMZ were chosen because it is widely used in the geospatial community and supported by the Open Geospatial Consortium (OGC) (Consortium 2010).

Three virtual globes were analyzed as candidates for use in the Digital Crust project: Google Earth, Cesium Globe, and World Wind. The advantages or disadvantages to using each of these virtual globes are discussed below as well as the reasons for deciding to use one over the others.

2.1 Google Earth

In addition to its widespread use and popularity, the Google Earth virtual globe has many advantages that made it a strong candidate for this project. It includes an API which is free for use and easily implemented into any website through JavaScript. This API is accompanied by large amounts of development support available through forums and also through the Google Code Playground (Google) which includes development support for many other APIs available from Google.

A download of the Google Earth plugin or some other offline Google Earth application is required for any machine to view the API hosted in a website. Google Earth offers a free standard version of this application for PCs and Macs which makes it able to meet the second criterion mentioned above, i.e., the ability to access the website without special licensing or payments. Given the history of Google Earth with the initial development of the Keyhole Markup Language it was obvious that the third requirement for this project to be able to support KML and KMZ files would be met (Wilson 2008).

The greatest challenge in finding a way to make use of the Google Earth virtual globe for this project was in meeting the first criterion, to allow the visualization of subsurface data through the use of a transparent ground surface. While Google Earth does allow 3D data to be created and displayed on the Google Earth globe, the visualization of any subsurface data is obscured by the ground surface. Attempts to change the transparency of the surface imagery

through the API were unsuccessful. Extensive searching was done for any information on any attempts to do this in the past whether they had succeeded or failed. It was discovered that older versions of the standard Google Earth application did support this ability in versions 5.0 through 6.2 but this feature was deprecated in later versions. It was also found that Google Earth layers are structured with a main ground surface layer and other layers may be located below the ground surface elevation, but the opacity of this main layer cannot be changed.

When it was determined that the standard API would not work, an API using a Google Earth Enterprise account was considered for use in this project. An Enterprise account with Google Earth allows the account holder more control over the imagery layers in Google Earth, specifically the ability to change the ordering and the opacity of the ground surface layer. It also allows users the ability to host their own maps for public viewing using the Google Earth Enterprise API. However, a license to an Enterprise account could not be obtained in time to complete this phase of the project.

2.2 Cesium Globe

Cesium Globe is a dynamic and attractive virtual globe that is easily implemented into customized websites. Cesium Globe is built on the WebGL platform and the source code needed to implement it into a website is given in JavaScript. It is an open source project which is compatible with PCs and Macs and does not require any plug-in downloads to view which means that it meets the project requirement for being freely accessible to the public without special licensing or payments.

Cesium has helpful and responsive support for developers through their sandcastle website (Cesium) which is similar to the Google Code Playground and also through forums where posted questions are responded to by the Cesium staff, typically within hours.

Cesium Globe allows developers to change the main surface imagery so that the globe can utilize base imagery from Bing, Open Street-Map, Esri, and others. The surface imagery can also be turned off, resulting in a transparent globe where 3D subsurface data can be viewed. This ability is easily implemented in Cesium Globe and fulfills the first criterion for this project.

The Cesium Globe does not support KML or KMZ files in the globe directly but boasts more advanced capabilities to show time dependent data compared to other virtual globes because of its use of the Cesium Language (CZML) file format. CZML files are geo-spatial data files like KML files. They have an extension of “.czml” and are completely indigenous to the Cesium Globe and are not used by any other virtual globes or anywhere else in the GIS industry and are not formalized by the OGC. CZML files are based on JavaScript object notation (JSON) (Crockford 2006) file structure which is a common data sharing file structure for many coding languages including but not limited to: C, C++, C#, Java, JavaScript, and Python. CZML files are structured to be more efficient for incremental streaming so that time-dependent geospatial data can be viewed in series (Cesium).

The Cesium website provides code examples and instructions for writing CZML files which can be shown in the Cesium globe in 2D or 3D, extending above or below the surface imagery. A conversion tool to convert from KML files to CZML files is also available. However, this converter does not support all of the features of KML that are required for this project, such as extrusions and some multi-geometry. Attempts to show 3D subsurface data in the Cesium Globe, which was written originally in KML and then converted to CZML using this conversion tool, were unsuccessful.

2.3 World Wind

World Wind is an open-source, cross-platform-compatible virtual globe developed by NASA Ames Research Center (Boschetti, Roy et al. 2008). Older versions of the World Wind virtual globe have been developed in the .NET framework as a desktop application. The current version is written in Java and the source code may be downloaded from the World Wind website (Wind) and then used to develop a virtual globe as a Java application or Java applet.

The World Wind virtual globe was developed for the geoscience and earth-science communities to use as a data processing tool (Butler 2006). It has been used by other organizations including the OEF mentioned in the previous section.

Because World Wind is open source, custom developed versions of the virtual globe allow it to be hosted in publicly accessible websites. This fulfills the second project criterion for free access to the virtual globe through a website.

The preferred method of hosting either the Java application or the Java applet for World Wind is done through the use of Java Web Start (Zukowski 2002). Java Web Start software allows web users to run applications from a web server. A World Wind application run through Java Web Start does not show up in the browser window. It shows up in another window similar to the Google Earth application that is installed locally.

Java Web Start software is also used to run Java applets that differ from Java applications because they have no independent viewing window and so are displayed within the user's web browser. A World Wind applet run through Java Web Start would show up in the web site similar to the Google Earth API.










The World Wind globe also has the ability to change or remove surface imagery layers so that subsurface data loaded into the globe can be viewed, which satisfies the first criterion. The

World Wind globe also supports the visualization of both KML and KMZ files which fulfills the third project criterion.

2.4 Selection of Globe for Prototype

Table 1 below summarizes how well each of the three globes reviewed satisfy the project criteria. The Google Earth API globe could not meet all of the criteria for this project because of its inability to allow the opacity of the surface imagery to be manipulated. The Cesium Globe did not fulfil all of the requirements of this project because of its lack of support for KML files. The current conversion tools from KML files to CZML files work for most 2D data but not 3D data. If Cesium continues to develop more comprehensive tools for converting from KML and other OGC standard geo-spatial data types to CZML then the Cesium Globe may become a more viable option. This would also be true if Cesium would develop a Cesium globe that would support KML data natively. The decision was made to move forward in development using the World Wind virtual globe because it was the only one to meet all of the minimum requirements of the project scope with the ability to accomplish a significant amount of development within the timeframe given for this project.

Table 1: Summary of how virtual globe criteria were met with the X mark indicating a criteria which was not met and the check mark indicating a criteria which was met

Virtual Globe	Shows Subsurface Data	Can be Hosted Online	Supports KML/KMZ
Google Earth			
Cesium Globe			
World Wind			

3 DEVELOPING A DIGITAL CRUST PROTOTYPE

The second part of this research project is to develop a working prototype to demonstrate how the EarthCube program could use the selected virtual globe, World Wind, to accomplish the first two data tasks for subsurface data specifically. Doing this requires creating a prototype website where the World Wind globe can be accessed and used to discover and show subsurface data hosted on a server. Because this Digital Crust project is not intended to create an entire data network only a few data files are used from the server to demonstrate a use case for this.

The development of two geoprocessing routines is also discussed in this chapter. One is to convert shapefiles from a user machine to KML files on the server and the other is to interpolate soil cross-sections based on borehole log data. Both of these return a KML file of the results to the user for viewing in the World Wind globe. These two geoprocessing routines were developed in addition to the scope of the Digital Crust project to demonstrate the third data task.

3.1 Developing a World Wind Virtual Globe

The World Wind virtual globe is an open-source visualization platform developed for the geoscience and earth-science communities for use as a customized data processing tool (Butler 2006). The source code is written in Java and may be downloaded from the World Wind website (Wind). This website also has tutorials and code snippets available to help beginning World Wind developers understand the many features available within the source code and how to

access different types of World Wind functionality. However, even with these examples, the help available to go from downloading the source code to launching a World Wind globe is not complete enough on the World Wind site to make it possible to do without help from other sources. It is useful then to describe how the process was accomplished in this project as a reference to anyone attempting to build a World Wind globe. The development of the World Wind globe as an application or an applet is also discussed as well as a brief explanation of how to launch them through Java Web Start (Zukowski 2002). The main components necessary to begin development of any World Wind virtual globe are listed as follows:

1. The necessary source code.
 - a. The World Wind source code
 - b. An up-to-date version of the Java Development Kit (JDK) to support the Java code
2. An integrated development environment (IDE) with which to write, organize, and compile the code

When preparing these two components for development in this project, information from several sources was used to prepare the source code and choose an appropriate IDE. Again, no existing sources could be found which describe all the necessary information and procedures to do this so it is useful to discuss how it was done for this project specifically.

3.1.1 Obtaining the Source Code

The World Wind source code can be downloaded from the World Wind web site (Wind Aug. 2013) at <http://worldwind.arc.nasa.gov/java/>. On this page there are two downloads available, “World Wind Java SDK 1.5” and “World Wind Java SDK 2.0 Daily Builds.” The SDK 1.5 download was used for the development of this project.

The Java Development Kit (JDK) is the library of the core functionality of Java programming. An up to date version of the JDK required to support the development of Java code can be downloaded from: <http://www.oracle.com/technetwork/java/index.html> on the official Java website. Before beginning development with the World Wind source code or any Java code this JDK must be downloaded and installed. After installing the JDK certain environment variables on the machine used to develop the code may need to be set up before development begins. The process for setting up these environment variables may differ based on the JDK version installed and the operating system of the machine being used. This project used a Windows 7 operating system and the JDK labeled on the website as “jdk1.7.0_21” because it was the most up to date JDK available at the time. After the JDK is installed and the environment variables are set up a Java IDE can be installed to begin the development process.

3.1.2 Selecting an IDE

When selecting an IDE to use for developing with the World Wind Java source code it is important to use a 32-bit compatible IDE because errors will be encountered when attempting to compile the World Wind Java source code in a 64-bit IDE.

For this project, development was done using the 32-bit Eclipse Juno IDE because it is available for free and it is widely used which made finding solutions to coding problems much easier. It also has the distinct advantage of having other plugins like Aptana Studio, which is used for web development, and PyDev, which is used for Python scripting. These features became very useful for this project because it includes code development in Java, Python, and web design.

After an appropriate Java IDE has been selected and installed, the libraries from the source code, which will be used in the development of the World Wind globe, need to be set up for referencing.

3.1.3 Java Code Organization

Before development of a World Wind globe begins, a Java project needs to be created in an IDE with a coding library which includes DLL files from the source code downloaded from the World Wind website. Instructions to organize these files are not provided on the World Wind website because the process for doing this changes slightly depending on the IDE being used and the type of project being developed. It is useful then to provide the specific steps taken to set up the Java project and its source code libraries for this research to assist others who may seek help developing a World Wind virtual globe. These steps are described in APPENDIX A with some brief explanation given below.

Java code is organized into a hierarchy of projects, packages, and classes. A project contains several folders to organize all of the content. Typically there are two folders within a project which are named “src” and “lib” to hold the code developed for the project and coding libraries respectively.

The “src” folder contains packages. The “lib” folder contains other content referred to by classes such as dynamic-link libraries (DLL files), text files, images, etc. More information about Java code organization can be found at the official Java website.

3.1.4 Application or Applet

The World Wind virtual globe was developed as both an application and an applet for the Digital Crust to show the differences in use and capabilities of each. Both may be run from the

server through Java Web Start. Because they are different from both the user and a development perspective it is useful to point out some of the main differences now before describing how they were developed. From a user perspective the main difference is that an applet run through Java Web Start appears within the web browser and inherits its viewing frame dimensions from the HTML div tag it is assigned to. It does not require any downloads or installations.

An application run through Java Web Start requires the download of a JNLP file. When this file is run the application appears in its own viewing frame outside of the web browser. This JNLP file will run the application whether the website it originated from is opened a web browser or not as long as there is an Internet connection. Applications also tend to handle more memory better than applets so a World Wind globe with extensive features and capabilities would be better suited as an application than an applet.

From a development perspective, both must contain classes that act as the viewing frame object in which the globe and the other controls will be shown. The viewing frame class of an applet must be extended from the JApplet class to appear in a web browser and the viewing frame of an application must be extended from the JPanel class to appear in its own viewing frame. Both the JApplet class and the JPanel class are part of the JDK. A World Wind Application is easier to start with because there are several examples of applications with different capabilities already created that come with the download of the World Wind source code. The differences between a World Wind applet and application are discussed further in Section 3.3.

3.1.4.1 Creating a Basic Application

Once the libraries for the source code are established, creating applications with the demo examples provided with the source code becomes very easy. The application built for this

research project was built with the intent of allowing the user to generate a subsurface soil cross section within the virtual globe by doing the following steps.

1. Identify an area where borehole log data already exists in a database.
2. Draw a polyline within the extent of that area.
3. Return and load a KML of the soil profile below the polyline drawn as interpolated from the existing borehole log data.

This project took specific advantage of the “KMLViewer.java” demo in the “examples/kml” folder, the “Line Builder.java” demo found in the “examples” folder and the “WMSLayerManager.java” demo found in the “examples/kml” folder of the World Wind source code downloaded as mentioned in APPENDIX A.

The code from these three demos were added as new Java files to the project package and became the KMLViewer class, Line Builder class, and WMSLayerManager class respectively. The KMLViewer class has controls in a menu bar that allows the user to add KML files as overlays to the World Wind globe. The Line Builder class has a control panel with buttons that allows the user to draw a polyline on the World Wind globe surface within the application. The WMSLayerManager class allows the user to show different overlays of tile imagery for the World Wind globe from different databases.

The functionality of these classes was combined by creating all the classes within the same package and extending the WMSLayerManager.AppFrame class from the LineBuilder.AppFrame class then extending the LineBuilder.AppFrame class from KMLViewer.AppFrame class. Because the AppFrame class of each of these parent classes was the viewing frame object of each, extending one from the other combined all the parts of each class’ viewing frame into one. By extending classes for the viewing frames in this manner the

application contained all of the features of the KMLViewer demo, the LineBuilder demo, and the WMSLayerManager demo.

Other custom classes were also written to perform some intermediate functionality which was used in passing the coordinates from the polyline drawn to Python Scripts on the server to interpolate the soil cross sections and return them to the World Wind globe application. The code written for this application is provided in Appendix B.1.

3.1.4.2 Creating a Basic Applet

The applet created for this research project was built to demonstrate how a World Wind globe applet could be used in a webpage to show KML files of 3D subsurface data. It allows the user to load KML files directly from the user machine or from a web source.

The World Wind Java source code includes two applet demos within the “examples/applets/” folder mentioned in APPENDIX A. The code in the “WWJApplet.java” file was used to create the applet with adjustments were made to add the layers menu and the menu bar for adding KML and KMZ files.

The code example from the KMLViewer class mentioned in Section 3.1.4.1 above was used to help develop this additional functionality. Extensions for the viewing frame classes could not be used in this case because the structure of an application window and an applet window are different in the Java JDK, as mentioned in Section 3.1.4. In order to apply the functionality of the KMLViewer class to an applet the code and all of its methods and the origins of those methods had to be understood and written into the applet. The code for the applet developed in this project is included in Appendix B.2.

3.1.5 Launching Through Java Web Start

The application and applet developed for this project are launched through the website using Java Web Start. Other methods may be used to run Java applications or applets but this is the currently preferred and most broadly accepted method today because it is the most supported by browsers. Information about Java Web Start software is available at the official Java website (Java 2014) but it is discussed here briefly to assist in the explanation of other material in this thesis.

Java Web Start software does not require any installation because it is implemented through JavaScript and HTML code in a website. Java Web Start implementations allow web users to run applications from a web server through the use of compiled Java executable files called JAR files with the “.jar” extension. The JAR files exist on the host server and are referenced through another file called a JNLP file with the “.jnlp” extension which is downloaded by the user. JNLP files are typically very small in size and are downloaded quickly. A JNLP file establishes the link between the user computer, or browser, and the server.

To run an application or an applet through Java Web Start three files are required: a compiled executable JAR file of the applet or application, a keystore file, and a JNLP file referencing the JAR file. The process to create a JAR file will depend on the IDE being used for development. JNLP files are readable code which may be written in a plain text editor. All of these files should exist somewhere within the root folder where the web files are hosted by the website.

3.1.5.1 Creating a Keystore

To run any JAR file through Java Web Start it needs to be signed and a keystore needs to be generated to sign the JAR file. This can be done in the command line by directing the

command line to the file location of the JAR file with the *cd* command and creating a keystore with the *keytool* command. Once a keystore has been generated it can be used to sign a JAR file using the *jarsigner* command. Examples of how to create a keystore and use it to sign a JAR file are provided in APPENDIX C.

Signing JAR files can be done with a self-signature or a certified signature. Since the new version of Java was released in January, 2014 a self-signed JAR file is not considered secure and will be blocked by the user's local machine unless special permission is given to allow it (2014). The JAR files for the application and applet used in this research project are self-signed so instructions for allowing these to run are provided in the home page of the site where the applet and application are being hosted at (Smith and Whitman 2013).

3.1.5.2 Creating JNLP Files

Once the JAR file has been created a JNLP file must be created to reference it. JNLP files are necessary for launching applications and applets through Java Web Start and the structure for a JNLP file differs slightly for applications and applets. The JNLP files created for this project were based on examples provided in the World Wind source code.

The example used to run the application is found within the main folder of the World Wind source code under the “demosite” folder. The example used to run the applet in the website using JavaScript is also found within the main folder of the World Wind source code under the following path: ...src/gov/nasa/worldwindx/examples/applet/.

The JNLP files are referenced within the website through HTML and JavaScript code. This code also differs between the application and the applet because applets are launched as part of the website and must be provided with a div tag and specifically dimensioned while the

application has its own window and is run through a button or link which references the application's JNLP file.

Examples for the JNLP files used specifically for this project as well as the HTML and JavaScript code to run them are provided in APPENDIX C.

3.2 Geoprocessing Using ArcGIS and AHGW

The World Wind application and applet serve as visualization platforms for geospatial data but are not intended to perform heavy-duty data processing. The data processing aspect of this project demonstrates the prototype's ability to communicate with data on the server and return subsurface data to the virtual globe for visualization. The ability to do this kind of geoprocessing online also makes the website more attractive for use to the geoscience community which may encourage more data sharing.

To accomplish the data-processing routines for this research project another software system was installed onto the server which hosts the website and World Wind globes. Two processing routines were built, one for the application and another for the applet. The routine for the application takes longitude and latitude coordinates forming a polyline to return a KML file of subsurface soil cross-sections below the polyline. The routine for the applet takes a set of files that form one shapefile and returns a KML file version of the shapefile loaded into the applet globe.

Esri's ArcGIS software was chosen as the software system to for use in these geoprocessing routines because it has the advantage of using its tools through Python scripting through the arcpy library. The arcpy library is a Python-based library of all the tools available in the ArcGIS software package. Using ArcGIS also allows the use of Arc Hydro Groundwater

(AHGW) software. AHGW is a suite of geoprocessing tools distributed by Aquaveo that works as a plugin to Esri's Arc Desktop software including Arc Map and Arc Scene. It is specifically designed for subsurface data processing and visualization and includes tools for generating soil cross-sections from borehole log data, which is ideal for this research project. Its tools are also available for use through Python scripting through the arcpy library with some initial setup.

The early development of this project was begun using ArcGIS 10.0 but it was found that the 10.0 version lacked some desired features offered by ArcGIS 10.1. Serious development of geoprocessing tools was done using ArcGIS 10.1 but complications arose in the web service development that led to attempts to use ArcGIS 10.2, the newest version of the ArcGIS suite. The complications with using different versions of ArcGIS for running these processing routines are described here because the differences which caused these complications are not advertised by Esri.

