

University of Redlands

InSPIRe @ Redlands

MS GIS Program Major Individual Projects

Theses, Dissertations, and Honors Projects

12-2009

Custom GIS Tools for Enhancing Wilderness Search and Rescue

Kylie Donia

University of Redlands

Follow this and additional works at: https://inspire.redlands.edu/gis_gradproj



Part of the [Geographic Information Sciences Commons](#), and the [Recreation, Parks and Tourism Administration Commons](#)

Recommended Citation

Donia, K. (2009). *Custom GIS Tools for Enhancing Wilderness Search and Rescue* (Master's thesis, University of Redlands). Retrieved from https://inspire.redlands.edu/gis_gradproj/36



This work is licensed under a [Creative Commons Attribution 4.0 License](#).

This material may be protected by copyright law (Title 17 U.S. Code).

This Thesis is brought to you for free and open access by the Theses, Dissertations, and Honors Projects at InSPIRe @ Redlands. It has been accepted for inclusion in MS GIS Program Major Individual Projects by an authorized administrator of InSPIRe @ Redlands. For more information, please contact inspire@redlands.edu.

University of Redlands

Custom GIS Tools for Enhancing Wilderness Search and Rescue

A Major Individual Project submitted in partial satisfaction of the requirements
for the degree of Master of Science in Geographic Information Systems

by

Kylie Donia

Fang Ren, Ph.D., Committee Chair

Mark Stewart, M.S

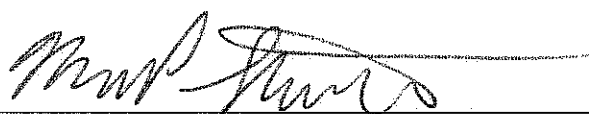
December 2009

Custom GIS Tools for Enhancing Wilderness Search and Rescue

Copyright © 2009

by
Kylie Donia

The report of Kylie Donia is approved.

A handwritten signature in cursive script, appearing to read "Mark Stewart", written above a horizontal line.

Mark Stewart, M.S.

A handwritten signature in cursive script, appearing to read "Fang Ren", written above a horizontal line.

Fang Ren, Ph.D., Committee Chair

December 2009

Acknowledgements

When starting the MS GIS program at University of Redlands you repeatedly hear that you should avoid major changes in your life for the course of the program. I didn't listen, and over the course of this program I got engaged, moved, brought home a little girl dog, got married, started a new business, had major triathlon races, and adopted a baby girl dog. None of this produced regrets; instead, it involved the bulk of the support crew that I would like to thank.

Thank you Mike, my wonderful husband, for all the support you have given me. Taking the dogs out when I'd just gotten to bed, and cooking dinners when even making cereal seemed like work to me: I noticed, and thank you. To my little girl dog Gracie, I don't know how you did it but you sensed when I went back to the computer in the middle of the night, and you always followed me, curled up, and just made sure all was ok. And Annie, my baby puppy, your typing was always helpful, as was your support of my elbow as I tried to work.

To my triathlon coach, Mark Van Akkeren, thank you for making me stronger and faster. You did all the thinking for me, taking the mental stress and distraction from my job as an athlete and letting me just do the physical training work and have fun.

Mom, thanks for editing my papers, no matter how short the notice and how major the updates and changes.

Outside my personal support crew, thanks to Professor Fang Ren for her help throughout the program. I'd also like to thank Molson and its support of James MacKay of ESRI as he helped me learn to program against the geodatabase. I also appreciate the support of a couple other people from ESRI: Noman Nawajish, who shared his spatial analysis and geoprocessing expertise, and Melita Kennedy, who helped me determine what projection to use for my data.

Last, but definitely not least, thanks to Jon Pedder, and Sierra Madre Search and Rescue, for providing a fun project and teaching me about their searching.

Abstract

Custom GIS Tools for Enhancing Wilderness Search and Rescue

by

Kylie Donia

Search and rescue teams are relied on to save the lives of people injured or lost in the wilderness. The Sierra Madre Search and Rescue (SMSR) team needed assistance incorporating a GIS into searches to increase the information available for search decisions and analysis in order to improve their search success rate. This project focused on the home search area of SMSR and created a GIS that integrates into their operations. A custom ArcMap experience was delivered, including custom analysis tools to supplement SMSR's current field procedures, as well as supporting user documentation and training. The system provides information previously not available during searches and also serves to begin familiarizing the team with GIS.

Table of Contents

Chapter 1 – Introduction	1
1.1 Client	1
1.2 Problem Statement	1
1.3 Proposed Solution	2
1.3.1 Goals and Objectives	2
1.3.2 Scope	2
1.3.3 Methods	3
1.4 Audience	3
1.5 Overview of the Rest of this Report	4
Chapter 2 – Background and Literature Review	5
2.1 Search and Rescue Missions	5
2.2 Sierra Madre Search and Rescue’s Current Workflow	6
2.3 Why Add GIS into Search and Rescue?	7
2.4 Existing Search and Rescue GIS Computer Applications	8
2.5 Beyond Software: Case Studies	8
2.6 Additional Tools for Search and Rescue Success	9
2.7 Summary	9
Chapter 3 – Systems Analysis and Design	11
3.1 Problem Statement	11
3.2 Requirements Analysis	11
3.2.1 Functional Requirements	11
3.2.2 Nonfunctional Requirements	12
3.3 System Design	13
3.4 Project Plan	14
3.4.1 Original Project Plan	14
3.4.2 Actual Project Plan	17
3.5 Summary	18
Chapter 4 – Data and Database Design	19
4.1 Structure of the Databases	19
4.1.1 Base Database	19
4.1.2 Search Database	20
4.2 Data Sources	22
4.3 Data Scrubbing and Loading	23
4.4 Summary	23
Chapter 5 – Implementation	25
5.1 The Search and Rescue Map Document	25
5.2 Search and Rescue Toolbar	26
5.3 Custom Sierra Madre Commands and Tools	27
5.3.1 New Search Map File Command	27
5.3.2 Add Team Command	28
5.3.3 Define Search Region Tool	29

5.3.4	Add Clue Tool	30
5.3.5	Add Route Tool	32
5.3.6	Add Beacon Reading Tool.....	33
5.3.7	Beacon Location Calculation Command	36
5.4	Search Route Visibility Analysis Model.....	39
5.4.1	Parameters	39
5.4.2	Processing	39
5.4.3	Resulting information	40
5.5	Documentation	41
5.6	Summary	41
Chapter 6	- Results and Analysis.....	43
6.1	Technical Limitations.....	43
6.2	Feedback from Sierra Madre Search and Rescue.....	43
6.3	An Example Search Using the Search and Rescue GIS.....	44
6.4	Summary	48
Chapter 7	- Conclusions and Future Work	49
Works Cited	51
Appendix A	. Application Source Code	53
A.1	The Toolbar.....	53
A.2	New Search Map File Command	54
A.3	Add Team Command	56
A.4	Define Search Region Tool	57
A.5	Add Clue Tool	59
A.6	Add Route Tool	62
A.7	Add Beacon Reading Tool.....	65
A.8	Beacon Location Calculation Command	69
A.9	Shared Code that Supports Mapping Operations.....	74
A.10	Shared Code that Supports Geodatabase Operations.....	85
A.11	Shared Code that Supports Dialog Interaction.....	94

Table of Figures

Figure 3-1:	Original Project Workflow.	16
Figure 3-2:	Original Project Schedule Chart.	17
Figure 3-3:	Final Project Schedule Chart.	17
Figure 4-1:	The Base Database Design.....	20
Figure 4-2:	The Base Database in ArcCatalog.	20
Figure 4-3:	The Search Database Design.	21
Figure 4-4:	The Search Database in ArcCatalog.	21
Figure 5-1:	The Sierra Madre Search Area MXD Default Table of Contents.....	25
Figure 5-2:	The Navigation Tools on the Search and Rescue Toolbar.....	26
Figure 5-3:	The Printing Tools on the Search and Rescue Toolbar.....	26
Figure 5-4:	The Map Interaction Tools on the Search and Rescue Toolbar.....	26
Figure 5-5:	The New Search Map File Command and the Save Tool on the Search and Rescue Toolbar.....	26
Figure 5-6:	The Add Team Command, Define Search Region Tool, Add Clue Tool, Add Route Tool, Add Beacon Reading Tool, and Beacon Location Calculation Command on the Search and Rescue Toolbar (left to right).	27
Figure 5-7:	The Help Command and the Base Geodatabase Label.....	27
Figure 5-8:	The Set Base Geodatabase Dialog.	28
Figure 5-9:	The Add Team Dialog.	28
Figure 5-10:	The Define Search Region Dialog.	29
Figure 5-11:	An Example Search Region Attribute Table.	30
Figure 5-12:	The Add Clue Dialog set for UTM Coordinate Entry.	30
Figure 5-13:	The Add Clue Dialog set for Latitude/Longitude Coordinate Entry.	31
Figure 5-14:	An Example Clue Attribute Table.	32
Figure 5-15:	The Add Search Route Dialog.	33
Figure 5-16:	An Example Search Route Attribute Table.	33
Figure 5-17:	The Add Beacon Reading Dialog set for UTM Coordinate Entry.....	34
Figure 5-18:	The Add Beacon Reading Dialog set for Latitude/Longitude Coordinate Entry.....	35
Figure 5-19:	An Example Beacon Reading Attribute Table.....	36
Figure 5-20:	Spacing of Beacon Signal Readings as Angles.....	37
Figure 5-21:	The Predict Beacon Location Dialog.....	38
Figure 5-22:	The Search Route Visibility Analysis Model.	40
Figure 6-1:	The New Search MXD.....	45
Figure 6-2:	Example Search Regions.	46
Figure 6-3:	Example Clues.	46
Figure 6-4:	Example Search Routes.	47
Figure 6-5:	Example Beacon Readings and Their Beacon Location Prediction.	48

List of Tables

Table 1.	Functional System Requirements.	12
Table 2.	Nonfunctional System Requirements.	13

List of Acronyms and Definitions

CHM	Microsoft Compiled HTML Help (file extension)
DEM	Digital elevation model
GIS	Geographic information system
MXD	Map document (file extension)
SAR	Search and Rescue
SMSR	Sierra Madre Search and Rescue

Chapter 1 – Introduction

Each year, people get lost or become injured while in the wilderness. Search and rescue teams such as Sierra Madre Search and Rescue (SMSR) are called in to find the missing persons and help them to safety. Before this project, SMSR operated on paper maps and only had available the information they could discern off those maps. The analysis they did to determine where to focus their search efforts did not take into account all the details of the terrain on which they searched. They had a basic idea of where they had covered but had no additional details about the area that was visible over the course of their search routes. Having that information will help the team to rescue people faster and even to save more lives.

To correspond to that need, this project created a simple GIS experience for the team. It included tools for the team to perform spatial analysis, using their search data along with the terrain to generate information previously unavailable during searches.

This chapter includes an introduction to the client. It then discusses the problem of the client that this project addressed. The proposed solution is then presented, including the project goals, the scope, and the methods used to create the solution. Finally, the audience targeted with this project is identified and an overview of the structure of this report is presented.

1.1 Client

The client for this project was Sierra Madre Search and Rescue (SMSR). The team was founded in 1951 as the first mountain rescue team in California. Then and now, it is a group of volunteers who can be called upon to help lost, injured, or missing people in the wilderness. In recent years, the team of about 24 people has performed an average of one search per week. In 2008, they participated in 50 searches. These searches took place mainly within a couple hours driving of their base location in Sierra Madre, California. However, they are also called to serve anywhere in California, Arizona, and Nevada, and on occasion even anywhere in the United States, or potentially the world.

Today the team is under the jurisdiction of the Sheriff's department. The contacts for this project were: Arnold Gaffrey, Senior Operations Leader; Art Fortini, Team President; and Jon Pedder, a team member who initiated the investigation into what GIS can do to make the team more successful.

1.2 Problem Statement

SMSR is tasked with finding people who are lost in the wilderness. Operations can be as simple as a rescue or short search, where the team walks down the trail to an injured person and helps that person out of the wilderness. Operations can also be as complicated as a major search, involving multiple reevaluations of the search area and the team's search focus. Before this project SMSR tracked their search progress and plans on paper maps. Paper maps are sufficient for a rescue or short search, but not for a major search. During a major search, it is difficult for the team to accurately analyze and refocus their searching when working on paper maps because multiple maps have to be created to look

at different aspects of the search, and the impact of the terrain on their search information is not obvious. Therefore, SMSR wished to develop GIS tools to present and analyze search information.

However, SMSR does not have previous GIS experience and did not know how to best manage the data that they already had or the data that is created during their search efforts. They did not know how to make use of the analytical power of a GIS to learn more about the search as it progresses. Therefore, to fulfill their need, a simple GIS application that allows them to gather additional search information without disrupting their existing workflow was required to enable them to carry out rescue tasks more effectively.

1.3 Proposed Solution

Sierra Madre Search and Rescue (SMSR) needed a tool to help them find missing people in the wilderness. This project focused on the home search area of SMSR, introducing GIS into their existing workflow.

1.3.1 Goals and Objectives

This project aimed to develop a workflow and toolset for SMSR to begin integrating GIS into their workflow. The product enables the SMSR team to use the analytical power of a GIS to provide additional information to their major searches, supporting their ability to assist people lost in the wilderness. The goal was not to change how SMSR operated, but to introduce GIS into their existing workflow. Specifically, the project had the following objectives:

1. Design a simplified GIS experience that lets SMSR get information about their searches. This was done by providing a map document (MXD) with a toolbar and some custom tools for ArcMap.
2. Provide a collection of user documentation to facilitate use of their customized GIS experience.
3. Train the SMSR team in the use of their GIS.

1.3.2 Scope

SMSR can be called to assist anywhere in the world, although the vast majority of their searches are close to their home base in Sierra Madre, CA. The searches can involve from ten to hundreds of searchers with geotechnology experience ranging from no GIS knowledge to professional GIS experience. Available technology at a given operation may be a single laptop or a mobile lab of machines, radios, and other electronic devices that can provide geographic coordinates.

To keep this project manageable in a single year, the focus was on SMSR's home area and search team. SMSR is based in Sierra Madre, CA, and they are the main search and rescue team for the nearby San Gabriel Mountains. The team includes about 24 members, none of whom has extensive GIS training or knowledge. In the field they use clipboards primarily but also have a laptop available. In most searches they have a team assigned to stay at their home base where they have a desktop complete with ArcView; however, some searches are managed from the field in remote locations.

Data acquisition was not the focus of this project. The prototype needed some basic topographic data of the San Gabriel Mountains near Sierra Madre, as well as other base data for that area. Since SMSR members have an extensive knowledge of their home area, the GIS is not to teach them the area or guide the searchers as they are in the field and searching. Instead the focus was to provide additional information about the search they are on by giving details that their paper maps cannot provide.

The final deliverables, including an ArcMap document (MXD), a custom ArcMap toolbar, and documentation on the tools, were handed off to SMSR with the completion of both this project and their training in the system.

1.3.3 Methods

To create the custom ArcMap experience for SMSR, an MXD and a custom Search and Rescue toolbar were created. In addition, analysis tools were provided for the team. The MXD was authored in ArcMap using basic mapping design principles.

The toolbar, and thus customization of ArcMap, was created using C# and objected-oriented programming. The toolbar makes use of a number of ArcObjects, the same software components that form the basis for the ArcMap application. In particular, the application required heavy use of the ESRI.ArcGIS.Geodatabase, ESRI.ArcGIS.DataSourcesGDB, and ESRI.ArcGIS.DataSourcesRaster assemblies, containing ArcObjects that provide interaction with the geodatabase—including creating, data access, and data modification. In order to do the visibility analysis, a geoprocessing model was created with ModelBuilder, and geoprocessing tools included in ArcToolbox were executed. The tools used are available through the ESRI.ArcGIS.AnalysisTools, ESRI.ArcGIS.SpatialAnalysisTools, and ESRI.ArcGIS.DataManagementTools assemblies. The ESRI.ArcGIS.Geoprocessing and ESRI.ArcGIS.Geoprocessor assemblies provided methods for executing both the model and the built-in tools. To provide customization of the symbolization and display on the map, the ArcObjects available in the ESRI.ArcGIS.Carto and ESRI.ArcGIS.Display assemblies were used. Classes for functionality to support mapping, geodatabase operations, and dialog functionality were created to optimize code reuse. The forms were created with a standard Windows look and feel.

Information that the team didn't have available on their paper maps was calculated with custom tools in the GIS application. For example, to determine what was in sight during a search route, visibility analysis can be performed, along with intermediate buffering. Predicting the beacon locations can also be conducted with visibility analysis, since the transmission signals are line of sight.

1.4 Audience

The intended audience of this paper is a reader with either a background in search and rescue looking to bring GIS into their own organization or a reader with a background in GIS and an interest in assisting a search and rescue team. The reader will benefit from this report by seeing how GIS was introduced to and now assists Sierra Madre Search Rescue, and by providing the reader information for their own integration of GIS and SAR.

1.5 Overview of the Rest of this Report

This report details the entire lifecycle of this project. Background information and literature review are in Chapter 2. Chapter 3 contains the system analysis and design, and information on the data and the design of the geodatabase are discussed in Chapter 4. Implementation of the system, including the geoprocessing tool designs and implementations, is included in Chapter 5. For the results of the project, refer to Chapter 6. This report is concluded with the project summary and the future research as described in Chapter 7.

Chapter 2 – Background and Literature Review

When people get lost or injured in the wilderness, search and rescue (SAR) teams are called upon to save them. The faster the team locates the person and gets them the needed help, the more likely the victim will survive. There is nothing novel about this concept, nor about the approach of using GIS to improve the information available to the searchers.

Yet during a search the rescuers are stressed, physically and mentally. A system that does not integrate into their existing workflows and structure will not be successful. Before implementing a system for the team it was necessary to develop a high-level understanding of search and rescue, and then a more detailed understanding of how the Sierra Madre Search and Rescue team in particular operates their searches. To avoid mistakes already made and duplication of research, it was important to be familiar with past work done in GIS applications focused on search and rescue. To create a system that the team could use in the foreseeable future, it was also important to keep in mind the current cutting edge devices and tools available in the search and rescue field. All of these considerations enabled creation of the best GIS for SMSR.

This chapter opens defining the missions that Search and Rescue (SAR) teams undertake, and then detailing the current workflow that SMSR follows. From there the utility of GIS in SAR is examined, and existing SAR GIS applications are evaluated. Some case studies of GIS use in SAR are presented, followed by other tools that can be incorporated into SAR.

2.1 Search and Rescue Missions

SAR teams have one goal: saving lives. But there are a few types of missions (operations) they undertake to reach this goal. These are differentiated mainly by how well the location of the victim is known: a rescue, a short search, and a long search. In a rescue, someone is hurt or stranded, but their location is known. This type of SAR is usually completed in a few hours. A rescue involves very little mapping unless it is a complicated extraction (such as airlifting) or is in an area unfamiliar to the search team. Generally maps and global positioning systems (GPS) devices are used to follow a path to the victim's location, where the searchers determine how to help the victim. Sometimes the victim can be helped out on foot while at other times the terrain dictates that a helicopter is needed.

In a short search, the missing person is overdue from a trip along a known route or is lost and calls for help on a cell phone. A short search is also completed fairly quickly since the team has a good idea where to start looking. A map and GPS devices are used to keep a record of where the team searches in case the person is not found at the location predicted by the known route or based on the information in the phone call.

In a long search, someone is overdue from an outing and only their car's location or their drop-off location, or occasionally only the area which they frequent, is known. A long search, often called a major search, can be a multi-day effort as SAR teams are sent in all directions to find clues as to which way the person went. Major searches often involve multiple SAR teams working together to find the victim. As they search, they use more involved mapping than is used in a rescue or short search. Maps are used to track

the search area, both planned and actual, and to identify focus areas for each group of the team to search. Each group reports their findings, including geographic coordinates where appropriate, and reports the chance that they would have seen the victim if located in their search area. If further searching is needed, analysis of all information gathered so far is examined on a map to reevaluate where to focus efforts in the next round of searching.

A complication arises since any operation can escalate into a major search. Any workflow needs to be able to transform along with the mission of the SAR team. Last year in Green Valley Lake, CA, a man called 911 after becoming disoriented in dense fog. This was first classified as a short search, and the SAR team expected a relatively straightforward operation. Yet when the teams arrived in the field, they were unable to locate the man, and soon a major search was underway. Unfortunately, even with the assistance of a GIS system, they were unable to find him in time (DigitalGlobe, 2008; Patterson, 2008a; Patterson, 2008b).

2.2 Sierra Madre Search and Rescue's Current Workflow

The best way to get an understanding of how a major search operates is to experience one. SMSR invited the author to one of their training days. The day started with a phone call to the station, and a report that aircraft passing through the area were receiving emergency beacon transmissions, indicating a downed aircraft in the area. Right away, the operations leader knew that it was going to be a big search effort, as there were no reports of where the crash might have been or the route the aircraft had been following. The search teams were called in, and this joint training day brought together members of Sierra Madre Search and Rescue Team, San Dimas Mountain Search and Rescue, and Altadena Mountain Rescue Team. The teams gathered at the Sierra Madre Search and Rescue Station where the control center for the training event was located.

The group was divided into search teams, and assignments were handed out. Teams were provided with a paper map with a highlighted planned search route on it and were tasked to go to various high points around the valley. There they took readings on the emergency beacon signal, including the direction the strongest signal was coming from, to determine the area where the beacon itself was located and where the search for the downed aircraft would focus. While they were in transit, satellite information came in that provided proposed coordinates of where the beacon had landed, which are usually accurate to within a couple miles. This helped refocus the teams taking readings where necessary. It also informed the teams that in fact there were two separate beacons they should be looking for, and not a single crash.

Beacon signal readings were called back into the control center, including the direction from which the signal was the strongest, the location at which the reading was taken, and the signal strength. This information was plotted on a paper map in the command center with the beacon signal paths drawn to see where they were intersecting. There were a couple stumbling points here: (1) the team was trying plot on their computer, but while one person plotted, the other data wasn't visible, so the plotted data wasn't initially used in the search decisions, and (2) the reports coming in were sometimes given in terms of magnetic north, and sometimes true north, due to different operating standards in the different SAR teams.

The plotted data helped to redirect the search teams, and over the radio new tasks were sent out. People were sent to the areas where the beacons, and thus the crash sites, were predicted to be located. While searching there, the teams radioed back in indications that they were nearing the crash site or its victims, including conversations with people they met who had heard possible crash noises, footprints and other clues found on the ground.

Over the course of the searching, both crash sites were found along with three of the victims. A fourth victim had wandered off from the crash site, and he was not found in the course of the exercise, although when the weekend training was called to a close the search team was getting closer and closer to him.

2.3 Why Add GIS into Search and Rescue?

As wilderness activities are increasing in popularity, with hiking seeing a 46% increase from 1994 through 2002, the number of lives that SAR can potentially save also rises (Ela, 2004). The workflow that SAR teams traditionally use inherently contains uncertainty as shown in the example in the previous section. Using paper maps to identify terrain characteristics and narrow the region victims are potentially located in contributes to the uncertainty. In addition, the traditional way of conducting searches has limitations.

“Outdated maps and a disconnect between documented and planned search efforts also reduce the efficiency in SAR operations” (Ferguson, 2008). Well-executed operations, with wisely deployed resources, have a higher success rate and are completed faster. Streamlined searches also accrue fewer expenses (Abi-Zeid and Frost, 2005). As reported by Adams et al. (2007), faster operations are also safer for the rescuers, who while searching face the same risks as the victim. Enhanced communication during the search and enabling the command post to keep track of the rescuers also further protect the SAR team members (Ferguson, 2008; Patterson, 2008a; Patterson, 2008b; Sayda, Wittmann, Kandawasvika, and Wang, 2005).

Despite challenges, a GIS solution holds promise and merits development to fit the workflow of a particular agency. SAR land search efficiency can be greatly increased by having a method to create maps of searched areas, solving “an issue that is present in most land SAR operations: the inability of searchers to search in the assigned area or to inaccurately represent the area that was searched” (Ferguson, 2008). A GIS would also facilitate integration of data from various sources while “it provides a mechanism for viewing, querying, and analyzing this data, thus creating useful information to assist in decision making and resource management” (Ferguson, 2008). When working on paper maps, search and rescue teams limit the information they have available during their searches. The details of the terrain are obscured, and the team has to create multiple maps, a time consuming process, to view different combinations of the data gathered during their search. However, a GIS can automate the calculation of required information and make the map creation and analysis much easier and faster. In addition, a GIS helps to keep the search maps and operations synchronized. With these benefits, searches done with GIS support will have better success rates and be safer for the searchers.

2.4 Existing Search and Rescue GIS Computer Applications

Using GIS to make SAR operations more efficient and successful is not a novel approach. There are a number of computer applications that target GIS use for SAR. They include water, aerial, and land search methodologies. SearchMaster and CaseMaster, respectively, are the mapping and database of case information applications implemented for the Canadian Rescue Coordination Centers by Payette and Wood (1997). Based on ArcView 3.0, the system uses scripts to facilitate using map data during search operations, incorporating both raster and vector maps. SARPlan, developed for the Canadian National Search and Rescue, focuses on search theory, calculating the probabilities of success with different searches, allowing different searches to be analyzed and finding “near-optimal solutions” for resource distribution and searches (Abi-Zeid and Frost, 2005). Similarly, the application developed by Soylemez and Usul (2006) generates probability maps based on a variety of inputs and uses GIS to generate both search areas and routes to reach them. Another application—the SARPA application—presented by Rodriguez-Rodriguez and Aleman-Flores (2001) is a bit different in that it is designed to aid the user in individual tasks that are a part of the regular workflow. SARPA provided a configurable environment with a system where the users would be helped in executing multiple smaller tasks instead of automating a complete task. In addition to the applications developed for searches on the land there are applications that were developed for searchers on water. For example, a computer application created by Howlett and Spaulding (2004) focuses on water-rescue for the US Coast Guard.