3.2.1 Setting Up the ArcGIS Environment to Use AHGW Tools in Python Scripts

Python scripts using the arcpy library require the AHGW tools to be imported specifically and importing all of the AHGW tools would take an unnecessarily long amount of time compared to importing only a few. A new toolbox was created for this project that contained only the AHGW tools necessary for the geoprocessing routine to be accomplished. This was to facilitate the importation of only the AHGW tools that will be used. The method for importing this new toolbox is illustrated in the Python code of Appendix E.2.b.

The purpose of using AHGW, as previously described in Section 3.1.4.1, was to return a KML of subsurface cross-sections below a specified polyline where the polyline was within the geographical extent of some borehole log data existing on the server. The data used for this project were taken from the Aquaveo tutorials (Aquaveo 2014) using the "Horizons" tutorial.

This tutorial has datasets already set up for creating geosections using soil profile raster data in the Roseville, CA area.

3.2.2 Initial Problems with Arc 10.0

Arc Desktop has a tool called “LayerToKML” which can be used to convert Esri-supported files such as feature classes, shapefiles, and layer files to KMZ files. In Arc Desktop versions 9.3 and 10.1 there is an option with this tool to specify whether or not the converted file should be “clamped to ground” meaning the file data will only show on the ground surface and not extend above or below it. This selection should be checked for files such as simple polygons, points, polylines, and surface imagery which are intended to only represent surface data. However, this option should be disabled when converting 3D data existing above and/or below the surface.

Version 10.0 does not have this option and makes all conversions “clamped to ground” by default. There are no options in the desktop tool or the Python code to change that setting. This excludes the 3D aspects of any data used by this tool. Because the application and applet developed for this project support KML files and this project is focused on displaying 3D subsurface data, version 10.1 of the ArcGIS suite was investigated as a solution.

3.2.3 Developments in Arc 10.1

The 10.1 version of ArcGIS introduced some problems for the geoprocessing and also for web services because of Esri’s transition of Arc Server from 32-bit compiling to 64-bit compiling.

In version 10.0 the installation of Arc Desktop and Arc Server included a 32-bit version of Python 2.6 and both made use of the same Python executable file. This made using the desktop plugin AHGW tools through Arc Server very easy. In the 10.1 version the Arc Desktop

installation comes with a 32-bit Python 2.7 executable file and the Arc Server version comes with a 64-bit Arc Python 2.7 executable file. This presented several problems, most of which were overcome, but one that was not.

The web service requests for the Python scripts would not work when using the 32-bit Python executable installed by Arc Desktop 10.1. The Python code would work only until reaching the “import arcpy” statement and then break. The exact reason for the code not being able to use the arcpy tools from the 32-bit installation of Python 2.7 from Arc Desktop 10.1 is still unknown. The web service did work for Python scripts using the 64-bit Python executable file installed by Arc Server 10.1 but had problems being accessed through web service requests.

Although web services could be run using the Arc Server 10.1 installation of the 64 bit Python 2.7 executable the geoprocessing capabilities were limited. ArcGIS supports several database types but the 64-bit version of ArcGIS does not support personal databases with the ‘.mdb’ extension. The geoprocessing workflows developed for this project used the personal databases originally and had to be adjusted to use file geodatabases with the ‘.gdb’ extension in order to accommodate the 64-bit compilation of ArcGIS. Another limitation to the 64-bit version of ArcGIS is the support of the AHGW plugin. Because the AHGW plugin is compiled in 32 bit the web services using the 64-bit Python executable and the 64-bit ArcGIS could not recognize the AHGW tools.

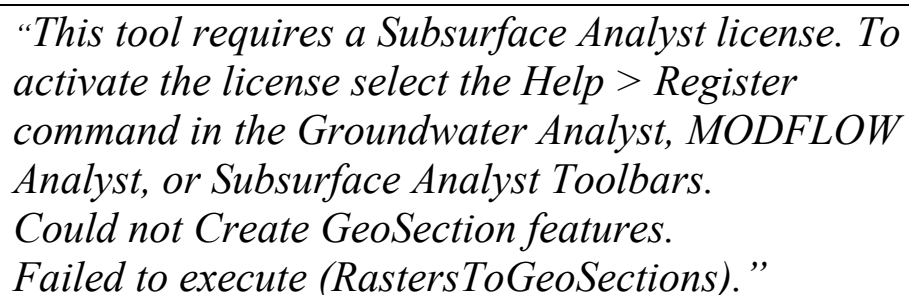
The inability to use AHGW through web services using ArcGIS 10.1 made it necessary to investigate the new 10.2 version of ArcGIS in favor of version 10.1.

3.2.4 Using Arc 10.2

Like the 10.1 version, the 10.2 version of Arc Desktop and Server were compiled differently as 32 bit and 64 bit programs that also came with different Python 2.7 executable files

in 32-bit and 64-bit respectively. The Python scripts were able to run through web services which called the 32-bit Python 2.7 executable installed with Arc Desktop 10.2 and successfully imported the arcpy library. However this did not completely solve the issue of accessing the AHGW tools.

Different tools in the AHGW plug in require different levels of licensing. The two tools used in the Python script for generating soil cross sections are AssignHydroIDGW and RastersToGeosections. The license to use the AssignHydroIDGW tool is free with any valid ArcGIS user license because it is part of the Groundwater Analyst toolbox (Aquaveo 2014). The RastersToGeosections tool is part of the Subsurface Analyst toolbox so it requires a higher level license. For this research project the full license was obtained which included a license to use the Subsurface Analyst tools and the Python script worked as expected with no errors when it was test run from directly within the host server. However, when the Python script was run from a web request it would fail at the RastersToGeosections tool with the error message shown in Figure 1.



“This tool requires a Subsurface Analyst license. To activate the license select the Help > Register command in the Groundwater Analyst, MODFLOW Analyst, or Subsurface Analyst Toolbars. Could not Create GeoSection features. Failed to execute (RastersToGeoSections).”

Figure 1: Error message given when trying to run RastersToGeosections tool from a web request

This message reveals that the higher Subsurface Analyst license registered to the host server is not honored when the AHGW tools are initiated by a request from a web service. The managing company of the AHGW plugin, Aquaveo, was contacted about this issue but a solution has not yet been found.

3.2.5 ArcGIS/AHGW Summary of Conflicts

The AHGW tools could not be utilized for this project with the resources available because the necessary criteria to access these tools and convert outputs to KMZ could not be met by any available version of ArcGIS. These necessary criteria are as listed below.










1. Licensed version of ArcGIS installed onto server
2. LayerToKML arcpy tool which supports 3D KMZ
3. 32 bit Python library
4. AHGW licensing working with web requests

Table 2 below summarizes the conflicts found with creating a web service through PHP for the AHGW tools with different versions of the ArcGIS suite. The red 'X' marks indicate where there was a conflict, the blue check mark indicates where there was no conflict, and the yellow question mark indicates where progress had not gone far enough to find out if there would have been a conflict or not.

This left the geoprocessing side of the project at a standstill because the AHGW tools could not be used by any version of ArcGIS by a web service to create soil cross sections.

To show the potential ability for using ArcGIS in a web service, a processing tool to convert shapefile data to KML and load the KML into the globe was developed using version 10.2 of ArcGIS. This did not require any of the AHGW tools but still accomplished the task of demonstrating some kind of processing using ArcGIS through Python scripts

Table 2: Summary of problems with using different versions of ArcGIS with a PHP Web Service for AHGW

Version of Arc GIS	3D KMZ Conversion	Access to 32bit arcpy through PHP	AHGW License with Web Request
10.0			
10.1			
10.2			

3.3 Website Development

The website developed for the Digital Crust can be accessed from is hosted through a virtual machine at BYU campus using the Windows Server 2008R2 operating system and an IIS 7 server. The home page for this website is: <http://digcrust.byu.edu/> (Smith and Whitman 2013).

Scripts were developed to allow communication between both World Wind globes and the server and accomplish the respective tasks which the application and applet were designed to perform, namely generating geosections and displaying subsurface KML data. A mixture of programming languages was used to accomplish the communications between the globes, server, processing software, and website.

PHP is a programming language used for server-side web scripting (Lerdorf, Tatroe et al. 2006). It is able to pass information between the server and the web user and was used in the development of this website as the primary means of passing information between the World Wind globes and ArcGIS software on the server.

Command line codes were executed on the server using PHP to run Python scripts responsible for data processing using ArcGIS and AHGW. The PHP scripts also capture all

printed outputs from the command line and Python scripts. These output messages can be checked in PHP to direct the web services to other functions if necessary.

JavaScript and HTML were used for the main website development including the layout of the website, navigation controls, and implementing the World Wind application and applet through Java WebStart. Both an application and an applet are available through the website to demonstrate the advantages and disadvantages and abilities of each. These advantages and disadvantages as well as the processing routines for the application and applet are described in the following below.

3.3.1 World Wind Applications

Using the World Wind virtual globe through an application as opposed to an applet offers some advantages. Applications are run by the client machine through an Internet link to the JAR file from the web host server. This allows the application to communicate closely with the client machine to browse through, open, and read client side files. Applications also run outside of web browsers so they do not need to be cross browser compatible and they run faster than applets and can handle more memory. Another advantage of using applications is that the host server can handle several applications with different specific abilities. This way, users can run the application that pertains to their needs and save necessary information to the server so that it can be loaded into the application or the server later.

For the World Wind application specifically, there is a distinct advantage for developers because many demo applications are downloaded with the source code. These demo applications provide examples of how to use the World Wind Java source code and can be easily customized to suit more specific needs.

The application developed for this project has three distinct abilities which enable it to accomplish its goal to allow the user to generate a subsurface soil cross section within the virtual globe as mentioned earlier in Section 3.1.4.1.

1. It can load raster data from a preloaded WMS server provided with the World Wind source code and from another WMS server specified by the user.
2. It can draw polylines on the globe surface and send a geoprocessing request to the server through PHP.
3. It can load KML or KMZ files from a client side file or from a URL.

The first ability, loading data from a WMS server, demonstrates the application's ability to provide large amounts of global scale data to users where there is data available online. The application also starts up with the data from NASA's WMS server already loaded. If a WMS server existed which hosted locations of available borehole log data then that server could also be loaded using this capability. In the absence of such a server, the application contains an option in the main menu labeled "Add Data Boundaries" seen in Figure 2 to load a small polygon into the globe as shown in Figure 3. This polygon represents the extent of borehole log data existing in Roseville, CA from the AHGW Horizons tutorial mentioned in Section 3.2.1.

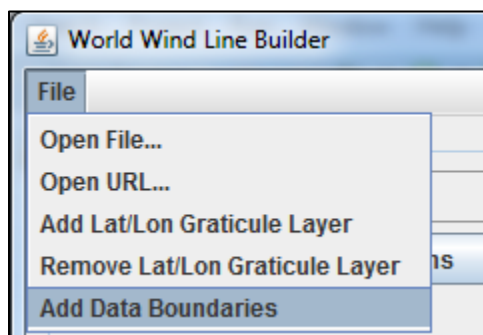


Figure 2: Application menu option which will load a file showing the extent of borehole log data on the server which may be used to generate soil cross sections

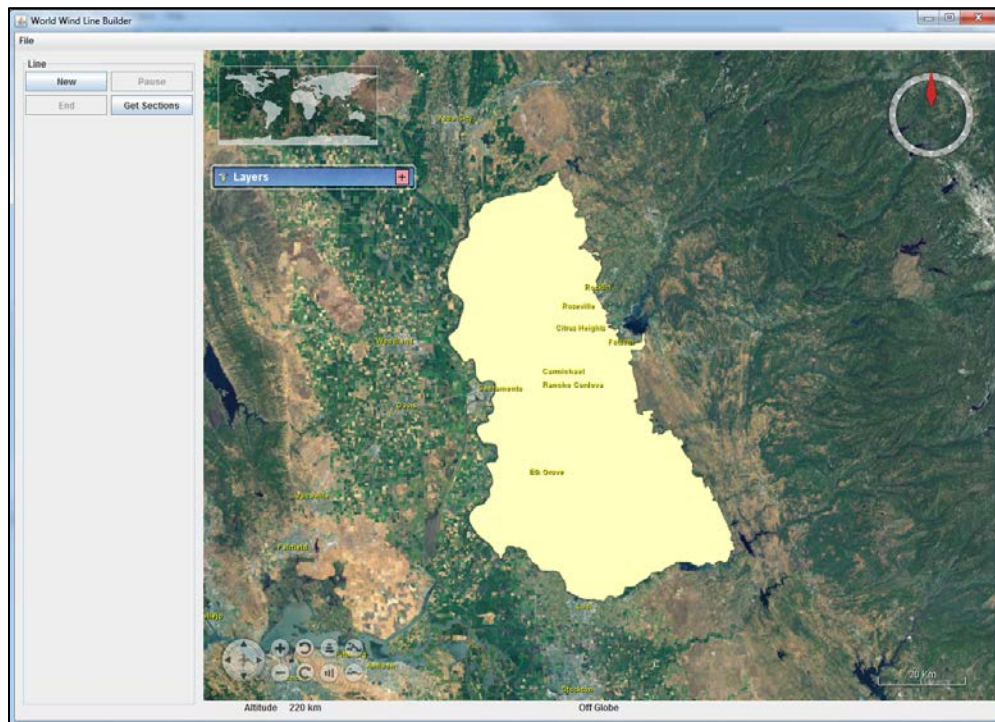


Figure 3: A polygon loaded from selecting the menu option in Fig2. This polygon represents the extent of available borehole log data on the server

The second ability, to draw polylines in the globe, allows the user to specify the location of the cross sections to be created through the buttons shown in Figure 4 and in the upper right hand corner of Figure 3.

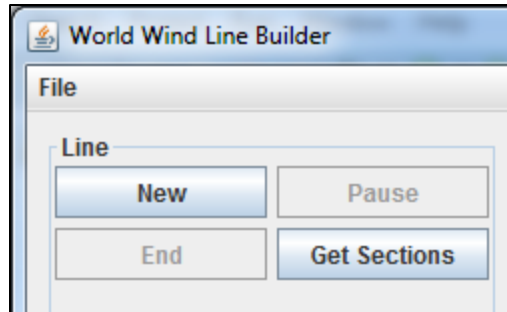


Figure 4: Buttons used for polyline drawing controls in application

The “New” button turns on the polyline drawing mode. The polyline is drawn by clicking and dragging to different points on the globe. As the polyline is drawn the coordinates of the line appear below the buttons on the left. The “End” button exits the polyline drawing mode. If the “New” button is clicked again after clicking the “End” button, the previous line and the coordinates will be erased. The “Pause” button pauses the drawing mode to allow the user to pan on the globe and changes its text to “Resume.” If the “Resume” button is pressed the drawing mode resumes and pan mode is disabled. The “Get Sections” button sends the coordinates of the polyline to the server to generate soil cross sections below the polyline based on the existing borehole log data. This will also open a web page where the cross section can be downloaded in KML format after the processing is complete.

The third ability, loading KML or KMZ files into the globe, allows the user to load the cross section data generated into the globe with the “Open File...” option shown in Figure 2 above.

3.3.1.1 Creating a Geosection

When the user clicks the “Get Section” button in the application a geoprocessing request is initiated which creates a KML file of a soil cross section below the polyline drawn. The process is described here and illustrated in Figure 5 below.

The “Get Section” button initiates the geoprocessing request by opening a browser window to a PHP page. The URL of the PHP page includes the latitudinal and longitudinal coordinates of the polyline drawn on the globe.

The PHP page accesses the coordinate values passed through the URL and redirects to another PHP page, while sending the coordinate values to the second PHP page through the URL again. The second PHP page also accesses the coordinates from the URL and uses them to create a single string variable that represents a command line argument for running a Python script. The purpose of using two PHP pages in this way is to allow the user to see a waiting screen on the first PHP page while the Python script is running.

While the second PHP page is loading the server command line is running a Python script which creates the cross section based on the coordinates of the polyline drawn. The Python script first uses the coordinates to create a comma-separated values (CSV) file. The CSV file is used to recreate a polyline as an ArcGIS feature class section line. The section line is then used with the AHGW tools to create a geosection showing the interpolated soil profiles below the section line from the raster data. A KML of the geosection is then created on the server at a location where it can be downloaded by a web browser. Once the Python script has finished running, the second

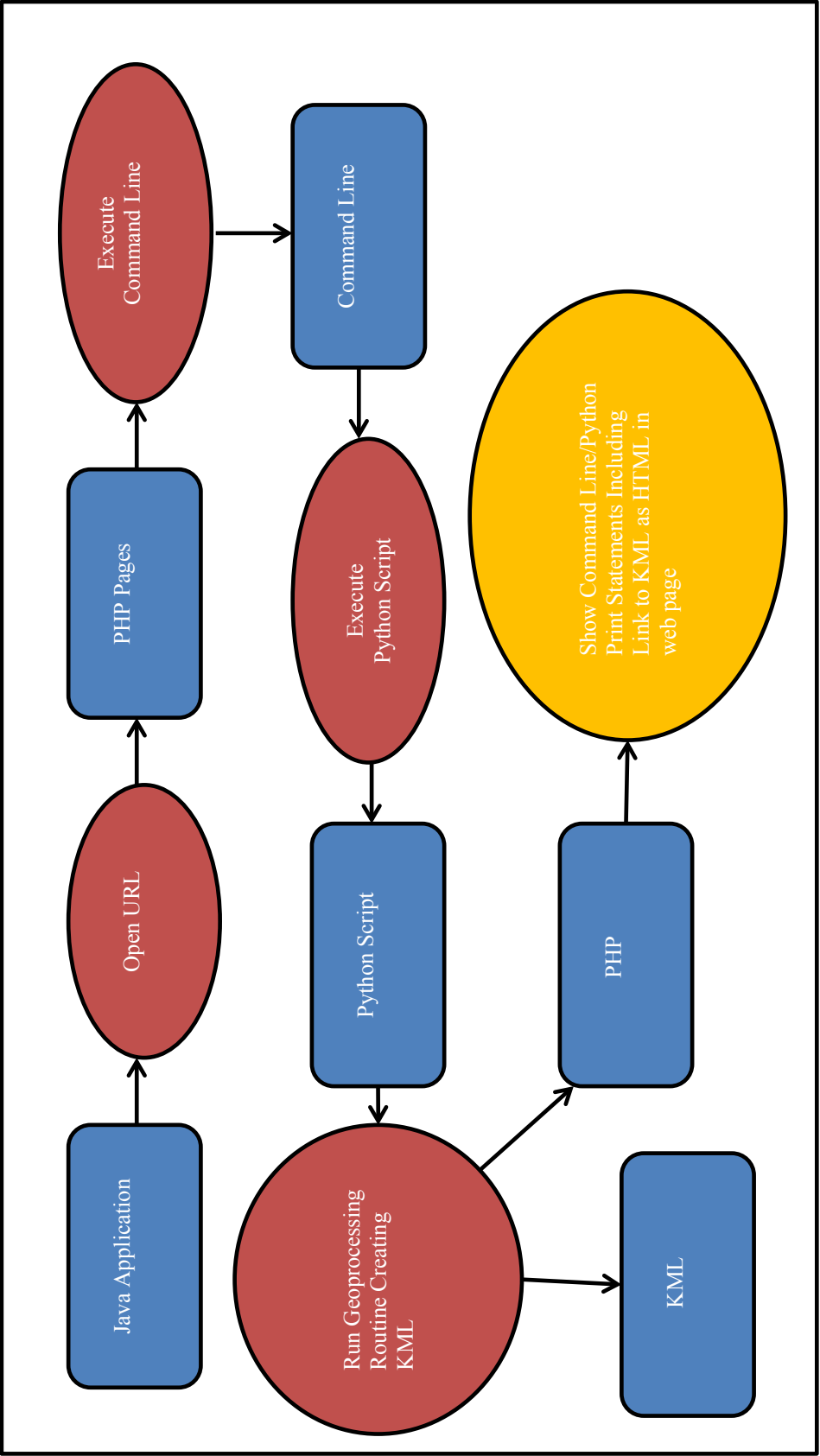


Figure 5: General processing workflow for application. The blue boxes represent elements of the workflow and the red ovals represent events within the workflow between elements. The orange oval represents the last workflow to be accomplished

PHP page finishes loading and displays information about the success or failure of the process with a link to download the new KML if it is successful.

This process could not be completed to allow a full demonstration because of the problems encountered with using the AHGW. Despite these problems a theoretically workable Python script was written for use with ArcGIS 10.2 that could be easily implemented when a compatible version of AHGW becomes available. The codes used to write all of the PHP and Python scripts used in this process are provided in Appendix E.2.

3.3.2 World Wind Applet

In addition to developing a World Wind application, a World Wind applet was also developed. The applet is limited in its performance because it runs within a web browser and some web browsers, such as Google Chrome for Macs, may not support Java or support Java Web Start fully (Java 2014). Applets are also limited in communication with client machines. Security issues prevent applets in general from browsing client side files. There is also very little documentation provided with the World Wind Java source code to help with developing an applet.

Despite these limitations the applet also offers the distinct advantage of being displayed directly within the website. Because the applet is displayed in the website, the JavaScript of the web page where the applet is hosted inherits some of the Java methods that are included in the applet JAR file. This means that the inability for the applet to communicate with client side files may be overcome by using JavaScript/HTML controls to do it for you and then communicating with the applet via JavaScript. This gives developers the opportunity to maximize viewing space for the virtual globe by leaving the applet as only a viewing window that is not cluttered by menu bars, buttons, menu trees and other control items. Rather, all of these controls can be created as

HTML form controls that can be part of a collapsible menu form in the webpage allowing the control options to be expanded, contracted, and used without compromising the viewing space of the virtual globe window as in Figure 6.

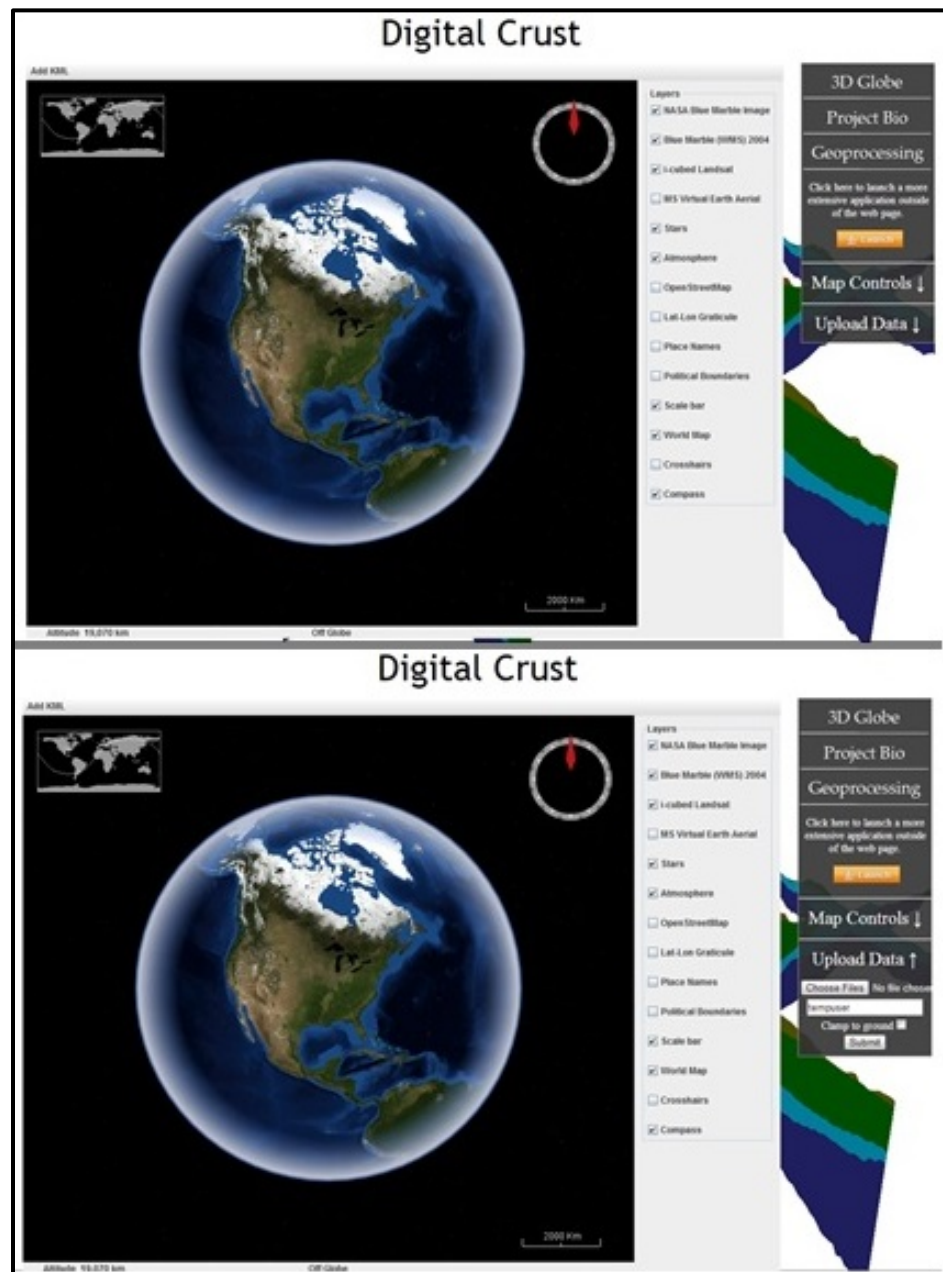


Figure 6: Example of collapsed controls menu. Top - Menu is fully compressed. Bottom - Menu is partially expanded

The applet developed in this project has several tile layers available in which are shown in the right checkbox menu contained within the applet window. The surface imagery may be turned off by unchecking the top three layers “NASA Blue Marble Image”, “Blue Marble (WMS) 2004”, “i-cubed Landsat”. The “Map Controls” section to the right of the applet window can be expanded to show several controls to change the viewing altitude and direction of the virtual globe as well as the ability to fly to locations of specific interest. The first two locations of specific interest are Roseville, CA and Chesapeake Bay, VA as shown in the fully expanded “Map Controls” section in Figure 7. These two locations contain preloaded soil cross sections which may be viewed by unchecking the surface layer imagery options as mentioned above then clicking the “Fly to location” button when the item is selected.

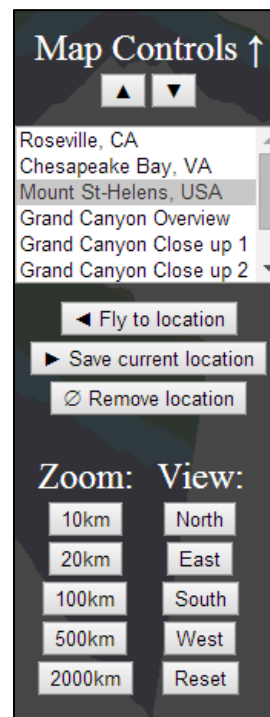


Figure 7: Image of the expanded map controls for the applet

KML and KMZ files can be uploaded to the globe through the “Add” option in the menu bar at the top of the applet window. These files may be from a web source or local to the user computer. Local files are entered by selecting the options Add>>KMZ or KML and then entering the full file location. Choosing the same options but entering a URL to an online file instead of a local file location will add a file from a web source. This Add option only accepts files formatted as KML or KMZ files.

To allow users to view data which is not already in KML or KMZ format and to demonstrate the capabilities for the applet, website, and server to pass and send information a geoprocessing routine was developed for this project which accepts shapefile data, converts it to KML and loads it into the applet’s virtual globe.

Shapefile data can be uploaded using the “Upload Data” section of the menu bar to the right of the applet window shown in Figure 8. The controls in this menu are HTML form controls that call JavaScript functions. These controls allow the client to upload data and send a request to the server to initiate a processing routine.

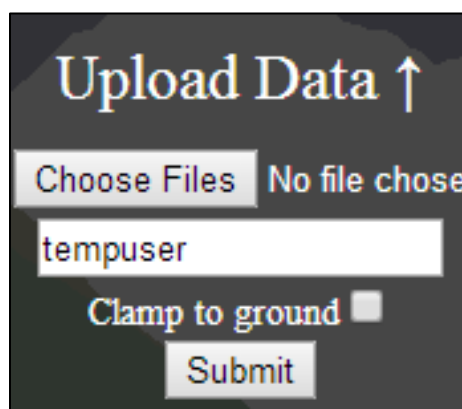


Figure 8: Image of the HTML controls to upload shapefiles into the applet globe

This processing routine converts uploaded shapefiles to KML and loads the KML into the applet globe. The specifics of this process are described in the section below and are also illustrated in Figure 9.

3.3.2.1 Converting Shapefiles to KML from the Website

The first step in this process is to receive inputs. An HTML form was created on the web site where the World Wind applet is hosted which allows users to upload multiple files (Figure 8). The “Choose Files” button in this form opens a file browser showing client-side files. The user can navigate to a file and select all the files associated with one shapefile dataset. Although the browser will allow the user to select any file type, only files with the extensions associated with shapefile data are actually uploaded to the server. The permitted extensions are – .shp; .shx; .dbf; .prj; .sbn; .sbx; .fbn; .fbx; .ain; .ixs; .mxs; .atx; .shp.xml; .cpg.

A checkbox is also available as part of this form to allow the user to specify whether or not the resultant KML should be clamped to the ground. If this checkbox is checked the resulting KML file will not appear in three dimensions. After completing these two form items the “Submit” button may be clicked which initiates two processes. One of these processes runs the geoprocessing request and the other process checks for a result from the geoprocessing request every two seconds.

The geoprocessing request is begun when the “Submit” button is clicked and opens a new tab to a PHP page. This PHP page specifies the criteria for acceptable file extensions and uploads the allowable files to the server. The page then shows a waiting screen with an animated GIF image and a message about the uploaded content and the success or failure of each file upload. This page also creates a single string variable representing arguments to pass to the Python code and navigates to the next PHP page with the variable being passed through the URL.

The next PHP page accepts the variable in the URL through the “\$_GET[]” array and appends the string to another string which represents a command line prompt. This new string is used in the “exec()” command which sends a request to the server command line to run a Python script with three arguments. The arguments are the file path on the server where the shapefile data has been saved, the name of the shapefile data, and an integer of value 0 or 1 to indicate whether the checkbox on the form is checked.

Once the second PHP page is initiated, and while it is still loading, the server command line is running a Python script which accepts the three arguments passed to it. This Python script uses the “LayerToKML” tool in the arcpy library to save the shapefile data as a file with the ‘.lyr’ extension (layer file) and converts the new layer file to a KMZ. The KMZ is then unzipped to a KML in a location on the server where it can be downloaded from a web browser. When the Python code is finished running, the second PHP page then finishes loading and shows information about the failure or success of the geoprocessing result including a link to download the KML file generated.

A text file is also created by the Python script which contains information about the newly created KML file including the URL to download it at and a time stamp to mark when the file was created. This text file is important for the other process initiated by the HTML “Submit” button.

The other process initiated by the HTML “Submit” button intermittently checks for the creation of the new text file with the information about the KML. This process begins with a JavaScript function inside of the HTML page where the World Wind applet is hosted. This JavaScript function uses the jQuery (De Volder 2006) library to run another PHP page which is not opened in the user’s browser and receives a variable back.

The PHP page run by the JavaScript function reads the text file created and parses through it to identify the URL location of the text file and the time stamp. The URL and time stamp are passed back to the JavaScript. If the timestamp for the text file is more recent than the time the submit button was pressed the JavaScript will open a message to the user from the web page hosting the virtual globe applet. This message asks if the user would like to upload the KML file into the virtual globe. If the user selects “OK” the KML is loaded from the URL location passed to the JavaScript through the jQuery call to the PHP page. It also generates a new message informing the user that a link to download the KML is available on the PHP page that has now finished loading. If the user selects “Cancel” then the message about the link disappears but the KML is not loaded.

This process has many pieces but when run by the user the process appears to be quick and seamless. From the user’s perspective only the page with the World Wind applet, one more tab, and an options message appear. This is done without closing, refreshing, or navigating away from the page with the World Wind applet. This is important because any KML files loaded into the applet’s virtual globe are lost when the page is reloaded or refreshed. The current process allows the user to load KML files without refreshing or reloading so that multiple files can be loaded in one session.

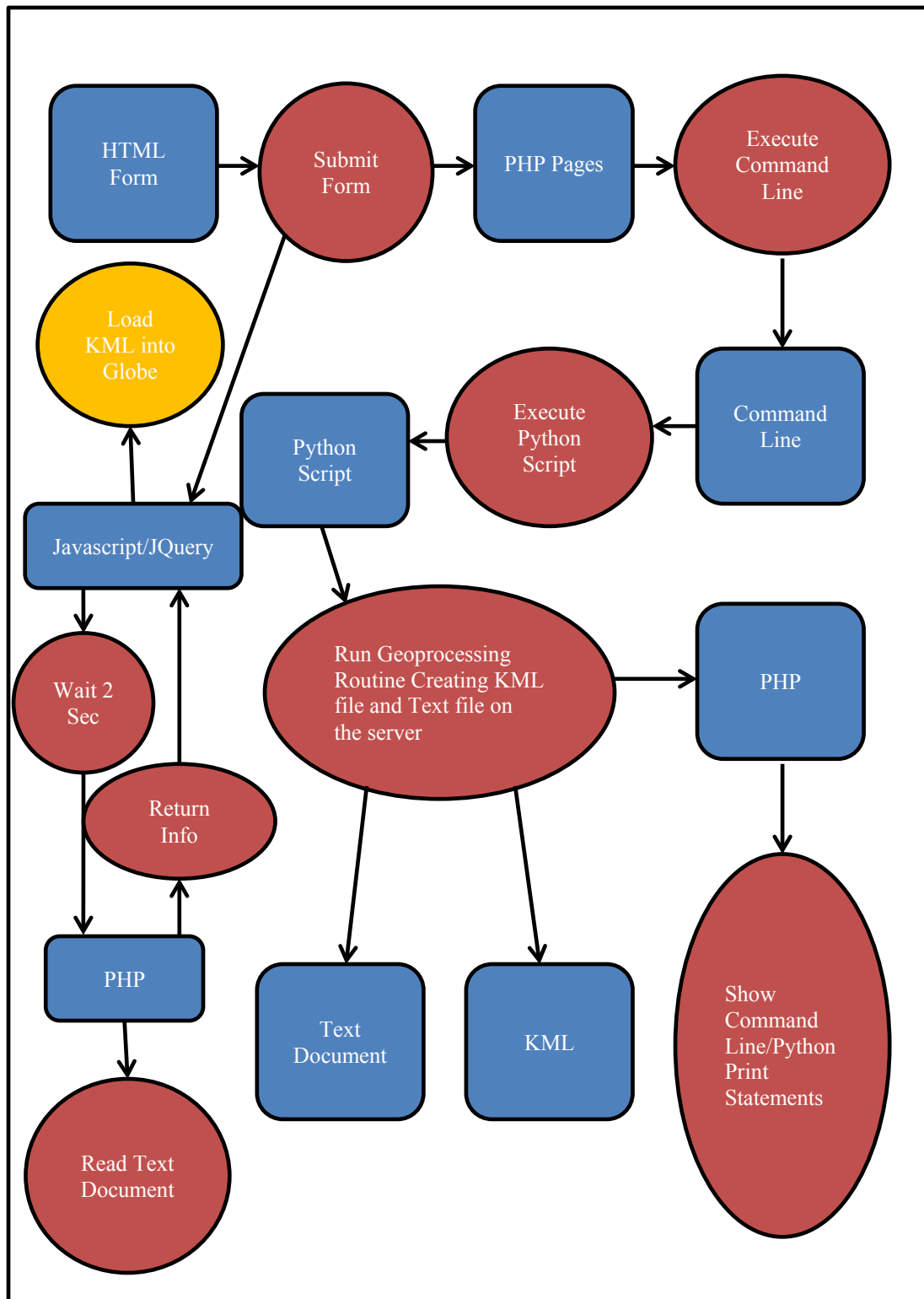


Figure 9: General processing workflow for applet. The blue boxes represent elements of the workflow and the red ovals represent events within the workflow between elements. The orange oval represents the last workflow to be accomplished

4 CONCLUSIONS

The purpose of this project was to develop a prototype website for discovering and visualizing 3D subsurface data in a cloud-based environment. This was done through researching current 3D subsurface visualization methods. A virtual globe was selected to act as a visualization platform for the subsurface data. Two processing routines were also developed to extend the work using the ArcGIS and AHGW software packages. Finally, a prototype website was created to manage the virtual globe and processing requests to the server.

4.1 Selection of a Virtual Globe

Three virtual globes were considered as potential candidates for use in this project - Google Earth, Cesium Globe, World Wind. That candidacy was determined based on the following three main criteria:

1. The virtual globe must allow the visualization of subsurface geospatial data as it exists below the Earth's surface.
2. The virtual globe must be hosted in a publicly accessible website with no requirement for special licensing or payments.
3. The virtual globe must be able to support file types with the Keyhole Markup Language such as KML and KMZ which have the file extensions “.kml” and “.kmz” respectively.

Google Earth was not used because of its inability to make the surface imagery transparent without a Google Enterprise account which could not be obtained within the project time-frame. Cesium Globe could not be used because of its lack of support for KML and other OGC standard data types. The World Wind globe was found to meet all the requirements of the project for the selection of a virtual globe. It also offered the ability to develop a virtual globe as both a Java applet in a web page and a Java application run outside of a web browser.

4.2 Development of World Wind Applets and Applications

The applet and application developed in this project demonstrate the ability for the World Wind globe to show subsurface geospatial data as well as communicate with servers and user machines to run data processing routines.

The applet was able to successfully load KML files from user machines as well as from online sources. The application was able to successfully load data from the server and other WMS databases, draw a polyline on the globe surface, and initiate a processing routine on the server through a PHP web request.

4.3 Using ArcGIS and AHGW for Web Processing

ArcGIS with the AHGW plugin was used for data processing routines through Python scripts to automatically run processing tools. Different versions of ArcGIS were found to react differently to web requests to run these Python scripts making them more or less compatible with the scope of this project. ArcGIS version 10.2 was found to be the best fit for running web services through PHP and creating KML files from the processing results.

A process for converting shapefiles from a user machine to a KML file was successfully implemented which also loads the generated KML file into the applet globe.

A process for generating cross sections from a polyline drawn in the application globe was developed. This process successfully initiated a Python script which could run ArcGIS and AHGW tools, however only some of the tools from the AHGW toolbox were able to be used due to licensing restrictions. Despite this setback the ability for the process to take place was demonstrated successfully because the process was initiated and the data about the process was returned to the user on the website.

4.4 Prototype Website Developed

A website was successfully developed which uses HTML for management of web content and JavaScript for hosting the virtual globes through Java WebStart. It also uses PHP scripts for handling web requests to send information to and return information from the server the website is hosted at. This website successfully demonstrates the ability to use the World Wind virtual globe for subsurface data visualization, data discovery, and processing in a cloud-based environment.