2.5 Beyond Software: Case Studies

While most of the applications presented in the previous section focus on what the computer can provide to a search and rescue operation, the case studies on SAR using GIS in practice stress a combination of GIS techniques and user experience when incorporating GIS into an existing workflow.

For example, one study used a combination of RedZone software, ArcGIS, GPS-enabled radios, and technicians in a mobile mapping center (Patterson, 2008a; Patterson, 2008b). “The maps helped determine where to concentrate the search for Christy [the victim], pinpoint the area that had already been covered by searchers to avoid retracing the same routes, and brief Christy’s family about the progress of the search” (Patterson, 2008a). In the same search, GIS was used to provide documentation in case of lawsuits (DigitalGlobe, 2008). The teams had a complete record of where they had gone and the information behind their decisions.

An indicator of how successful an emergency response GIS application will be in practice is how well it fits into the organization making use of it (Rodriguez-Rodriguez and Aleman-Flores, 2001; Cambell and Masser, 1995; Gilfoyle and Thorpe, 2004). This is complicated since SAR teams often combine forces (Patterson, 2008a; Patterson, 2008b; Johnson, 2003). Any application used in the field must be unique to the workflow of the searching organization, while also being intuitive enough that members of other SAR teams and other assisting organizations are able to contribute information without their searches being compromised (National Research Council of the National Academies, 2007; Radke, Cova, Sheridan, Troy, Lan, and Johnson, 2000). “The vast

majority of first responders (such as police, fire, emergency medical personnel) is not that familiar with GIS, nor are they likely to use these systems in the immediate response or rescue phase” (Cutter, 2003). Therefore, when developing a GIS for a particular SAR organization, it is important to adapt the developed GIS application to the existing search workflow and provide the necessary training to the crew.

2.6 Additional Tools for Search and Rescue Success

GIS is not the only area being researched to enhance SAR success rates. The use of “camera-equipped mini unmanned aerial vehicles” (Goodrich, et al., 2008) has potential to provide search information over larger areas of ground without requiring excessive manpower to operate. Instead of having human teams cover all of the ground in searches over large regions, the robotic searchers can be used to cover the area.

GPS devices have utility outside that of simply tracking the routes searchers are taking; they can also be useful when GPS receivers are carried by the missing persons, especially in the case of avalanche victims. Investigation into the use of GPS beacon devices to locate avalanche victims has found that a GPS beacon device under 55cm of snow provides a very accurate and usable signal, and that an accurate reading can be obtained from a device buried under up to a meter of compacted snow (Stepanek and Claypool, 1997).

A service, PARAMOUNT, that combines information and support for hikers, including emergency SAR transmissions, has also been developed (Sayda et al., 2005). It is a Location Based Service that provides map visualization, routing assistance, up-to-date information about the local area, and the ability to make emergency calls. Hikers are provided with a mobile client to assist them while hiking. SAR teams have both a mobile and a command center application. In the mobile client they have similar features to the hikers, but also the ability to locate other hikers using the system. In the command center application the SAR teams and search information, including search history, are presented.

2.7 Summary

Even after SAR teams respond, 3-6% of the cases result in deaths (Adams et al., 2007; Hung and Townes, 2007; Heggie and Heggie, 2009). Unfortunately, a GIS doesn’t guarantee a search’s success; unsuccessful searches have also incorporated GIS (DigitalGlobe, 2008; Patterson, 2008a; Patterson, 2008b; Payette and Wood, 1997). Yet it has been repeatedly noted that a GIS contributes information that enables searchers to make better decisions and have safer searches. A variety of SAR applications have been created, each noting it is crucial that the application integrates into the organization using the system. Successful systems combine the experience of the searchers with the additional information from the analyses.

With this in mind, the system for Sierra Madre Search and Rescue was created with a focus on the particulars of their organization. It is an application that fits how they currently operate, supplementing the information available during searches without requiring the team to greatly change their operations. It automates parts of their workflow previously done on paper maps, and contributes additional information (such as visibility analysis results) to the search.

Chapter 3 – Systems Analysis and Design

Sierra Madre Search and Rescue was operating on paper maps. While this kept the team on familiar ground, it also limited the information available to them as they searched. A GIS benefits the team by performing analysis that couldn't reasonably be performed using paper maps, thus providing information that would otherwise not be part of their search decisions. However, while the team is on a search, they are in a high-stress situation and do not need additional distractions and processes to focus on. While initially this project included developing a workflow for GPS data collection and use, it was too much of a deviation from SMSR's existing operating procedures and was removed from the plan. The focus transitioned to creating a simple, relatively easy to use GIS that integrates into the operations of the team. A map document (MXD) with a simplified user experience and custom tools fills these needs.

This chapter defines the problem that this project addressed and the requirements for the solution. The project plan outlines the tasks and the schedule for completing the project, both in original form and the updated, and followed, plan.

3.1 Problem Statement

Sierra Madre Search and Rescue (SMSR) previously managed searches on paper maps. However, this limited their knowledge during a search to that the searchers were able to derive from the paper maps. Additional search information to assist in decisions, and ultimately to increase the chance of finding people before it is too late, can be extracted from digital maps using a GIS that fits into SMSR's existing workflow. Basically, SMSR's GIS developed in this project provides two analysis tools: one for analyzing the terrain searchers have viewed and the other to determine where emergency beacons could be located based on signal readings. Additional tools support the team operating with the GIS maps similar to the operations with paper maps: having teams, placing clues and defining search regions. It then presents the system design.

3.2 Requirements Analysis

To be a success, the GIS created for SMSR needed to not only fulfill their functional requirements describing how the system itself works, but also the nonfunctional requirements which detail how the users interact with the system. The success of the project depended on how well the system integrated into the workflow of the searchers and encouraged its use; the field testing, training, and documentation were key.

3.2.1 Functional Requirements

SMSR needed a system to allow them to incorporate additional information into their searches. The system was required to provide guidance as a search develops by showing how well surrounding areas have been covered in the search for a missing person. The system needed to be intuitive for the searchers to use, and must provide some advanced analysis capabilities for users with additional training in the system. Since the team may

be called to search anywhere in the world, their GIS needed to be easy to extend for other regions by providing the necessary data and making minor (non-programmatic) updates to the tools.

As mentioned earlier, SMSR performs wilderness search and rescue. This brings with it a couple unique requirements: (1) low wait times, and (2) no connectivity dependencies. When there are lives at stake, the team will not wait for a computer to run an analysis. If the results are not delivered quickly, the team will move on and continue searching. If that is the case, the result, once ready, will no longer necessarily contain all the current search information.

At times the search is managed in the field at remote locations. Thus the system needed to be able to run on a standard laptop in the field, and without a network connection. Yet many searches have a team that is at the home base, complete with an internet connection and networked computers. The system is also able to take advantage of that setup when it is available.

Table 1. Functional System Requirements.

Functional Requirements	
Provide detailed spatial search route analysis	Required
Visually display the search route analysis	Required
Provide analysis of beacon signal readings	Required
Visually display the predicted beacon location	Required
Run in the field (without internet support)	Requested
Extendible with data for additional areas, and documentation on how to incorporate it	Requested
Store and manage associated data	Required
Low wait times for analysis results	Required

3.2.2 Nonfunctional Requirements

A system that fulfills the above requirements is only useful if it is a system that the team can easily use. As such, the nonfunctional requirements, outlining how a user interacts with the system, are also critical to the success of the project (Table 2). The user interface for the GIS needed to be simple because many of the searchers have limited to no GIS knowledge, experience, or training. Although the interface focuses on the map of the search area, it also provides access to tools to perform actions on that search area. The interface also needed to support the workflow of the more advanced and more trained search director, providing him with tools to reanalyze the search's focus. Since Microsoft

Word and the Google Maps site are commonly used by searchers, the GIS application navigation mimics both. This means that toolbars organize the tasks searchers want to take right on the screen, and that the user can pan and zoom the map as done in Google Maps. Errors (such as those that occur during integration of search routes and findings) are handled automatically where possible, and if input is needed from the user there are dialogs to guide through the correction.

Table 2. Nonfunctional System Requirements.

Nonfunctional Requirements	
Simple user interface, matching common applications	Required
Automated error handling	Requested
Training courses available	Required
Documentation available	Required

The goal of the system is to assist SMSR by enhancing the information available to them as they make search decisions. To meet this goal the team was given basic training in using the system, as well as documentation about the system. In addition, the system itself was deployed onto SMSR’s machine and is operational for the team.

3.3 System Design

Sierra Madre Search and Rescue is new to the GIS arena. They needed an easy to use system to bring GIS into their organization and inform their search decisions. The user interface of their system is ArcMap, which is used with a specific map document, or MXD. As this project focused on their home search area of the San Gabriel Mountains, the MXD displays terrain, transportation, and other relevant data for that area. The MXD contains a custom Search and Rescue toolbar so that the team has a single place to look for the tools they use most frequently. It combines some of the ArcMap navigation tools, some custom data entry tools, and two tools which perform spatial analysis. The custom data entry tools help the team to enter areas of interest, clues, routes, and beacon signal readings. The route data entry tool also performs spatial analysis, analyzing the visibility of a given search route and providing the team with information about what areas they have visually covered during a search. The second spatial analysis tool will predict the area to focus on when searching for a downed aircraft based off readings taken on the emergency beacon’s transmission. In order to have a clean MXD to start with, the MXD provided is not writeable and is never opened by the team. Instead, they create new Search Map Files using a provided tool. Complete details of this system are provided in Chapter 5. Since the ArcMap navigation is similar to that used in Google Maps, it will have a familiar feel to the team members. In addition, the toolbar and tool structure is similar to that the team will be familiar with through programs such as Word and Internet Explorer.

For this system to be of most use to the team, the system also includes documentation (both electronic and paper). It gives basic guidance on using all of the tools on the Search and Rescue toolbar to get the most information from the GIS. It includes more detailed help on using the custom geoprocessing tools, including guidelines on setting the parameters of each tool.

Underneath, the MXD is powered by data stored both in geodatabases and on disk. When the MXD is first opened for a search, it will display only the basic area data for the San Gabriel Mountains. That data comes from a single geodatabase, the base database, as well as from imagery of quad maps which are on disk. However, the results of running the different analysis tools are stored in a second geodatabase that is particular to a given search, keeping that information separate and minimizing the data at the team's fingertips to that relevant for their current operation. These databases are described in detail in Chapter 4. This structure also enables the team to extend the system for other geographic areas by providing a different base geodatabase, or adding additional data to the existing base geodatabase (benefits and disadvantages of each approach are presented to the team in the system documentation).

3.4 Project Plan

Over the course of the year the project plan had to be adjusted. Although the basic workflow and goals remained unchanged, the plan for reaching them had to be reevaluated.

3.4.1 Original Project Plan

Initially the focus of the project was on incorporating a GIS into the Sierra Madre Search and Rescue workflow, including a workflow for using data collected with GPS devices over the course of the search. The original plan included four major deliverables: a GPS collection workflow, a geodatabase, an application to use to perform the search analysis, and a transfer of knowledge to the team through both documentation and training. To implement the original plan, this project followed a traditional development approach in order to leave the client time to provide feedback, particularly regarding how well the application fits into their workflow. The following steps were used to complete the project:

1. Obtain Proposal Approval: SMSR was provided with details of the project and agreed to the goals.
2. Conduct Application Requirements Analysis: The author met with the client to further outline the client's goals for this project and what the client wanted in their GIS system. This also involved learning the client's existing workflow. This task was coordinated to line up with one of the client's field exercises to present a live look at the team's workflow.
3. Collect Data and Design Geodatabase: The author collected topographic and transportation data for the San Gabriel Mountains near Sierra Madre. The data was then brought into a geodatabase.
4. (Removed) Develop GPS Collection Workflow: The author originally planned to develop a workflow for collecting GPS data of a search as it progressed, including when and how to import that data into the GIS. However, once SMSR provided

insight into their existing processes, it became clear to both the group and the author that the team needed a more basic GIS workflow to introduce them to the use and capabilities of GIS. Once the team is familiar with using a GIS to automate parts of their existing workflow, then a GPS workflow should be investigated. As a result, this task was removed from the project. Instead, the time for that task was focused on additional training for the SMSR team.

5. **Develop Application:** The GIS system for SMSR's use during their operations includes an MXD and custom toolset. The MXD was designed and created. Tools to supplement search operations for SMSR were developed and incorporated into the user interface through a custom toolbar.
6. **Prepare Documentation:** SMSR, as well as collaborating teams, need to be able to use the GIS system with minimal training. Documentation was written to provide the information the searchers need to know about interacting with the system.
7. **Test Prototype:** The prototype was tested by inexperienced GIS users known by the author. SMSR was also be given a chance to try out the proposed system and provide their feedback. Although there was not a training event in a timeframe that worked for this testing, the testing was done at a training meeting of SMSR.
8. **Modify and Finalize:** The input from SMSR and the other testers was brought into the system.
9. **Deploy Solution and Train SMSR:** Once the GIS system was ready, the workflow, documentation, and application, including supporting code, were provided to SMSR. In addition the author provided training in the system to SMSR.

A graphic outline of the original tasks follows (Figure 3-1). The key decision points are also included. The GPS workflow step is shown in this original project workflow although it was later removed.

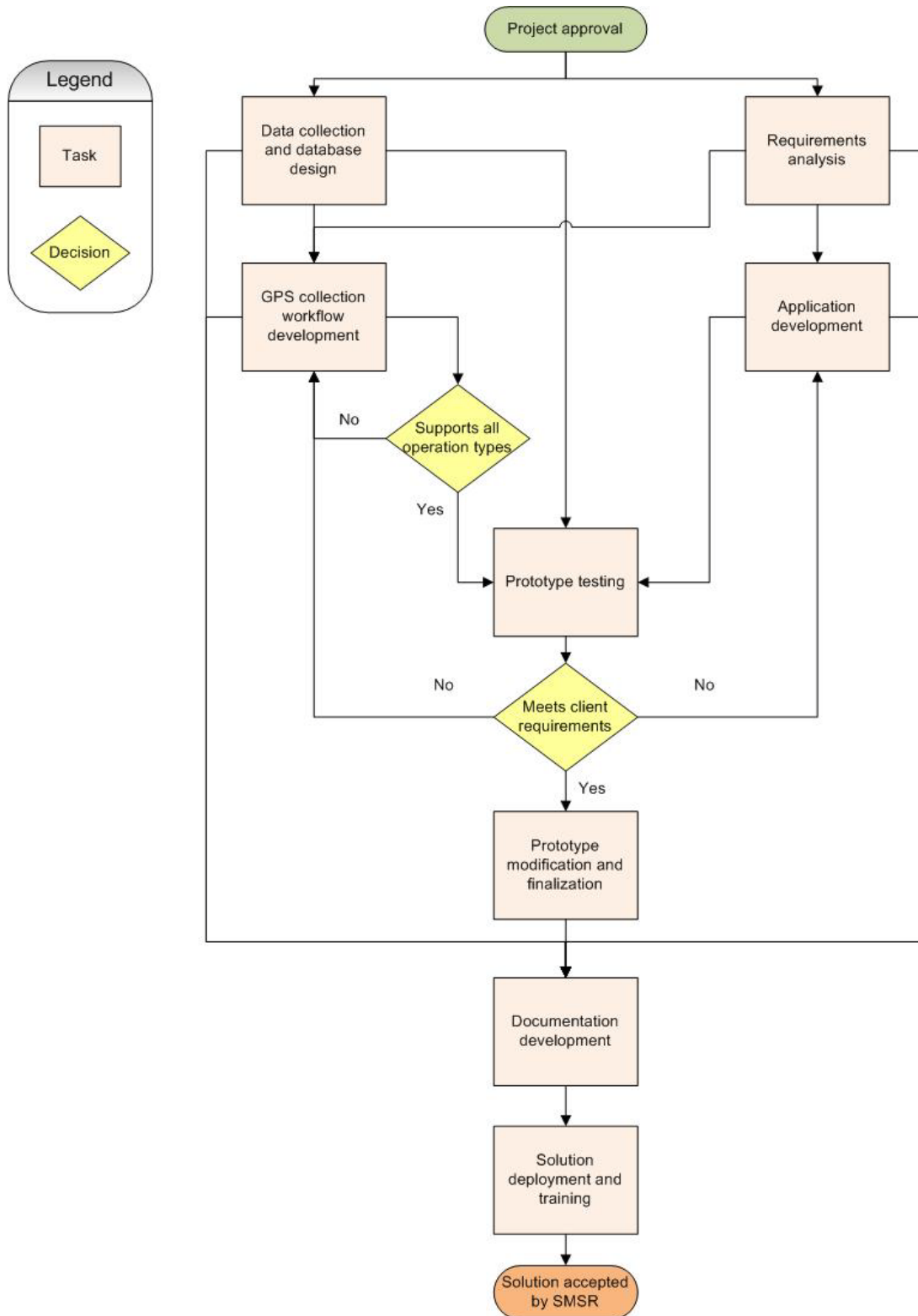


Figure 3-1: Original Project Workflow.

One of the strict requirements for this project was that it fit within a year timeframe. To keep within that schedule, the project plan organized the time to allot to each major

task of the project as shown in Figure 3-2. Over the course of the project, modifications were made to the deliverables and focus, and those adjustments are reflected in the modified project plan.

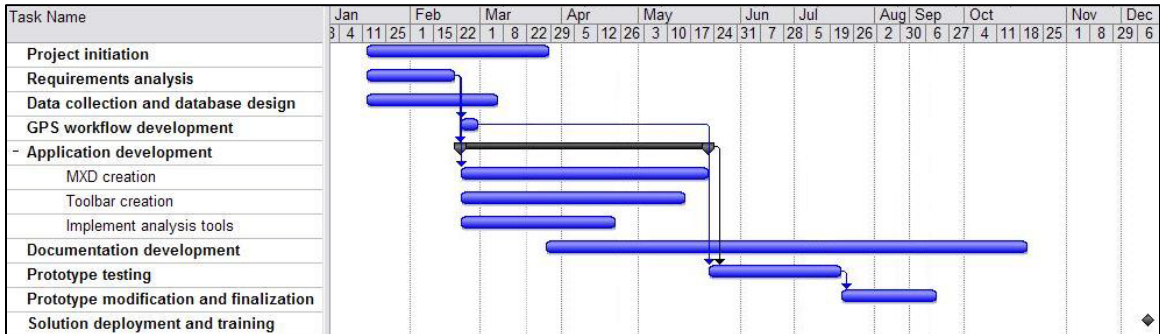


Figure 3-2: Original Project Schedule Chart.

3.4.2 Actual Project Plan

As the project progressed, it became clear that the team needed to break the project into phases and that this year would focus on the first phase. Existing research on integrating GIS into operations emphasized that the biggest factor for success is integration into an existing workflow. SMSR was not yet using GPS data collected during searches, nor were they regularly collecting it. Instead of focusing on the GPS workflow in the first phase of enhancing SMSR’s workflows with GIS, this project was refocused to emphasize automating processes the team currently performed on paper maps. Plans for a GPS workflow were removed from the project, and the time for that work was allotted to the knowledge transfer, particularly additional GIS training of the team (Figure 3-3). The project focus became familiarizing SMSR with GIS and its basic use, as well as providing them with some tools that would automate tasks they already perform.

This allowed the custom application to be refined, and the plan updated for the creating of an MXD with a custom Search and Rescue toolbar containing two custom geoprocessing tools.

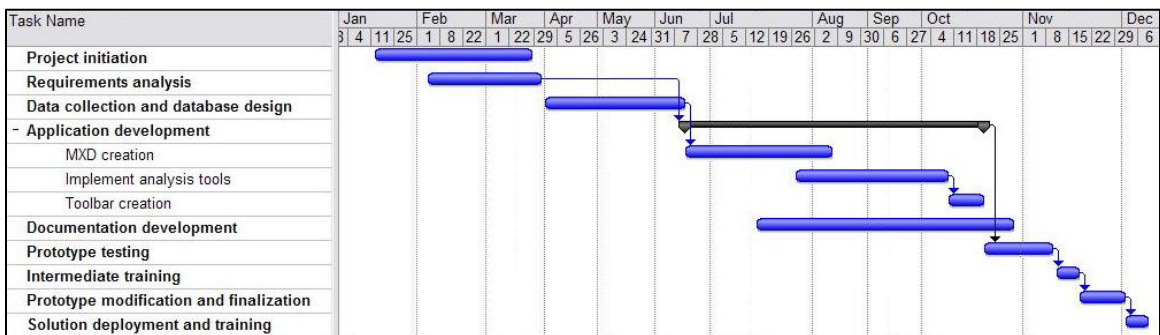


Figure 3-3: Final Project Schedule Chart.

3.5 Summary

When the Sierra Madre Search and Rescue team is looking for a missing person, paper maps provide limited information. A GIS enhances the information available and allows the team to make better informed decisions. An MXD containing a custom toolbar and custom geoprocessing tools provides information to their searches. In particular, the custom visibility analysis tools provided enable them to determine: (1) areas searchers have visually covered, and (2) the location of an emergency beacon based on the signals received from the beacon.

This year-long project started with one focus being creation of a workflow for managing GPS data. After getting a better feeling of where SMSR was in GIS use and understanding, it was determined that focusing on that workflow would result in too many changes to the organization's existing workflow. That component of the project was removed from the current phase and became a future recommendation.

Chapter 4 – Data and Database Design

The GIS created for Sierra Madre Search and Rescue (SMSR) is driven by data detailing the terrain they search as well as the findings of particular search operations. While often a single geodatabase drives a GIS, the team has two distinct sets of data: that for the terrain and other features of a search area, and that pertaining to a specific search. To best handle the reuse of the search area data, and also support archiving of data about a particular search, a multiple file geodatabase design was used.

This chapter describes the structure of both the Base and Search geodatabases, as well as the sources of the data to populate the geodatabases and the methods used to collect them in the geodatabase.

4.1 Structure of the Databases

The terrain and search area data is housed in the Base geodatabase, and the data specific to a particular operation lives in the Search geodatabase.

4.1.1 Base Database

The Base geodatabase houses the data for the areas searched. It includes both 10 and 30 meter digital elevation models (DEMs) to provide information about the terrain as well as transportation route feature classes including roads and trails. While georeferenced quad maps are also used in the MXD created for the team, the data behind them is not part of the geodatabase. It is MrSID image data that was already on the machine of the team, and with the good performance of that data, as well as the large size of the dataset, it was determined unnecessary to bring that into the geodatabase.

Currently, the 10 meter DEM is the only base data that is used in the analysis. A 30 meter DEM is included in case less terrain detail and faster analysis is required for some searches. However, the team requested the other data, such as the transportation feature classes and quad map imagery, be included in order to give them a familiar frame of reference.

For purposes of this year long project, the geographical area of interest was limited to the home area of SMSR, the San Gabriel Mountains, and as such a single Base geodatabase was required and created. In future extensions of this system, SMSR may extend the system by adding additional geographic areas to that same Base geodatabase, or, as it makes sense, by creating additional Base geodatabases for geographically distinct search areas. Either extending the existing Base geodatabase, or creating a new Base geodatabase, would be done using ArcCatalog. The steps to take for this extension, as well as a discussion of the two approaches, are included in the system documentation. The exact contents of the Base geodatabase for each area will depend on the information available: the DEM is a key component in the analysis, and the only required piece. However, the structure shown in Figure 4-1 is followed in the Base geodatabase provided to the team and is encouraged for future additions:

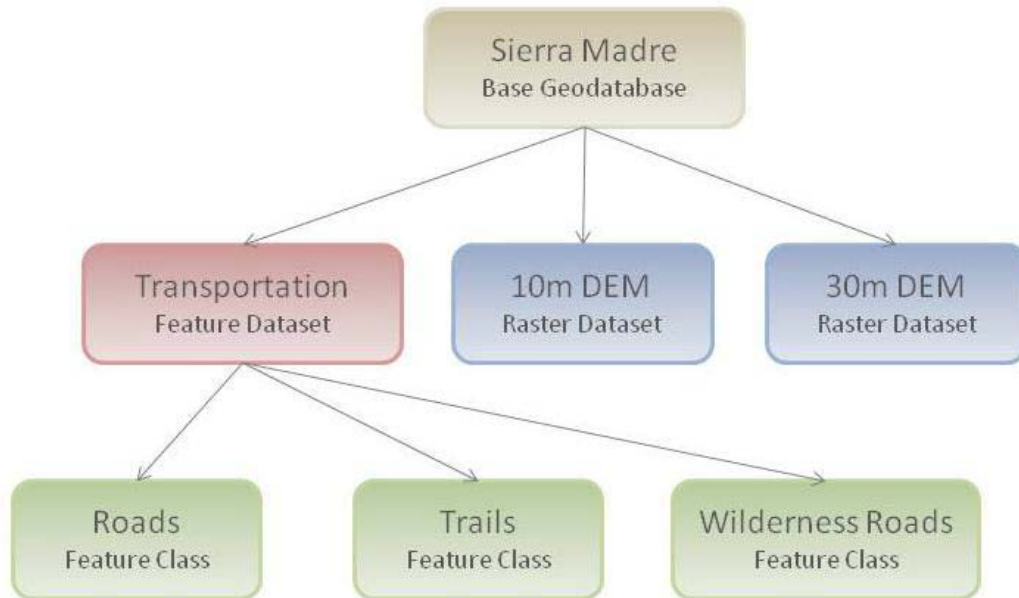


Figure 4-1: The Base Database Design.

The designed Base database model was then developed in ArcCatalog. Figure 4-2 shows the structure of the Base geodatabase implemented for SMSR.

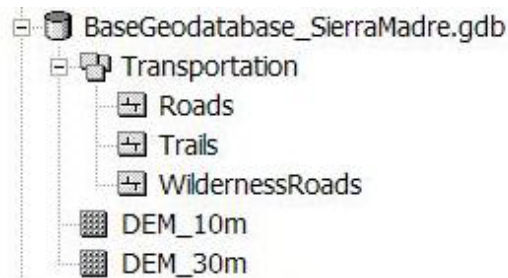


Figure 4-2: The Base Database in ArcCatalog.

4.1.2 Search Database

The Search database houses the information particular to a single search. It includes routes that the team digitizes based on where the searchers go, as well as the analyses of those routes. It also can contain emergency beacon signal data, including signal strength and source direction, when beacon signal readings are taken as a part of the search.

The Search database will be created during a search. While an example has been provided for the team along with this project, creating that structure will be up to the team as they search. The guidelines for that creation are summarized in the following paragraph and are detailed in the system documentation, including directions on how to create the different components.

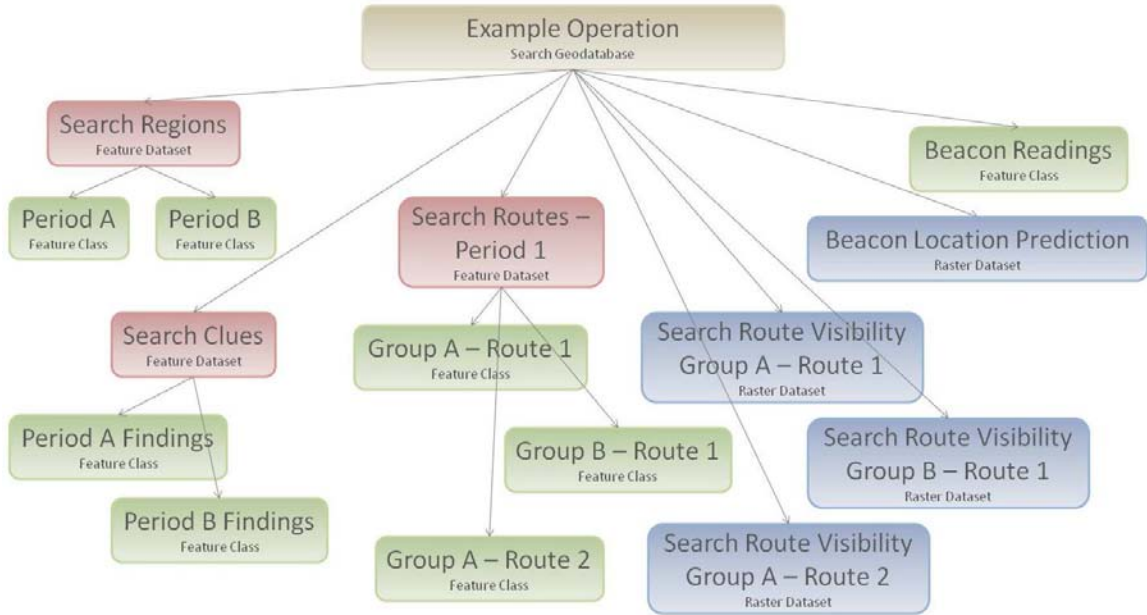


Figure 4-3: The Search Database Design.

The Search database (Figure 4-3), once created during an operation, will be named based on the date and time the search started and the region in which the search took place. It will contain a number of feature datasets (collections of data), each containing a number of feature classes (related features, or pieces of data), as well as a number of raster datasets. Feature datasets are created for each group of data from different parts of the search, or operational periods. The structure shown in Figure 4-3 is followed in the example Search geodatabase, and that structure is created when using the MXD and its tools, accepting the default settings for data creation and output. The implemented Search geodatabase is shown in Figure 4-4.

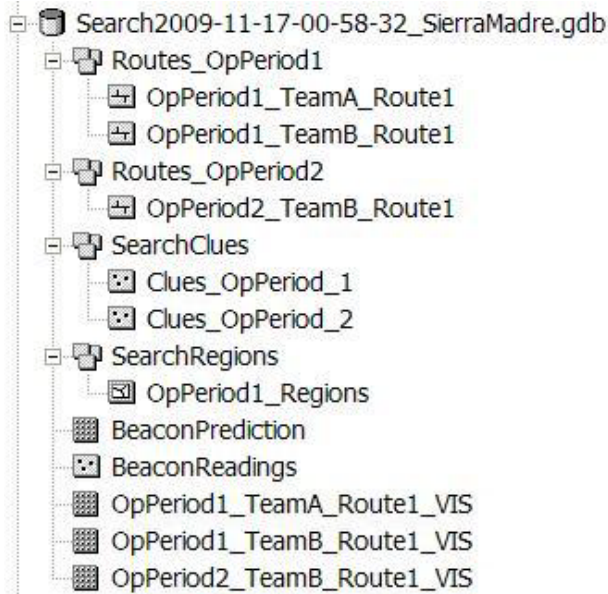


Figure 4-4: The Search Database in ArcCatalog.

In this example Search database (Figure 4-4), a feature dataset (SearchClues) contains a feature class for each set of clues from each operational period, including Clues_OpPeriod_1 and Clues_OpPeriod2. Another feature dataset, SearchRegions, contains a feature class for each set of search regions from each operational period. In the example shown in Figure 4-4, the regions from operational period 1 are stored in OpPeriod1_Regions. Additionally, the routes followed in operational period 1 are contained in the Routes_OpPeriod1 feature dataset, and that feature dataset contains a polyline feature class for each route taken by each team. In the example, they are OpPeriod1_TeamA_Route1 and OpPeriod1_TeamB_Route1. The routes from the second operational period are in the feature dataset Routes_OpPeriod2, and in this example that was the single route OpPeriod2_TeamB_Route1 that was taken by Team B. The geodatabase will also include the visibility analysis raster datasets, named similarly to the feature class for the route, but with _VIS appended at the end. An example from Figure 4-4 is the raster dataset OpPeriod1_TeamA_Route1_VIS, indicating the visibility result for the route OpPeriod1_TeamA_Route1 that Team A followed in the first operational period.

If the search involves taking beacon readings, a point feature class Beacon Readings containing all the beacon reading sites will be created. When the beacon location prediction analysis runs, a BeaconPrediction raster dataset will also be created in the search geodatabase.

Although the team will rarely, if ever, have to look at the contents of the geodatabase, the naming scheme was designed to keep the data gathered by operational periods, teams, and the type of data that is contained within the dataset.

4.2 Data Sources

Although data was not the focus of this project, it was necessary to have some basic data about the search terrain in order to provide Sierra Madre Search and Rescue with examples of how the GIS could assist them. The following datasets were collected for the San Gabriel Mountains area from free downloads at the <http://seamless.usgs.gov> site:

- 10 meter and 30 meter DEMs
- Roads

In addition, the US Forest Service's more detailed fire roads and trails of the Sierra Madre area were downloaded from <http://www.fs.fed.us/r5/rsl/projects/frdb/>.

For purposes of this year long project, the geographical area of interest was limited to the home area of SMSR, the San Gabriel Mountains. When downloading data by a specific extent, the following limits were used: 34.2716 N, -118.20627 W, 34.14544 S, and -117.83937 E. Not all of the datasets span the entire area, but enough data is provided for proof-of-concept analysis. In addition, information was provided to SMSR along with this project to facilitate their retrieval of similar datasets for other areas of interest in the future.

SMSR declined to provide additional data for use in the geodatabase; however, they did provide quality quad map imagery. That data is used by the team in a MapTech system, but it can also be brought into and displayed in ArcMap. Although it is not being included in the geodatabase, it is being used in the MXD to provide the team with a familiar look at the area as it matches the paper maps they are used to seeing. SMSR also

has other agency contacts that they could use in the future for additional data as the need arises.

4.3 Data Scrubbing and Loading

Used to working on paper maps, Sierra Madre Search and Rescue prefers the 1983 North American Datum (NAD 83). To avoid deviations from what they are familiar with, the data in the geodatabases was projected to this datum before inclusion. When the team enters UTM coordinates for search data, the coordinates are already in NAD 83 from reading them off the quad maps. If coordinates are provided from a GPS, or using Latitude and Longitude as reported by aircraft, those are often in World Geodetic System 1984 (WGS 84). However, in the continental United States the difference between NAD83 and WGS 84 coordinates is less than a meter. For purposes of search and rescue applications, that is an acceptable margin of error. In Hawaii and Alaska, that error can be as large as two meters. If working outside of the United States, the team will need to evaluate what datum to use for their base and search data, and a change would require some updates to their tools as well.

4.4 Summary

To handle the unique data requirements of a search and rescue team, a multiple geodatabase solution was created. Base geodatabases contain general information about a search area, including terrain, transportation, and other non-search specific information. Each Search geodatabase contains information about a particular search. This allows the team to archive information no longer needed in the field while reusing their Base data.

Chapter 5 – Implementation

To bring GIS into Sierra Madre Search and Rescue’s workflow, ArcMap was customized for their use. A map document, or MXD, was created to display the relevant information and to provide interaction with the map. The MXD contained a toolbar, simplifying the ArcMap experience and focusing it to the needs of the search and rescue team. On that toolbar were a number of tools for navigation, digitization of search routes and search findings, and most importantly the tools to do visual analysis of the terrain and provide information that was not available to the team when working on paper maps.

In this chapter, the deliverables are described, starting with the Map Document. Next the toolbar is presented, along with the tools for both data entry and analysis. The models used in the analysis tools are also detailed. It wraps up with a look at the documentation included for users of the system.

5.1 The Search and Rescue Map Document

The basic environment of Sierra Madre Search and Rescue’s GIS is a map of their search area. The key dataset for the analysis is the digital elevation model (DEM) of the region. However, the team has to be able to identify locations and digitize routes that they took, so they also need more visually-friendly background information of the search area to help team members reference themselves to the map (Figure 5-1). The background information was included in the Base geodatabase and on disk beside that geodatabase and is displayed in ArcMap as shown in Figure 5-1. These visual-aid background features include roads and trails, which are turned off by default, and wilderness roads where there is data available. The final dataset included to assist the searchers in understanding what they are seeing is a georeferenced quad map. This provides the same look, feel, and view of the area that the team is used to from their paper maps. Since the DEM is not a useful visible layer, it is turned off by default.

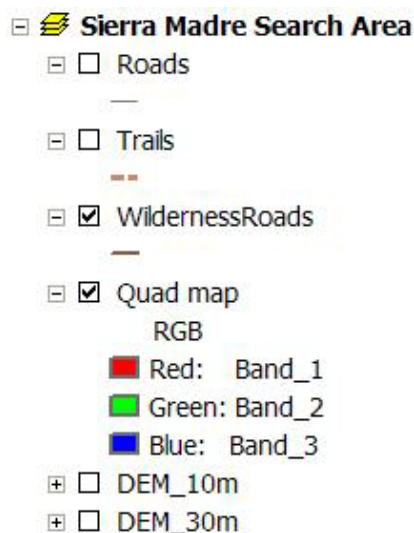


Figure 5-1: The Sierra Madre Search Area MXD Default Table of Contents.

A Search and Rescue toolbar, detailed in the next section, is also included in the MXD. It enables the team to interact with the map: navigating around it, adding data, and performing analysis. Unlike the default ArcMap experience, there is only the single toolbar available by default; the others are closed to provide a simplified experience.

5.2 Search and Rescue Toolbar

The search and rescue team needs to be able to navigate around the map, interact with it and provide route information, as well as analyze the information that it contains. To support all of these functions, the Search and Rescue toolbar, the “one stop shopping” location for SMSR, contains some of ArcMap’s navigation tools as well as custom data addition tools, custom drawing tools, and the custom geoprocessing analysis tools. The toolbar also contains a shortcut for launching the help system provided for the team.

Some basic ArcMap tools are on the toolbar to provide navigation around the map (Figure 5-2), printing (Figure 5-3), and for map interaction (Figure 5-4). This includes panning and zooming tools: pan, zoom in, zoom out, and zoom to full extent. It also includes the previous and next extent tools to navigate between recent views. There are also tools for printing out the maps that are created.



Figure 5-2: The Navigation Tools on the Search and Rescue Toolbar.



Figure 5-3: The Printing Tools on the Search and Rescue Toolbar.



Figure 5-4: The Map Interaction Tools on the Search and Rescue Toolbar.

For the team to start or save a map document for a particular search, they use the New Search Map File command to start, and the basic ArcMap save command to save (Figure 5-5).



Figure 5-5: The New Search Map File Command and the Save Tool on the Search and Rescue Toolbar.

Custom tools are provided for the team to add information to the map. They include a custom command for the team to define the teams that are participating in the search (Figure 5-6, far left icon). This information is used by many of the other tools to set the dialog options for associating a team. To the right of that command on the toolbar is a

tool to define search regions by drawing polygons on the map. The third icon shown in the toolbar segment in Figure 5-6 is a tool to add point locations to represent clues found during the search. Next the team has an add route tool that lets them draw the route followed by searchers onto the map. When the route is entered, the tool also calculates the visibility along that route. The toolbar also has a couple tools for searches which include taking readings of beacon signals and analyzing those readings. The first of the beacon tools adds beacon signal readings to the map (the second tool from the right in Figure 5-6). The second beacon tool, which is the final tool shown in Figure 5-6, runs a custom geoprocessing tool to calculate the possible beacon locations based on the readings entered. These commands and tools are described in detail in Section 5-3.



Figure 5-6: The Add Team Command, Define Search Region Tool, Add Clue Tool, Add Route Tool, Add Beacon Reading Tool, and Beacon Location Calculation Command on the Search and Rescue Toolbar (left to right).

The final items on the Search and Rescue toolbar are a command to launch the help for the system and a label that notifies what Base geodatabase is associated with this particular search (Figure 5-7).

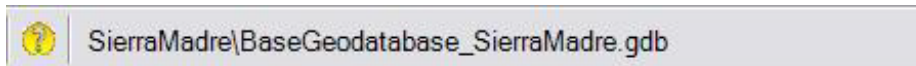


Figure 5-7: The Help Command and the Base Geodatabase Label.

A key benefit of delivering a toolbar, and a customized version of ArcMap, is that while the team has the simplified GIS experience that will help them integrate GIS into their workflow, they also have the full ArcMap capabilities. Although some of the tools and commands that are in ArcMap are hidden from view, they are still available to the team. As the team learns more about GIS, they will be able to make use of more areas of the system without requiring a completely redefined application and basic experience. In addition, it allows them to make use of the basic display, navigation, and printing which are available in ArcMap.

5.3 Custom Sierra Madre Commands and Tools

SMSR collects data during their searches and needs to import it to the GIS and run some analysis on the data entered. While the default ArcMap editing interface is complicated in order to support all different kinds of editing, they only need to do some basic digitizing and specify some predetermined attributes. To keep the search team's GIS experience simple, custom tools were written to assist them in these tasks.

5.3.1 New Search Map File Command

The team keeps a search map file for each search that it does. The search map file is an MXD which is based on the MXD for the region in which the search is occurring. To

start a search, the button is clicked to create the Search geodatabase and also an MXD for information relating to that search. When the New Search Map File command is clicked, the Set Base Geodatabase dialog prompts the user for the Base geodatabase to use with the search (Figure 5-8). The options in the dialog are generated based on all the geodatabases present in the Base data folder in the application's directory.

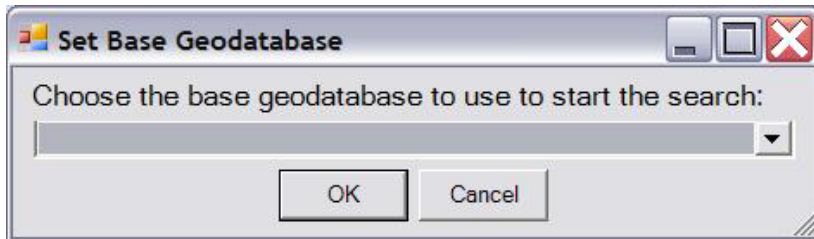


Figure 5-8: The Set Base Geodatabase Dialog.

When “OK” is clicked in the Set Base Geodatabase dialog, the time and date is converted to a string to use to name the Search geodatabase and also the Search's MXD. They are created in the Search data folder in the application's directory, and are named following the pattern Search_SEARCHDATE_SEARCHREGION.mxd where SEARCHDATE is the date in the format year-month-day-hour-minute-seconds with the time using a 24 hour clock, and SEARCHREGION is the name of the region in which the search began (based off the Base geodatabase selected). Then the MXD for that region is copied to an MXD with the new name and stored in the application's Search data folder. If no such MXD exists for the region, an empty MXD is created with the new name. After the MXD is created for the search, a Search geodatabase with the same naming is created, and the MXD is opened and is ready to record and analyze a new search. If no Base geodatabase is selected a message is returned and no Search geodatabase or MXD is created. Key sections of the code for this command can be found in Appendix A.

5.3.2 Add Team Command

As a search begins, the Search and Rescue team divides into search teams. These are often designated by letters. For example, if there are three teams, they are often Teams A, B, and C. However, the names can be different. This command opens a dialog (Figure 5-9) which lets the search and rescue team add the team names that they will be using during their search. If additional teams form over the course of the search, they can be added using this dialog at any point during the search.

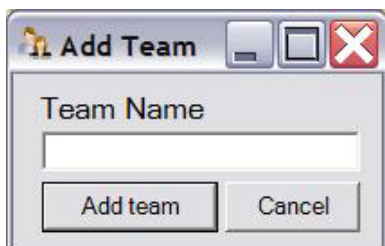


Figure 5-9: The Add Team Dialog.

This command works by creating a domain in the search’s geodatabase and populating it with the team names entered as options. That allows the team names to be persisted across ArcMap sessions using this search database. However, the fields that store team names are not set to use this domain, as domains that are used cannot be updated. It also ensures that the team uses a consistent naming of the team across the different data dialogs. Key sections of the code for this command can be found in Appendix A.

5.3.3 Define Search Region Tool

One task that the search and rescue team is often assigned is to define the map into regions. The team then determines which region to focus on by voting, assigning each region a popularity score. This tool allows the team to draw polygon region on the map and then, in the dialog that opens (Figure 5-10), give them names, assign them popularities, and indicate which operational period during which they were defined.

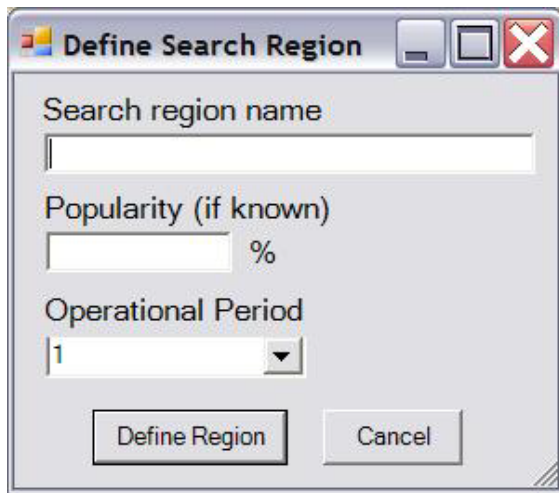


Figure 5-10: The Define Search Region Dialog.

This command works by using an ArcMap tool to draw a single polygon region on the map and when the sketch is completed it opens the dialog shown in Figure 5-10. In that dialog, the search region’s name, popularity, and operational period are entered. The operational period defaults to the last operational period used, but has a “Next” option if a new operational period has started. Operational period choices, like team names, are stored in a domain in the search geodatabase. When the “Define Region” button is clicked, the tool determines if an appropriate feature class exists to add this polygon to or if a new feature class needs to be created. It looks in the feature dataset SearchRegions for a polygon feature class named OpPeriodX_Regions, where X is the operational period. If no SearchRegions feature dataset exists, it is created. If no such feature class exists, one is created with REGION_NAME and POPULARITY attributes (Figure 5-11), and those values are set based on those entered in the dialog. If the feature class does exist, the polygon and its REGION_NAME and POPULARITY attributes are added to the existing feature class.

OBJECTID *	SHAPE *	REGION_NAME	POPULARITY	SHAPE_Length	SHAPE_Area
1	Polygon	WilsonPeak	27	9.68453510741431E-02	-4.82637552157511E-04
2	Polygon	EatonCanyon	34	0.146903830518784	-1.32972706526246E-03

Figure 5-11: An Example Search Region Attribute Table.

When the feature class is added to the map, a custom renderer is also added. It sets the polygons to unique colors with transparency of 50%. The layer of the search regions is also set to display map tips of the region names. When a feature is hovered over, the region name appears as a map tip. Key sections of the code for this tool can be found in Appendix A.

5.3.4 Add Clue Tool

When clues to the location of the victim are found, the teams radio the location and details of the clue back to the search command. This tool allows the team to add that information to the map. When the tool is clicked, the Add Clue Dialog opens (Figures 5-12 and 5-13). When the dialog opens the team can select how the coordinates of the clue will be entered:

1. Using UTM coordinates and the zone.
2. Using latitude and longitude in degrees, minutes, seconds.
3. By clicking on the map, which sets the UTM coordinates and switches to that location display.

The dialog is then used to add the name of the team which found the clue, the operational period in which the clue was found, and a description of the clue.

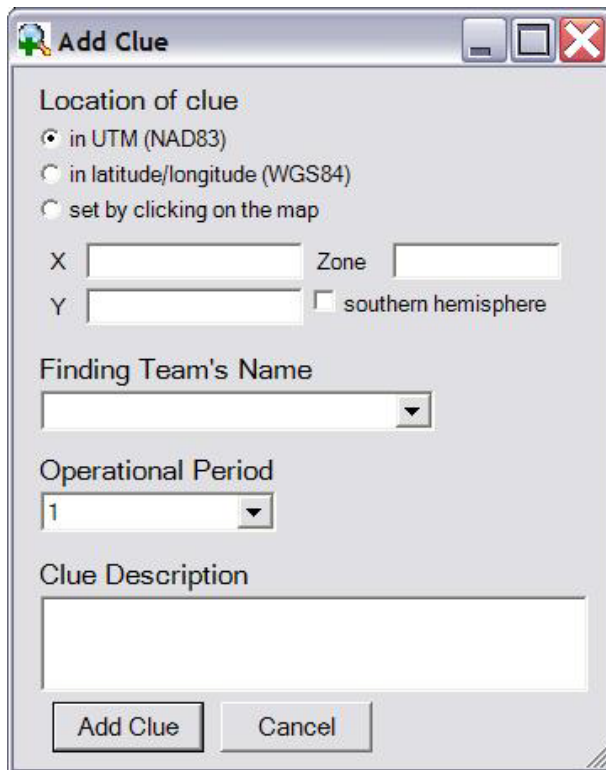


Figure 5-12: The Add Clue Dialog set for UTM Coordinate Entry.

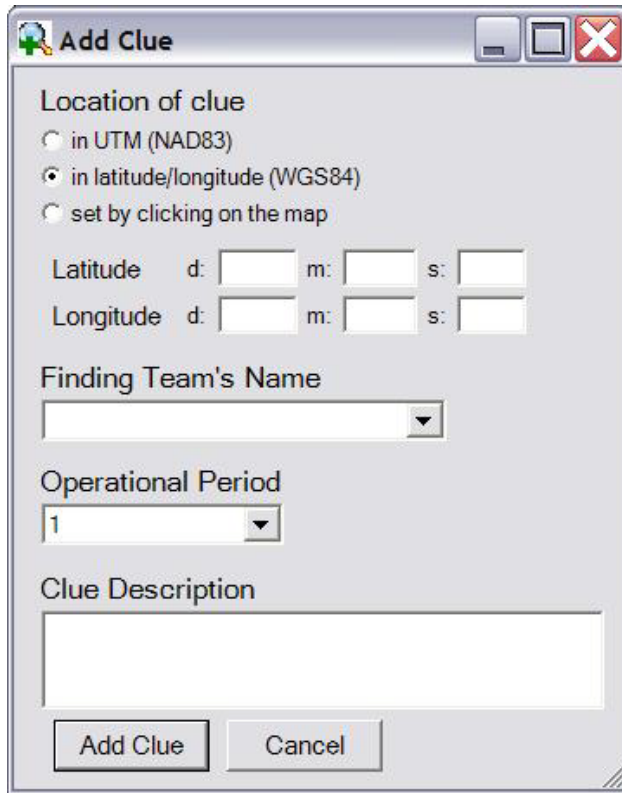


Figure 5-13: The Add Clue Dialog set for Latitude/Longitude Coordinate Entry.

This tool works by opening the Add Clue dialog. If the user is entering UTM coordinates, the dialog shown in Figure 5-12 is shown and the X and Y, along with the zone, and if the clue is in the southern hemisphere the check box is checked. If the user selects latitude/longitude coordinate entry, the dialog adjusts and appears as in Figure 5-13 and the user is prompted for the degrees, minutes, seconds format of the latitude and longitude values. If a positive value is entered for the longitude, the system warns the user. If the user selects adding the clue by clicking on the map, the dialog disappears and the current tool is set to an ArcMap tool that returns the coordinates of the point clicked on the map. When the sketch is completed, the dialog shown in Figure 5-12 returns with the UTM coordinate options populated based on the click on the map, and the user is prompted to set the zone and check if the point is in the southern hemisphere. The team which found the clue is also entered, and then the operational period is set. The operational period defaults to the last operational period used but has a “Next” option if a new operational period has started. Operational period choices, like team names, are stored in a domain in the search geodatabase, and selecting “Next” in this dialog updates the options available for the domain. Finally a short description of the clue found is entered in the dialog, and “Add Clue” is clicked.

When the “Add Clue” button is clicked, the coordinates are converted to UTM, and the tool determines if an appropriate feature class exists to add the point to or if a new feature class needs to be created. It looks in the feature dataset SearchClues for a point feature class named Clues_OpPeriod_X, where X is the operational period. If no SearchClues feature dataset exists, it is created. If no such feature class exists, one is created with TEAM, DESCRIPTION, LATITUDE_DMS, LONGITUDE_DMS,

UTM_X, UTM_Y, and UTM_ZONE attributes (Figure 5-14). Those values are set based on those entered in the dialog. If the feature class does exist, the point and its attributes are added to the existing feature class.