REFERENCES

- (2014). "Java 7 Release Highlights." Retrieved April, 28, 2014, from https://www.java.com/en/download/faq/release_changes.xml.
- Aquaveo (2014). "Arc Hydro Groundwater | Aquaveo.com." Retrieved April, 30, 2014, from <http://aquaveo.com/software/ahgw-archydro-groundwater-introduction>.
- Aquaveo (2014). "Arc Hydro Groundwater Tutorials | Aquaveo.com." 2013, from <http://aquaveo.com/software/ahgw-learning-tutorials>.
- Baru, C., et al. (2008). Integrating diverse geophysical and geological data to construct multi-dimensional earth models: the open earth framework. AGU Fall Meeting Abstracts.
- Bell, D. G., et al. (2007). NASA World Wind: opensource GIS for mission operations. Aerospace Conference, 2007 IEEE, IEEE.
- Boschetti, L., et al. (2008). "Using NASA's World Wind virtual globe for interactive internet visualization of the global MODIS burned area product." *International Journal of Remote Sensing* **29**(11): 3067-3072.
- Brodaric, B., et al. (2009). Groundwater Information Network: enabling online access and analysis of Canadian groundwater information. AGU Spring Meeting Abstracts.
- Butler, D. (2006). "Virtual globes: The web-wide world." *Nature* **439**(7078): 776-778.
- Cesium. "Cesium Language (CZML) Guide." 2012, from <https://github.com/AnalyticalGraphicsInc/cesium/wiki/CZML-Guide>.
- Consortium, O. G. (2010). "Inc.(OGC)." Url: <http://www.opengeospatial.org>.
- Crockford, D. (2006). JSON: The fat-free alternative to XML. *Proc. of XML*.
- De Volder, K. (2006). JQuery: A generic code browser with a declarative configuration language. *Practical Aspects of Declarative Languages, Springer*: 88-102.
- Google. "Code Playground." Retrieved 2012, 2012, from <https://code.google.com/apis/ajax/playground/>.

- Java, O. (2014). "Java and Google Chrome Browser." 2013, from <http://java.com/en/download/faq/chrome.xml>.
- Lerdorf, R., et al. (2006). Programming Php, O'Reilly Media.
- Mathers, S. and H. Kessler (2010). "GSI3D. Version 2.6 user manual."
- Meertens, C., et al. (2006). The GEON IDV (Integrated Data Viewer) for data exploration and discovery in the geosciences. AGU Fall Meeting Abstracts.
- Smith, K. and D. Whitman (2013). "Digcrust Home." from <http://digcrust.byu.edu/>.
- Userguide, I. (2009). "Unidata's Integrated Data Viewer." Version 2: u2.
- Wilson, T. (2008). "OGC Keyhole Markup Language, 2.2. 0." Open GIS Consortium.
- Wind, W. "goworldwind.org." 2012, from <http://goworldwind.org/>.
- Workflows, L. and G. Portal "GEON."
- Zukowski, J. (2002). "Deploying software with jnlp and java web start." Retrieved December 3: 2007.

APPENDIX A SETTING UP SOURCE CODE LIBRARIES IN ECLIPSE

The following steps illustrate how to set up the source code libraries and basic application code for the World Wind Java source code.

1. Create a new Eclipse project:
 - a. In the top menu select File>>New>>Project.
 - b. The New Project wizard should pop up. Select Java>>Java Project and click “Next.”
 - c. Assign a project name such as “testWorldWind” and select “Finish.”
2. Create the “lib” folder under this project:
 - a. The project you created should show up in the “Package Explorer” window on the left. Right click on the project name and select “Properties.”
 - b. A new window should pop up for your project properties. From the menu on the left select “Java Build Path.”
 - c. Select the “Source” tab and click the “Add Folder...” then in the new window click “Create New Folder...” and name the folder “Lib” and click “Finish” and “OK” and “OK” in the last remaining window.
 - d. In the “Package Explorer” window on the left expand the contents under your new project by clicking the triangle next to the name. You should see a folder named “src” another named “Lib” and the “JRE System Library.”
3. Put required libraries and DLLs into the “Lib” folder:
 - a. Navigate to the main folder where you have saved the Java source code on your machine.
 - b. Copy the following files onto your clip board: gluegen-rt.jar, gluegen-rt.dll, jogl.jar, jogl.dll, worldwind.jar, jogl_awt.dll, jogl_cg.dll
 - i. For this project the plugin.jar & worldwindx.jar files were also copied but are not necessary for a very basic application.
 - c. Paste all of these files into the “Lib” folder inside of the “Package Explorer” window.
4. Add all JARs in the “Lib” folder into the “Referenced Libraries” tree:
 - a. Expand the “Lib” folder in the “Package Explorer” window and right click on “jogl.jar” then select Build Path>>Add to Build Path.

- b. Repeat this for all of the JAR files in the “Lib” folder one at a time.
 - c. The “Referenced Libraries” tree should appear under your project. Expand it to show all of the JARs you have added.
 - d. Right click on “jogl.jar” in the “Referenced Libraries tree and select BuildPath>>Configure Build Path.
 - e. A new “Properties” dialog box will appear. Select the “Libraries” tab and expand the “jogl.jar” tree.
 - f. Select “Native Library location” and click on the “Edit” button on the right.
 - g. In the new window “Native Library Folder Configuration” click the “Workspace...” button.
 - h. Locate your project in the new window and expand it to show the “Lib” folder. Select the “Lib” folder and click “OK” then “OK” again then “OK” again.
5. Create a package and class and add code:
 - a. In the “Package Explorer” window create a new package by right clicking on the “src” folder and selecting New>>Package.
 - b. Assign a name for your package like “testWorldWind.” You should see your package appear in the “Package Explorer” window under the “src” folder.
 - c. Right click on the new package and select New>>Class. Enter the name “SimplestPossibleExample” then uncheck all options under the question “Which method stubs would you like to create?” and click on “Finish.”
 - i. Some code should pop up in the Eclipse code window stating the package and creating a public class.
 - ii. Delete all of the code other than the first line stating the package.
 - d. Navigate to the main folder of your World Wind source code and navigate from there to ...src/gov/nasa/worldwindx/examples/
 - e. Open the file “SimplestPossibleExample.java” with some sort of text editor.
 - f. Copy the contents of that file other than the first line stating the package.
 - g. Paste it into the code window under the line stating the package and save your changes.
6. Run your code.
 - a. If you have done everything right you should be able to run your code by pressing the green play button in the tool bar on the top.
 - b. If you have named your class something other than “SimplestPossibleExample” you may need to make other adjustments in your code to match the class name you assigned.
 - c. These instructions will work for the applications and applets. You can create a very basic applet by repeating step 5 using the class name “WWJAppletMinimal” and the file in the source code at
...src/gov/nasa/worldwindx/examples/applet/WWJAppletMinimal.java

APPENDIX B CREATING APPLICATIONS AND APPLETS

This appendix shows the code developed to create the application and applet in used in this research project.

B.1 Application Code Examples

This section shows the classes used to develop the application used in this research project. The names of the subsections that follow indicate the name of the Java file class which the code is for.

B.1.a KMLViewer

```
package digcrustApplications;
//imports from KMLViewer
import gov.nasa.worldwind.WorldWind;
import gov.nasa.worldwind.avlist.AVKey;
import gov.nasa.worldwind.retrieve.RetrievalService;
import gov.nasa.worldwindx.examples.ApplicationTemplate;
import gov.nasa.worldwindx.examples.kml.*;
import gov.nasa.worldwindx.examples.util.*;
import gov.nasa.worldwind.util.layertree.*;
import gov.nasa.worldwind.layers.*;
import gov.nasa.worldwind.ogc.kml.*;
import gov.nasa.worldwind.ogc.kml.impl.KMLController;
import gov.nasa.worldwind.render.Offset;
import gov.nasa.worldwind.render.Renderable;
import gov.nasa.worldwind.util.*;
import javax.swing.*;
import javax.swing.filechooser.*;
import javax.xml.stream.XMLStreamException;
import java.awt.*;
import java.awt.event.*;
import java.beans.*;
import java.io.*;
import java.net.URL;
//imports from ExtrudedShapes
import gov.nasa.worldwind.View;
import gov.nasa.worldwind.avlist.AVKey;
import gov.nasa.worldwind.event.*;
import gov.nasa.worldwind.geom.Position;
import gov.nasa.worldwind.layers.RenderableLayer;
import gov.nasa.worldwind.render.*;
```

```

import gov.nasa.worldwind.render.airspaces.*;
import gov.nasa.worldwind.render.airspaces.Polygon;
import gov.nasa.worldwind.util.WWIO;
import gov.nasa.worldwindx.examples.util.ExampleUtil;
import java.io.*;
import java.util.*;
import java.util.zip.*;

public class KMLViewer extends ApplicationTemplate
{
    /**
     *
     */
    public static Boolean addornot;

    public static class AppFrame extends ApplicationTemplate.AppFrame
    {
        protected LayerTree layerTree;
        protected RenderableLayer hiddenLayer;

        protected HotSpotController hotSpotController;
        protected KMLApplicationController kmlAppController;
        protected BalloonController balloonController;

        protected JPanel WebPanel;

        public AppFrame()
        {
            super(true, false, false); // Don't include the layer panel; we're
using the on-screen layer tree.

            // Add the on-screen layer tree, refreshing model with the
WorldWindow's current layer list. We
            // intentionally refresh the tree's model before adding the layer
that contains the tree itself. This
            // prevents the tree's layer from being displayed in the tree
itself.
            this.layerTree = new LayerTree(new Offset(20d, 160d, AVKey.PIXELS,
AVKey.INSET_PIXELS));

this.layerTree.getModel().refresh(this.getWwd().getModel().getLayers());
            this.WebPanel = new JPanel();
            WebPanel.setSize(100, 100);

            // Set up a layer to display the on-screen layer tree in the
WorldWindow. This layer is not displayed in
            // the layer tree's model. Doing so would enable the user to hide
the layer tree display with no way of
            // bringing it back.
            this.hiddenLayer = new RenderableLayer();
            this.hiddenLayer.addRenderable(this.layerTree);
            this.getWwd().getModel().getLayers().add(this.hiddenLayer);

            // Add a controller to handle input events on the layer selector
and on browser balloons.
            this.hotSpotController = new HotSpotController(this.getWwd());

            // Add a controller to handle common KML application events.
            this.kmlAppController = new
KMLApplicationController(this.getWwd());

            // Add a controller to display balloons when placemarks are
clicked. We override the method addDocumentLayer

```

```

// so that loading a KML document by clicking a KML balloon link
displays an entry in the on-screen layer
// tree.
this.balloonController = new BalloonController(this.getWwd())
{
    @Override
    protected void addDocumentLayer(KMLRoot document)
    {
        addKMLLayer(document);
    }
};

// Give the KML app controller a reference to the BalloonController
so that the app controller can open
// KML feature balloons when feature's are selected in the on-
screen layer tree.
this.kmlAppController.setBalloonController(balloonController);

// Size the World Window to take up the space typically used by the
layer panel.
Dimension size = new Dimension(800, 800);
this.setPreferredSize(size);
this.pack();
WWUtil.alignComponent(null, this, AVKey.CENTER);

makeMenu(this);

// Set up to receive SSLHandshakeExceptions that occur during
resource retrieval.
WorldWind.getRetrievalService().setSSLExceptionListener(new
RetrievalService.SSLExceptionListener()
{
    public void onException(Throwable e, String path)
    {
        System.out.println(path);
        System.out.println(e);
    }
});
}

/**
 * Adds the specified <code>kmlRoot</code> to this app frame's
<code>WorldWindow</code> as a new
 * <code>Layer</code>, and adds a new <code>KMLLayerTreeNode</code> for
the <code>kmlRoot</code> to this app
 * frame's on-screen layer tree.
 * <p/>
 * This expects the <code>kmlRoot</code>'s
<code>AVKey.DISPLAY_NAME</code> field to contain a display name
 * suitable for use as a layer name.
 *
 * @param kmlRoot the KMLRoot to add a new layer for.
 */
protected void addKMLLayer(KMLRoot kmlRoot)
{
    // Create a KMLController to adapt the KMLRoot to the World Wind
renderable interface.
    KMLController kmlController = new KMLController(kmlRoot);

    // Adds a new layer containing the KMLRoot to the end of the
WorldWindow's layer list. This
    // retrieves the layer name from the KMLRoot's DISPLAY_NAME field.
    RenderableLayer layer = new RenderableLayer();

```

```

        layer.setName((String) kmlRoot.getField(AVKey.DISPLAY_NAME));
        layer.addRenderable(kmlController);
        String lyrDescription = layer.getValues().toString();
        this.getWwd().getModel().getLayers().add(layer);

        // Adds a new layer tree node for the KMLRoot to the on-screen
layer tree, and makes the new node visible
        // in the tree. This also expands any tree paths that represent
open KML containers or open KML network
        // links.
        KMLLayerTreeNode layerNode = new KMLLayerTreeNode(layer, kmlRoot);
        this.layerTree.getModel().addLayer(layerNode);
        this.layerTree.makeVisible(layerNode.getPath());
        layerNode.expandOpenContainers(this.layerTree);

        // Listens to refresh property change events from KML network link
nodes. Upon receiving such an event this
        // expands any tree paths that represent open KML containers. When
a KML network link refreshes, its tree
        // node replaces its children with new nodes created from the
refreshed content, then sends a refresh
        // property change event through the layer tree. By expanding open
containers after a network link refresh,
        // we ensure that the network link tree view appearance is
consistent with the KML specification.

layerNode.addPropertyChangeListener(AVKey.RETRIEVAL_STATE_SUCCESSFUL, new
PropertyChangeListener()
    {
        public void propertyChange(final PropertyChangeEvent event)
        {
            if (event.getSource() instanceof KMLNetworkLinkTreeNode)
            {
                // Manipulate the tree on the EDT.
                SwingUtilities.invokeLater(new Runnable()
                {
                    public void run()
                    {
                        ((KMLNetworkLinkTreeNode)
event.getSource()).expandOpenContainers(layerTree);
                        getWwd().redraw();
                    }
                });
            }
        }
    });
}

/** A <code>Thread</code> that loads a KML file and displays it in an
<code>AppFrame</code>. */
public static class WorkerThread extends Thread
{
    /** Indicates the source of the KML file loaded by this thread.
Initialized during construction. */
    protected Object kmlSource;
    /** Indicates the <code>AppFrame</code> the KML file content is
displayed in. Initialized during construction. */
    protected AppFrame appFrame;

    /**
     * Creates a new worker thread from a specified <code>kmlSource</code>
and <code>appFrame</code>.

```

```

        *
        * @param kmlSource the source of the KML file to load. May be a {@link
File}, a {@link URL}, or an {@link
        *                               java.io.InputStream}, or a {@link String}
identifying a file path or URL.
        * @param appFrame the <code>AppFrame</code> in which to display the
KML source.
    */
    public WorkerThread(Object kmlSource, AppFrame appFrame)
    {
        this.kmlSource = kmlSource;
        this.appFrame = appFrame;
    }

    /**
     * Loads this worker thread's KML source into a new <code>{@link
gov.nasa.worldwind.ogc.kml.KMLRoot}</code>,
     * then adds the new <code>KMLRoot</code> to this worker thread's
<code>AppFrame</code>. The
     * <code>KMLRoot</code>'s <code>AVKey.DISPLAY_NAME</code> field
contains a display name created from either the
     * KML source or the KML root feature name.
     * <p/>
     * If loading the KML source fails, this prints the exception and its
stack trace to the standard error stream,
     * but otherwise does nothing.
    */
    public void run()
    {
        try
        {
            KMLRoot kmlRoot = this.parse();

            // Set the document's display name
            kmlRoot.setField(AVKey.DISPLAY_NAME, formName(this.kmlSource,
kmlRoot));

            // Schedule a task on the EDT to add the parsed document to a
layer

            final KMLRoot finalKMLRoot = kmlRoot;
            SwingUtilities.invokeLater(new Runnable()
            {
                public void run()
                {
                    appFrame.addKMLLayer(finalKMLRoot);
                }
            });
        }
        catch (Exception e)
        {
            e.printStackTrace();
        }
    }

    /**
     * Parse the KML document.
     *
     * @return The parsed document.
     *
     * @throws IOException if the document cannot be read.
     * @throws XMLStreamException if document cannot be parsed.
    */

```



```

        protected KMLRoot parse() throws IOException, XMLStreamException
        {
            // KMLRoot.createAndParse will attempt to parse the document using
            a namespace aware parser, but if that
            // fails due to a parsing error it will try again using a namespace
            unaware parser. Note that this second
            // step may require the document to be read from the network again
            if the kmlSource is a stream.
            return KMLRoot.createAndParse(this.kmlSource);
        }
    }

    protected static String formName(Object kmlSource, KMLRoot kmlRoot)
    {
        KMLAbstractFeature rootFeature = kmlRoot.getFeature();
        if (rootFeature != null && !WWUtil.isEmpty(rootFeature.getName()))
            return rootFeature.getName();

        if (kmlSource instanceof File)
            return ((File) kmlSource).getName();

        if (kmlSource instanceof URL)
            return ((URL) kmlSource).getPath();

        if (kmlSource instanceof String && WWIO.makeURL((String) kmlSource) !=
null)
            return WWIO.makeURL((String) kmlSource).getPath();

        return "KML Layer";
    }

    protected static void makeMenu(final AppFrame appFrame)
    {
        final JFileChooser fileChooser = new JFileChooser();
        fileChooser.setMultiSelectionEnabled(true);
        fileChooser.addChoosableFileFilter(new FileNameExtensionFilter("KML/KMZ
File", "kml", "kmz"));

        JMenuBar menuBar = new JMenuBar();
        appFrame.setJMenuBar(menuBar);
        JMenu fileMenu = new JMenu("File");
        menuBar.add(fileMenu);

        JMenuItem openFileMenuItem = new JMenuItem(new AbstractAction("Open
File...")
        {
            public void actionPerformed(ActionEvent actionEvent)
            {
                try
                {
                    int status = fileChooser.showOpenDialog(appFrame);
                    if (status == JFileChooser.APPROVE_OPTION)
                    {
                        for (File file : fileChooser.getSelectedFiles())
                        {
                            //add the kml file
                            new WorkerThread(file, appFrame).start();
                            String filePath = file.toString();
                            System.out.println(filePath);
                            /*THIS PART ZOOMS TO THE APX CENTER OF THE KML FILE

                            * ONLY WORKS WITH KML FILES NOT KMZ
                            View view = appFrame.getWwd().getView();

```

```

        double lon =
ReadingFiles.CoordinateGeometry("file", filePath, "lon");
        double lat =
ReadingFiles.CoordinateGeometry("file", filePath, "lat");
        double alt =
ReadingFiles.CoordinateGeometry("file", filePath, "alt");
        view.setEyePosition(Position.fromDegrees(lat, lon,
alt));
        System.out.println(lon + ", " + lat + ", " +
alt);*/
    }
}
}
catch (Exception e)
{
    e.printStackTrace();
}
});

fileMenu.add(openFileMenuItem);

JMenuItem openURLMenuItem = new JMenuItem(new AbstractAction("Open
URL...")
{
    public void actionPerformed(ActionEvent actionEvent)
    {
        try
        {
            String status = JOptionPane.showInputDialog(appFrame,
"URL");

            if (!WWUtil.isEmpty(status))
            {
                //add the kml from URL
                new WorkerThread(status.trim(), appFrame).start();
                String file = status.toString().trim();
                System.out.println(file);
                /*THIS PART ZOOMS TO THE APX CENTER OF THE KML FILE BUT
IT
                * ONLY WORKS WITH KML FILES NOT KMZ
                View view = appFrame.getWwd().getView();
                double lon = ReadingFiles.CoordinateGeometry("url",
file, "lon");
                double lat = ReadingFiles.CoordinateGeometry("url",
file, "lat");
                double alt = ReadingFiles.CoordinateGeometry("url",
file, "alt");
                view.setEyePosition(Position.fromDegrees(lat, lon,
alt));
                System.out.println(lon + ", " + lat + ", " + alt);*/
            }
        }
        catch (Exception e)
        {
            e.printStackTrace();
        }
    }
});

fileMenu.add(openURLMenuItem);
}

public static void main(String[] args)