OBJECTID *	SHAPE *	TEAM	DESCRIPTION
1	Point	A	Hat
2	Point	B	wallet
3	Point	B	shoe

LATITUDE_DMS	LONGITUDE_DMS	UTM_X	UTM_Y	UTM_ZONE
34 deg 12 min 20 sec	-118 deg 4 min 36 sec	400802.316883049	3785472.97974916	11
34 deg 12 min 10 sec	-118 deg 3 min 33 sec	402426.324638619	3785155.64490037	11
34 deg 12 min 31 sec	-118 deg 3 min 53 sec	401903.655475907	3785790.31459795	11

Figure 5-14: An Example Clue Attribute Table.

When the feature class is added to the map, a custom renderer is also added. It sets the points to render with a magnifying glass symbol. The layer of the clues is also set to display map tips of the clue description, so when a feature is hovered over, the description appears as a map tip. Key sections of the code for this tool can be found in Appendix A.

5.3.5 Add Route Tool

The teams also track the routes that they take during the course of the search. This tool allows the team to add that information to the map. This command works by first tracking a polyline on the map. When the sketch is completed, the polyline is modified to have a vertex every 10 meters. This improves the visibility analysis since the visibility is only calculated at vertices. Then the dialog shown in Figure 5-15 opens, and the user is prompted to set the attributes of the route. The team which found the clue is entered, and then the operational period is set. The operational period defaults to the last operational period used but has a “Next” option if a new operational period has started. Operational period choices, like team names, are stored in a domain in the search geodatabase, and as with the Add Clue Tool selecting “Next” in this dialog updates the options available for the domain. The team also enters the time the route was followed. The visibility along the route is estimated by the searchers based on the weather and brush density. The search group knows what the visibility range was as they searched. For example, on a foggy day, the visibility along a route might only be 30 meters. The same route on a clear day might have visibility of 250 meters.

Finally a probability of detection is entered, indicating the chance that the victim would have been found if he was along that route. This value is also one that the teams regularly predict and use in searches on paper maps. When the attributes are filled out, the “Add Search Route” button is clicked.

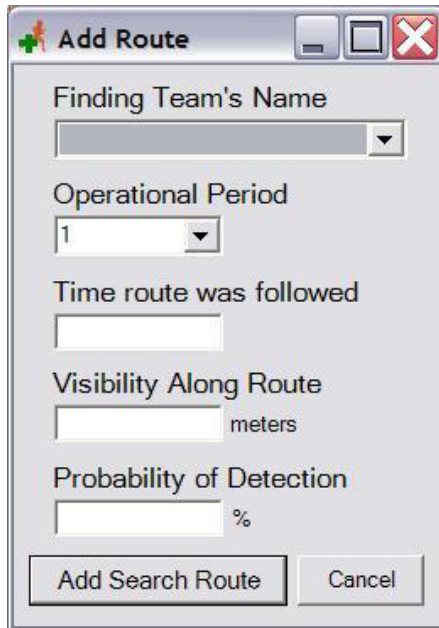


Figure 5-15: The Add Search Route Dialog.

When the “Add Search Route” button is clicked, the tool creates a new feature class for the polyline drawn on the map. First it checks for a feature dataset Routes_OpPeriodX, where X is the operational period. If no such feature dataset exists, it is created. Then in that feature dataset a feature class named OpPeriodX_TeamY_RouteZ is created, where X is the operational period, Y is the name of the team, and Z is an automatically detected numerical count of what route that is for the team during that operational period. The feature class created has the attributes TEAM, TIME, VISIBILITY, RADIUS2, and PROB_OF_DETECTION fields (Figure 5-16). Those values are set based on those entered in the dialog, with VISIBILITY and RADIUS2 having the same value. This is done so that the team can easily read the visibility, but RADIUS2 is used by the visibility calculation on the route.

OBJECTID *	SHAPE *	TEAM	TIME	VISIBILITY	RADIUS2	PROB_OF_DETECTION	SHAPE_Length
1	Polyline	A	5:00:00 AM	200	200	80	4682.95408

Figure 5-16: An Example Search Route Attribute Table.

Before the route feature is added to the map, the visibility along the route is calculated. This is done by running the custom geoprocessing model to create a route visibility raster. Details of this geoprocessing tool are in Section 5-4. Once the geoprocessing tool completes, the raster visibility surface is added to the map with a custom raster classified renderer. Key sections of the code for this tool can be found in Appendix A.

5.3.6 Add Beacon Reading Tool

During searches for downed aircraft, the team travels to high points around the valley and takes readings on the emergency beacon transmissions. They then report the location at which the reading was taken and the direction of the beacon’s signal (if detectable from

that location). That information is then used to determine areas where the aircraft may have crashed. Each time a search group takes a reading on the emergency beacon transmission, the location of the reading and the direction of the strongest signal are radioed back to the command center. This tool allows the team to add that information to the map. When the tool is clicked, the Add Beacon Reading dialog opens (Figures 5-17 and 5-18). When the dialog opens the team can select how the coordinates of the clue will be entered:

1. Using UTM coordinates and the zone.
2. Using latitude and longitude in degrees, minutes, seconds.
3. By clicking on the map, which sets the UTM coordinates and switches to that location display.

The dialog is then used to add the name of the team which took the beacon signal reading and information about the signal reading. The beacon signals are directional and are basically line of sight. Thus the locations from which the signal could be transmitted can be located by doing visibility analysis of the reading locations and the signal direction which the teams radio back to the command center. However, there is a margin of error in the direction reading, and so that direction should not be taken as the exact answer. Instead, the tool uses a range of degrees on either side of that particular direction, entered as the angle of error.

Add Beacon Reading

Location reading was taken

- in UTM (NAD83)
- in latitude/longitude (WGS84)
- set by clicking on the map

X Zone

Y southern hemisphere

Reading Team's Name

Direction of Strongest Signal

True North

Magnetic North - offset

Signal Strength

Angle of error (to each side)

degrees

Distance over ground the reading was taken

meters

Figure 5-17: The Add Beacon Reading Dialog set for UTM Coordinate Entry.

Figure 5-18: The Add Beacon Reading Dialog set for Latitude/Longitude Coordinate Entry.

This tool works by opening the Add Beacon Reading dialog. If the user is entering UTM coordinates, the dialog shown in Figure 5-17 is shown and the X and Y, along with the zone, and if the clue is in the southern hemisphere the check box is checked. If the user selects latitude/longitude coordinate entry, the dialog adjusts and appears as in Figure 5-18, and the user is prompted for the degrees, minutes, seconds format of the latitude and longitude values. If a positive value is entered for the longitude, the system warns the user. If the user selects adding the clue by clicking on the map, the dialog disappears and the current tool is set to an ArcMap tool that returns the coordinates of the point clicked on the map. When the sketch is completed, the dialog shown in Figure 5-17 returns with the UTM coordinate options populated based on the click on the map, and the user is prompted to set the zone and check if the point is in the southern hemisphere. The team which took the reading is also entered. Then the direction that had the strongest beacon signal reading is entered, along with if that reading is true north or magnetic north, and the offset if magnetic north is used for the signal direction. The strength of the signal reading is entered, and then the angle on each side of the beacon reading direction that should be checked for the beacon. Finally the distance over the ground that the reading was taken is entered in the dialog, and “Add Beacon Reading” is clicked.

When the “Add Beacon Reading” button is clicked, the coordinates are converted to UTM, and the tool determines if an appropriate feature class exists to add the point to or if a new feature class needs to be created. It looks for the feature class BeaconReadings. If no such feature class exists, one is created with TEAM, SIGNAL_DIRECTION, SIGNAL_STRENGTH, AZIMUTH1, AZIMUTH2, OFFSETA, LATITUDE_DMS, LONGITUDE_DMS, UTM_X, UTM_Y, UTM_ZONE, and INCLUDED_IN_ANALYSIS attributes (Figure 5-19). If the feature class does exist, the point and its attributes are added to the existing feature class. The attribute values are set based on those entered in the dialog. In the case of the AZIMUTH1, that value is calculated by taking the entered signal direction and subtracting the angle of error. Similarly, AZIMUTH2 is calculated by adding the signal direction and the angle of error. OFFSETA represents the distance over the ground the reading was taken. These three field names (AZIMUTH1, AZIMUTH2, and OFFSETA) are used based on the requirements of the geoprocessing Viewshed tool which is used to determine the area in which the beacon itself is located. The INCLUDED_IN_ANALYSIS attribute is set to 1 on addition of the new reading. During the beacon location prediction it is updated and then used to filter out readings that are too close together.

OBJECTID *	SHAPE *	TEAM	SIGNAL_DIRECTION	SIGNAL_STRENGTH
1	Point	A	135	Very Strong
2	Point	A	75	Very Strong
3	Point	B	120	Very Strong

AZIMUTH1	AZIMUTH2	OFFSETA	LATITUDE_DMS	LONGITUDE_DMS
115	155	3	34 deg 12 min 12 sec	-118 deg 2 min 42 sec
55	95	3	34 deg 11 min 28 sec	-118 deg 2 min 4 sec
100	140	3	34 deg 12 min 11 sec	-118 deg 3 min 36 sec

UTM_X	UTM_Y	UTM_ZONE	INCLUDED_IN_ANALYSIS
403725.041947692	3785191.41552058	11	0
404687.921804259	3783828.61531857	11	1
402342.815985589	3785183.51003894	11	1

Figure 5-19: An Example Beacon Reading Attribute Table.

When the feature class is added to the map, a custom renderer is also added. It sets the points to render with a beacon symbol. The layer of the beacon readings regions is also set to display map tips of the signal direction, so when a feature is hovered over, the signal direction appears as a map tip. Key sections of the code for this tool can be found in Appendix A.

5.3.7 Beacon Location Calculation Command

During searches for downed aircraft the team travels to high points around the valley and takes readings on the emergency beacon transmissions. The readings are entered into the geodatabase using the Add Beacon Reading tool discussed in Section 5.3.6. The utility of

having those readings in the GIS is that the system can calculate a prediction of where the beacon itself is most likely located, and thus the crash site location.

By taking a number of readings and getting readings from angles all around where the beacon is transmitting from, the SMSR team narrows the area of interest (Figure 5-20). The possible locations of the emergency beacon are determined based on each of the beacon signal readings. Those areas of interest are combined, refining the predicted location of the beacon. With that information, SMSR can send in search teams to gather additional information about the area containing the beacon and continue the process of finding the aircraft and assisting the passengers.

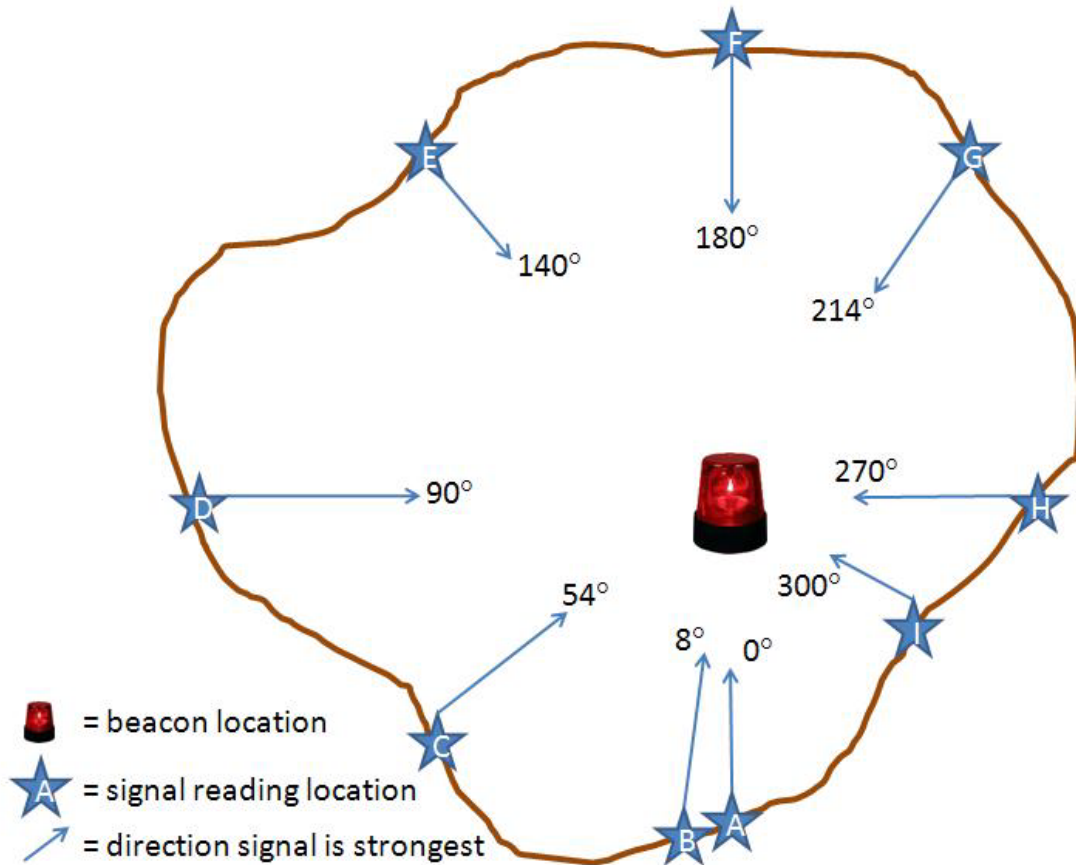


Figure 5-20: Spacing of Beacon Signal Readings as Angles.

When the tool is run it uses the BeaconReadings feature class as input to the analysis. Figure 5-20 demonstrates how the readings relate to each other. They are taken from points all around the beacon’s location and include information about the direction in which the signal is the strongest. The direction in which the signal is the strongest will be an angle between 0 and 360 degrees and will point to the beacon’s location. However, not all the readings will be significant. When two readings are “very close” to each other, the information provided by them is actually duplicated. To address this issue, this analysis tool was designed to allow the team to define the angle at which two readings are regarded as “very close” using the Predict Beacon Location dialog (Figure 5-21). By default, this is set to 10 degrees.

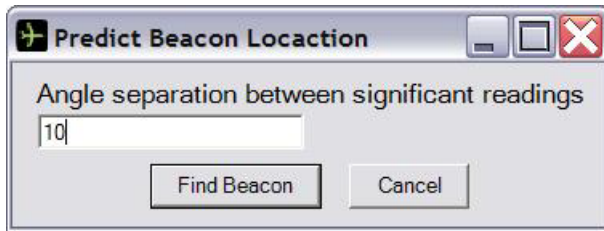


Figure 5-21: The Predict Beacon Location Dialog.

To determine the significance of the readings, the team enters a parameter into the tool that represents the required spacing between readings, or what is considered “very close”. In this example, let’s use an angle of 10° as a significant difference in the readings. In order to determine which readings are of interest, we will order all the readings from smallest to largest angle of signal strength. Based on the numbers presented in Figure 5-20, this gives our example a list: 0° , 8° , 54° , 90° , 140° , 180° , 214° , 270° , and 300° (corresponding with readings at points A—I in Figure 5-20). To determine significant readings, the angles of the first two readings in the list are compared. With a difference of 8° between them, they are not both significant, so the second reading is not used in the combined analysis and is removed from the list. The direction of the reading at point C, 54° , is then compared to the direction of the reading at point A. As they are 54° apart, both are left in place. Reading D is then compared to reading C, and being 50° apart, again both are significant. The list of readings is processed in this manner, and as all other readings are over 10° apart, only reading B is excluded from the combined analysis.

Internally to the tool, this exclusion of points is done by looking at the list of angles, sorting them, and then creating a list of the significant readings by comparing adjacent readings. These significant readings are then copied into a temporary feature class which is then passed as input into the viewshed geoprocessing tool. Unfortunately, the viewshed tool does not honor selections when run from within C# code, but only when run from ArcToolbox.

The geoprocessing viewshed tool also automatically uses the AZIMUTH1 and AZUMITH2 values to indicate the minimum and maximum angles of the field of view and the OFFSETA to indicate the height off the ground at which the reading was taken. The viewshed geoprocessing tool also takes in the terrain raster on which to calculate the viewshed, and here the DEM_10m for the Base geodatabase is the terrain raster used. This returns a raster surface that indicates the combined visibility of the beacon readings: the more beacons that could see a location, the higher the priority assigned to the location. The areas visible to more beacon readings indicate the areas the team should focus on when looking for the beacon.

After the tool completes, the team is provided with an area to focus on when searching for the beacon’s source. It shows the area of the terrain where the signal readings intersect and thus where the GIS predicts the beacon itself is located, and that information is automatically added to the map. It provides the team with the area to begin their detailed search for the missing aircraft.

Key sections of the code for this command can be found in Appendix A.

5.4 Search Route Visibility Analysis Model

A search group follows a route looking for the victim and then reports back any findings. Once a route has been followed, the search group also knows the visibility range of the route they took, taking into account the vegetation and weather. This information can help determine locations that merit further searching. To determine the area visibly covered during a route, a custom geoprocessing model which links together a number of geoprocessing tools was created.

5.4.1 Parameters

This tool requires two parameters: the search route to analyze and the visibility along the route. Often, the search headquarters will know the route taken, as they assigned it. However, at times the actual route followed will differ, and the search group will provide that information. The route is entered into the GIS by drawing it onto the map, and then that route is used when the visibility tool is run. In the future, this tool will be modified to be able to link to imported GPS routes. Although the beacon analysis, which also uses the viewshed tool, required entry of a reading height, that is not the case for the search route visibility. During the route the searchers are walking or driving. The default height above the ground used by the tool is one meter, which is appropriate for a walking or driving reading.

5.4.2 Processing

Once route information is provided by the search team that information is combined with the terrain to show where the search team visibly covered. First the search route is buffered by the visibility distance. Then the terrain that falls within that buffer is extracted for visibility analysis. A viewshed calculation is done, which internally uses the RADIUS2 parameter to restrict the distance seen from each vertex along the line (Figure 5-22). As the viewshed tool processes, it calculates what is visible within the RADIUS2 amount (example: 200 meters) from each vertex along the line

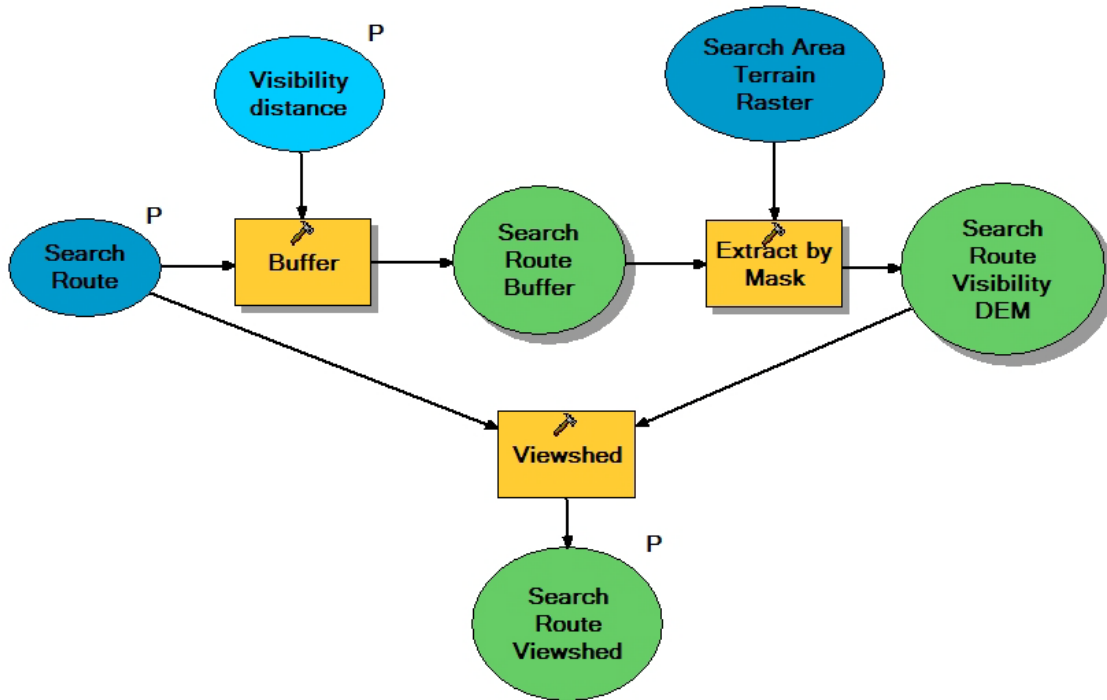


Figure 5-22: The Search Route Visibility Analysis Model.

Using a buffer before the viewshed analysis is done drastically improves the performance of the system; when the entire dataset is used in the viewshed analysis, a single route took on average over 5 hours for the tool to complete, which is much longer than the team can afford to wait. With the buffering, however, it took the same routes an average of under 5 minutes, proving that this optimization makes the tool usable for SMSR.

5.4.3 Resulting information

After the Search Route Visibility tool runs, the team has a new piece of data for their GIS: information about the area a team has visually covered. It shows the parts of the terrain that the team has seen while looking for the victim. If the victim is still missing after some rounds of searching, these datasets can be viewed to highlight where in the focus area searchers have not covered.

A limitation of this analysis is that it only shows the area the searchers estimated they could see. If their estimation is a bit off, the tool might display visual coverage of an area that was not yet viewed or not display visual coverage of an area that has been seen. This could be adjusted by automatically adding or removing a percentage of the distance to account for searchers systematically over- or underestimating their distances. Yet as each searcher will act differently, there is no way to know which would be more important, and instead the search team's analysis of what could be seen is trusted.

5.5 Documentation

If SMSR can't use their GIS, this project will not be a success. In order for the team to be able to make the most of their system, documentation is crucial. This documentation includes both an electronic version, provided as a Microsoft Compiled HTML Help (CHM) file, and a paper version, provided as a bound booklet.

The documentation includes instructions on using the default ArcMap tools provided on the Search and Rescue toolbar as well as steps for using the custom tools provided on that toolbar. While the documentation gives the basics of using each of the tools, the documentation for the custom geoprocessing tools is more detailed. An understanding of how they work is required to provide good inputs and to get the best results from those tools.

In addition to covering how to use the system, the documentation also includes information on expanding the system to cover other geographic areas. This includes where the data included comes from and where to find similar data for other areas, how the data was prepared for the geodatabase, and how it was all collected together. In addition it includes a comparison of two different approaches to expanding the geographic coverage of the GIS: expanding the coverage of the existing base geodatabase, and creating an additional base geodatabase for the new geographic area.

5.6 Summary

Together the MXD, toolbar, tools, and documentation provide for SMSR a GIS that they can use within their workflow. Without making major changes to how they approach a search the team can gather additional information about the search as it progresses. The additional information better informs their search decisions and allows the teams to have all the information possible at their fingertips as they make decisions that often amount to the difference in someone living or dying.

Chapter 6 – Results and Analysis

A GIS is only useful if it is used to answer questions. For the system to answer questions, it has to work as expected and be useable by the team for which it was designed. Some technical limitations arose while the system was in development. Once the tools were in place, the system was introduced to more of the search team at Sierra Madre Search and Rescue (SMSR).

In this chapter the limitations that arose are presented, as well as a summary of the team's feedback. Some of that was incorporated into the system, and those adjustments to the GIS are also presented here. The chapter concludes with an example of how the team might use the system.

6.1 Technical Limitations

Creating this GIS application for Search and Rescue highlighted a couple limitations of the ArcGIS software and its analysis tools. First, the visibility tool only calculates the visibility along a line at the vertices of the line. Obviously this causes issues as the team needs a true visibility from all points along the line, not just the locations where they clicked while digitizing their routes. However, this becomes less significant once GPS devices are used to enter the routes as the devices store the routes with much denser vertices. When the visibility is done on a route entered from a GPS, the visibility analysis uses all of those vertices, and the small gaps between the vertices have an insignificant impact on the visibility result.

Another technical limitation identified during the creation of the tools also relates to the way the visibility tools work: while the viewshed tool does honor the selections when it runs from the ArcToolbox in ArcMap, the viewshed tool does not honor selections made through ArcMap if the tool is running from within a .NET program. Thus in the beacon analysis, the original model resulted in all the beacons being used in the visibility analysis, even those excluded as insignificant readings. As a result, the beacon location calculation had to be revisited in order to only make use of those readings deemed significant.

6.2 Feedback from Sierra Madre Search and Rescue

There were two key points at which the team provided feedback on the system: the first was their training day, where they provided insight into how they run their operations by allowing the author to see it. The second was at a meeting where the beta version of the created GIS system was introduced.

On their training day, it became clear that the team needed to adjust their goals for this project: instead of integrating GPS devices into their workflow, this project transitioned to a more basic introduction to GIS for the team. While the tools created are greatly appreciated, and will be used to make the incorporation of GIS a simpler process, they do still want to learn the default ArcMap system. If they are searching to assist another team, they might be using that team's computers, and their custom tools might not be available.

The beta system was presented to the majority of the SMSR team at one of their meetings. After a brief introduction to GIS, the tools were demonstrated to the team and were well-received by the members. The team found it useful to be able to overlay search data, as well as to add and remove information with a click, as opposed to having to create a whole separate map as when working on paper maps. In addition, they were able to immediately see the value in the visibility analysis tools and the information that they didn't previously have available while searching. They were pleased to see the detail in the analysis results, clearly highlighting areas seen and not seen during their search routes, and similarly identifying possible beacon locations and areas where it would not be found.

The team also had some great feedback on the tools. Although the dialogs were useable, they realized some enhancements that would make the system easier for them to use. One such enhancement was being able to click on the map to enter a clue, not just use coordinates that were radioed back to the command center. It also became clear that it was hard to tell when tools were running, and as a result wait cursors (such as the familiar hourglass) were added to the tools to indicate the application was working.

Some miscommunication between the team and the author were also discovered: they wanted to be able to enter UTM coordinates, not just latitude and longitude as the author originally understood. In addition, they wanted to have the coordinates (both UTM and latitude/longitude) recorded in the attribute tables to make it easier to retrieve that information. The team also expressed a desire to be able to enter the signal direction of the beacon readings in either true or magnetic north, specifying an offset to true north when providing the direction as magnetic north. When entering search routes, the team had another parameter they wanted to be able to specify: probability of detection, which they use to indicate the chance that if the victim was along the route he would have been seen. They also had some more minor enhancement requests once they saw the system as a whole, including: (1) the use of military time in naming and dialogs, and (2) displaying the coordinates in the status bar without decimal places. This feedback was incorporated into the final version of the GIS tools.

While they were pleased with the tools presented, the team also showed their enthusiasm for the next phase of spatially enabling their operations. They had a number of suggestions on how to enhance the application moving forward, and these will be presented in the next chapter.

6.3 An Example Search Using the Search and Rescue GIS

To understand how the team will work with the system, let's look at a simulated search. This simulation focuses on the results the team will get from using each tool. When the team commences a new search, they will start by opening ArcMap and clicking the New Search Map File command. This will create their search geodatabase and MXD named by the time the search starts. In this example, the search started on November 17, 2009, at 00:58:32 and is based in Sierra Madre. Thus their search geodatabase is named Search2009-11-17-00-58-32_SierraMadre.gdb, and the search MXD is Search2009-11-17-00-58-32_SierraMadre.mxd. It is built on the provided base data, and when it opens it looks like Figure 6-1.

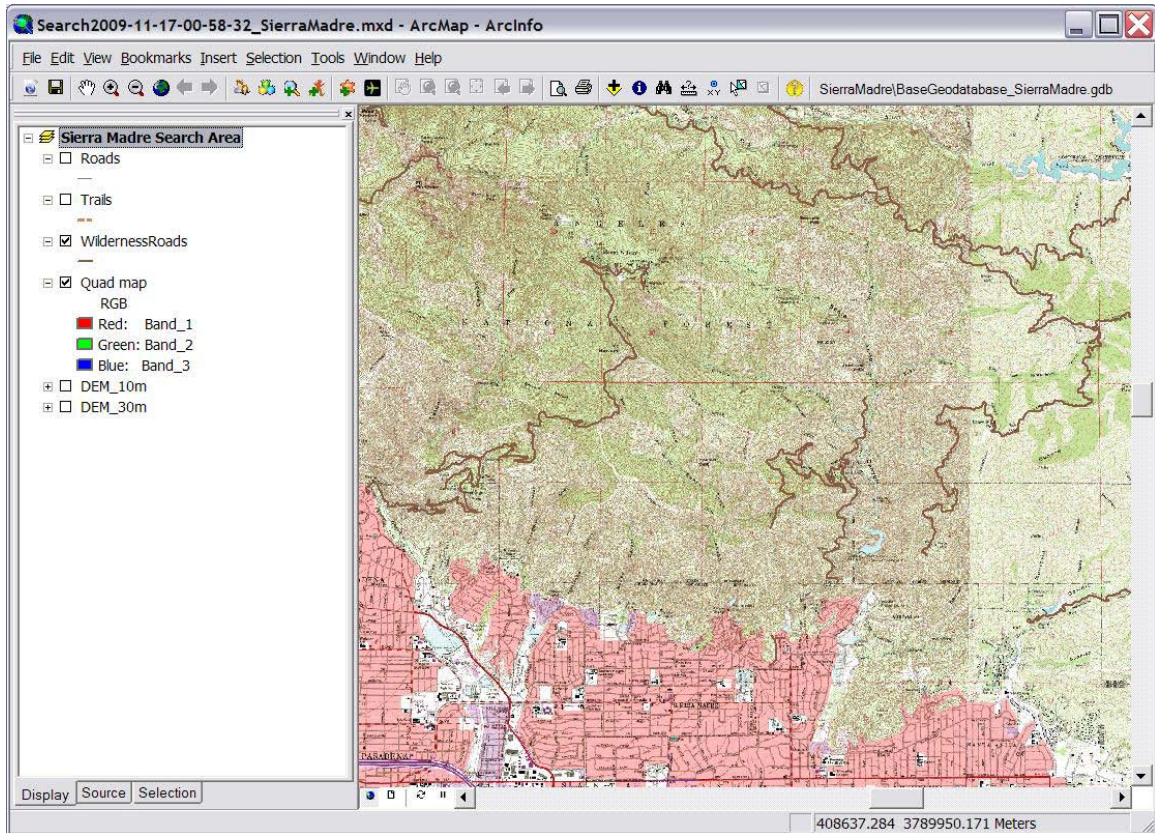


Figure 6-1: The New Search MXD.

While one searcher at the command center set up the GIS for this search, another divided the team into three teams, A, B, and C, and provided them with initial search directions. Once the teams had names, the Add Team command was used to add the names into the GIS.

The search area also had to be divided into areas of interest, which the team did by digitizing the regions on the map using the Define Search Region tool. The feature dataset and feature class were automatically created, and the feature class was loaded to the map. Once on the map, a region's name is visible when the mouse is over the feature (Figure 6-2).

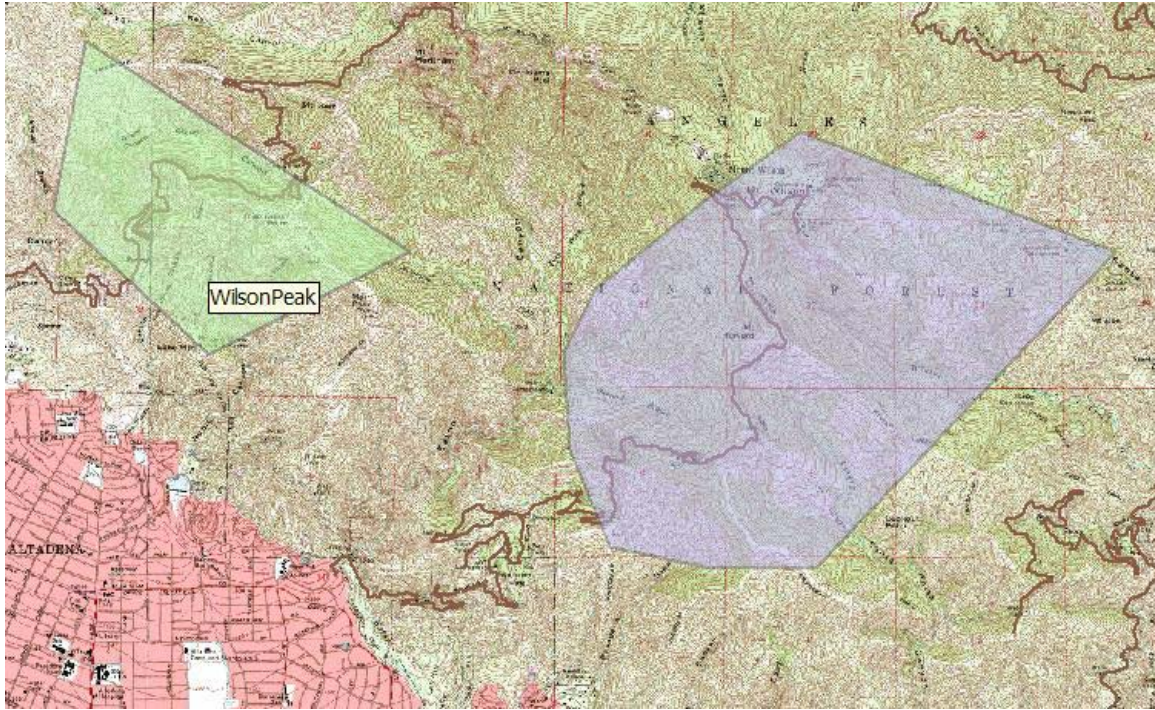


Figure 6-2: Example Search Regions.

Soon the teams were at their tasks in the field, and teams were sending clues back to the operations center. Added to the map, the clues present as magnifying glasses, and hovering over them displays the description of the clue (Figure 6-3).

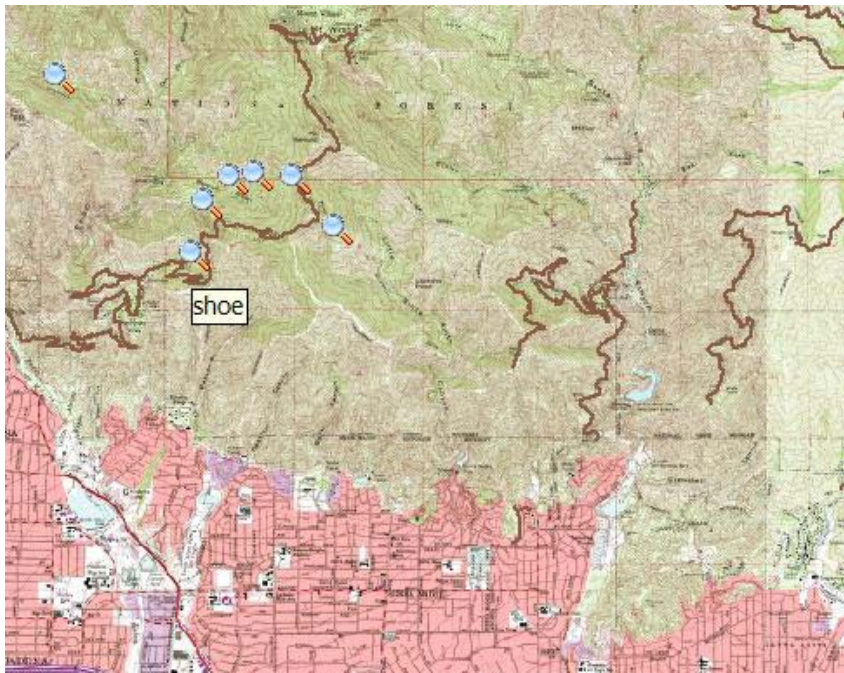


Figure 6-3: Example Clues.

Other teams radioed in that they had followed their assigned search routes, and those were added to the map. When the routes were added, the visibility of the route was also calculated and added to the map (Figure 6-4). The darker the orange area presented in the route visibility analysis, the more times the location was seen while following the route.



Figure 6-4: Example Search Routes.

In this search, an aircraft was missing, and some of the search teams were out taking readings on the beacon signal. Those readings were radioed back to the command center, added to the map, and analyzed to locate where to search for the beacon, and thus the crash site (Figure 6-5). The darker areas indicate a higher probability of containing the beacon, and hovering over a beacon shows the angle at which the signal is strongest from that location.

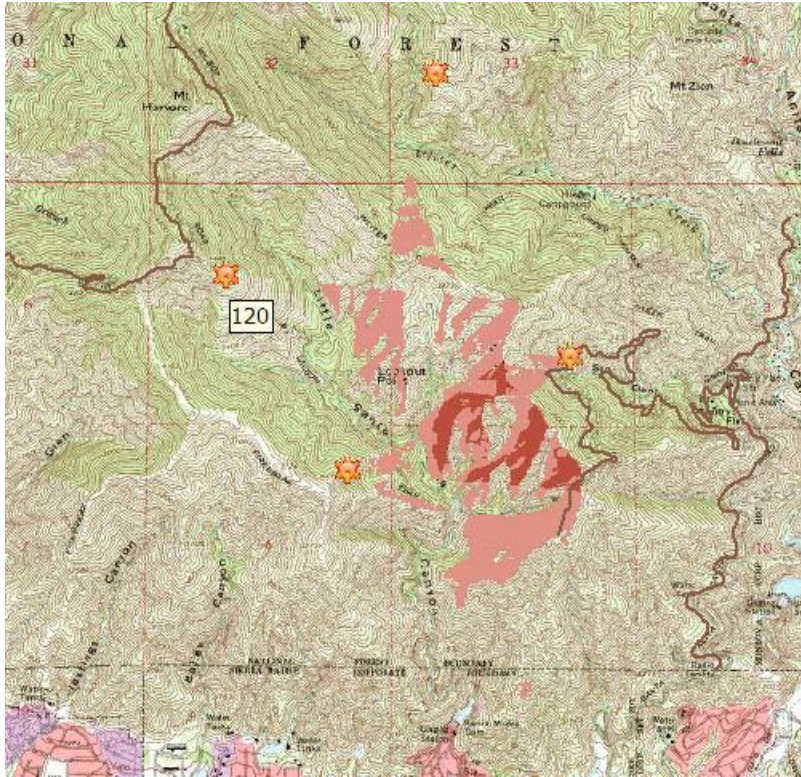


Figure 6-5: Example Beacon Readings and Their Beacon Location Prediction.

6.4 Summary

Although the developed GIS system had some areas that needed work before it would be ready for use, the members of Sierra Madre Search and Rescue wanted to get the beta version of the GIS onto their computer so that they could begin familiarizing with it. The team provided useful feedback upon seeing the beta version of the system, and all the suggestions were incorporated into the final version of the system or documented for enhancements in the next phase. The success of the project depended on how well the system integrated into the workflow of the searchers and encouraged its use. By incorporating the team's feedback, it became easier for the team to integrate the system to their workflow and therefore the projected reached the goal. Yet the real value of this project for the team has been having a GIS consultant: someone to ask their questions of, and someone to help introduce them to the overwhelming system. The focus transitioned to a sharing of knowledge. The system created for the team was just one delivery mechanism for that information.

Chapter 7 – Conclusions and Future Work

Over the course of this project, Sierra Madre Search and Rescue became familiar with GIS and now is incorporating it into their searches. They knew that GIS could enhance the knowledge available during their searches, and now that information is a part of their searches. They have a toolbar that provides a simple GIS experience and provides additional information during their major searches. The first of the main tools analyzes the visibility coverage of their search routes, and the second tracks and integrates together beacon readings taken when searching for downed aircraft. While this does differ from the original project of providing GPS data management and integration, it does satisfy the overarching goal of introducing SMSR to GIS and starting them on the path to further workflow enhancements.

Although Sierra Madre Search and Rescue had an application before, they were unable to successfully use it as it disrupted their workflow. For example, one person could enter data, but other team members were unable to analyze the data already in place during the data entry. Now when a team is searching in the field, one team member can be entering data at one workstation while other team members are printing maps of that data and running analysis tasks on the data already in the system. Using file geodatabases for simultaneous editing and reading will rarely result in an error for the reader. However, waiting a couple seconds and trying the read again will succeed. Having this information available the team can make better informed decisions about how an operation should proceed and can better do their job—saving lives.

The current custom toolbar and experience is just the beginning for the team. When SMSR increases their GIS familiarity in the future, they can continue to develop their geospatial tools and workflow. In particular the following future extensions of this system would benefit the team:

1. Enhance the Add Route tool: When digitizing a route on the map it would be a nicer interface if the drawing was continuous while moving the mouse with the button held down.
2. Enhancing the Add Clue and Add Beacon Reading tools: When using the Add Clue or Add Beacon tool, it would help the team to place the beacon by clicking if they could see the elevation of the current mouse cursor location. In a future version of these tools the status bar will present the elevation. In addition, the tool could provide better assistance to the users by reporting coordinates outside of the base data area.
3. Incorporating GPS: With this system the team is more equipped to incorporate GPS devices. They will need to develop a workflow for GPS data collection in the field during operations, including how to make use of the data both during the search and in the later documentation of the search.
4. Incorporating other geospatial technologies such as GPS-enabled radios: Data entry would be simplified if the team were to incorporate GPS-enabled devices where the data from those devices on transmission were to populate the database without the manual entry steps.
5. Multi-editor geodatabase: In major searches, it is foreseeable that the team would want to be able to have multiple members in the control center entering search

data into the database. The team would need to look at moving to a multi-editor geodatabase to make that possible.

6. Improving the symbolization: While the analysis shows what the team members could have seen, and where the beacons are predicted to be coming from, there is a measure of uncertainty in those readings. To make this more apparent, it would be useful if the system were to make use of the methods for displaying uncertainty investigated in Payne's project (2009).
7. Using beacon signal strength: Currently all beacons are treated equally if they are determined to be significant readings. Their significance is only determined based on the angles of separation from other readings. However, each beacon reading has a signal strength (very weak, weak, moderate, strong, and very strong) that is not currently used in the location analysis. The signal strength should be a factor in both determining the significant readings and determining the areas of interest for further searching. In addition, locations where readings indicate that there is no beacon signal should be accounted for in the analysis.
8. Multi-source beacon analysis: Cases of missing aircraft are not always a single missing aircraft. For instance, if there is a collision, there would be multiple missing aircrafts and multiple beacons as a result. The beacon tool could be enhanced in the future to manage readings taken for multiple beacon sources, and provide multiple areas of interest for detailed searching.
9. Collecting base data for additional regions: Without base data for the search region, the system cannot be used to assist the team in finding the victim. By creating Base geodatabases for the areas in which the team is most often called to help, the team will have all the information required to best handle a situation, even if they are assisting in an area with which they are less familiar.

While more enhancements are expected for the future, Sierra Madre Search and Rescue is now GIS-enabled: they understand the value and the data, and they have a set of tools that they can use for their searches. With this GIS system, SMSR has a better chance of successfully saving the lives in their hands.

Works Cited

- Abi-Zeid, I., & Frost, J. (2005). SARPlan: A decision support system for Canadian search and rescue operations. *European Journal of Operational Research*, 162, 630-653. doi:10.1016/j.ejor.2003.10.029
- Adams, A., Schmidt, T., Newgard, C., Federiuk, C., Christie, M., Scorvo, S., et al. (2007). Search is a time-critical event: when search and rescue missions may become futile. *Wilderness and Environmental Medicine*, 18(2), 95-101. doi:10.1580/06-WEME-OR-035R1.1
- Campbell, H., & Masser, I. (1995). *GIS and organizations*. Pennsylvania: Taylor & Francis.
- Cutter, S. (2003). GI science, distasters, and emergency management. *Transactions in GIS*, 7(4), 439-445. doi:10.1111/1467-9671.00157
- DigitalGlobe. (2008). Search and rescue – getting where from here – Case study draft 2008-113 San Bernardino SAR. Retrieved February 6, 2008, from http://www.esri.com/partners/alliances/digitalglobe/digitalglobe_search-rescue.pdf
- Ela, G. (2004). Epidemiology of wilderness search and rescue in New Hampshire, 1999-2001. *Wilderness and Environmental Medicine*, 15(1), 11-17. Retrieved February 6, 2009 from the Wilderness Medical Society web site: <http://www.wemjournal.org/pdfserv/i1080-6032-015-01-0011.pdf>
- Ferguson, D. (2008, February). *GIS for wilderness search and rescue*. Paper presented at the 2008 ESRI Federal User Conference, Washington, D.C.
- Gilfoyle, I., & Thorpe, P. (2004). *Geographic information management in local government*. London: CRC Press.
- Goodrich, M., Morse, B., Gerhardt, D., Cooper, J., Quigley, M., Adams, J., & Humphrey, C. (2008). Supporting wilderness search and rescue using a camera-equipped mini UAV. *Journal of Field Robotics*, 25(1), 89-110. doi:10.1002/rob.20226
- Heggie, T., & Heggie, M. (2009). Search and rescue trends associated with recreational travel in US National Parks. *Journal of Travel Medicine*, 16 (1), 23-27. doi:10.1111/j.1708-8305.2008.00269.x
- Howlett, E., & Spaulding, M. (2004). SAROPS: The next generation search and rescue optimal planning system for the US Coast Guard [PowerPoint slides]. Retrieved February 6, 2009, from the LFremer web site: <http://www.ifremer.fr/web-com/stw2004/sar/presentations/spaulding.ppt>
- Hung, E., & Townes, D. (2007). Search and rescue in Yosemite National Park: A 10-year review. *Wilderness and Environmental Medicine*, 18(2), 111-116. doi:10.1580/06-WEME-OR-022R1.1
- Johnson, R. (2003). Persistence and technology save hiker. *ArcUser Online*. Retrieved February 6, 2009, from <http://www.esri.com/news/arcuser/0703/search1of2.html> and <http://www.esri.com/news/arcuser/0703/search2of2.html>
- National Research Council of the National Academies (2007). *Successful response starts with a map: Improving geospatial support for disaster management*. Washington, D.C.: The National Academies Press.
- Patterson, T. (2008a). GIS Aids in search for man lost in California's rugged San Bernardino Mountains. *ArcWatch*. Retrieved February 6, 2009, from <http://www.esri.com/news/arcwatch/0408/feature.html>

- Patterson, T. (2008b). GIS aids in search for lost hiker. *GovTech*. Retrieved February 6, 2009, from the GovTech web site: <http://www.govtech.com/em/articles/4043777>
- Payette, R., & Wood, S. (1997). *Search and rescue planning – a GIS solution*. Paper presented at the Seventeenth Annual ESRI User Conference, San Diego, CA.
- Payne, J. (2009). *GIS Tools for Cartographic Representation of Spatial Data Uncertainty*. Paper presented at the North American Cartographic Information Society 2009 Annual Meeting, Sacramento, CA.
- Radke, J., Cova, T., Sheridan, M., Troy, A., Lan, M., & Johnson, R. (2000). Application challenges for geographic information science: implications for research, education, and policy for emergency preparedness and response. *URISA Journal*, 12(2). Retrieved February 6, 2009, from the URISA web site: <http://www.urisa.org/files/WigginsVol12No2-4.pdf>
- Rodriguez-Rodriguez, A., & Aleman-Flores, M. (2001). *A framework for the search and rescue domain*. Paper presented at the 14th International Conference of Applications of Prolog, Tokoyo, Japan. Retrieved February 6, 2009, from http://www.ifcomputer.co.jp/inap/inap2001/program/sol_rodriguez.pdf
- Sayda, F., Wittmann, E., Kandawasvika, A., & Wang, F. (2005, July). *PARAMOUNT – A LBS prototype for hikers*. Paper presented at the Twenty-fifth Annual ESRI User Conference, San Diego, CA.
- Schmidt, T., Federiuk, C., Zechnich, A., Forsythe, M., Christie, M., & Andrews, C. (1996). Advanced life support in the wilderness: 5-year experience of the Reach and Treat Team. *Wilderness and Environmental Medicine*, 7(3), 208-215. Retrieved February 6, 2009, from the Wilderness Medical Society web site: <http://www.wemjournal.org/pdfserv/i1080-6032-007-03-0208.pdf>
- Schvartz, S., Abi-Zeid, I., & Tourigny, N. (2007). Knowledge engineering for modeling reasoning in a diagnosis task: application to search and rescue. *Canadian Journal of Administrative Sciences*, 24, 196-211. doi:10.1002/CJAS.24
- Soylemez, E., & Usul, N. (2006). *Utility of GIS in search and rescue operations*. Paper presented at the Twenty-sixth Annual ESRI User Conference, San Diego, CA.
- Stepanek, J., & Claypool, D. (1997). GPS signal reception under snow cover: A pilot study establishing the potential usefulness of GPS in avalanche search and rescue operations. *Wilderness and Environmental Medicine*, 8(2), 101-104. Retrieved February 6, 2009, from the Wilderness Medical Society web site: <http://www.wemjournal.org/pdfserv/i1080-6032-008-02-0101.pdf>

Appendix A. Application Source Code

This section contains some of the key sections of the code used to create this application. The formatting of this section, while close to that required by Visual Studio, does introduce line breaks that will stop the code from compiling. Long lines of code broken across multiple lines in this appendix will need to be placed on a single line for the code to compile.