```

```

        {
            //noinspection UnusedDeclaration
            final AppFrame af = (AppFrame) start("World Wind KML Viewer",
AppFrame.class);
        }
    }
}

```

B.1.b ReadingFiles

```

package digcrustApplications;

import java.io.*;
import java.net.URL;
import java.util.*;

import javax.swing.JOptionPane;

import org.w3c.dom.NodeList;

import com.sun.corba.se.impl.orbutil.graph.Node;
import com.sun.org.apache.xerces.internal.impl.xpath.XPath;

/**
 * This class is created to extract coordinate information from a kml file
 * This class structure assumes that only one type of tag is on any single line
 * it watches for the case when the opening and closing coordinate tags are on
the same line together with the coordinate numbers
 * and also for the case when the the opening and closing coordinate tags are
on separate lines with the coordinate numbers
 * on the lines in between
 * @author D.Whitman
 *
 */
public class ReadingFiles
{
    //each element of CoordinateSets holds all the coordinate information for
a single coordinate tag
    protected static ArrayList <String> CoordinateSets = new
ArrayList<String>();

    public static ArrayList <String> ReadingFiles (String FileOrUrl, String
path) throws IOException
    {
        try
        {
            DataInputStream in = null;
            if (FileOrUrl == "file")
            {
                FileInputStream fstream = new FileInputStream(path);
                in = new DataInputStream(fstream);
            }
            else if (FileOrUrl == "url")
            {
                InputStream input = new URL(path).openStream();
                in = new DataInputStream(input);
            }
            BufferedReader br = new BufferedReader(new
InputStreamReader(in));

            String strLine;
            String front = "<coordinates>";
            String back = "</coordinates>";
            String comma = ",";

```

```

        String whiteFront = "";
        int count = 0; //counts the number of instances where the
front variable is found
        int frontIndex = 0;
        int backIndex = 0;

        boolean junkLine = true; //true for lines where no
coordinate information is found i.e. junk
        while ((strLine=br.readLine()) != null) //loops through
each line of code
        {
            frontIndex = strLine.indexOf(front);
            backIndex = strLine.indexOf(back);
            if(frontIndex != -1) //true if opening coordinate tag
is on this line
            {
                junkLine = false;
                count++;
            }
            if(backIndex != -1) // true if closing coordinate tag
is on this line
            {
                junkLine = true;
            }

            if(frontIndex != -1 && backIndex != -1) //true if
opening and closing tags are both on the same line
            {
                whiteFront = strLine.substring(frontIndex +
front.length(),backIndex); //print substring without tags (only coordinates)
                whiteFront = whiteFront.trim();
                CoordinateSets.add(whiteFront);
            }
            else if (junkLine == false && strLine.indexOf(comma)
!= -1) //true if after opening coordinate tag and has commas (or coordinate numbers)
            {
                whiteFront = strLine.trim();
                CoordinateSets.add(whiteFront);
            }
        }
        in.close();
    } //end try
    catch (Exception e)
    {
        System.err.println("Error: " + e.getMessage());
        JOptionPane.showMessageDialog(null, "Error: " +
e.getMessage());
    } //end catch
    return CoordinateSets;
} //end of main

/**This method returns the approximate center latitude and longitude of
the specified kml file
 * as well as the approximate altitude at which the extent of the data
can be viewed from the
 * center location looking straight down.
 * @param FileOrUrl
 * Specify whether a file or a url is being read with the string
inputs "file" or "url" passed into
 * this argument.
 * @param path
 * Pass in a string equal to "lon", "lat", or "alt" to return the center
latitude, longitude, or
 * viewing altitude respectively.

```

```

        * @param LonOrLatOrAlt
        *     Pass in a string equal to "lon", "lat", or "alt" to return the
center latitude, longitude, or
        *     viewing altitude respectively.
        * @return
        *     Returns a double.
        * @throws IOException
        */
        public static double CoordinateGeometry(String FileOrUrl, String path,
String LonOrLatOrAlt) throws IOException
        {
            ReadingFiles(FileOrUrl, path);

            double centerLon = 0;
            double centerLat = 0;
            double centerAlt = 0;
            double centerReturn=0;
            int countGeometries=0;
            for(int index1=0; index1<CoordinateSets.size(); index1++)
            {
                for(int index2=0;
index2<FindSingleSet(index1,CoordinateSets).size(); index2++)
                {
                    centerLon=centerLon+FindLonLatAlt(index1,index2,0);
                    centerLat=centerLat+FindLonLatAlt(index1,index2,1);
                    countGeometries++;
                }
            }
            centerLon=centerLon/countGeometries;
            centerLat=centerLat/countGeometries;
            double maxDist = 0; //maximum distance from center location to
another location
            double iDist = 0; //instantaneous distance from center location to
instantaneous location
            double iLon = 0; //instantaneous longitude being compared to
centerLon
            double iLat = 0; //instantaneous latitude being compared to
centerLat
            for(int indexA=0; indexA<CoordinateSets.size(); indexA++)
            {
                for(int indexB=0;
indexB<FindSingleSet(indexA,CoordinateSets).size(); indexB++)
                {
                    iLon = FindLonLatAlt(indexA,indexB,0);
                    iLat = FindLonLatAlt(indexA,indexB,1);
                    iDist = Math.sqrt(Math.pow(iLon-
centerLon,2)+Math.pow(iLat-centerLat,2));
                    if (iDist > maxDist)
                    {
                        maxDist = iDist;
                    }
                }
            }
            centerAlt = maxDist*100000/Math.tan(0.402);

            if(LonOrLatOrAlt == "lon")
            {
                centerReturn = centerLon;
            }
            else if (LonOrLatOrAlt == "lat")
            {
                centerReturn = centerLat;
            }
        }
    }

```

```

        else if (LonOrLatOrAlt == "alt")
        {
            centerReturn = centerAlt;
        }

        return centerReturn;
    }
    /**This returns a string containing the Longitude,Latitude,Altitude of
the
        * specified index from the ArrayList CoordinateSets
        */
    public static List<String> FindSingleSet(int index, ArrayList<String>
CoordinateSets)
    {

        String set = CoordinateSets.get(index);
        //each element of Location holds a single lon,lat,alt string from
CoordinateSets

        List<String> SingleSet = Arrays.asList(set.trim().split(" "));
        return SingleSet;
    }
    /**This returns a double representing the latitude, longitude, or
altitude of the
        * specified set of indexes.
        * @param index1
        *     Integer representing the index of coordinates tag within kml file.
        * @param index2
        *     Integer representing the index of a specific coordinate set within
a specific coordinate
        *     tag specified by index1.
        * @param index3
        *     Integer 0,1, or 2 only to represent the index of the longitude,
latitude, or altitude
        *     respectively within a specific coordinate set of a specific
coordinate tag within the
        *     kml file.
        * @return
        *     Returns a double for longitude, latitude, or latitude.
        */
    public static double FindLonLatAlt (int index1, int index2, int index3)
    {

        List<String> SingleSet = FindSingleSet(index1,CoordinateSets);
        String xyz = SingleSet.get(index2);
        List<String> LonLatAlt = Arrays.asList(xyz.trim().split(","));
        String numString = LonLatAlt.get(index3);
        double numLocation = Double.parseDouble(numString);
        return numLocation;
    }
}
} //end of ReadingFiles class

```

B.1.c LineBuilder

```

package digcrustApplications;

import gov.nasa.worldwind.*;
import gov.nasa.worldwind.event.*;
import gov.nasa.worldwind.avlist.*;
import gov.nasa.worldwind.geom.*;
import gov.nasa.worldwind.layers.*;
import gov.nasa.worldwind.render.*;
import gov.nasa.worldwind.util.Logging;

```

```

import gov.nasa.worldwind.util.WWIO;

import javax.swing.*;
import javax.swing.border.*;
import javax.xml.transform.TransformerConfigurationException;

import java.awt.*;
import java.awt.event.*;
import java.beans.*;
import java.io.BufferedWriter;
import java.io.File;
import java.io.FileWriter;
import java.io.IOException;
import java.io.InputStream;
import java.util.*;

/**
 * A utility class to interactively build a polyline. When armed, the class
monitors mouse events and adds new positions
 * to a polyline as the user identifies them. The interaction sequence for
creating a line is as follows: <ul> <li> Arm
 * the line builder by calling its {@link #setArmed(boolean)} method with an
argument of true. </li> <li> Place the
 * cursor at the first desired polyline position. Press and release mouse
button one. </li> <li> Press button one near
 * the next desired position, drag the mouse to the exact position, then
release the button. The proposed line segment
 * will echo while the mouse is dragged. Continue selecting new positions this
way until the polyline contains all
 * desired positions. </li> <li> Disarm the <code>LineBuilder</code> object by
calling its {@link #setArmed(boolean)}
 * method with an argument of false. </li> </ul>
 * <p/>
 * While the line builder is armed, pressing and immediately releasing mouse
button one while also pressing the control
 * key (Ctl) removes the last position from the polyline. </p>
 * <p/>
 * Mouse events the line builder acts on while armed are marked as consumed.
These events are mouse pressed, released,
 * clicked and dragged. These events are not acted on while the line builder is
not armed. The builder can be
 * continuously armed and rearmed to allow intervening maneuvering of the globe
while building a polyline. A user can
 * add positions, pause entry, maneuver the view, then continue entering
positions. </p>
 * <p/>
 * Arming and disarming the line builder does not change the contents or
attributes of the line builder's layer. </p>
 * <p/>
 * The polyline and a layer containing it may be specified when a
<code>LineBuilder</code> is constructed. </p>
 * <p/>
 * This class contains a <code>main</code> method implementing an example
program illustrating use of
 * <code>LineBuilder</code>. </p>
 *
 * @author tag
 * @version $Id: LineBuilder.java 1 2011-07-16 23:22:47Z dcollins $
 */
public class LineBuilder extends AVLListImpl
{
    private final WorldWindow wwd;
    private boolean armed = false;

```

```

private static ArrayList<Position> positions = new ArrayList<Position>();
private final RenderableLayer layer;
private final Polyline line;
private boolean active = false;

/**
 * Construct a new line builder using the specified polyline and layer and
drawing events from the specified world
 * window. Either or both the polyline and the layer may be null, in which
case the necessary object is created.
 *
 * @param wwd the world window to draw events from.
 * @param lineLayer the layer holding the polyline. May be null, in which
case a new layer is created.
 * @param polyline the polyline object to build. May be null, in which
case a new polyline is created.
 */
public LineBuilder(final WorldWindow wwd, RenderableLayer lineLayer,
Polyline polyline)
{
    this.wwd = wwd;

    if (polyline != null)
    {
        line = polyline;
    }
    else
    {
        this.line = new Polyline();
        this.line.setFollowTerrain(true);
    }
    this.layer = lineLayer != null ? lineLayer : new RenderableLayer();
    this.layer.addRenderable(this.line);
    this.wwd.getModel().getLayers().add(this.layer);

    this.wwd.getInputHandler().addMouseListener(new MouseAdapter()
    {
        public void mousePressed(MouseEvent mouseEvent)
        {
            if (armed && mouseEvent.getButton() == MouseEvent.BUTTON1)
            {
                if (armed && (mouseEvent.getModifiersEx() &
MouseEvent.BUTTON1_DOWN_MASK) != 0)
                {
                    if (!mouseEvent.isControlDown())
                    {
                        active = true;
                        addPosition();
                    }
                }
                mouseEvent.consume();
            }
        }

        public void mouseReleased(MouseEvent mouseEvent)
        {
            if (armed && mouseEvent.getButton() == MouseEvent.BUTTON1)
            {
                if (positions.size() == 1)
                    removePosition();
                active = false;
                mouseEvent.consume();
            }
        }
    }

```



```

    }

    public void mouseClicked(MouseEvent mouseEvent)
    {
        if (armed && mouseEvent.getButton() == MouseEvent.BUTTON1)
        {
            if (mouseEvent.isControlDown())
                removePosition();
            mouseEvent.consume();
        }
    }
});

this.wwd.getInputHandler().addMouseMotionListener(new
MouseMotionAdapter()
{
    public void mouseDragged(MouseEvent mouseEvent)
    {
        if (armed && (mouseEvent.getModifiersEx() &
MouseEvent.BUTTON1_DOWN_MASK) != 0)
        {
            // Don't update the polyline here because the wwd current
            // cursor position will not
            // have been updated to reflect the current mouse position.
            // Wait to update in the
            // position listener, but consume the event so the view
            // doesn't respond to it.
            if (active)
                mouseEvent.consume();
        }
    }
});

this.wwd.addPositionListener(new PositionListener()
{
    public void moved(PositionEvent event)
    {
        if (!active)
            return;

        if (positions.size() == 1)
            addPosition();
        else
            replacePosition();
    }
});
}

/**
 * Returns the layer holding the polyline being created.
 *
 * @return the layer containing the polyline.
 */
public RenderableLayer getLayer()
{
    return this.layer;
}

/**
 * Returns the layer currently used to display the polyline.
 *
 * @return the layer holding the polyline.
 */

```

```

public Polyline getLine()
{
    return this.line;
}

/**
 * Removes all positions from the polyline.
 */
public void clear()
{
    while (this.positions.size() > 0)
        this.removePosition();
}

/**
 * Identifies whether the line builder is armed.
 *
 * @return true if armed, false if not armed.
 */
public boolean isArmed()
{
    return this.armed;
}

/**
 * Arms and disarms the line builder. When armed, the line builder monitors
user input and builds the polyline in
 * response to the actions mentioned in the overview above. When disarmed,
the line builder ignores all user input.
 *
 * @param armed true to arm the line builder, false to disarm it.
 */
public void setArmed(boolean armed)
{
    this.armed = armed;
}

private void addPosition()
{
    Position curPos = this.wwd.getCurrentPosition();
    if (curPos == null)
        return;

    this.positions.add(curPos);
    this.line.setPositions(this.positions);
    this.firePropertyChange("LineBuilder.AddPosition", null, curPos);
    this.wwd.redraw();
}

private void replacePosition()
{
    Position curPos = this.wwd.getCurrentPosition();
    if (curPos == null)
        return;

    int index = this.positions.size() - 1;
    if (index < 0)
        index = 0;

    Position currentLastPosition = this.positions.get(index);
    this.positions.set(index, curPos);
    this.line.setPositions(this.positions);
}

```

```

        this.firePropertyChange("LineBuilder.ReplacePosition",
currentLastPosition, curPos);
        this.wwd.redraw();
    }

    private void removePosition()
    {
        if (this.positions.size() == 0)
            return;

        Position currentLastPosition = this.positions.get(this.positions.size()
- 1);
        this.positions.remove(this.positions.size() - 1);
        this.line.setPositions(this.positions);
        this.firePropertyChange("LineBuilder.RemovePosition",
currentLastPosition, null);
        this.wwd.redraw();
    }

    // ===== Control Panel ===== //
    // The following code is an example program illustrating LineBuilder usage.
It is not required by the
    // LineBuilder class, itself.

    private static class WebPanel extends JPanel
    {
        protected static final String BROWSER_BALLOON_CONTENT_PATH =
"WebsiteTest.html";

        private final WorldWindow wwd;

        public WebPanel(WorldWindow wwd)
        {
            this.wwd = wwd;
            this.setSize(200, 200);
            String htmlString = null;
            InputStream contentStream = null;

            try
            {
                // Read the URL content into a String using the default
encoding (UTF-8).
                contentStream =
WWIO.openFileOrResourceStream(BROWSER_BALLOON_CONTENT_PATH, this.getClass());
                htmlString = WWIO.readStreamToString(contentStream, null);
            }
            catch (Exception e)
            {
                e.printStackTrace();
            }
            finally
            {
                WWIO.closeStream(contentStream, BROWSER_BALLOON_CONTENT_PATH);
            }

            if (htmlString == null)
                htmlString =
Logging.getMessage("generic.ExceptionAttemptingToReadFile",
BROWSER_BALLOON_CONTENT_PATH);
        }
    }

    private static class LinePanel extends JPanel
    {

```

```

private final WorldWindow wwd;
private final LineBuilder lineBuilder;
private JButton newButton;
private JButton pauseButton;
private JButton endButton;
private JLabel[] pointLabels;

public LinePanel(WorldWindow wwd, LineBuilder lineBuilder)
{
    super(new BorderLayout());
    this.wwd = wwd;
    this.lineBuilder = lineBuilder;
    this.makePanel(new Dimension(100, 100));
    lineBuilder.addPropertyChangeListener(new PropertyChangeListener()
    {
        public void propertyChange(PropertyChangeEvent
propertyChangeEvent)
        {
            fillPointsPanel();
        }
    });
}

private void makePanel(Dimension size)
{
    JPanel buttonPanel = new JPanel(new GridLayout(2, 2, 5, 5));

    ////////////////////////////////////example of how to add a button
    newButton = new JButton("New");
    newButton.addActionListener(new ActionListener()
    {
        public void actionPerformed(ActionEvent actionEvent)
        {
            lineBuilder.clear();
            lineBuilder.setArmed(true);
            pauseButton.setText("Pause");
            pauseButton.setEnabled(true);
            endButton.setEnabled(true);
            newButton.setEnabled(false);
            ((Component)
wwd).setCursor(Cursor.getPredefinedCursor(Cursor.CROSSHAIR_CURSOR));
        }
    });
    buttonPanel.add(newButton);
    newButton.setEnabled(true);
    ////////////////////////////////////end button example

    pauseButton = new JButton("Pause");
    pauseButton.addActionListener(new ActionListener()
    {
        public void actionPerformed(ActionEvent actionEvent)
        {
            lineBuilder.setArmed(!lineBuilder.isArmed());
            pauseButton.setText(!lineBuilder.isArmed() ? "Resume" :
"Pause");
            ((Component) wwd).setCursor(Cursor.getDefaultCursor());
        }
    });
    buttonPanel.add(pauseButton);
    pauseButton.setEnabled(false);

    endButton = new JButton("End");
    endButton.addActionListener(new ActionListener()

```

```

    {
        public void actionPerformed(ActionEvent actionEvent)
        {
            lineBuilder.setArmed(false);
            newButton.setEnabled(true);
            pauseButton.setEnabled(false);
            pauseButton.setText("Pause");
            endButton.setEnabled(false);
            ((Component) wwd).setCursor(Cursor.getDefaultCursor());
        }
    });
    buttonPanel.add(endButton);
    endButton.setEnabled(false);

    JPanel pointPanel = new JPanel(new GridLayout(0, 1, 0, 10));
    pointPanel.setBorder(BorderFactory.createEmptyBorder(5, 5, 5, 5));

    this.pointLabels = new JLabel[20];
    for (int i = 0; i < this.pointLabels.length; i++)
    {
        this.pointLabels[i] = new JLabel("");
        pointPanel.add(this.pointLabels[i]);
    }
    ////////////////////////////////////THIS IS MY OWN BUTTON
    newButton = new JButton("Get Sections");
    newButton.addActionListener(new ActionListener()
    {
        public void actionPerformed(ActionEvent actionEvent)
        {
            /**This part of the code is to create a url
            which passes lon,lat variables to a PHP page
            * through the URL. Those variables are then
            passed into a python script through the command
            * line. The python script creates a csv file
            which is used in an arcpy script to
            * create a polyline feature class which is
            used to create a GeoSection kml file.
            */
            String coordUrl =
            CoordsUrl.CoordsUrl(positions);
            coordUrl =
            "http://digcrust.byu.edu/lonlatvar.php?" + coordUrl;
            //open up webpage to run the python script
            try {
                java.awt.Desktop.getDesktop().browse(java.net.URI.create(coordUrl));
            } catch (IOException e) {
                // TODO Auto-generated catch block
                e.printStackTrace();
            }
            //end opening up webpage
            System.out.println(coordUrl); //not needed.
            only for debugging purposes.
        }
    });
    buttonPanel.add(newButton);
    newButton.setEnabled(true);
    ////////////////////////////////////END OWN BUTTON

    // Put the point panel in a container to prevent scroll panel from
    stretching the vertical spacing.
    JPanel dummyPanel = new JPanel(new BorderLayout());
    dummyPanel.add(pointPanel, BorderLayout.NORTH);

```

```

        // Put the point panel in a scroll bar.
        JScrollPane scrollPane = new JScrollPane(dummyPanel);
        scrollPane.setBorder(BorderFactory.createEmptyBorder(0, 0, 0, 0));
        if (size != null)
            scrollPane.setPreferredSize(size);

        // Add the buttons, scroll bar and inner panel to a titled panel
        that will resize with the main window.
        JPanel outerPanel = new JPanel(new BorderLayout());
        outerPanel.setBorder(
            new CompoundBorder(BorderFactory.createEmptyBorder(9, 9, 9, 9),
new TitledBorder("Line")));
        outerPanel.setToolTipText("Line control and info");
        outerPanel.add(buttonPanel, BorderLayout.NORTH);
        outerPanel.add(scrollPane, BorderLayout.CENTER);
        this.add(outerPanel, BorderLayout.CENTER);
    }

    private void fillPointsPanel()
    {
        int i = 0;

        for (Position pos : lineBuilder.getLine().getPositions())
        {
            if (i == this.pointLabels.length)
                break;

            String las = String.format("Lat %7.4f\u00B0",
pos.getLatitude().getDegrees());
            String los = String.format("Lon %7.4f\u00B0",
pos.getLongitude().getDegrees());
            pointLabels[i].setText(las + " " + los);
            i++;
        }
        for (; i < this.pointLabels.length; i++)
            pointLabels[i++].setText("");
    }
}

/**
 * Marked as deprecated to keep it out of the javadoc.
 *
 * @deprecated
 */
public static class AppFrame extends KMLViewer.AppFrame
{
    public AppFrame()
    {
        super();

        LineBuilder lineBuilder = new LineBuilder(this.getWwd(), null,
null);
        this.getContentPane().add(new LinePanel(this.getWwd(),
lineBuilder), BorderLayout.WEST);
        this.getContentPane().add(new WebPanel(this.getWwd()),
BorderLayout.SOUTH);
        this.pack();
    }
}

/**
 * Marked as deprecated to keep it out of the javadoc.