A.1 The Toolbar

The following code places the commands and tools onto the Search and Rescue toolbar:

```
public Toolbar()
{
    // Map document tools
    AddItem("SearchAndRescue.NewSearchMapFile"); // new Search Map
File
    AddItem("esriArcMapUI.MxFileMenuItem", 3); // Save

    // ArcMap map navigation tools: pan, zoom, full extent
    BeginGroup();
    AddItem("esriArcMapUI.PanTool");
    AddItem("esriArcMapUI.ZoomInTool");
    AddItem("esriArcMapUI.ZoomOutTool");
    AddItem("esriArcMapUI.FullExtentCommand");
    AddItem("esriArcMapUI.ZoomToLastExtentBackCommand");
    AddItem("esriArcMapUI.ZoomToLastExtentForwardCommand");

    // add data tools
    BeginGroup();
    AddItem("SearchAndRescue.AddTeam"); // add a team
    AddItem("SearchAndRescue.DefineSearchRegion"); // define a search
region
    AddItem("SearchAndRescue.AddClue"); // add clue
    AddItem("SearchAndRescue.AddRoute"); // add route

    // Beacon reading tools
    BeginGroup();
    AddItem("SearchAndRescue.AddBeaconReading"); // add beacon reading
    AddItem("SearchAndRescue.BeaconLocationCalc"); // Beacon location
analysis

    // ArcMap PageLayout navigation tools
    BeginGroup();
    AddItem("esriArcMapUI.PagePanTool");
    AddItem("esriArcMapUI.PageZoomInTool");
    AddItem("esriArcMapUI.PageZoomOutTool");
    AddItem("esriArcMapUI.ZoomWholePageCommand");
    AddItem("esriArcMapUI.ZoomPageToLastExtentBackCommand");
    AddItem("esriArcMapUI.ZoomPageToLastExtentForwardCommand");

    // Print
    BeginGroup();
```

```

AddItem("esriArcMapUI.MxFileMenuItem", 6); // File_PrintPreview
AddItem("esriArcMapUI.MxFileMenuItem", 7); // File_Print

// data tools
BeginGroup();
AddItem("esriArcMapUI.AddDataCommand");
AddItem("esriArcMapUI.IdentifyTool");
AddItem("esriControls.ControlsMapFindCommand");
AddItem("esriControls.ControlsMapMeasureTool");
AddItem("esriControls.ControlsMapGoToCommand");
AddItem("esriArcMapUI.SelectFeaturesTool");
AddItem("esriArcMapUI.ClearSelectionCommand");

// Help
BeginGroup();
AddItem("SearchAndRescue.LaunchHelp");

// Base geodatabase selection
BeginGroup();
AddItem("SearchAndRescue.BaseGeodatabaseLabel");
}

```

A.2 New Search Map File Command

The following code is run when the command is clicked:

```

public override void OnClick()
{
    // create a copy of the Search MXD for a new search.
    // give a default name based on the date and time. If it already
    // exists, append a sequential number to it
    DateTime searchDateStart = DateTime.Now;
    string month = "";
    if (searchDateStart.Month < 10)
        month += "0";
    month += searchDateStart.Month;
    string day = "";
    if (searchDateStart.Day < 10)
        day += "0";
    day += searchDateStart.Day;
    string hours = "";
    if (searchDateStart.Hour < 10)
        hours += "0";
    hours += searchDateStart.Hour;
    string minutes = "";
    if (searchDateStart.Minute < 10)
        minutes += "0";
    minutes += searchDateStart.Minute;
    string seconds = "";
    if (searchDateStart.Second < 10)
        seconds += "0";
    seconds += searchDateStart.Second;
    string searchDateString = searchDateStart.Year + "-" + month + "-"
+ day + "-" + hours + "-" + minutes + "-" + seconds;

    // set the starting base geodatabase

```

```

    string regionFolder = SetBaseGDB();
    if (String.IsNullOrEmpty(regionFolder))
        return;

    string regionString = regionFolder.Substring(0,
regionFolder.IndexOf(@"\"));
    string newSearchMapFileName = System.IO.Path.Combine(@"C:\Program
Files\GISearcher\Data\Searches", "Search" + searchDateString + "_" +
regionString + ".mxd");
    string gdbName =
System.IO.Path.Combine(System.IO.Path.GetDirectoryName(newSearchMapFile
Name), System.IO.Path.GetFileNameWithoutExtension(newSearchMapFileName)
+ ".gdb");
    int count = 2;
    while (System.IO.File.Exists(newSearchMapFileName) ||
System.IO.Directory.Exists(gdbName))
    {
        newSearchMapFileName =
System.IO.Path.Combine(System.IO.Path.GetDirectoryName(newSearchMapFile
Name), System.IO.Path.GetFileNameWithoutExtension(newSearchMapFileName)
+ "_" + count + ".gdb");
        gdbName =
System.IO.Path.Combine(System.IO.Path.GetDirectoryName(gdbName),
System.IO.Path.GetFileNameWithoutExtension(gdbName) + "_" + count +
".gdb");
        ++count;
    }

    // get the MXD from that base region folder and copy it, if it
exists
    if (!String.IsNullOrEmpty(regionFolder))
    {
        string[] mxds =
System.IO.Directory.GetFiles(GeodatabaseSupport.BaseDataFolder + @"\" +
System.IO.Path.GetDirectoryName(regionFolder), "*.mxd",
System.IO.SearchOption.TopDirectoryOnly);
        if (mxds.Length >= 1)
        {
            If
(!System.IO.Directory.Exists(System.IO.Path.GetDirectoryName(newSearchM
apFileName)))

System.IO.Directory.CreateDirectory(System.IO.Path.GetDirectoryName(new
SearchMapFileName));
            System.IO.File.Copy(mxds[0], newSearchMapFileName);
        }
    }

    // Create the Search Geodatabase
    // Instantiate a file geodatabase workspace factory and create a
file geodatabase.
    // The Create method returns a workspace name object.
    try
    {
        Type t =
Type.GetTypeFromProgID("esriDataSourcesGDB.FileGDBWorkspaceFactory");

```



```

System.Object obj = Activator.CreateInstance(t);
IWorkspaceFactory workspaceFactory = obj as IWorkspaceFactory;

IWorkspaceName workspaceName =
workspaceFactory.Create(System.IO.Path.GetDirectoryName(gdbName),
System.IO.Path.GetFileName(gdbName), null, 0);
GeodatabaseSupport.SearchWorkspace =
GeodatabaseSupport.OpenGDB(workspaceName.PathName);
// set the base geodatabase relative path (regionFolder) for
the search GDB

GeodatabaseSupport.CreateAndUpdateSearchDomain(GeodatabaseSupport.BaseG
eodatabaseDomainName, regionFolder);

// load the mxd (after creating one if none to copy)
if (!System.IO.File.Exists(newSearchMapFileName))
{
    string searchAndRescueMxT = @"c:\Program
Files\GISearcher\SearchMapFile.mxt";
    m_application.NewDocument(false, searchAndRescueMxT);
    m_application.SaveDocument(newSearchMapFileName);
}
m_application.OpenDocument(newSearchMapFileName);
}
catch (Exception e)
{
    System.Windows.Forms.MessageBox.Show(e.Message);
}
}

```

A.3 Add Team Command

The following code is run when the Add Team button is clicked:

```

private void AddTeamButton_Click(object sender, EventArgs e)
{
    // set busy cursor
    this.Cursor = Cursors.WaitCursor;

    try
    {
        // create/update domain
        GeodatabaseSupport.DomainCreationStatus status =
GeodatabaseSupport.CreateAndUpdateSearchDomain(GeodatabaseSupport.TeamD
omainName, teamNameMTB.Text);
        if (status ==
GeodatabaseSupport.DomainCreationStatus.VALUE_EXISTED)
        {
            System.Windows.Forms.MessageBox.Show("Team already
exists.", "Duplicate entry", MessageBoxButtons.OK,
MessageBoxIcon.Warning);
            teamNameMTB.Select();
            teamNameMTB.SelectAll();
            return;
        }
    }
}

```

```

catch (Exception ex)
{
    System.Windows.Forms.MessageBox.Show(ex.Message);
}
teamNameMTB.Text = "";
teamNameMTB.Select();
this.Close();

// cursor back to default
this.Cursor = Cursors.Default;
}

```

A.4 Define Search Region Tool

The following code is run with the Define Search Region tool is selected and the mouse is clicked on the map:

```

public override void OnMouseDown(int Button, int Shift, int X, int Y)
{
    // get the active view
    ESRI.ArcGIS.ArcMapUI.IMxDocument mxDocument =
    (ESRI.ArcGIS.ArcMapUI.IMxDocument)m_application.Document;
    // create a collection of the points of the polyline
    defineSearchRegionDialog.SearchRegion =
    DrawPolygon(mxDocument.ActiveView);

    defineSearchRegionDialog.UpdateOperationalPeriods();
    defineSearchRegionDialog.ShowDialog();
}

```

The following code tracks the polygon drawn on the screen:

```

public IPolygon DrawPolygon(ESRI.ArcGIS.Carto.IActiveView activeView)
{
    if (activeView == null)
    {
        return null;
    }

    ESRI.ArcGIS.Display.IScreenDisplay screenDisplay =
    activeView.ScreenDisplay;

    // Constant.
    screenDisplay.StartDrawing(screenDisplay.hDC, (System.Int16)
    ESRI.ArcGIS.Display.esriScreenCache.esriNoScreenCache); //
Explicit Cast
    ESRI.ArcGIS.Display.IRgbColor rgbColor = new
    ESRI.ArcGIS.Display.RgbColorClass();
    rgbColor.Red = 255;

    ESRI.ArcGIS.Display.IColor color = rgbColor; // Implicit cast.
    ESRI.ArcGIS.Display.ISimpleFillSymbol simpleFillSymbol = new
    ESRI.ArcGIS.Display.SimpleFillSymbolClass();
    simpleFillSymbol.Color = color;
}

```

```

    ESRI.ArcGIS.Display.ISymbol symbol = simpleFillSymbol as
        ESRI.ArcGIS.Display.ISymbol; // Dynamic cast.
    ESRI.ArcGIS.Display.IRubberBand rubberBand = new
        ESRI.ArcGIS.Display.RubberPolygonClass();
    ESRI.ArcGIS.Geometry.IGeometry geometry =
    rubberBand.TrackNew(screenDisplay,
        symbol);
    screenDisplay.SetSymbol(symbol);
    screenDisplay.DrawPolygon(geometry);
    screenDisplay.FinishDrawing();

    return (IPolygon)geometry;
}

```

The following code is run when the Define Region button is clicked:

```

private void defineRegionButton_Click(object sender, EventArgs e)
{
    // set to busy cursor
    this.Cursor = Cursors.WaitCursor;

    try
    {
        // add the feature drawn as a featureclass to the geodatabase -
        - in a feature dataset for the operational period
        string featureDatasetName = "SearchRegions";
        IFeatureDataset featureDataset =
        GeodatabaseSupport.CreateAndOpenSearchFeatureDataset(featureDatasetName
        );
        string featureClassName = "OpPeriod" +
        operationalPeriodCB.SelectedItem + "_Regions";
        IFeatureClass featureClass =
        GeodatabaseSupport.CreateAndOpenSearchFeatureClass(featureClassName,
        featureDataset, esriGeometryType.esriGeometryPolygon, fieldParams);

        // create the line feature & set its info based on the dialog
        IFeature feature = featureClass.CreateFeature();
        feature.Shape = SearchRegion;

        feature.set_Value(featureClass.FindField("REGION_NAME"),
        searchRegionNameTB.Text);
        feature.set_Value(featureClass.FindField("POPULARITY"),
        popularityTB.Text);
        feature.Store();

        // add to the map if needed
        ILayer searchRegionLayer =
        MappingSupport.AddLayerToMapIfMissing(m_application, featureClass,
        null);
        IFeatureRenderer uniqueFillSymbol =
        MappingSupport.MakeUniqueValuePolygonRenderer(searchRegionLayer,
        "REGION_NAME", 50);
        MappingSupport.RenderLayer(searchRegionLayer,
        uniqueFillSymbol);
        ((IFeatureLayer)searchRegionLayer).DisplayField =
        "REGION_NAME";
    }
}

```

```

searchRegionLayer.ShowTips = true;

// refresh the display
MappingSupport.Refresh(m_application);
// refresh the TOC
((IMxDocument)m_application.Document).UpdateContents();

// close the dialog
this.Close();
}
catch (Exception ex)
{
    System.Windows.Forms.MessageBox.Show(ex.Message);
    this.Cursor = Cursors.Default;
}
searchRegionNameTB.Clear();
searchRegionNameTB.Select();
popularityTB.Clear();
this.Close();

// set back to default cursor
this.Cursor = Cursors.Default;
}

```

A.5 Add Clue Tool

The following code is run when the Add Clue tool is selected, coordinates are being used to enter the point, and the map is clicked on:

```

public override void OnMouseDown(int Button, int Shift, int X, int Y)
{
    if (addClueDialog.GetByClicking)
    {
        // get the active view
        ESRI.ArcGIS.ArcMapUI.IMxDocument mxDocument =
        (ESRI.ArcGIS.ArcMapUI.IMxDocument)m_application.Document;
        // get the coords of the clicked point
        IPoint screenCoordsPoint = new PointClass();
        screenCoordsPoint.PutCoords(X, Y);
        IPoint mapCoordsPoint =
        MappingSupport.GetMapCoordinatesFromScreenCoordinates(screenCoordsPoint
        , ((IMxDocument)m_application.Document).ActiveView);
        addClueDialog.SetUTM(mapCoordsPoint.X, mapCoordsPoint.Y);
        addClueDialog.ShowDialog();
    }
}

```

The following code is run when the Add Clue button is clicked:

```

private void addClueButton_Click(object sender, EventArgs e)
{
    // set to busy cursor
    this.Cursor = Cursors.WaitCursor;

    try

```

```

    {
        if (UTM_RB.Checked) // validate the UTM entries
        {
            if (!MappingSupport.ValidUTMCoordinates(utmXTB, utmYTB,
utmZoneTB))
                return;
        }
        else // validate the lat/long options if lat/long entered or
coords gotten by clicking
        {
            if (!MappingSupport.ValidLatitudeLongitude(latDegTB,
latMinTB, latSecTB, longDegTB, longMinTB, longSecTB))
                return;
        }
        if
(String.IsNullOrEmpty(operationalPeriodCB.SelectedItem.ToString()))
        {
            System.Windows.Forms.MessageBox.Show("An operational period
must be provided to enter a clue.", "Missing operational period",
MessageBoxButtons.OK, MessageBoxIcon.Error);
            operationalPeriodCB.Select();
            return;
        }

        // get the coordinates (lat, long, utm)
        double latitude_decimalDegrees, longitude_decimalDegrees,
utm_x, utm_y;
        int zone = -1;
        bool isSouthernHemisphere;
        if (UTMPanel.Visible) // utm coords are provided
        {
            utm_x = System.Convert.ToDouble(utmXTB.Text);
            utm_y = System.Convert.ToDouble(utmYTB.Text);
            zone = System.Convert.ToInt32(utmZoneTB.Text);
            double latitude_radians, longitude_radians;
            if (!CoordinateConversion.UTMXYToLatLong(utmXTB.Text,
utmYTB.Text, utmZoneTB.Text, southernHemisphereCheckBox.Checked, out
latitude_radians, out longitude_radians))
                return;
            latitude_decimalDegrees =
CoordinateConversion.RadiansToDecimalDegrees(latitude_radians);
            longitude_decimalDegrees =
CoordinateConversion.RadiansToDecimalDegrees(longitude_radians);
        }
        else // lat/long are provided
        {
            latitude_decimalDegrees =
CoordinateConversion.DegreesMinSecToDecimalDegrees(System.Convert.ToInt
32(latDegTB.Text), System.Convert.ToInt32(latMinTB.Text),
System.Convert.ToInt32(latSecTB.Text));
            longitude_decimalDegrees =
CoordinateConversion.DegreesMinSecToDecimalDegrees(System.Convert.ToInt
32(longDegTB.Text), System.Convert.ToInt32(longMinTB.Text),
System.Convert.ToInt32(longSecTB.Text));
            if
(!CoordinateConversion.LatLongToUTMXY(longitude_decimalDegrees.ToString

```

```

        (, latitude_decimalDegrees.ToString(), out utm_x, out utm_y, out zone,
        out isSouthernHemisphere))
            return;
    }

    // Check for feature dataset existence and create if doesn't
    exist
    IFeatureDataset featureDataset =
    GeodatabaseSupport.CreateAndOpenSearchFeatureDataset("SearchClues");

    // check for feature class existence and create if doesn't
    exist
    string featureClassName = "Clues_OpPeriod_" +
    operationalPeriodCB.SelectedItem;
    IFeatureClass featureClass =
    GeodatabaseSupport.CreateAndOpenSearchFeatureClass(featureClassName,
    featureDataset, esriGeometryType.esriGeometryPoint,
    requiredFieldParameters);

    // create a point feature
    IPoint cluePoint = new PointClass();
    cluePoint.PutCoords(utm_x, utm_y); // use the utm coords
    since our base data will always be in UTM

    IFeature feature = featureClass.CreateFeature();
    feature.Shape = cluePoint;
    feature.set_Value(featureClass.FindField("TEAM"),
    teamNameCB.SelectedItem);
    feature.set_Value(featureClass.FindField("DESCRIPTION"),
    clueDescriptionTB.Text);
    feature.set_Value(featureClass.FindField("LATITUDE_DMS"),
    CoordinateConversion.DecimalDegreesToDegreesMinSec(latitude_decimalDegr
    ees));
    feature.set_Value(featureClass.FindField("LONGITUDE_DMS"),
    CoordinateConversion.DecimalDegreesToDegreesMinSec(longitude_decimalDeg
    rees));
    feature.set_Value(featureClass.FindField("UTM_X"), utm_x);
    feature.set_Value(featureClass.FindField("UTM_Y"), utm_y);
    feature.set_Value(featureClass.FindField("UTM_ZONE"), zone);
    feature.Store();

    // add to the map if needed
    IFeatureRenderer symbolRenderer =
    MappingSupport.MakePictureSymbolRenderer("c:\\Program
    Files\\GISearcher\\symbols\\AddClueSymbol.bmp", 12);
    ILayer cluesLayer =
    MappingSupport.AddLayerToMapIfMissing(m_application, featureClass,
    symbolRenderer);
    // also set map tips to display the clue descriptions
    ((IFeatureLayer)cluesLayer).DisplayField = "DESCRIPTION";
    cluesLayer.ShowTips = true;

    // refresh the map
    MappingSupport.Refresh(m_application);
}
catch (Exception ex)
{

```

```

        System.Windows.Forms.MessageBox.Show(ex.Message);
        this.Cursor = Cursors.Default;
    }
    utmXTB.Clear();
    utmYTB.Clear();
    utmZoneTB.Clear();
    southernHemisphereCkBox.Checked = false;
    latDegTB.Clear();
    latMinTB.Clear();
    latSecTB.Clear();
    longDegTB.Clear();
    longMinTB.Clear();
    longSecTB.Clear();
    clueDescriptionTB.Clear();
    this.Close();

    // set cursor back to normal
    this.Cursor = Cursors.Default;

    // set the current tool to the pan tool
    DialogSupport.SetPanAsCurrentTool(m_application);
}

```

A.6 Add Route Tool

The following code is run when the Add Route tool is selected and the mouse is clicked on the map:

```

public override void OnMouseDown(int Button, int Shift, int X, int Y)
{
    // get the active view
    ESRI.ArcGIS.ArcMapUI.IMxDocument mxDocument =
(ESRI.ArcGIS.ArcMapUI.IMxDocument)m_application.Document;
    // create a collection of the points of the polyline
    IPolyline route = DrawPolyline(mxDocument.ActiveView);
    // before using that geometry add a vertex every 10m along the line
    // this improves the viewshed calculation since it only runs at
vertices
    route.Densify(10, 0); // put a vertex every 10 m
    addRouteDialog.Route = route;

    addRouteDialog.UpdateTeamNames();
    addRouteDialog.UpdateOperationalPeriods();
    addRouteDialog.ShowDialog();
}

```

The following code tracks the polyline drawn on the map:

```

public IPolyline DrawPolyline(ESRI.ArcGIS.Carto.IActiveView activeView)
{
    if (activeView == null)
    {
        return null;
    }
}

```

```

    ESRI.ArcGIS.Display.IScreenDisplay screenDisplay =
activeView.ScreenDisplay;

    // Constant.
    screenDisplay.StartDrawing(screenDisplay.hDC,
(System.Int16)ESRI.ArcGIS.Display.esriScreenCache.esriNoScreenCache);
// Explicit Cast
    ESRI.ArcGIS.Display.IRgbColor rgbColor = new
ESRI.ArcGIS.Display.RgbColorClass();
    rgbColor.Red = 255;

    ESRI.ArcGIS.Display.IColor color = rgbColor; // Implicit cast.
    ESRI.ArcGIS.Display.ISimpleLineSymbol simpleLineSymbol = new
ESRI.ArcGIS.Display.SimpleLineSymbolClass();
    simpleLineSymbol.Color = color;

    ESRI.ArcGIS.Display.ISymbol symbol =
(ESRI.ArcGIS.Display.ISymbol)simpleLineSymbol; // Explicit cast.
    ESRI.ArcGIS.Display.IRubberBand rubberBand = new
ESRI.ArcGIS.Display.RubberLineClass();
    ESRI.ArcGIS.Geometry.IGeometry geometry =
rubberBand.TrackNew(screenDisplay, symbol);
    screenDisplay.SetSymbol(symbol);
    screenDisplay.DrawPolyline(geometry);
    screenDisplay.FinishDrawing();

    return (IPolyline)geometry;
}

```

The following code is run when the Add Route button is clicked:

```

private void drawSearchRouteButton_Click(object sender, EventArgs e)
{
    try
    {
        // set to a busy cursor
        this.Cursor = Cursors.WaitCursor;

        // Set the scratchEnvironment env variable
        GP.SetEnvironmentValue("scratchWorkspace",
GeodatabaseSupport.SearchWorkspace.PathName);

        // add the feature drawn as a featureclass to the geodatabase -
- in a feature dataset for the operational period
        string featureDatasetName = "Routes_OpPeriod" +
operationalPeriodCB.SelectedItem;
        IFeatureDataset featureDataset =
GeodatabaseSupport.CreateAndOpenSearchFeatureDataset(featureDatasetName
);
        int routeNum = 1;
        string featureClassName = "OpPeriod" +
operationalPeriodCB.SelectedItem + "_Team" + teamNameCB.SelectedItem +
"_Route" + routeNum;
        while
(GeodatabaseSupport.SearchFeatureClassExists(featureClassName))
        {

```



```

        ++routeNum;
        featureClassName = "OpPeriod" +
operationalPeriodCB.SelectedItem + "_Team" + teamNameCB.SelectedItem +
"_Route" + routeNum;
    }

    IFeatureClass featureClass =
GeodatabaseSupport.CreateAndOpenSearchFeatureClass(featureClassName,
featureDataset, esriGeometryType.esriGeometryPolyline, fieldParams);

    // create the line feature & set its info based on the dialog
    IFeature feature = featureClass.CreateFeature();
    feature.Shape = Route;
    feature.set_Value(featureClass.FindField("TEAM"),
teamNameCB.SelectedItem);
    feature.set_Value(featureClass.FindField("TIME"), timeTB.Text);
    feature.set_Value(featureClass.FindField("VISIBILITY"),
visibilityTB.Text);
    feature.set_Value(featureClass.FindField("RADIUS2"),
visibilityTB.Text);
    feature.set_Value(featureClass.FindField("PROB_OF_DETECTION"),
System.Convert.ToDouble(probOfDetectionTB.Text));
    feature.Store();

    // add to the map if needed
    MappingSupport.AddLayerToMapIfMissing(m_application,
featureClass, null);

    // create the associated viewshed for this route by running the
SearchRouteViewshed tool
    SARTools.SearchRouteViewshed routeViewshed = new
SARTools.SearchRouteViewshed();
    bool process = true;
    if
(GeodatabaseSupport.RasterExists(GeodatabaseSupport.BaseGDBFullPath +
"\DEM_10m"))
        routeViewshed.Search_Area_Terrain_Raster =
GeodatabaseSupport.BaseGDBFullPath + "\DEM_10m";
    else if
(GeodatabaseSupport.RasterExists(GeodatabaseSupport.BaseGDBFullPath +
"\DEM_30m"))
        routeViewshed.Search_Area_Terrain_Raster =
GeodatabaseSupport.BaseGDBFullPath + "\DEM_30m";
    else
    {
        System.Windows.Forms.MessageBox.Show("No terrain raster
exists - 10m or 30m. Can't calculate route visibility.", "No Terrain
Data", MessageBoxButtons.OK, MessageBoxIcon.Error);
        process = false;
    }
    if (process)
    {
        bool extCheckoutNotNeeded = false;
        LicensingSupport.CheckOutSpatialAnalystExtension(ref
extCheckoutNotNeeded);

```

```

        routeViewshed.Search_Route =
System.IO.Path.Combine(System.IO.Path.Combine(GeodatabaseSupport.Search
Workspace.PathName, featureDatasetName), featureClassName);
        string outputRaster =
System.IO.Path.Combine(GeodatabaseSupport.SearchWorkspace.PathName,
featureClassName + "_VIS");
        routeViewshed.Search_Route_Viewshed = outputRaster;
        routeViewshed.Visibility_distance = visibilityTB.Text;
        IGeoProcessorResult gpResult =
(IGeoProcessorResult)GP.Execute(routeViewshed, null);
        if (!extCheckoutNotNeeded)
            LicensingSupport.CheckInSpatialAnalystExtension();

        // remove the buffer temporary feature class
GeodatabaseSupport.DeleteSearchFeatureClass(featureClassName
+ "_Buffe");

        // add the visibility of the route to the map
        if (gpResult.Status ==
ESRI.ArcGIS.esriSystem.esriJobStatus.esriJobSucceeded)
        {
            IRasterDataset rasterDataset =
GeodatabaseSupport.OpenGDBRasterDataset(outputRaster);
            IRasterRenderer uniqueValRasterRenderer =
MappingSupport.MakeClassifyRasterRenderer(rasterDataset);
            MappingSupport.SetRasterLayerRenderer(m_application,
rasterDataset, uniqueValRasterRenderer);
        }
        else
            System.Windows.Forms.MessageBox.Show("Unable to
determine the route visibility.", "Processing Error",
MessageBoxButtons.OK, MessageBoxIcon.Error);

        // refresh the display
MappingSupport.Refresh(m_application);
        // refresh the TOC
((IMxDocument)m_application.Document).UpdateContents();
    }
}
catch (Exception ex)
{
    System.Windows.Forms.MessageBox.Show(ex.Message);
}
timeTB.Clear();
visibilityTB.Clear();
probOfDetectionTB.Clear();
this.Close();

this.Cursor = Cursors.Default;
}

```

A.7 Add Beacon Reading Tool

The following code is run when the Add Beacon Reading tool is selected, coordinates are being used to enter the point, and the map is clicked on:

```

public override void OnMouseDown(int Button, int Shift, int X, int Y)
{
    if (addBeaconReadingDialog.GetByClicking)
    {
        // get the active view
        ESRI.ArcGIS.ArcMapUI.IMxDocument mxDocument =
(ESRI.ArcGIS.ArcMapUI.IMxDocument)m_application.Document;
        // get the coords of the clicked point
        IPoint screenCoordsPoint = new PointClass();
        screenCoordsPoint.PutCoords(X, Y);
        IPoint mapCoordsPoint =
MappingSupport.GetMapCoordinatesFromScreenCoordinates(screenCoordsPoint
, ((IMxDocument)m_application.Document).ActiveView);
        addBeaconReadingDialog.SetUTM(mapCoordsPoint.X,
mapCoordsPoint.Y);
        addBeaconReadingDialog.ShowDialog();
    }
}

```

The following code is run when the Add Beacon Reading button is clicked:

```

private void addBeaconReadingButton_Click(object sender, EventArgs e)
{
    // set to busy cursor
    this.Cursor = Cursors.WaitCursor;
    try
    {
        if (UTM_RB.Checked) // validate the UTM entries
        {
            if (!MappingSupport.ValidUTMCoordinates(utmXTB, utmYTB,
utmZoneTB))
                return;
        }
        else // validate the lat/long options
        {
            if (!MappingSupport.ValidLatitudeLongitude(latDegTB,
latMinTB, latSecTB, longDegTB, longMinTB, longSecTB))
                return;
        }

        if (System.Convert.ToDouble(azimuthTB.Text) < 0 ||
System.Convert.ToDouble(azimuthTB.Text) > 180)
        {
            System.Windows.Forms.MessageBox.Show("Angle of error must
be from 0 to 180.", "Invalid error angle", MessageBoxButtons.OK,
MessageBoxIcon.Error);
            azimuthTB.Select();
            azimuthTB.SelectAll();
            return;
        }
        if (System.Convert.ToDouble(readingHeightTB.Text) < 0)
        {
            System.Windows.Forms.MessageBox.Show("Reading height must
be greater than 0", "Invalid reading height", MessageBoxButtons.OK,
MessageBoxIcon.Error);
            readingHeightTB.Select();
        }
    }
}

```

```

        readingHeightTB.SelectAll();
        return;
    }

    // get the coordinates (lat, long, utm)
    double latitude_decimalDegrees, longitude_decimalDegrees,
utm_x, utm_y;
    int zone = -1;
    bool isSouthernHemisphere;
    if (UTM_RB.Checked) // utm coords are provided
    {
        utm_x = System.Convert.ToDouble(utmXTB.Text);
        utm_y = System.Convert.ToDouble(utmYTB.Text);
        zone = System.Convert.ToInt32(utmZoneTB.Text);
        double latitude_radians, longitude_radians;
        if (!CoordinateConversion.UTMXYToLatLong(utmXTB.Text,
utmYTB.Text, utmZoneTB.Text, southernHemisphereCkBox.Checked, out
latitude_radians, out longitude_radians))
            return;
        latitude_decimalDegrees =
CoordinateConversion.RadiansToDecimalDegrees(latitude_radians);
        longitude_decimalDegrees =
CoordinateConversion.RadiansToDecimalDegrees(longitude_radians);
    }
    else // lat/long are provided
    {
        latitude_decimalDegrees =
CoordinateConversion.DegreesMinSecToDecimalDegrees(System.Convert.ToInt
32(latDegTB.Text), System.Convert.ToInt32(latMinTB.Text),
System.Convert.ToInt32(latSecTB.Text));
        longitude_decimalDegrees =
CoordinateConversion.DegreesMinSecToDecimalDegrees(System.Convert.ToInt
32(longDegTB.Text), System.Convert.ToInt32(longMinTB.Text),
System.Convert.ToInt32(longSecTB.Text));
        if
(!CoordinateConversion.LatLongToUTMXY(longitude_decimalDegrees.ToString
()), latitude_decimalDegrees.ToString(), out utm_x, out utm_y, out zone,
out isSouthernHemisphere))
            return;
    }

    // check for feature class existence and create if doesn't
exist
    string featureClassName = "BeaconReadings";
    IFeatureClass featureClass =
GeodatabaseSupport.CreateAndOpenSearchFeatureClass(featureClassName,
null, esriGeometryType.esriGeometryPoint, requiredFieldParameters);

    // create a point feature
    IPoint beaconReadingPoint = new PointClass();
    beaconReadingPoint.PutCoords(utm_x, utm_y);

    IFeature feature = featureClass.CreateFeature();
    feature.Shape = beaconReadingPoint;
    feature.set_Value(featureClass.FindField("TEAM"),
teamNameCB.SelectedItem);

```

```

        double signalDir =
System.Convert.ToDouble(signalDirectionMTB.Text);
        if (magNorthRB.Checked)
            signalDir += System.Convert.ToDouble(magNorthOffsetMTB);
        feature.set_Value(featureClass.FindField("SIGNAL_DIRECTION"),
signalDir.ToString());
        feature.set_Value(featureClass.FindField("SIGNAL_STRENGTH"),
signalStrengthCB.SelectedItem);
        double lowerAngle = signalDir -
System.Convert.ToDouble(azimuthTB.Text);
        if (lowerAngle < 0)
            lowerAngle += 360;
        feature.set_Value(featureClass.FindField("AZIMUTH1"),
lowerAngle.ToString());
        feature.set_Value(featureClass.FindField("AZIMUTH2"),
((signalDir + System.Convert.ToDouble(azimuthTB.Text)) %
360).ToString());
        feature.set_Value(featureClass.FindField("OFFSETA"),
System.Convert.ToDouble(readingHeightTB.Text));
        feature.set_Value(featureClass.FindField("LATITUDE_DMS"),
CoordinateConversion.DecimalDegreesToDegreesMinSec(latitude_decimalDegr
ees));
        feature.set_Value(featureClass.FindField("LONGITUDE_DMS"),
CoordinateConversion.DecimalDegreesToDegreesMinSec(longitude_decimalDeg
rees));
        feature.set_Value(featureClass.FindField("UTM_X"), utm_x);
        feature.set_Value(featureClass.FindField("UTM_Y"), utm_y);
        feature.set_Value(featureClass.FindField("UTM_ZONE"), zone);

feature.set_Value(featureClass.FindField("INCLUDED_IN_ANALYSIS"), 1);
        feature.Store();

        // add to the map if needed
        IFeatureRenderer symbolRenderer =
MappingSupport.MakePictureSymbolRenderer("c:\\Program
Files\\GISearcher\\symbols\\beaconSymbol.bmp", 12);
        ILayer beaconReadingsLayer =
MappingSupport.AddLayerToMapIfMissing(m_application, featureClass,
symbolRenderer);
        // also set map tips to display the clue descriptions
        ((IFeatureLayer)beaconReadingsLayer).DisplayField =
"SIGNAL_DIRECTION";
        beaconReadingsLayer.ShowTips = true;

        // refresh the map
        MappingSupport.Refresh(m_application);
    }
    catch (Exception ex)
    {
        System.Windows.Forms.MessageBox.Show(ex.Message);
    }
    utmXTB.Clear();
    utmYTB.Clear();
    utmZoneTB.Clear();
    southernHemisphereCkBox.Checked = false;
    latDegTB.Clear();
    latMinTB.Clear();

```

```

latSectTB.Clear();
longDegTB.Clear();
longMinTB.Clear();
longSectTB.Clear();
signalDirectionMTB.Clear();
magNorthOffsetMTB.Clear();
signalStrengthCB.SelectedIndex = 0;
readingHeightTB.Clear();
azimuthTB.Clear();
this.Close();

// cursor back to default
this.Cursor = Cursors.Default;

// set the current tool to the pan tool
DialogSupport.SetPanAsCurrentTool(m_application);
}

```

A.8 Beacon Location Calculation Command

The following code is run when the Find Beacon button is clicked:

```

private void findBeaconButton_Click(object sender, EventArgs e)
{
    // set to busy cursor
    this.Cursor = Cursors.WaitCursor;

    try
    {
        if (System.Convert.ToDouble(sigReadingSeparationTB.Text) < 0)
        {
            System.Windows.Forms.MessageBox.Show("Significant
separation between readings must be greater than 0", "Invalid
significant separation", MessageBoxButtons.OK, MessageBoxIcon.Error);
            sigReadingSeparationTB.Select();
            sigReadingSeparationTB.SelectAll();
            return;
        }
        // check for feature class existence. End if doesn't exist
        string featureClassName = "BeaconReadings";
        IFeatureClass featureClass =
GeodatabaseSupport.OpenSearchFeatureClass(featureClassName);
        if (featureClass == null)
        {
            System.Windows.Forms.MessageBox.Show("No beacon readings
have been entered.", "No readings", MessageBoxButtons.OK,
MessageBoxIcon.Stop);
            sigReadingSeparationTB.Clear();
            this.Close();
            return;
        }

        // get all the angles at which readings were taken by reading
the signal direction field ("SIGNAL_DIRECTION")
        // remove the duplicate angles (based on reading separation
entry)

```

```

        string tempBeaconClassName = "BeaconReadingsProcessed";
        #region select and set all the significant reading features to
INCLUDED_IN_ANALYSIS = 1
        // also set excluded features to INCLUDED_IN_ANALYSIS = 0
        List<int> nonSigReadingOIDs =
NonSignificantBeaconReadingOIDs(featureClass);
        // update the value of that field for all features
        // Use IFeatureClass.Update to populate IFeatureCursor.
        using (ESRI.ArcGIS.ADF.ComReleaser comReleaser = new
ESRI.ArcGIS.ADF.ComReleaser())
        {
            IFeatureCursor updateCursor = featureClass.Update(null,
true);
            comReleaser.ManageLifetime(updateCursor);
            int typeFieldIndex =
featureClass.FindField("INCLUDED_IN_ANALYSIS");
            IFeature feature = null;
            try
            {
                while ((feature = updateCursor.NextFeature()) != null)
                {
                    if (nonSigReadingOIDs.Contains(feature.OID))
                        feature.set_Value(typeFieldIndex, 0);
                    else
                        feature.set_Value(typeFieldIndex, 1);
                    updateCursor.UpdateFeature(feature);
                }
            }
            catch (Exception exc)
            {
                System.Windows.Forms.MessageBox.Show("Unable to update
field INCLUDED_IN_ANALYSIS", "Can't update",
System.Windows.Forms.MessageBoxButtons.OK,
System.Windows.Forms.MessageBoxIcon.Error);
                sigReadingSeparationTB.Text = "10";
                sigReadingSeparationTB.Select();
                sigReadingSeparationTB.SelectAll();
                this.Close();

                // cursor back to default
                this.Cursor = Cursors.Default;

                return;
            }
        }
    }
    #endregion
    // Create a copy of the feature class, excluding the unused
features
    ESRI.ArcGIS.AnalysisTools.Select selectFeatures = new
ESRI.ArcGIS.AnalysisTools.Select();
    selectFeatures.in_features = featureClass;
    selectFeatures.where_clause = "\"INCLUDED_IN_ANALYSIS\" = 1";
    selectFeatures.out_feature_class = GeodatabaseSupport.SearchGDB
+ "\\\" + tempBeaconClassName;
    IGeoProcessorResult gpResult =
(IGeoProcessorResult)GP.Execute(selectFeatures, null);

```

```

        if (gpResult == null || gpResult.Status !=
ESRI.ArcGIS.esriSystem.esriJobStatus.esriJobSucceeded)
        {
            System.Windows.Forms.MessageBox.Show("Unable to predict the
beacon location.", "Processing Error", MessageBoxButtons.OK,
MessageBoxIcon.Error);
            sigReadingSeparationTB.Text = "10";
            sigReadingSeparationTB.Select();
            sigReadingSeparationTB.SelectAll();
            this.Close();

            // cursor back to default
            this.Cursor = Cursors.Default;
            return;
        }

        // archive older prediction
        string beaconPredictionRasterPath =
GeodatabaseSupport.SearchGDB + "\\\" +
GeodatabaseSupport.BeaconPredictionDatasetName;

GeodatabaseSupport.BackupSearchRasterDataset(GeodatabaseSupport.BeaconP
redictionDatasetName, m_application);

        // calculate the visibility of those significant features
        ESRI.ArcGIS.SpatialAnalystTools.Viewshed viewshedTool = new
ESRI.ArcGIS.SpatialAnalystTools.Viewshed();
        viewshedTool.in_observer_features =
GeodatabaseSupport.SearchGDB + "\\\" + tempBeaconClassName;

        bool process = true;
        if
(GeodatabaseSupport.RasterExists(GeodatabaseSupport.BaseGDBFullPath +
"\\DEM_10m"))
            viewshedTool.in_raster = GeodatabaseSupport.BaseGDBFullPath
+ "\\DEM_10m";
        else if
(GeodatabaseSupport.RasterExists(GeodatabaseSupport.BaseGDBFullPath +
"\\DEM_30m"))
            viewshedTool.in_raster = GeodatabaseSupport.BaseGDBFullPath
+ "\\DEM_30m";
        else
        {
            System.Windows.Forms.MessageBox.Show("No terrain raster
exists - 10m or 30m. Can't predict beacon location.", "No Terrain
Data", MessageBoxButtons.OK, MessageBoxIcon.Error);
            process = false;
        }
        if (process)
        {

            bool extCheckoutNotNeeded = false;
            LicensingSupport.CheckOutSpatialAnalystExtension(ref
extCheckoutNotNeeded);
            string outputRaster = GeodatabaseSupport.SearchGDB + "\\\" +
GeodatabaseSupport.BeaconPredictionDatasetName;
            viewshedTool.out_raster = outputRaster;

```



```

        gpResult = (IGeoProcessorResult)GP.Execute(viewshedTool,
null);
        if (!extCheckoutNotNeeded)
            LicensingSupport.CheckInSpatialAnalystExtension();

        // remove the temp feature class from the map and the gdb
        MappingSupport.RemoveLayerFromMap(m_application,
tempBeaconClassName);

GeodatabaseSupport.DeleteSearchFeatureClass(tempBeaconClassName);

        // add the beacon location prediction to the map
        if (gpResult != null && gpResult.Status ==
ESRI.ArcGIS.esriSystem.esriJobStatus.esriJobSucceeded)
        {
            IRasterDataset rasterDataset =
GeodatabaseSupport.OpenGDBRasterDataset(outputRaster);
            IRasterRenderer uniqueValRasterRenderer =
MappingSupport.MakeClassifyRasterRenderer(rasterDataset);
            MappingSupport.SetRasterLayerRenderer(m_application,
rasterDataset, uniqueValRasterRenderer);
        }
        else
        {
            System.Windows.Forms.MessageBox.Show("Unable to predict
the beacon location.", "Processing Error", MessageBoxButtons.OK,
MessageBoxIcon.Error);
            return;
        }
        // refresh the display
        MappingSupport.Refresh(m_application);
        // refresh the TOC
        ((IMxDocument)m_application.Document).UpdateContents();
    }
}
catch (Exception ex)
{
    System.Windows.Forms.MessageBox.Show("Unable to predict beacon
location at this time.", "Error locating beacon",
System.Windows.Forms.MessageBoxButtons.OK, MessageBoxIcon.Error);
}
sigReadingSeparationTB.Text = "10";
sigReadingSeparationTB.Select();
sigReadingSeparationTB.SelectAll();
this.Close();

// cursor back to default
this.Cursor = Cursors.Default;
}

```

The following code is used to determine which beacon readings are insignificant:

```

private List<int> NonSignificantBeaconReadingOIDs(IFeatureClass
featureClass)
{
    List<BeaconInfo> beaconReadings = new List<BeaconInfo>();

```

```

    // go through all the features and read the beacon reading value
    using (ESRI.ArcGIS.ADF.ComReleaser comReleaser = new
ESRI.ArcGIS.ADF.ComReleaser())
    {
        IFeatureCursor searchCursor = featureClass.Search(null, true);
        comReleaser.ManageLifetime(searchCursor);
        IFeature feature = null;
        try
        {
            int signalDirFieldID =
featureClass.Fields.FindField("SIGNAL_DIRECTION");
            while ((feature = searchCursor.NextFeature()) != null)
            {
                beaconReadings.Add(new BeaconInfo(feature.OID,
System.Convert.ToDouble(feature.get_Value(signalDirFieldID)));
            }
        }
        catch (Exception excep)
        {
            System.Windows.Forms.MessageBox.Show("Unable to read
features", "Can't find beacon location",
System.Windows.Forms.MessageBoxButtons.OK,
System.Windows.Forms.MessageBoxIcon.Error);
            return null;
        }
    }
    // process all the signal angles and get only significant ones
    beaconReadings.Sort();
    List<int> nonSigOIDs = new List<int>();

    // go through the rest and check if significant
    int i = 1;
    while (i < beaconReadings.Count - 1)
    {
        if (Math.Abs((beaconReadings[i].signalDirection -
beaconReadings[i - 1].signalDirection) % 360) >=
System.Convert.ToDouble(sigReadingSeparationTB.Text))
        {
            ++i;
        }
        else
        {
            nonSigOIDs.Add(beaconReadings[i].OID);
            beaconReadings.RemoveAt(i);
        }
    }
    // check the final addition to the sigReadings against the previous
and the first one, removing if necessary
    if (Math.Abs((beaconReadings[i].signalDirection - beaconReadings[i
- 1].signalDirection) % 360) <
System.Convert.ToDouble(sigReadingSeparationTB.Text) ||
        Math.Abs((beaconReadings[i].signalDirection -
(beaconReadings[0].signalDirection + 360)) % 360) <
System.Convert.ToDouble(sigReadingSeparationTB.Text))
        nonSigOIDs.Add(beaconReadings[i].OID);

```

```

    return nonSigOIDs;
}

```

A.9 Shared Code that Supports Mapping Operations

The following code is shared across the tools and supports interacting with the map and its display:

```

class MappingSupport
{
    public static string CurrentMXD;

    public static string GetMXD(IApplication m_application)
    {
        switch (m_application.Templates.Count)
        {
            case 2:
                return m_application.Templates.get_Item(1);
            case 3:
                return m_application.Templates.get_Item(2);
            default:
                System.Windows.Forms.MessageBox.Show("Unable to get the
MXD name.", "Error opening associated geodatabase",
System.Windows.Forms.MessageBoxButtons.OK,
System.Windows.Forms.MessageBoxIcon.Error);
                return null;
        }
    }

    public static ILayer AddLayerToMapIfMissing(IApplication app,
IFeatureClass featureClass, IFeatureRenderer featureRenderer)
    {
        // add the featureclass to the map if not yet there
        IMxDocument mxdoc = (IMxDocument)app.Document;
        IFeatureLayer featureLayer = new FeatureLayerClass();
        featureLayer.FeatureClass = featureClass;
        ILayer layer = (ILayer)featureLayer;
        layer.Name = featureLayer.FeatureClass.AliasName;
        // get all feature layers
        IEnumLayer layersEnum = mxdoc.FocusMap.get_Layers(null, true);
        bool layerPresent = false;
        ILayer layerFromMap = layersEnum.Next();
        while (layerFromMap != null)
        {
            if (layer.Name == layerFromMap.Name)
            {
                layerPresent = true;
                layerFromMap.Visible = true; // make sure visible
                return layerFromMap;
            }
            layerFromMap = layersEnum.Next();
        }
        if (!layerPresent)
        {
            if (featureRenderer != null)
            {

```

```

        IGeoFeatureLayer geoFeatureLayer =
        (IGeoFeatureLayer)featureLayer;
        geoFeatureLayer.Renderer = featureRenderer;
    }
    mxdoc.FocusMap.AddLayer(layer);
}

return layer;
}

public static void RemoveLayerFromMap(IApplication app, string
layerName)
{
    IMxDocument mxdoc = (IMxDocument)app.Document;
    // get all feature layers
    IEnumLayer layersEnum = mxdoc.FocusMap.get_Layers(null, true);
    ILayer layerFromMap = layersEnum.Next();
    while (layerFromMap != null)
    {
        if (layerName == layerFromMap.Name)
        {
            mxdoc.FocusMap.DeleteLayer(layerFromMap);
            return;
        }
    }
}

public static IRasterLayer SetRasterLayerRenderer(IApplication
m_application, IRasterDataset rasterDataset, IRasterRenderer
rasterRenderer)
{
    // see if a layer with that name already in the map
    IMxDocument mxdoc = (IMxDocument)m_application.Document;
    try
    {
        IEnumLayer layersEnum = mxdoc.FocusMap.get_Layers(null,
true); // get all layers
        ILayer layerFromMap = layersEnum.Next();
        //Create a raster layer from a raster dataset since no
matching layer found. You can also create a raster layer from a raster.
        ESRI.ArcGIS.Carto.IRasterLayer rasterLayer = new
RasterLayerClass();
        rasterLayer.CreateFromDataset(rasterDataset);
        ILayer layerToFind = (ILayer)rasterLayer;
        while (layerFromMap != null)
        {
            if (layerToFind.Name == layerFromMap.Name)
            {
                ((IRasterLayer)layerFromMap).Renderer =
rasterRenderer;
                layerFromMap.Visible = true;
                return (IRasterLayer)layerFromMap;
            }
            layerFromMap = layersEnum.Next();
        }
    }
}

```

```

        // If get this far, there was no matching layer in the map.
        So set the renderer on the layer created and add to the map.
        // The default renderer will be used if passing a null
value.
        if (rasterRenderer != null)
        {
            rasterLayer.Renderer = rasterRenderer;
        }
        // Add it to a map if the layer is valid.
        if (rasterLayer != null)
        {
            ESRI.ArcGIS.Carto.IMap map =
((IMxDocument)m_application.Document).ActiveView.FocusMap;
            map.AddLayer((ILayer)rasterLayer);
        }
        return rasterLayer;
    }
    catch
    {
        return null;
    }
}

public static void RenderLayer(ILayer layer, IFeatureRenderer
featureRenderer)
{
    IFeatureLayer featureLayer = (IFeatureLayer)layer;
    IGeoFeatureLayer geoFeatureLayer =
(IGeoFeatureLayer)featureLayer;
    layer.Visible = true;
    geoFeatureLayer.Renderer = featureRenderer;
}

public static void Refresh(IApplication m_application)
{
    // get the active view
    ESRI.ArcGIS.ArcMapUI.IMxDocument mxDocument =
(ESRI.ArcGIS.ArcMapUI.IMxDocument)m_application.Document;

mxDocument.ActiveView.PartialRefresh(esriViewDrawPhase.esriViewGeograph
y, null, null);
}

// validation for coordinate masked text boxes
public static bool ValidLatitude(string latitude)
{
    if (String.IsNullOrEmpty(latitude))
    {
        System.Windows.Forms.MessageBox.Show("Latitude must be
provided to enter a clue.", "Missing coordinates",
System.Windows.Forms.MessageBoxButtons.OK,
System.Windows.Forms.MessageBoxIcon.Error);
        return false;
    }
    try
    {
        double dLatitude = System.Convert.ToDouble(latitude);

```

```

        if (dLatitude < -90 || dLatitude > 90)
        {
            System.Windows.Forms.MessageBox.Show("Latitude must be
a number between -90 and 90.", "Invalid coordinate",
System.Windows.Forms.MessageBoxButtons.OK,
System.Windows.Forms.MessageBoxIcon.Error);
            return false;
        }
        return true;
    }
    catch
    {
        System.Windows.Forms.MessageBox.Show("Latitude must be a
number between -90 and 90.", "Invalid coordinate",
System.Windows.Forms.MessageBoxButtons.OK,
System.Windows.Forms.MessageBoxIcon.Error);
        return false;
    }
}

public static bool ValidLongitude(string longitude)
{
    if (String.IsNullOrEmpty(longitude))
    {
        System.Windows.Forms.MessageBox.Show("Latitude must be
provided to enter a clue.", "Missing coordinates",
System.Windows.Forms.MessageBoxButtons.OK,
System.Windows.Forms.MessageBoxIcon.Error);
        return false;
    }
    try
    {
        double dLongitude = System.Convert.ToDouble(longitude);
        if (dLongitude < -180 || dLongitude > 180)
        {
            System.Windows.Forms.MessageBox.Show("Longitude must be
a number between -180 and 180.", "Invalid coordinate",
System.Windows.Forms.MessageBoxButtons.OK,
System.Windows.Forms.MessageBoxIcon.Error);
            return false;
        }
        return true;
    }
    catch
    {
        System.Windows.Forms.MessageBox.Show("Longitude must be a
number between -180 and 180.", "Invalid coordinate",
System.Windows.Forms.MessageBoxButtons.OK,
System.Windows.Forms.MessageBoxIcon.Error);
        return false;
    }
}

public static bool ValidUTMCoordinates(System.Windows.Forms.TextBox
utmXTB, System.Windows.Forms.TextBox utmYTB,
System.Windows.Forms.TextBox utmZoneTB)
{

```

```

        try
        {
            double utm_x = System.Convert.ToDouble(utmXTB.Text);
        }
        catch
        {
            System.Windows.Forms.MessageBox.Show("UTM X coordinate must
be numeric.", "Invalid X Coordinate",
System.Windows.Forms.MessageBoxButtons.OK,
System.Windows.Forms.MessageBoxIcon.Error);
            utmXTB.Select();
            utmXTB.SelectAll();
            return false;
        }
        try
        {
            double utm_y = System.Convert.ToDouble(utmYTB.Text);
        }
        catch
        {
            System.Windows.Forms.MessageBox.Show("UTM Y coordinate must
be numeric.", "Invalid Y Coordinate",
System.Windows.Forms.MessageBoxButtons.OK,
System.Windows.Forms.MessageBoxIcon.Error);
            utmYTB.Select();
            utmYTB.SelectAll();
            return false;
        }
        try
        {
            int zone = System.Convert.ToInt32(utmZoneTB.Text);
            if (zone < 1 || zone > 60)
            {
                System.Windows.Forms.MessageBox.Show("UTM Zone must be
an integer 1-60.", "Invalid Zone",
System.Windows.Forms.MessageBoxButtons.OK,
System.Windows.Forms.MessageBoxIcon.Error);
                utmZoneTB.Select();
                utmZoneTB.SelectAll();
                return false;
            }
        }
        catch
        {
            System.Windows.Forms.MessageBox.Show("UTM Zone must be an
integer 1-60.", "Invalid Zone",
System.Windows.Forms.MessageBoxButtons.OK,
System.Windows.Forms.MessageBoxIcon.Error);
            utmZoneTB.Select();
            utmZoneTB.SelectAll();
            return false;
        }
        return true;
    }

    public static bool
ValidLatitudeLongitude(System.Windows.Forms.TextBox latDegTB,