```

```

*
* @param args the arguments passed to the program.
* @deprecated
*/
public static void main(String[] args)
{
    //noinspection deprecation
    KMLViewer.start("World Wind Line Builder", LineBuilder.AppFrame.class);
}

```

B.1.d CoordsUrl

```

package digcrustApplications;

import gov.nasa.worldwind.geom.Position;
import java.util.ArrayList;
import java.util.Arrays;
import java.util.List;

public class CoordsUrl
{
    public static String CoordsUrl(ArrayList <Position> positions)
    {
        String coordinates = "";
        String position = null;
        String deg = null;
        int comma = -1;
        for(int i = 0; i < positions.size(); i++)
        {
            position = positions.get(i).toString();
            System.out.println(position);
            //position is the array element at index i converted into a
string
            position = positions.get(i).toString();
            //remove the last character of position ")"
            position = position.substring(0,position.length()-1);
            //remove the first character of position "("
            position = position.substring(1,position.length());
            //find first comma between lat and lon
            comma = position.indexOf(",",0);
            //find degree symbol "°" just in front of comma
            deg = position.substring(comma-1,comma);
            //replace degree symbol with nothing
            position = position.replace(deg, "");
            //remove spaces from front and back in case they are there
            position = position.trim();
            //remove all spaces inside of string
            position = position.replace(" ", "");
            //split the lat,lon,alt into parts of a list
            List<String> LatLonAlt =
Arrays.asList(position.trim().split(","));
            //reorder the string to show: "&lon1=lon&lat1=lat"
            //example: "&lon1=-121.56&lat1=38.22"
            coordinates = coordinates + "&lon" + i + "=" +
LatLonAlt.get(1) + "&lat" + i + "=" + LatLonAlt.get(0);
        }
        coordinates = coordinates.substring(1,coordinates.length());
        //System.out.println(coordinates); //not needed. only for
debugging purposes
        return coordinates;
    }
}

```

B.1.e WMSLayerManager

```
package digcrustApplications;

import gov.nasa.worldwind.util.Logging;
import gov.nasa.worldwind.util.WWIO;
import gov.nasa.worldwindx.examples.WMSLayersPanel;

import javax.swing.*;
import javax.swing.event.*;

import java.awt.*;
import java.beans.*;
import java.io.IOException;
import java.io.InputStream;
import java.net.URISyntaxException;

public class WMSLayerManager
{
    protected static final String BROWSER_CONTENT_PATH = "WebsiteTest.html";

    protected static final String[] servers = new String[]
    {
        "http://neowms.sci.gsfc.nasa.gov/wms/wms",
    };

    protected static class AppFrame extends LineBuilder.AppFrame
//changed from ...extends ApplicationTemplate.AppFrame
    {
        protected final Dimension wmsPanelSize = new Dimension(400, 600);
        protected JTabbedPane tabbedPane;
        protected int previousTabIndex;

        public AppFrame()
        {
            this.tabbedPane = new JTabbedPane();

            this.tabbedPane.add(new JPanel());
            this.tabbedPane.setTitleAt(0, "+");
            this.tabbedPane.addChangeListener(new ChangeListener()
            {
                public void stateChanged(ChangeEvent changeEvent)
                {
                    if (tabbedPane.getSelectedIndex() != 0)
                    {
                        previousTabIndex = tabbedPane.getSelectedIndex();
                        return;
                    }

                    String server = JOptionPane.showInputDialog("Enter
wms server URL");

                    if (server == null || server.length() < 1)
                    {
                        tabbedPane.setSelectedIndex(previousTabIndex);
                        return;
                    }

                    // Respond by adding a new WMSLayerPanel to the
tabbed pane.

                    if (addTab(tabbedPane.getTabCount(), server.trim())
!= null)
                        tabbedPane.setSelectedIndex(tabbedPane.getTabCount() - 1);
                }
            })
        }
    }
}
```



```

        });
        ////////////////////////////////////TOP - add web content tab
        InputStream contentStream =
WWIO.openFileOrResourceStream(BROWSER_CONTENT_PATH, this.getClass());
        String htmlString = null;
        try {
            htmlString =
WWIO.readStreamToString(contentStream, null);
        } catch (IOException e) {
            // TODO Auto-generated catch block
            e.printStackTrace();
        }
        JPanel panel = new JPanel();
        JEditorPane editorPane = new JEditorPane();
        editorPane.setEditable(false);
        editorPane.setContentType("text/html"); //the "text/html"
document is part of the project library
        editorPane.setText(htmlString);
        panel.add(editorPane);
        tabbedPane.add(panel);
        ////////////////////////////////////BOTTOM - add web content tab

        // Create a tab for each server and add it to the tabbed
panel.
        for (int i = 0; i < servers.length; i++)
        {
            this.addTab(i + 2, servers[i]); // i+1 to place all server
tabs to the right of the Add Server and Web Help tabs
        }

        // Display the first server pane by default.

this.tabbedPane.setSelectedIndex(this.tabbedPane.getTabCount() > 0 ? 2 : 0);
        this.previousTabIndex = this.tabbedPane.getSelectedIndex();

        // Add the tabbed pane to a frame separate from the world
window.
        JFrame controlFrame = new JFrame();
        controlFrame.getContentPane().add(tabbedPane);
        controlFrame.pack();
        controlFrame.setDefaultCloseOperation(JFrame.EXIT_ON_CLOSE);
        controlFrame.setVisible(true);
    }

protected WMSLayersPanel addTab(int position, String server)
{
    // Add a server to the tabbed dialog.
    try
    {
        WMSLayersPanel layersPanel = new
WMSLayersPanel(AppFrame.this.getWwd(), server, wmsPanelSize);
        this.tabbedPane.add(layersPanel, BorderLayout.CENTER);
        String title = layersPanel.getServerDisplayString();
        this.tabbedPane.setTitleAt(position, title != null &&
title.length() > 0 ? title : server);

        // Add a listener to notice wms layer selections and tell
the layer panel to reflect the new state.

        layersPanel.addPropertyChangeListener("LayersPanelUpdated", new
PropertyChangeListener()
        {

```

```

        public void propertyChange(PropertyChangeEvent
propertyChangeEvent)
        {
AppFrame.this.getLayerPanel().update(AppFrame.this.getWwd());
        }
    });

    return layersPanel;
}
catch (URISyntaxException e)
{
    JOptionPane.showMessageDialog(null, "Server URL is
invalid", "Invalid Server URL", JOptionPane.ERROR_MESSAGE);
    tabbedPane.setSelectedIndex(previousTabIndex);
    return null;
}
}

public static void main(String[] args)
{
    KMLViewer.start("World Wind WMS Layers", AppFrame.class);
//changed from ApplicationTemplate.start...
}
}

```

B.2 Applet Code Example

This section shows the classes used to develop the application used in this research project. The names of the subsections that follow indicate the name of the Java file class which the code is for.

```

package applets;

//world wind imports for applet
import gov.nasa.worldwind.*;
import gov.nasa.worldwind.avlist.AVKey;
import gov.nasa.worldwind.awt.*;
import gov.nasa.worldwind.geom.Angle;
import gov.nasa.worldwind.geom.Position;
import gov.nasa.worldwind.layers.*;
import gov.nasa.worldwind.layers.Earth.*;
import gov.nasa.worldwind.util.*;
//java imports for applet
import javax.swing.*;
import javax.swing.border.*;
import java.awt.*;
import java.awt.event.*;
//imports from world wind KMLViewer
import gov.nasa.worldwind.WorldWind;
import gov.nasa.worldwindx.examples.kml.KMLApplicationController;
import gov.nasa.worldwindx.examples.util.*;

```

```

import gov.nasa.worldwind.util.layertree.*;
import gov.nasa.worldwind.layers.RenderableLayer;
import gov.nasa.worldwind.ogc.kml.*;
import gov.nasa.worldwind.ogc.kml.impl.KMLController;
import gov.nasa.worldwind.view.orbit.BasicOrbitView;
import gov.nasa.worldwind.view.orbit.OrbitView;
//imports from java KMLViewer
import javax.swing.filechooser.*;
import javax.swing.JFileChooser;
import netscape.javascript.JSObject;
import java.io.*;
import java.net.URL;

public class BasicDemo extends JApplet
{
    /**This adds layers to the layer tree of the applet.
    *The layers added here are added from the World Wind
    *source code.
    */
    private BasicDemo.LayerAction[] layers = new BasicDemo.LayerAction[]
    {
        //IMAGERY LAYERS
        new BasicDemo.LayerAction(new BMNGOneImage(), true),
        new BasicDemo.LayerAction(new BMNGWMSLayer(), true),
        new BasicDemo.LayerAction(new Landsat13WMSLayer(), true),
        new BasicDemo.LayerAction(new MSVirtualEarthLayer(), false),
        new BasicDemo.LayerAction(new StarsLayer(), true),
        new BasicDemo.LayerAction(new SkyGradientLayer(), true),
        new BasicDemo.LayerAction(new OSMMapnikLayer(), false),
        //DATA LAYERS
        new BasicDemo.LayerAction(new LatLonGraticuleLayer(), false),
        new BasicDemo.LayerAction(new NASAWFSPlaceNameLayer(), false),
        new BasicDemo.LayerAction(new CountryBoundariesLayer(), false),
        //TOOLS LAYERS
        new BasicDemo.LayerAction(new ScalebarLayer(), true),
        new BasicDemo.LayerAction(new WorldMapLayer(), true),
        new BasicDemo.LayerAction(new CrosshairLayer(), false),
        new BasicDemo.LayerAction(new CompassLayer(), true),
    };

    //This object represents the source of the kml file being loaded
    //by the addKMLfromURL() method.
    private Object kmlSource1;

    public void init()
    {
        ////////////////////////////////////TOP - Menu bar
        final JFileChooser fileChooser = new JFileChooser();
        fileChooser.setMultiSelectionEnabled(true);
        fileChooser.addChoosableFileFilter(new FileNameExtensionFilter("KML/KMZ File", "kml", "kmz"));

        JMenuBar menubar = new JMenuBar();
        JMenu menuAddKML = new JMenu("Add KML");

        //TOP - Example of how to make a menu item for loading a specific file from a URL
        /*JMenuItem utFile = new JMenuItem(new AbstractAction("Utah Outline")
        {
            public void actionPerformed(ActionEvent actionEvent)

```

```

        {
            try
            {
                String status =
"http://digcrust.byu.edu/digcrustPython/kml/UTOutline.kml";
                //new WorkerThread(status.trim(), appFrame).start();
                addKMLfromURL(status);
            }
            catch (Exception e)
            {
                e.printStackTrace();
            }
        }

    });
    menuAddKML.add(utFile);*/
    //BOTTOM - UTAH outline file
    /**This is the menu item which allows the user to add a KML
    *or KMZ to the applet from a URL directly.
    */
    //TOP - addURL
    JMenuItem addURL = new JMenuItem(new AbstractAction("KML URL")
    {
        public void actionPerformed(ActionEvent actionEvent)
    {
        try
        {
            //http://digcrust.groups.et.byu.net/kml/myUTPointsClamp.kmz
            String status = JOptionPane.showInputDialog(appFrame, "URL");
            if (!WWUtil.isEmpty(status))
            {
                //new WorkerThread(status.trim(), appFrame).start();
                addKMLfromURL(status);
            }
        }
        catch (Exception e)
        {
            e.printStackTrace();
        }
    }
    });
    menuAddKML.add(addURL);

    //the menu items, menus and menu bar all need
    // to be ADDED to something!
    menubar.add(menuAddKML);
    menuAddKML.add(addURL);
    //menuAddKML.add(utFile); //this adds the menu item to load specific file

    setJMenuBar(menubar);
    ////////////////////////////////////BOTTOM - Menu bar

    try
    {
        BasicDemo demo = new BasicDemo();
        appFrame = new BasicDemo.AppFrame(demo.layers, getContentPane());
    }
    ////////////////////////////////////TOP - Load the Virginia geo-sections in before starting
    /*String status2 = "http://digcrust.groups.et.byu.net/kml/VirginiaCrossSections01.kml";
    addKMLfromURL(status2);*/

```

```

//////////////////////////////////////BOTTOM - Load the Virginia geo-sections in before starting
}

catch (Exception e)
{
    e.printStackTrace();
}
}
/**This creates the frame or window for the applet and
 *adds in the container for the layers menu.
 */
public static class AppFrame
{
    protected LayerTree layerTree;
    protected RenderableLayer hiddenLayer;
    protected HotSpotController hotSpotController;
    protected KMLApplicationController kmlAppController;
    protected BalloonController balloonController;

    public final static WorldWindowGLCanvas wwd = new WorldWindowGLCanvas();

    public AppFrame(BasicDemo.LayerAction[] layers, Container c)
    {
        LayerList layerList = new LayerList();

        try
        {
            JPanel mainPanel = new JPanel();
            mainPanel.setLayout(new BorderLayout());
            wwd.setPreferredSize(new Dimension(800, 600));
            mainPanel.add(wwd, BorderLayout.CENTER);

            StatusBar statusBar = new StatusBar();
            statusBar.setEventSource(wwd);
            mainPanel.add(statusBar, BorderLayout.PAGE_END);
            c.add(mainPanel, BorderLayout.CENTER);

            JPanel eastContainer = new JPanel(new BorderLayout());
            {
                JPanel eastPanel = new JPanel(new GridLayout(0, 1, 0, 10));
                eastPanel.setBorder(BorderFactory.createEmptyBorder(9, 9, 9, 9));
                {
                    JPanel layersPanel = new JPanel(new GridLayout(0, 1, 0, 15));
                    layersPanel.setBorder(new TitledBorder("Layers"));
                    for (BasicDemo.LayerAction action : layers)
                    {
                        JCheckBox jcb = new JCheckBox(action);
                        jcb.setSelected(action.selected);
                        layersPanel.add(jcb);
                        layerList.add(action.layer);

                        if (action.layer instanceof TiledImageLayer)
                            ((TiledImageLayer) action.layer).setDrawTileBoundaries(false);

                        if (action.layer instanceof Landsat13WMSLayer)
                            ((TiledImageLayer) action.layer).setDrawBoundingVolumes(false);
                    }
                }
            }
        }
    }
}

```

```

        if (action.layer instanceof USGSDigitalOrtho)
            ((TiledImageLayer) action.layer).setDrawTileIDs(false);
        }
        eastPanel.add(layersPanel);
        eastContainer.add(eastPanel, BorderLayout.NORTH);
    }
}

c.add(eastContainer, BorderLayout.EAST);

Model m = (Model) WorldWind.createConfigurationComponent(AVKey.MODEL_CLASS_NAME);
m.setLayers(layerList);
m.setShowWireframeExterior(false);
m.setShowWireframeInterior(false);
m.setShowTessellationBoundingVolumes(false);
wwd.setModel(m);
}
catch (Exception e)
{
    e.printStackTrace();
}
}

/**
 * Adds the specified <code>kmlRoot</code> to this app frame's <code>WorldWindow</code> as a new
 * <code>Layer</code>, and adds a new <code>KMLLayerTreeNode</code> for the <code>kmlRoot</code>
 * to this app frame's on-screen layer tree.
 * <p/>
 * This expects the <code>kmlRoot</code>'s <code>AVKey.DISPLAY_NAME</code> field to contain a display
name
 * suitable for use as a layer name.
 *
 * @param kmlRoot the KMLRoot to add a new layer for.
 */
protected void addKMLLayer(KMLRoot kmlRoot)
{
    // Create a KMLController to adapt the KMLRoot to the World Wind renderable interface.
    KMLController kmlController = new KMLController(kmlRoot);

    // Adds a new layer containing the KMLRoot to the end of the WorldWindow's layer list. This
    // retrieves the layer name from the KMLRoot's DISPLAY_NAME field.
    RenderableLayer layer = new RenderableLayer();
    layer.setName((String) kmlRoot.getField(AVKey.DISPLAY_NAME));
    layer.addRenderable(kmlController);
    wwd.getModel().getLayers().add(layer);
}

protected static String formName(Object kmlSource, KMLRoot kmlRoot)
{
    KMLAbstractFeature rootFeature = kmlRoot.getFeature();

    if (rootFeature != null && !WWUtil.isEmpty(rootFeature.getName()))
        return rootFeature.getName();

    if (kmlSource instanceof File)
        return ((File) kmlSource).getName();
}

```

```

        if (kmlSource instanceof URL)
            return ((URL) kmlSource).getPath();

        if (kmlSource instanceof String && WWIO.makeURL((String) kmlSource) != null)
            return WWIO.makeURL((String) kmlSource).getPath();

        return "KML Layer";
    }

    static class LayerAction extends AbstractAction
    {
        private Layer layer;
        private boolean selected;

        public LayerAction(Layer layer, boolean selected)
        {
            super(layer.getName());
            this.layer = layer;
            this.selected = selected;
            this.layer.setEnabled(this.selected);
        }

        public void actionPerformed(ActionEvent actionEvent)
        {
            if (((JCheckBox) actionEvent.getSource()).isSelected())
                this.layer.setEnabled(true);
            else
                this.layer.setEnabled(false);

            appFrame.wwd.repaint();
        }
    }

    static
    {
        if (Configuration.isMacOS())
        {
            System.setProperty("apple.laf.useScreenMenuBar", "true");
            System.setProperty("com.apple.mrj.application.apple.menu.about.name", "World Wind Basic Demo");
            System.setProperty("com.apple.mrj.application.growbox.intrudes", "false");
        }
    }
    ///////////////////////////////////////////////////Top - add stuff for API
    public void start()
    {
        // Call javascript appletStart()
        try
        {
            JScript win = JScript.getWindow(this);
            win.call("appletStart", null);
        }
        catch (Exception ignore)
        {
        }
    }

    public void stop()
    {

```

```

// Call javascript appletStop()
try
{
    JSONObject win = JSONObject.getWindow(this);
    win.call("appletStop", null);
}
catch (Exception ignore)
{
}

// Shut down World Wind when the browser stops this Applet.
WorldWind.shutDown();
}

/**
 * Adds a layer to WW current layerlist, before a named layer. Target name can be a part of the layer name
 *
 * @param wwd    the <code>WorldWindow</code> reference.
 * @param layer  the layer to be added.
 * @param targetName the partial layer name to be matched - case sensitive.
 */
public static void insertBeforeLayerName(WorldWindow wwd, Layer layer, String targetName)
{
    // Insert the layer into the layer list just before the target layer.
    LayerList layers = wwd.getModel().getLayers();
    int targetPosition = layers.size() - 1;
    for (Layer l : layers)
    {
        if (l.getName().indexOf(targetName) != -1)
        {
            targetPosition = layers.indexOf(l);
            break;
        }
    }
    layers.add(targetPosition, layer);
}

/**
 * Move the current view position
 *
 * @param lat the target latitude in decimal degrees
 * @param lon the target longitude in decimal degrees
 */
public void gotoLatLon(double lat, double lon)
{
    this.gotoLatLon(lat, lon, Double.NaN, 0, 0);
}

/**
 * Move the current view position, zoom, heading and pitch
 *
 * @param lat    the target latitude in decimal degrees
 * @param lon    the target longitude in decimal degrees
 * @param zoom   the target eye distance in meters
 * @param heading the target heading in decimal degrees
 * @param pitch  the target pitch in decimal degrees
 */
public void gotoLatLon(double lat, double lon, double zoom, double heading, double pitch)

```



```

{
//////////BasicOrbitView view = (BasicOrbitView) this.wwd.getView();
    BasicOrbitView view = (BasicOrbitView) AppFrame.wwd.getView();
    if (!Double.isNaN(lat) || !Double.isNaN(lon) || !Double.isNaN(zoom))
    {
        lat = Double.isNaN(lat) ? view.getCenterPosition().getLatitude().degrees : lat;
        lon = Double.isNaN(lon) ? view.getCenterPosition().getLongitude().degrees : lon;
        zoom = Double.isNaN(zoom) ? view.getZoom() : zoom;
        heading = Double.isNaN(heading) ? view.getHeading().degrees : heading;
        pitch = Double.isNaN(pitch) ? view.getPitch().degrees : pitch;
        view.addPanToAnimator(Position.fromDegrees(lat, lon, 0),
            Angle.fromDegrees(heading), Angle.fromDegrees(pitch), zoom, true);
    }
}

/**
 * test out this method to see if it works in the applet when
 * called from javascript
 */
public void getMessage()
{
    String msg = "You need a message!";
    this.getMessage();
}

public void getMessage(String msg)
{
    JOptionPane.showMessageDialog(null, "MESSAGE: " + msg);
}

/**
 * Set the current view heading and pitch
 *
 * @param heading the target heading in decimal degrees
 * @param pitch the target pitch in decimal degrees
 */
public void setHeadingAndPitch(double heading, double pitch)
{
    BasicOrbitView view = (BasicOrbitView) AppFrame.wwd.getView();
    if (!Double.isNaN(heading) || !Double.isNaN(pitch))
    {
        heading = Double.isNaN(heading) ? view.getHeading().degrees : heading;
        pitch = Double.isNaN(pitch) ? view.getPitch().degrees : pitch;

        view.addHeadingPitchAnimator(
            view.getHeading(), Angle.fromDegrees(heading), view.getPitch(), Angle.fromDegrees(pitch));
    }
}

/**
 * Set the current view zoom
 *
 * @param zoom the target eye distance in meters
 */
public void setZoom(double zoom)
{
    BasicOrbitView view = (BasicOrbitView) AppFrame.wwd.getView();

```

```

        if (!Double.isNaN(zoom))
        {
            view.addZoomAnimator(view.getZoom(), zoom);
        }
    }

    /**
     * Get the WorldWindowGLCanvas
     *
     * @return the current WorldWindowGLCanvas
     */
    public WorldWindowGLCanvas getWW()
    {
        return AppFrame.wwd;
    }

    /**
     * This loads kml documents from a URL
     * @return
     */
    public void addKMLfromURL(String kmlSource1)
    {
        this.kmlSource1 = kmlSource1;
        AppFrame appFrame1 = BasicDemo.appFrame;
        try
        {
            KMLRoot kmlRoot = null;
            int end = kmlSource1.length();
            int start = end - 3;
            String ext = kmlSource1.substring(start, end);
            CharSequence ext2 = kmlSource1.subSequence(start, end);
            CharSequence kml = "kml";
            System.out.println();
            System.out.println(ext);
            if(ext.equals("kml"))
            {
                URL url = WWIO.makeURL(kmlSource1);
                kmlRoot = KMLRoot.createAndParse(url);
            }
            else
            {
                kmlRoot = KMLRoot.createAndParse(this.kmlSource1);
            }
            kmlRoot.setField(AVKey.DISPLAY_NAME, formName(this.kmlSource1, kmlRoot));
            final KMLRoot finalKMLRoot = kmlRoot;
            appFrame1.addKMLLayer(finalKMLRoot);
        }
        catch (Exception e)
        {
            e.printStackTrace();
            JOptionPane.showMessageDialog(null, "Your file did not load properly. Please make  

            sure your file has an extension of '.kml' or '.kmz' and try again."); //message error 1
            JOptionPane.showMessageDialog(null, e.toString()); //message error 2
        }
    }

    /**
     * Get the current OrbitView
     *

```

```

    * @return the current OrbitView
    */
    public OrbitView getOrbitView()
    {
        if (AppFrame.wwd.getView() instanceof OrbitView)
            return (OrbitView) AppFrame.wwd.getView();
        return null;
    }

    /**
     * Get a reference to a layer with part of its name
     *
     * @param layerName part of the layer name to match.
     *
     * @return the corresponding layer or null if not found.
     */
    public Layer getLayerByName(String layerName)
    {
        for (Layer layer : AppFrame.wwd.getModel().getLayers())
        {
            if (layer.getName().indexOf(layerName) != -1)
                return layer;
        }
        return null;
    }

    //////////////////////////////////////Bottom - add stuff for API
    public static BasicDemo.AppFrame appFrame;
}

```

APPENDIX C Putting Applications and Applets Online

The following example command line prompts in this appendix demonstrate how to create a keystore called “WWkeystore” with the alias “mykey” for a JAR file with the file location and name of “C:\JARfiles\WWApp.jar” which can represent the JAR file exported for an application or an applet.

C.1 Creating a Keystore in the Command Line

If necessary direct the command line to the file location: *cd C:\JARfiles*

Create the keystore: *keytool -genkey -keystore WWkeystore -alias mykey*

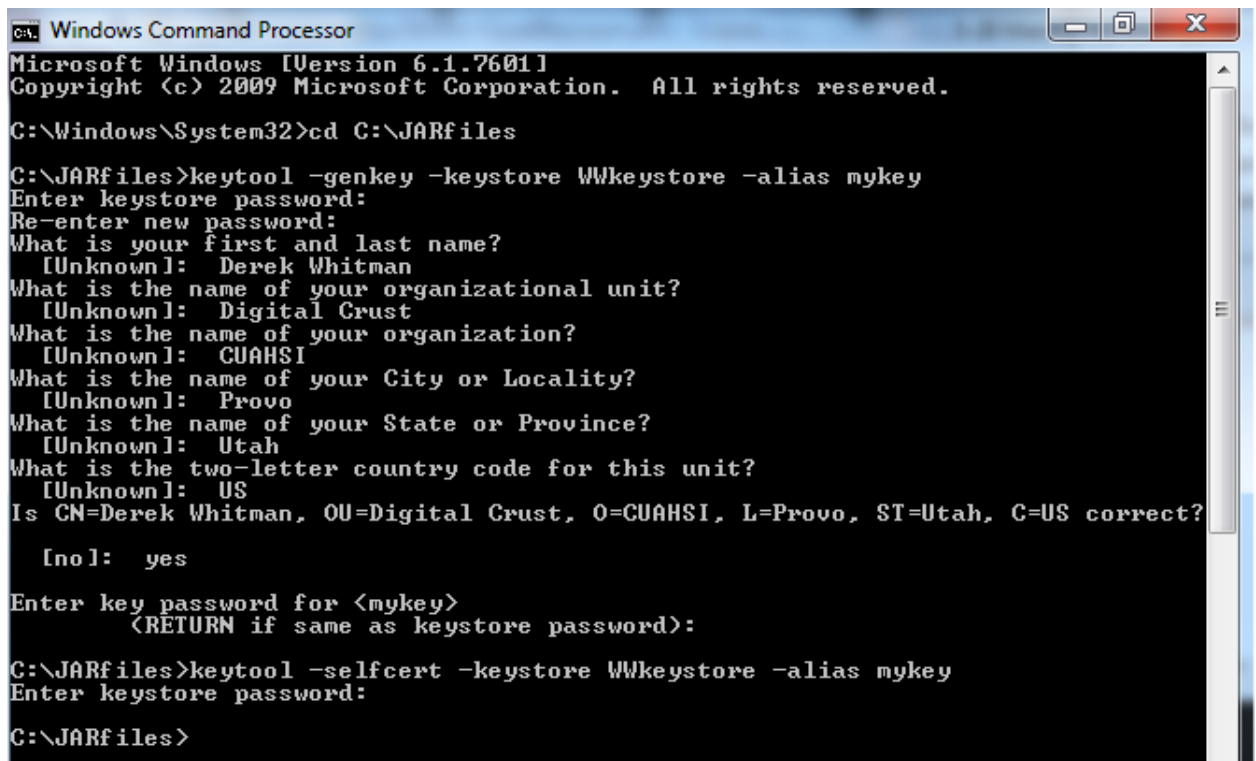
Enter a password then enter it again and continue to input other information as prompted.

The last prompt will ask for another password for the particular alias you have created. You may hit “Enter” to indicate that it will be the same as the password previously entered or you may enter a new password.

Self-certify your keystore: *keytool -selfcert -keystore WWkeystore -alias mykey*

Then enter your password for your keystore. Once your keystore is generated you can use it to sign multiple JAR files without creating a new keystore.

Your command line window should be similar to that shown below in Figure 10.



```
C:\Windows\System32>cd C:\JARfiles

C:\JARfiles>keytool -genkey -keystore WWkeystore -alias mykey
Enter keystore password:
Re-enter new password:
What is your first and last name?
[Unknown]: Derek Whitman
What is the name of your organizational unit?
[Unknown]: Digital Crust
What is the name of your organization?
[Unknown]: CUAHSI
What is the name of your City or Locality?
[Unknown]: Provo
What is the name of your State or Province?
[Unknown]: Utah
What is the two-letter country code for this unit?
[Unknown]: US
Is CN=Derek Whitman, OU=Digital Crust, O=CUAHSI, L=Provo, ST=Utah, C=US correct?
[no]: yes

Enter key password for <mykey>
(RETURN if same as keystore password):

C:\JARfiles>keytool -selfcert -keystore WWkeystore -alias mykey
Enter keystore password:

C:\JARfiles>
```

Figure 10: Example of command line inputs and outputs after creating and self-certifying a keystore

C.2 Signing a JAR file in the Command Line

If necessary direct the command line to the file location: `cd C:\JARfiles`

Sign your JAR file with your keystore: `jarsigner -keystore WWkeystore WWApp.jar mykey`

Enter your password for your keystore again. When finished you should receive a message that the signer certificate will expire within six months. Multiple JAR files can be signed this way with the same keystore. Your command line window should be similar to the one shown in Figure 11 below.

```

C:\ Windows Command Processor
Microsoft Windows [Version 6.1.7601]
Copyright (c) 2009 Microsoft Corporation. All rights reserved.

C:\Windows\System32>cd C:\JARfiles

C:\JARfiles>jarsigner -keystore WWkeystore WWApp.jar mykey
Enter Passphrase for keystore:

Warning:
The signer certificate will expire within six months.

C:\JARfiles>

```

Figure 11: Example of command line window inputs and outupts after signing a JAR file

C.3 Example of JNLP File for Application

```

<?xml version="1.0" encoding="utf-8"?>
<jnlp spec="1.0"
    <!-- Change the codebase to match the URL file location of this JNLP -->
    codebase="http://digcrust.byu.edu/webstartFiles/"
    <!-- Change the href here to match the JNLP name of this file. -->
    href="FinalApplication.jnlp">
    <!--
        Change the content in the information to match your Application's
        description. This does not affect the workability of this code. It
        is only meta data.
    -->
    <information>
        <title>World Wind Java Application</title>
        <vendor>Brigham Young University</vendor>
        <homepage href="http://digcrust.byu.edu/index.html"/>
        <description>World Wind Java Application</description>
        <description kind="short">World Wind Java Application</description>
        <offline-allowed/>
    </information>
    <security>
        <all-permissions/>
    </security>

    <resources>
        <j2se href="http://java.sun.com/products/autodl/j2se" version="1.6+" initial-heap-size="512m"
            max-heap-size="512m"/>
        <property name="sun.java2d.noddraw" value="true"/>
        <!-- Change the href here to match the JAR being referenced -->
        <jar href="FinalApplication.jar" main="true"/>
        <extension name="worldwind" href="http://worldwind.arc.nasa.gov/java/1.5.0/webstart/worldwind.jnlp"/>
        <extension name="worldwindx" href="http://worldwind.arc.nasa.gov/java/2.0.0/webstart/worldwindx.jnlp"/>
        <extension name="gdal" href="http://worldwind.arc.nasa.gov/java/2.0.0/webstart/gdal.jnlp"/>
    </resources>
    <!-- Change the main-class here to match your application's package.class name-->
    <application-desc main-class="digcrustApplications.WMSLayerManager"/>
</jnlp>

```

C.4 Launching Application Through Web Start in HTML

```
<script src=http://www.java.com/js/deploy.js></script>
<script type="text/javascript"></script>
<script>
    //this line gets the URL location of this HTML page
    var dir = location.href.substring(0,location.href.lastIndexOf('/')+1);
    //this line appends the actual file location of the JNLP
    var url = dir + "webstartFiles/FinalApplication.jnlp";
    //this line deploys the applet using JRE 1.6+
    deployJava.createWebStartLaunchButton(url, '1.6.0');
</script>
```

C.5 Example of JNLP File for Applet

```
<?xml version="1.0" encoding="utf-8"?>
<!--
~ Copyright (C) 2013 United States Government as represented by the Administrator of the
~ National Aeronautics and Space Administration.
~ All Rights Reserved.
-->

<!-- Change the href here to match the JNLP name of this file. -->
<jnlp href="BasicDemo9.jnlp">

  <!--
  Change the content in the information to match your Application's
  description. This does not affect the workability of this code. It
  is only meta data.
  -->
  <information>
    <title>Digital Crust World Wind Applet</title>
    <vendor>Brigham Young University</vendor>
    <homepage href="http://digcrust.byu.edu/">
    <description>Subsurface Viewing Globe</description>
    <description kind="short">This is a prototype for the Digital Crust, a virtual globe viewing web site for
subsurface data.</description>
    <offline-allowed/>
  </information>

  <security>
    <all-permissions/>
  </security>

  <resources>
    <java version="1.6+" max-heap-size="512m"/>
    <!-- Change the href here to match the JAR being referenced. -->
    <jar href="BasicDemo9.jar" main="true"/>
    <extension name="worldwind" href="http://worldwind.arc.nasa.gov/java/2.0.0/webstart/worldwind.jnlp"/>
    <extension name="worldwindx" href="http://worldwind.arc.nasa.gov/java/2.0.0/webstart/worldwindx.jnlp"/>
    <extension name="gdal" href="http://worldwind.arc.nasa.gov/java/2.0.0/webstart/gdal.jnlp"/>
  </resources>


  <!-- Change the main-class here to match your applet's package.class name. -->
  <applet-desc main-class="worldwinddemo.BasicDemo"
    name="Layers Demo"
    width="800" height="600">
    <param name="separate_jvm" value="true"/>
  </applet-desc>
</jnlp>
```

C.6 Launching Applet Through Web Start in HTML

```
<div id = "map3d" style="height: 760px; width: 1000px;" onmouseover="noscrolling()" onmouseout="scrolling()">
<script src=http://www.java.com/js/deployJava.js></script>
<script>
  var attributes = {id:'wwjApplet', mayscript:'true', width:'100%', height:'100%'};
  var parameters = [jnlp_href:'webstartFiles/BasicApplet9.jnlp'];
  deployJava.runApplet(attributes, parameters, '1.6'); //runApplet automatically looks for JRE 1.6+
</script>
</div>
```


APPENDIX D Create a Custom Toolbox with AHGW Tools in ArcGIS

The following steps in this appendix describe how to create a custom toolbox in Arc Catalog and add AHGW tools into it for use in Python scripts. This is done using Arc Catalog within the Arc Desktop environment of Arc Map or Arc Scene even though it may be done in Arc Catalog directly.

1. Be sure that both Arc Desktop and AHGW have been installed and Arc Desktop has been licensed correctly and AHGW tools have been added and registered correctly.
2. Open a new or existing map file for Arc Map or Arc Scene, Arc Scene was often used in this project because of the 3D visualization.
3. Within either of these Arc Desktop environments, open the Arc Catalog window by clicking the Catalog Window icon. 
4. Rightclick on the “Home” folder or any folder you would like to create the toolbox inside of then select New>>Toolbox.
5. Type a name for the toolbox and press “Enter” on the keyboard. Then rightclick on the new toolbox and select “Properties...” Under the “General” tab enter an alias for the toolbox in the textbox labeled “Alias.” This alias will be used to refer to the toolbox in the Python script so a shorter is recommended. Click the “OK” button to close the properties window.
6. Rightclick again on the new toolbox and select Add>>Tool... to open the “Add Tool” window. In this window click the ‘+’ signs to expand the toolboxes and the ‘-’ signs to compress them. Navigate to the tools or toolboxes you want to add and check the empty box next to them. More than one tool or toolbox may be selected. Select the “OK” button when finished to close the window.
 - a. Remember that all of the tools and toolboxes selected will be imported with your custom toolbox into the python script so to keep script run times down only select what you need. Also, avoid selecting entire toolboxes unless you will be using everything inside of them.

APPENDIX E Cloud-Based Processing

The code behind the geoprocessing methods including JavaScript, PHP, HTML, and Python are provided here.

E.1 Converting Shapefile

The subsections included here contain examples of code used for the geoprocessing routine. The code provided has been adjusted or minimized to show only what is most pertinent to the geoprocessing routine. The section title naming convention includes the coding language used followed by a ‘-’ then the file name followed by a description in parentheses. The file names are included make reading the code easier because file names are integrated into the code.

E.1.a HTML/JavaScript – index.html (homepage of website)

```
<!--This is the form to upload the shapefile data in the HTML page. -->
<form enctype="multipart/form-data" method="POST" target="blank" action="upload_shp.php" id="uploadForm">
    <input type="file" name="upload[]" multiple="multiple"><br />
    <input type="text" name="user" id="user" value="tempuser"><br />
    Clamp to ground<input type="checkbox" name="clamp" id="clamp" /></a>
    <input type="submit" name="Submit" value="Submit" onclick="setDate()"><br />
</form>

<!-- This is the Javascript initiated by the "Submit" button -->

<!-- Reference Javascript libraries for jquery and custom functions. -->
<script language="Javascript" type="text/javascript" src="jquery-1.10.2.js"></script>
<script language="javascript" type="text/javascript" src="MainJavascript.js"></script>
<script>

//global variables
var kmlrul;
var kmltime;
var oldTime =0;
var date;

function setDate()
{
    //get a fresh time stamp every time the "Submit" button is pressed
```

```

        date = new Date();
        checkText();
    }

function checkText()
{
    //these two variables are for checking the run time of the Python code
    var count = date.getTime();
    var countDif = 0;
    //the Python code creates a text file with a timestamp and a URL pointing to
    //the KML generated. This text file is named with the username in the text box element
    //referenced here.
    var user = document.getElementById('user').value;
    //this references a PHP page which checks the text file URL and time stamp
    var url = http://digcrust.byu.edu/checkText.php?user= + user;

    //This begins the jquery code which runs the PHP page to check the text file
    //every 2 second for up to 2 minutes (120000 milliseconds)
    $.getJSON(url, function(result)
    {
        var results = result[0].directory;
        var resultsArr = results.split(",");
        var url = resultsArr[0];
        var newTime = resultsArr[1];

        //oldTime is the time stamp from the last text file generated by the Python code
        //if the user has already uploaded shapefile data in this session. newTime is the
        //time stamp in the current text file. If a new text file has not been made then the
        //two variables will be equal. If a new text file has been made they will not be equal.

        if (oldTime == 0)
        {
            oldTime = newTime;
        }
        if (newTime != oldTime)
        {
            oldTime = newTime;
            //set runtime of Python code to over 2 minutes to exit the loop of
            //checking the PHP page to check the text file
            countDif = 120001;
            abortTimer();
            var msg = confirm("Your shapefile has been converted....load it now?");
            if(msg == true)
            {
                //this references the applet and a Java method inside of it
                //see section 10.2 of this thesis.
                getWWApplet().addKMLfromURL(url);
            }
        }
        else if (countdif < 120000)
        {
            //use countDif to check the runtime of the Python script
            //run again in 2 seconds if less than 2 minutes
            var date2 = new Date();
            var count2 = date2.getTime();
            countDif = count2 - count;
            tid = setTimeout(checkText, 2000);
        }
    });
}

function abortTimer()

```

```

{
    clearTimeout(tid);
}

</script>

```

E.1.b JavaScript - MainJavascript.js (JavaScript page referenced in the web pages)

```

var theApplet = null;
//this function points to the applet so that its methods can be accessed
function getWWJApplet()
{
    if (theApplet == null)
    {
        theApplet = document.getElementById('wwjApplet');
    }
    return theApplet;
}

```

E.1.c PHP - checkText.php (run by checkText() function in JavaScript)

```

<?php
    header("Content-Type: application/json");

    //The user name is taken from the HTML form text box in JavaScript and passed to this PHP
    //page through the URL. The $_GET method reads it from the URL.
    $user = $_GET['user'];

    //get one line from the text file
    $directory = file_get_contents('http://digcrust.byu.edu/userDirectories/' . $user . '.txt');

    //echo this one line which contains the URL to the KML and a time stamp separated by a comma.
    //This echo is passed into an array which is then read by the JavaScript and passed into the "results" variable.
    echo json_encode(array(array("directory"=>$directory)));
?>

```

E.1.d PHP - upload_shp.php (first visible PHP page with loading gif)

```

<html>
    <!-- this gif was downloaded from http://preloaders.net/en/search/windows%208 -->
    <body onload="MM_preloadImages('loaderwin8white.gif')">
    <div id="loadcontainer">
        <div id="loadcontent">
            
        </div>
        <div id="content" style="position: relative; top: 250px; margin-left: auto; margin-right: auto">
    <!-- This is the beginning of the "content" div tag ----->
    <!-- NOW BEGIN PHP CONTENT WITHIN THIS DIV TAG -->
    <?php
        $accept = array("shp", "shx", "dbf", "prj", "sbn", "sbx", "fbn", "ain", "aih", "ixs", "mxs", "atx", "xml", "cpg");
        $a = 0;
        $user = $_POST['user'];
        $args = null;
        $clamp = $_POST['clamp'];

        echo "Your files have been uploaded. Username is <b>" . $user . "</b><br>";

        for ($i=0; $i < count($_FILES["upload"]["name"][$i]); $i++)
        {
            $ext = end(explode(".", $_FILES["upload"]["name"][$i]));
            echo "<br>File extension:&nbsp;" . $ext;
            //check file type
            if ($_FILES["upload"]["size"][$i] < 999999 and in_array($ext, $accept))
            {
                if ($_FILES["upload"]["error"][$i] > 0)
                {

```

```

        echo "Error: " . $_FILES["upload"]["error"][$i] . "<br>";
    }
    else //uploads the file
    {
        //look for .xml files and only accept if extension is .shp.xml
        if ($ext == "xml" and substr($_FILES["upload"]["name"][$i], -8) !=
        ".shp.xml")

        {
            echo "Your .xml file must have the subextension .shp<br>";
        }
        else
        {
            //Get the temporary file path to upload files to
            $tmpFilePath = $_FILES["upload"]["tmp_name"][$i];
            //make sure we have a temporary filepath
            if ($tmpFilePath != "")
            {
                //Set up new file path
                $newFilePath = "C:/Conversions/uploads/" .

                //Upload the file into the temporary directory
                move_uploaded_file($tmpFilePath, $newFilePath);
                //create $args variable for the Python code using the file with the
                //'shp' extension.
                if ($ext == "shp")
                {
                    $shpFile = $_FILES["upload"]["name"][$i];
                    $name = substr($shpFile,0,strlen($shpFile)-4);
                    $args = $user . " " . $newFilePath . " " . $name;
                }
            }
        }
    }
}

else
{
    echo "<br>INVALID FILE";
}

}

//set variable for clamped to ground in Python arguments
if ($clamp == "on")
{
    $args = $args . " 1";
}
else
{
    $args = $args . " 0";
}

//check to make sure that at least three files have been uploaded
if (count($_FILES["upload"]["name"]) > 2)
{
    $convert = true;
}
?>

<!-- THIS ENDS THE PHP CODE IN THIS DIV TAG -->
<!--
Navigate to the next PHP page using JavaScript so that the gif loaded by JavaScript will load
before
the "window.location.replace()" function is run. Also, wait one second before navigating away just
to make sure that all the JavaScript has loaded.
-->

<script type = "text/javascript">

```

```

        setTimeout(function()
        {
            var yes = <?php echo $convert?>;
            if (yes == 1)
            {
                var arguments = <?php echo “” . $args . “”;?>;
                window.location.replace(‘http://digcrust.byu.edu/SaveToLyrToKMZ.php?args=’ +
arguments);
            }
        }, 1000 //waits one second before executing function in ‘setTimeout’
    );
</script>

</div><!-- This is the beginning of the “content” div tag ----->
----->
</div>

</html>

```

E.1.e PHP - SaveToLyrToKMZ.php (second visible PHP page)

```

<html>
<div id="content" style="width:600px; margin-left: auto; margin-right: auto; position:relative; top:50px">
    <?php
        //get variables from the url
        $args = $_GET['args'];

        //set maximum time for Python script be allowed to run
        ini_set('max_execution_time', 300); //300 seconds = 5 minutes
        ini_set('safe_mode',1);

        //create an array which will be filled by the command line & Python print statements
        $messages = array();
        echo "running python script...<br />";
        $pythonScript = "C:\Python26\ArcGIS10.0\python.exe C:\Conversions\SaveToKMZ10.py " . $args;

        //This is the line that actually executes the python script
        exec($pythonScript, $messages);

        //use the line below to show all print statements instead of using while loop below
        //echo implode("<br />", $messages);

        $j = 5;

        //look for the text “finished” in the last array location of the $messages array to confirm that
        //the Python script completed correctly
        if (end($messages) == "finished")
        {
            while ($j>1)
            {
                echo "<br />" . $messages[count($messages)-$j];
                $j--;
            }

            //echo the download link from the array (second to last print statement from Python)
            $link = $messages[count($messages)-2];
            echo '<br /><br /><a href="" . $link . "">Click here to download your KML.</a>';
        }

        else
        {
            echo "<br />We are sorry but something went wrong. Please try again.<br />";
        }
    }

```

```
?>
</div>
</html>
```

E.1.f Python - SaveToLyrToKMZ.py (conversion process for applet)

```
# -*- coding: utf-8 -*-
# -----
# SaveToLyrToKMZ.py
# Created on: 2013-08-07 11:47:57.00000
# Description: Accepts arguments for shapfile data and converts it to
#             KMZ using arcpy library then unzips to KML
# -----
#Import time module
import time
import zipfile
#get a time stamp
start = time.time()
#
#get system type
import sys
#
#Import arcpy module
import arcpy
import os
arcpy.env.overwriteOutput = True
#
#Get arguments from command line in the form:
args = sys.argv[1:]
username = args[0]
shapePath = args[1]
shapeName = args[2]

if str(args[3]) == "0":
    clamp = "ABSOLUTE"
else:
    clamp = "CLAMPED_TO_GROUND"
#
#set up workspace
arcpy.env.workspace = "C:\\Conversions"
workspace = arcpy.env.workspace
#####

#
# Local variables:
Uploaded = shapePath
LayerFile = workspace + "\\layers\\" + shapeName + "Lyr.lyr"
KMZ = workspace + "\\kmz\\" + shapeName + ".kmz"
kml = "C:\\inetpub\\wwwroot\\kml"
mxd = arcpy.mapping.MapDocument(workspace + "\\Conversions.mxd")
df = arcpy.mapping.ListDataFrames(mxd)[0]
newLayer = arcpy.mapping.Layer(Uploaded)
#
#add uploaded file 'newLayer' to dataframe 'df' in map 'mxd'
arcpy.mapping.AddLayer(df,newLayer)
countLyrs = arcpy.mapping.ListLayers(mxd,"*",df)
for l in countLyrs:
    count = count + 1
myLyr = countLyrs[count-1]
#
```

```

# Process: Save To Layer File
arcpy.SaveToLayerFile_management(myLyr, LayerFile, "", "CURRENT")
#
# Process: Layer To KML
##arcpy.LayerToKML_conversion(LayerFile, KMZ, "1", "false", "DEFAULT", "1024", "96", clamp) #this is the old
one for 10.1 & 10.2
arcpy.LayerToKML_conversion(LayerFile, KMZ, "1", "false", "DEFAULT", "1024", "96")#
#####
#
#now get the kmz to unzip and be renamed
#
#####
#unzip file
zipTest = zipfile.ZipFile(KMZ).extractall(kml)
#
#rename unzipped file
#
#see if the kml file already exists
if os.path.exists(kml + "\\" + shapeName + ".kml"):
    arcpy.Delete_management(kml + "\\" + shapeName + ".kml")
#
os.rename(kml + "\\doc.kml", kml + "\\" + shapeName + ".kml")
#
#delete text file if it exists
if os.path.exists(workspace + "\\" + username + ".txt"):
    arcpy.Delete_management(workspace + "\\" + username + ".txt")
#
#Write to an existing text file to add new KML URL
urldatabase = open("C:\\inetpub\\wwwroot\\userDirectories\\" + username + ".txt", "w")
#Write one line into new text file which contains the url and time stamp separated by a comma.
#This line is read by the "checkText.php" page which is initiated by JavaScript using jquery.
urldatabase.write("http://digcrust.byu.edu/kml/" + shapeName + ".kml," + str(time.time()))
#
#close text file
urldatabase.close()
#
#####
#get a time stamp
end = time.time()
print "<br />Total run time: " + str(end - start) + " seconds" # print runtime
#
#show KML file location
print "clamp: " + clamp
print "Your KML file is at the following location:"
print "http://digcrust.byu.edu/kml/" + shapeName + ".kml"
#
#give a statement for the PHP to check against
print "finished"

```

E.2 Creating a Geosection

The subsections included here contain examples of code used for the geoprocessing routine. The code provided has been adjusted or minimized to show only what is most pertinent to the geoprocessing routine. The section title naming convention includes the coding language

used followed by a ‘-’ then the file name followed by a description in parentheses. The file names are included make reading the code easier because file names are integrated into the code.

This section begins with a PHP page which is opened by the World Wind application developed for this project. The line of code in the application which accesses this PHP page from the application is part of the “LineBuilder” class in Appendix B.1.c.

E.2.a PHP - lonlatvar.php (sends processing request from application to server)

This example shows code from only one PHP page rather than two PHP pages designed to show a waiting screen as in the geoprocessing routine to convert shapefile data to KML described in Section 3.3.1.1.

```
<html>
  <?php
    //php getting python
    ini_set('max_execution_time', 3000); //300 seconds = 5 minutes
    ini_set('safe_mode',1);
    $messages = array();
    $cmdstr = "";
    $a = 0;
    //this loop gets the lat, lon coordinates from the URL and appends them to the
    //$cmdstr variable which is used to pass variables to the Python code in the
    //command line
    while ($_GET["lon" . $a] != NULL)
    {
        $cmdstr = $cmdstr . " " . $_GET["lon" . $a] . " " . $_GET["lat" . $a];
        $a++;
    }

    //Below shows a path to the python executable installed by ArcGIS version 10.2
    //then a space and the name of the Python script file. This file is shown as a relative location
    //because it is in the same file location as this PHP page. If the Python file is else where the full
    //file path should be used. Then a space is given and then the arguments to be passed to the python
    //script all separated by spaces.
    $pythonScript = "C:\Python27\ArcGIS10.2\python.exe PyToCsv.py" . $cmdstr;

    exec($pythonScript, $messages);
    //echo all of the print statements from Python and the command line
    echo implode("<br />", $messages);
  ?>
</html>
```

E.2.b Python - PyToCsv.py (process for creating geosections using AHGW)

```
# -----
# xyToSec3.py
# Created on: 2013-08-29 13:37:47.00000
# (generated by ArcGIS/ModelBuilder)
# Description:
# -----
#import necessary libraries
import os
import sys
```

```

import arcpy
import zipfile
#
#set up workspace
arcpy.env.workspace = "C:\\Horizons100"
workspace = arcpy.env.workspace
print "workspace set to: " + str(workspace)
#####
#Create a CSV file
#####
#create array of coordinates
coord = sys.argv[1:]
#
#create variables for looping and check to make sure the list has
#an even number of items
i=0
j=len(coord)-4
if j%2 == 0:
    #create csv file and headers
    csvfile = open(workspace + "\\linecoordinates1.csv", "w")
    csvfile.write("longitude1,latitude1,longitude2,latitude2");
    #loop through list of coordinates and write into csv file
    while i<=j:
        csvfile.write("\n" + coord[i] + "," + coord[i+1] + "," + coord[i+2] + "," + coord[i+3])
        i+=2
    #close csv file
    csvfile.close()
    print "CSV file created"
    if os.path.exists(workspace + "\\linecoordinates2.csv"):
        print "existing file found"
        os.remove(workspace + "\\linecoordinates2.csv")
        print "Existing file deleted"
    #
    os.rename("C:\\inetpub\\wwwroot\\linecoordinates1.csv",workspace + "\\linecoordinates2.csv")
    print "File was moved"
else:
    print "Must have an even number of arguments to execute."
#####
# Load required toolboxes
arcpy.ImportToolbox(workspace + "\\DigcrustTools.tbx")

#set Overwrite to true
arcpy.env.overwriteOutput = True
print "Overwrite set to: " + str(arcpy.env.overwriteOutput)

# Local variables:
dirDatabase = workspace + "\\Horizons.mdb"
linecoordinates = workspace + "\\linecoordinates2.csv"
SectionLine = dirDatabase + "\\Data\\SectionLine"
UniqueID = dirDatabase + "\\UniqueID"
GeoRasters = dirDatabase + "\\GeoRasters"
GeoSection = dirDatabase + "\\Data\\GeoSection"
xyLine_shp = workspace + "\\xyLine.shp"
shapeName = "TestGeosection"
LayerFile = workspace + "\\layers\\" + shapeName + "Lyr.lyr"
KMZ = workspace + "\\kmz\\" + shapeName + ".kmz"
KML = "C:\\inetpub\\wwwroot\\kml"
print "Local variables set"

# Process: XY To Line
arcpy.XYToLine_management(linecoordinates, xyLine_shp, "longitude1", "latitude1", "longitude2", "latitude2",
"GEODESIC", "",

```

```

"GEOGCS['GCS_WGS_1984',DATUM['D_WGS_1984',SPHEROID['WGS_1984',6378137.0,298.257223563]],PRIMEM['Greenwich',0.0],UNIT['Degree',0.0174532925199433]];-400 -400 1000000000;-100000 10000;-100000 10000;8.98315284119522E-09;0.001;0.001;IsHighPrecision")
    print "finished xy to line"
    # Process: Add Field
    arcpy.AddField_management(xyLine_shp, "HydroID", "DOUBLE", "", "", "", "", "NON_NULLABLE",
"NON_REQUIRED", "")
    print "finished add field"
    # Process: Delete Rows
    arcpy.DeleteRows_management(SectionLine)
    print "section line rows deleted"
    # Process: Append
    arcpy.Append_management("C:\\Horizons100\\xyLine.shp", SectionLine, "NO_TEST", "HydroID \"HydroID\" true
true false 4 Long 0 0 ,First,#,C:\\Horizons100\\xyLine.shp,HydroID,-1,-1;SName \"SName\" true true false 255 Text 0 0
,First,#,HydroCode \"HydroCode\" true true false 255 Text 0 0 ,First,#,FType \"FType\" true true false 255 Text 0 0
,First,#,VertExag2D \"Vertical Exaggeration\" true true false 8 Double 0 0 ,First,#,SHAPE_Length \"SHAPE_Length\" false true
true 8 Double 0 0 ,First,#, "")
    print "append finished"
    # Process: Calculate Field
    arcpy.CalculateField_management(SectionLine, "VertExag2D", "20", "PYTHON", "")
    print "calculation finished"
    # Process: Assign HydroID GW
    arcpy.da.AssignHydroIDGW(UniqueID,SectionLine,"HydroID","OVERWRITE")
    print "hydro id assigned"
    # Process: Rasters to GeoSections
    arcpy.da.RastersToGeoSections(SectionLine,
GeoRasters,"HorizonID",1000,GeoSection,"Clip","Fill","HGUID","OVERWRITE")
    print "rasters to geosections finished"

#Create feature layer
newLayer = arcpy.MakeFeatureLayer_management(GeoSection, workspace + "\\layers\\" + shapeName + "Lyr")
print "feature layer made"
# Process: Save To Layer File
arcpy.SaveToLayerFile_management(newLayer, LayerFile)
print "Layer file saved" # print lyr saved
# Process: Layer to KML
arcpy.LayerToKML_conversion(LayerFile,KMZ,"1","false","DEFAULT","1024","96","ABSOLUTE")
print "GeoSection KMZ created"

#see if a doc.kml file is there and delete it
if os.path.exists(KML + "\\doc.kml"):
    arcpy.Delete_management(KML + "\\doc.kml")
    print "doc.kml File deleted" # print deleted
#unzip file
zipTest = zipfile.ZipFile(KMZ).extractall(KML)
print "File unzipped" # print unzipped

#rename unzipped file
#see if the kml file already exists
if os.path.exists(KML + "\\" + shapeName + ".kml"):
    arcpy.Delete_management(KML + "\\" + shapeName + ".kml")
    print shapeName + ".kml file deleted" # print deleted
print "Passed deletion" # print deleted
os.rename(KML + "\\doc.kml", KML + "\\" + shapeName + ".kml")
print "File renamed" # print renamed
#
print "Click <a href='http://digcrust.byu.edu/kml/' + shapeName + ".kml'>here</a> to download your KML."

```