```

```

System.Windows.Forms.TextBox latMinTB, System.Windows.Forms.TextBox
latSecTB, System.Windows.Forms.TextBox longDegTB,
System.Windows.Forms.TextBox longMinTB, System.Windows.Forms.TextBox
longSecTB)
    {
        try
        {
            int test = System.Convert.ToInt32(latDegTB.Text);
            if (test < -90)
            {
                System.Windows.Forms.MessageBox.Show("Latitude degrees
must be greater than or equal to -90.", "Invalid Latitude Degrees",
System.Windows.Forms.MessageBoxButtons.OK,
System.Windows.Forms.MessageBoxIcon.Error);
                latDegTB.Select();
                latDegTB.SelectAll();
                return false;
            }
            if (90 < test)
            {
                System.Windows.Forms.MessageBox.Show("Latitude degrees
must be less than or equal to 90.", "Invalid Latitude Degrees",
System.Windows.Forms.MessageBoxButtons.OK,
System.Windows.Forms.MessageBoxIcon.Error);
                latDegTB.Select();
                latDegTB.SelectAll();
                return false;
            }
        }
        catch
        {
            System.Windows.Forms.MessageBox.Show("Latitude degrees must
be an integer between -90 and 90.", "Invalid Latitude Degrees",
System.Windows.Forms.MessageBoxButtons.OK,
System.Windows.Forms.MessageBoxIcon.Error);
            latDegTB.Select();
            latDegTB.SelectAll();
            return false;
        }
        try
        {
            int test = System.Convert.ToInt32(latMinTB.Text);
        }
        catch
        {
            System.Windows.Forms.MessageBox.Show("Latitude minutes must
be an integer.", "Invalid Latitude Minutes",
System.Windows.Forms.MessageBoxButtons.OK,
System.Windows.Forms.MessageBoxIcon.Error);
            latMinTB.Select();
            latMinTB.SelectAll();
            return false;
        }
        try
        {
            int test = System.Convert.ToInt32(latSecTB.Text);
        }
    }

```



```

        catch
        {
            System.Windows.Forms.MessageBox.Show("Latitude seconds must
be an integer.", "Invalid Latitude Seconds",
System.Windows.Forms.MessageBoxButtons.OK,
System.Windows.Forms.MessageBoxIcon.Error);
            latSecTB.Select();
            latSecTB.SelectAll();
            return false;
        }
        try
        {
            int test = System.Convert.ToInt32(longDegTB.Text);
            if (test < -180)
            {
                System.Windows.Forms.MessageBox.Show("Longitude degrees
must be greater than or equal to -180.", "Invalid Longitude Degrees",
System.Windows.Forms.MessageBoxButtons.OK,
System.Windows.Forms.MessageBoxIcon.Error);
                longDegTB.Select();
                longDegTB.SelectAll();
                return false;
            }
            if (180 < test)
            {
                System.Windows.Forms.MessageBox.Show("Longitude degrees
must be less than or equal to 180.", "Invalid Longitude Degrees",
System.Windows.Forms.MessageBoxButtons.OK,
System.Windows.Forms.MessageBoxIcon.Error);
                longDegTB.Select();
                longDegTB.SelectAll();
                return false;
            }
        }
        catch
        {
            System.Windows.Forms.MessageBox.Show("Longitude degrees
must be an integer between -180 and 180.", "Invalid Longitude Degrees",
System.Windows.Forms.MessageBoxButtons.OK,
System.Windows.Forms.MessageBoxIcon.Error);
            longDegTB.Select();
            longDegTB.SelectAll();
            return false;
        }
        try
        {
            int test = System.Convert.ToInt32(longMinTB.Text);
        }
        catch
        {
            System.Windows.Forms.MessageBox.Show("Longitude minutes
must be an integer.", "Invalid Longitude Minutes",
System.Windows.Forms.MessageBoxButtons.OK,
System.Windows.Forms.MessageBoxIcon.Error);
            longMinTB.Select();
            longMinTB.SelectAll();
            return false;
        }
    }
}

```

```

    }
    try
    {
        int test = System.Convert.ToInt32(longSecTB.Text);
    }
    catch
    {
        System.Windows.Forms.MessageBox.Show("Longitude seconds
must be an integer.", "Invalid Longitude Seconds",
System.Windows.Forms.MessageBoxButtons.OK,
System.Windows.Forms.MessageBoxIcon.Error);
        longSecTB.Select();
        longSecTB.SelectAll();
        return false;
    }
    return true;
}

public static IPoint GetMapCoordinatesFromScreenCoordinates(IPoint
screenPoint, IActiveView activeView)
{
    if (screenPoint == null || screenPoint.IsEmpty || activeView ==
null)
    {
        return null;
    }

    ESRI.ArcGIS.Display.IScreenDisplay screenDisplay =
activeView.ScreenDisplay;
    ESRI.ArcGIS.Display.IDisplayTransformation
displayTransformation =
        screenDisplay.DisplayTransformation;

    return
displayTransformation.ToMapPoint((System.Int32)screenPoint.X,
        (System.Int32)screenPoint.Y); // Explicit Cast
}

public static IPictureMarkerSymbol PictureMarkerSymbol(string
fullPathToBmp, double symbolSize)
{
    // Creates a picture marker symbol from a .bmp file.

    // Set the transparent background color for the .bmp based
    // picture marker symbol to white.
    ESRI.ArcGIS.Display.IRgbColor rgbColorCls = new
ESRI.ArcGIS.Display.RgbColorClass();
    rgbColorCls.Red = 255;
    rgbColorCls.Green = 255;
    rgbColorCls.Blue = 255;

    // Create the bitmap marker and assign properties.
    ESRI.ArcGIS.Display.IPictureMarkerSymbol
bitmapPictureMarkerSymbolCls = new
ESRI.ArcGIS.Display.PictureMarkerSymbolClass();

```

```

bitmapPictureMarkerSymbolCls.CreateMarkerSymbolFromFile(ESRI.ArcGIS.Display.esriIPictureType.esriIPictureBitmap, fullPathToBmp);
    bitmapPictureMarkerSymbolCls.Angle = 0;
    bitmapPictureMarkerSymbolCls.BitmapTransparencyColor =
rgbColorCls;
    bitmapPictureMarkerSymbolCls.Size = symbolSize;
    bitmapPictureMarkerSymbolCls.XOffset = 0;
    bitmapPictureMarkerSymbolCls.YOffset = 0;

    return bitmapPictureMarkerSymbolCls;
}

public static IFeatureRenderer MakePictureSymbolRenderer(string
fullPathToBmp, double symbolSize)
{
    ISimpleRenderer simpleRenderer = new SimpleRendererClass();
    simpleRenderer.Symbol =
(ISymbol)PictureMarkerSymbol(fullPathToBmp, symbolSize);
    return (IFeatureRenderer)simpleRenderer;
}

public static IFeatureRenderer
MakeUniqueValuePolygonRenderer(ILayer layer, string fieldName, short
transparency)
{
    IGeoFeatureLayer geoFeatureLayer = (IGeoFeatureLayer)layer;
    IRandomColorRamp randomColorRamp = new RandomColorRampClass();
    //Make the color ramp for the symbols in the renderer.
    randomColorRamp.MinSaturation = 20;
    randomColorRamp.MaxSaturation = 40;
    randomColorRamp.MinValue = 85;
    randomColorRamp.MaxValue = 100;
    randomColorRamp.StartHue = 76;
    randomColorRamp.EndHue = 188;
    randomColorRamp.UseSeed = true;
    randomColorRamp.Seed = 43;

    //Make the renderer.
    IUniqueValueRenderer uniqueValueRenderer = new
UniqueValueRendererClass();

    ISimpleFillSymbol simpleFillSymbol = new
SimpleFillSymbolClass();
    simpleFillSymbol.Style = esriSimpleFillStyle.esriSFSSolid;
    simpleFillSymbol.Outline.Width = 0.4;

    ILayerEffects layerEffects =
(ILayerEffects)((IFeatureLayer)layer);
    layerEffects.Transparency = transparency;

    //These properties should be set prior to adding values.
    uniqueValueRenderer.FieldCount = 1;
    uniqueValueRenderer.set_Field(0, fieldName);
    uniqueValueRenderer.DefaultSymbol = simpleFillSymbol as
ISymbol;
    uniqueValueRenderer.UseDefaultSymbol = false;
}

```

```

        IDisplayTable displayTable = geoFeatureLayer as IDisplayTable;

        int fieldIndex;
        using (ESRI.ArcGIS.ADF.ComReleaser comReleaser = new
ESRI.ArcGIS.ADF.ComReleaser())
        {
            IFeatureCursor featureCursor =
displayTable.SearchDisplayTable(null, false) as IFeatureCursor;
            IFeature feature = featureCursor.NextFeature();

            bool ValFound;

            IFields fields = featureCursor.Fields;
            fieldIndex = fields.FindField(fieldName);
            while (feature != null)
            {
                ISimpleFillSymbol classSymbol = new
SimpleFillSymbolClass();
                classSymbol.Style = esriSimpleFillStyle.esriSFSSolid;
                classSymbol.Outline.Width = 0.4;

                string classValue;
                classValue = feature.get_Value(fieldIndex) as string;

                //Test to see if this value was added
                //to the renderer. If not, add it.
                ValFound = false;
                for (int i = 0; i <= uniqueValueRenderer.ValueCount -
1; i++)
                {
                    if (uniqueValueRenderer.get_Value(i) == classValue)
                    {
                        ValFound = true;
                        break; //Exit the loop if the value was found.
                    }
                }
                //If the value was not found, it is new and it will be
added.
                if (ValFound == false)
                {
                    uniqueValueRenderer.AddValue(classValue, fieldName,
classSymbol as ISymbol);
                    uniqueValueRenderer.set_Label(classValue,
classValue);
                    uniqueValueRenderer.set_Symbol(classValue,
classSymbol as ISymbol);
                }
                feature = featureCursor.NextFeature();
            }
            //Since the number of unique values is known,
            //the color ramp can be sized and the colors assigned.
            randomColorRamp.Size = uniqueValueRenderer.ValueCount;
            bool bOK;
            randomColorRamp.CreateRamp(out bOK);

```

```

        IEnumColors enumColors = randomColorRamp.Colors;
        enumColors.Reset();
        for (int j = 0; j <= uniqueValueRenderer.ValueCount - 1; j++)
        {
            string xv;
            xv = uniqueValueRenderer.get_Value(j);
            if (xv != "")
            {
                ISimpleFillSymbol simpleFillColor =
                uniqueValueRenderer.get_Symbol(xv) as ISimpleFillSymbol;
                simpleFillColor.Color = enumColors.Next();
                uniqueValueRenderer.set_Symbol(xv, simpleFillColor as
                ISymbol);
            }
        }

        /*** If you didn't use a predefined color ramp
        /*** in a style, use "Custom" here. Otherwise,
        /*** use the name of the color ramp you selected.
        uniqueValueRenderer.ColorScheme = "Custom";
        ITable table = displayTable as ITable;
        bool isString = (table.Fields.get_Field(fieldIndex).Type ==
        esriFieldType.esriFieldTypeString);
        uniqueValueRenderer.set_FieldType(0, isString);
        geoFeatureLayer.Renderer = uniqueValueRenderer as
        IFeatureRenderer;

        //This makes the layer properties symbology tab
        //show the correct interface.
        IUID uid = new UIDClass();
        uid.Value = "{683C994E-A17B-11D1-8816-080009EC732A}";
        geoFeatureLayer.RendererPropertyPageClassID = uid as UIDClass;

        return (IFeatureRenderer)uniqueValueRenderer;
    }

    public static IRasterRenderer
    MakeClassifyRasterRenderer(ESRI.ArcGIS.Geodatabase.IRasterDataset
    rasterDataset)
    {
        try
        {
            //Create the classify renderer.
            IRasterClassifyColorRampRenderer classifyRenderer = new
            RasterClassifyColorRampRendererClass();
            IRasterRenderer rasterRenderer =
            (IRasterRenderer)classifyRenderer;

            // Set up the renderer properties.
            IRaster raster = rasterDataset.CreateDefaultRaster();
            rasterRenderer.Raster = raster;
            classifyRenderer.ClassCount = 6;
            rasterRenderer.Update();

            // Set the color ramp for the symbology.
            IAlgorithmicColorRamp colorRamp = new
            AlgorithmicColorRampClass();

```

```

        // Create start and end colors
        IRgbColor fromColor = new RgbColorClass();
        IRgbColor toColor = new RgbColorClass();
        fromColor.Red = 255;
        fromColor.Green = 176;
        fromColor.Blue = 98;
        toColor.Red = 210;
        toColor.Green = 105;
        toColor.Blue = 0;
        colorRamp.FromColor = fromColor;
        colorRamp.ToColor = toColor;
        // set the ramping algorithm
        colorRamp.Algorithm =
esriColorRampAlgorithm.esriCIELabAlgorithm;
        colorRamp.Size = 6;
        bool createColorRamp;
        colorRamp.CreateRamp(out createColorRamp);

        // Create the symbols for the classes.
        IFillSymbol fillSymbol = new SimpleFillSymbolClass();
        // color 0 is clear
        IColor transparent = new RgbColorClass();
        transparent.Transparency = 0;
        fillSymbol.Color = transparent;
        classifyRenderer.set_Symbol(0, (ISymbol)fillSymbol);
        classifyRenderer.set_Label(0, Convert.ToString(0));
        // other colors come from the colorramp
        for (int i = 1; i < classifyRenderer.ClassCount; i++)
        {
            fillSymbol.Color = colorRamp.get_Color(i);
            classifyRenderer.set_Symbol(i, (ISymbol)fillSymbol);
            classifyRenderer.set_Label(i, Convert.ToString(i));
        }
        return rasterRenderer;
    }
    catch (Exception ex)
    {
        System.Diagnostics.Debug.WriteLine(ex.Message);
        return null;
    }
}
}
}

```

A.10 Shared Code that Supports Geodatabase Operations

The following code is shared across the tools and supports interacting with the geodatabases:

```

public class GeodatabaseSupport
{
    public static IWorkspace SearchWorkspace;
    public static string SearchGDB;
    public static ISpatialReference SearchSpatialReference;

    public static string BaseDataFolder = @"C:\Program
Files\GISearcher\Data\Base";
}

```

```

private static string baseGDBRelPath;
public static string BaseGDBRelPath
{
    get { return baseGDBRelPath; }
}
public static string BaseGDBFullPath
{
    get { return System.IO.Path.Combine(BaseDataFolder,
baseGDBRelPath); }
}

public static string TeamDomainName = "TEAM_NAME";
public static string OperationalPeriodDomainName =
"OPERATIONAL_PERIOD";
public static string BaseGeodatabaseDomainName = "BASE_GDB";
public enum DomainCreationStatus { VALUE_EXISTED, SUCCESS, FAIL }

public static string BeaconPredictionDatasetName =
"BeaconPrediction";

public struct FieldParameters
{
    public string Name;
    public int Length;
    public esriFieldType Type;
    public FieldParameters(string name, int length, esriFieldType
type)
    {
        this.Name = name;
        this.Length = length;
        this.Type = type;
    }
}

public static void SetBaseGDBRelPath(string relPath)
{
    baseGDBRelPath = relPath;
}

public static string GetSearchGDB(string currentMXD)
{
    // get the geodatabase that matches the open MXD
    string searchGDB =
System.IO.Path.Combine(System.IO.Path.GetDirectoryName(currentMXD),
System.IO.Path.GetFileNameWithoutExtension(currentMXD) + ".gdb");
    if (!System.IO.Directory.Exists(searchGDB))
        return null;
    return searchGDB;
}

// work with domains
public static DomainCreationStatus
CreateAndUpdateSearchDomain(string domainName, string domainValue)
{
    try
    {

```

```

        ICodedValueDomain codedValueDomain =
GeodatabaseSupport.GetSearchDomain(domainName);

        if (!domainName.Equals(BaseGeodatabaseDomainName))
        {
            List<string> codedValues =
GeodatabaseSupport.GetSearchDomainChoices(domainName);
            for (int i = 0; i < codedValueDomain.CodeCount; ++i)
            {
                if
(codedValueDomain.get_Name(i).Equals(domainValue,
StringComparison.CurrentCultureIgnoreCase))
                {
                    return DomainCreationStatus.VALUE_EXISTED;
                }
            }
            codedValues.Add(domainValue.Trim());
            codedValues.Sort();
            while (codedValueDomain.CodeCount > 0)

codedValueDomain.DeleteCode(codedValueDomain.get_Value(0));
            foreach (string newVal in codedValues)
                codedValueDomain.AddCode(newVal, newVal);
        }
        else // get rid of all entries so only a single one
(current value) exists
        {
            while (codedValueDomain.CodeCount > 0)

codedValueDomain.DeleteCode(codedValueDomain.get_Value(0));
            codedValueDomain.AddCode(domainValue, domainValue);

            // also set the base gdb rel path
            baseGDBRelPath = domainValue;
        }
        // Update the domain
        IWorkspaceDomains2 wkspDomains2 =
(IWorkspaceDomains2)SearchWorkspace;
        wkspDomains2.AlterDomain((IDomain)codedValueDomain);
        return DomainCreationStatus.SUCCESS;
    }
    catch
    {
        return DomainCreationStatus.FAIL;
    }
}

private static ICodedValueDomain GetSearchDomain(string domainName)
{
    IWorkspaceDomains workspaceDomains =
(IWorkspaceDomains)SearchWorkspace;
    IDomain domain = workspaceDomains.get_DomainByName(domainName);
    if (domain == null)
    {
        // The code to set the common properties for the new coded
value domain.

```



```

        ICodedValueDomain codedValueDomain = new
CodedValueDomainClass();
        domain = (IDomain)codedValueDomain;
        domain.Name = domainName;
        domain.FieldType = esriFieldType.esriFieldTypeString;
        domain.SplitPolicy = esriSplitPolicyType.esriSPTDuplicate;
        domain.MergePolicy =
esriMergePolicyType.esriMPTDefaultValue;
        workspaceDomains.AddDomain(domain);

        // if the operational period domain, add 1 and Next as
items in list
        if
(domainName.Equals(GeodatabaseSupport.OperationalPeriodDomainName))
        {
            codedValueDomain.AddCode("1", "1");
            codedValueDomain.AddCode("Next", "Next");
            // Update the domain
            IWorkspaceDomains2 wkspDomains2 =
(IWorkspaceDomains2)SearchWorkspace;
            wkspDomains2.AlterDomain((IDomain)codedValueDomain);
        }
    }
    return (ICodedValueDomain)domain;
}

public static List<string> GetSearchDomainChoices(string
domainName)
{
    List<string> choices = new List<string>();
    ICodedValueDomain codedValueDomain =
GetSearchDomain(domainName);
    for (int i = 0; i < codedValueDomain.CodeCount; ++i)
    {
        choices.Add(codedValueDomain.get_Name(i));
    }
    return choices;
}

public static ISpatialReference GetBaseGDBSpatialReference()
{
    IRasterDataset dem;
    try
    {
        dem =
OpenGDBRasterDataset(System.IO.Path.Combine(BaseGDBFullPath,
"DEM_10m"));
    }
    catch
    {
        // try for the 30m DEM
        try
        {
            dem =
OpenGDBRasterDataset(System.IO.Path.Combine(BaseGDBFullPath,
"DEM_30m"));
        }
    }
}

```

```

        catch
        {
            System.Windows.Forms.MessageBox.Show("No base raster
DEM (DEM_10m or DEM_30m) exists, so can't set spatial reference.",
"Spatial reference error", System.Windows.Forms.MessageBoxButtons.OK,
System.Windows.Forms.MessageBoxIcon.Error);
            return null;
        }
    }
    try
    {
        // get the spatial reference from the DEM
        IRasterBandCollection demBandCollection =
(IRasterBandCollection)dem;
        IRasterBand demBand = demBandCollection.Item(0);
        IRasterProps demProps = (IRasterProps)demBand;
        return demProps.SpatialReference;
    }
    catch
    {
        System.Windows.Forms.MessageBox.Show("Unable to get spatial
reference from the base raster DEM.", "Spatial reference error",
System.Windows.Forms.MessageBoxButtons.OK,
System.Windows.Forms.MessageBoxIcon.Error);
        return null;
    }
}

public static IWorkspace OpenGDB(string gdbFullName)
{
    Type factoryType =
Type.GetTypeFromProgID("esriDataSourcesGDB.FileGDBWorkspaceFactory");
    IWorkspaceFactory workspaceFactory =
(IWorkspaceFactory)Activator.CreateInstance(factoryType);
    return workspaceFactory.OpenFromFile(gdbFullName, 0);
}

public static IFeatureDataset
CreateAndOpenSearchFeatureDataset(string featureDSName)
{
    IWorkspace2 workspace2 = (IWorkspace2)SearchWorkspace;
    IFeatureWorkspace featureWorkspace =
(IFeatureWorkspace)SearchWorkspace;
    if
(!workspace2.get_NameExists(esriDatasetType.esriDTFeatureDataset,
featureDSName))
    {
        featureWorkspace.CreateFeatureDataset(featureDSName,
SearchSpatialReference);
    }
    return featureWorkspace.OpenFeatureDataset(featureDSName);
}

public static IFeatureClass CreateAndOpenSearchFeatureClass(string
featureClassName, IFeatureDataset featureDataset, esriGeometryType
geomType, List<FieldParameters> requiredFieldParameters)
{

```

```

        IWorkspace2 workspace2 = (IWorkspace2)SearchWorkspace;
        IFeatureWorkspace featureWorkspace =
(IFeatureWorkspace)SearchWorkspace;
        if
(!workspace2.get_NameExists(esriDatasetType.esriDTFeatureClass,
featureClassName))
        {
            // create the fields
            IFeatureClassDescription fcDesc = new
FeatureClassDescriptionClass();
            IObjectClassDescription ocDesc =
(IOObjectClassDescription)fcDesc;

            // Create a new fields collection.
            IFields fieldsCollection = ocDesc.RequiredFields;

            // Set featureclass geometry
            int shapeFieldIndex =
fieldsCollection.FindField(fcDesc.ShapeFieldName);
            IField shapeField =
fieldsCollection.get_Field(shapeFieldIndex);
            IGeometryDefEdit geomDefEdit =
(IGeometryDefEdit)shapeField.GeometryDef;
            geomDefEdit.GeometryType_2 = geomType;
            if (featureDataset == null && SearchSpatialReference !=
null) // need to set the spatial reference since the feature dataset
won't
                geomDefEdit.SpatialReference_2 =
SearchSpatialReference;

            // Cast to IFieldsEdit to modify the properties of the
fields collection.
            IFieldsEdit fieldsEdit = (IFieldsEdit)fieldsCollection;

            // Create the fields
            foreach (GeodatabaseSupport.FieldParameters fieldParams in
requiredFieldParameters)
            {
                IField field =
GeodatabaseSupport.CreateField(fieldParams);
                fieldsEdit.AddField(field);
            }

            // Use IFieldChecker to create a validated fields
collection.
            IFieldChecker fieldChecker = new FieldCheckerClass();
            IEnumFieldError enumFieldError = null;
            IFields validatedFields = null;
            if (featureDataset != null)
                fieldChecker.ValidateWorkspace =
featureDataset.Workspace;
            else
                fieldChecker.ValidateWorkspace = SearchWorkspace;
            fieldChecker.Validate(fieldsCollection, out enumFieldError,
out validatedFields);

```

```

        // The enumFieldError enumerator can be inspected at this
point to determine
        // which fields were modified during validation.
        IFeatureClass featureClass;
        if (featureDataset != null)
        {
            featureClass =
featureDataset.CreateFeatureClass(featureClassName, validatedFields,
            ocDesc.InstanceCLSID, ocDesc.ClassExtensionCLSID,
esriFeatureType.esriFTSimple, fcDesc.ShapeFieldName, "");
        }
        else
        {
            featureClass =
featureWorkspace.CreateFeatureClass(featureClassName, validatedFields,
            ocDesc.InstanceCLSID, ocDesc.ClassExtensionCLSID,
esriFeatureType.esriFTSimple, fcDesc.ShapeFieldName, "");
        }
        return featureWorkspace.OpenFeatureClass(featureClassName);
    }

    public static IFeatureClass OpenSearchFeatureClass(string
featureClassName)
    {
        try
        {
            IFeatureWorkspace featureWorkspace =
(IFeatureWorkspace)SearchWorkspace;
            return featureWorkspace.OpenFeatureClass(featureClassName);
        }
        catch
        {
            return null;
        }
    }

    public static void DeleteSearchFeatureClass(string
featureClassName)
    {
        IWorkspace2 workspace2 = (IWorkspace2)SearchWorkspace;
        IFeatureWorkspace featureWorkspace =
(IFeatureWorkspace)SearchWorkspace;
        if
(!workspace2.get_NameExists(esriDatasetType.esriDTFeatureClass,
featureClassName))
            return;

        IFeatureClass featureClass =
OpenSearchFeatureClass(featureClassName);
        IDataset fcDataset = (IDataset)featureClass;
        fcDataset.Delete();
    }

    public static IRasterDataset OpenGDBRasterDataset(string
pathToRasterDatasetInGDB)
    {

```

```

        string gdbPath =
System.IO.Path.GetDirectoryName(pathToRasterDatasetInGDB);
        string datasetName =
System.IO.Path.GetFileName(pathToRasterDatasetInGDB);

        try
        {
            Type t =
Type.GetTypeFromProgID("esriDataSourcesGDB.FileGDBWorkspaceFactory");
            System.Object obj = Activator.CreateInstance(t);
            IWorkspaceFactory workspaceFactory = obj as
IWorkspaceFactory;
            IRasterWorkspaceEx rasterWorkspaceEx =
(IRasterWorkspaceEx)workspaceFactory.OpenFromFile(gdbPath, 0);
            return rasterWorkspaceEx.OpenRasterDataset(datasetName);
        }
        catch
        {
            return null;
        }
    }

    public static bool SearchRasterDatasetExists(string
rasterDatasetName)
    {
        IWorkspace2 workspace2 = (IWorkspace2)SearchWorkspace;
        return
workspace2.get_NameExists(esriDatasetType.esriDTRasterDataset,
rasterDatasetName);
    }

    public static bool SearchFeatureClassExists(string
featureClassName)
    {
        IWorkspace2 workspace2 = (IWorkspace2)SearchWorkspace;
        return
workspace2.get_NameExists(esriDatasetType.esriDTFeatureClass,
featureClassName);
    }

    public static bool RasterExists(string pathToRasterDatasetInGDB)
    {
        string gdbPath =
System.IO.Path.GetDirectoryName(pathToRasterDatasetInGDB);
        string datasetName =
System.IO.Path.GetFileName(pathToRasterDatasetInGDB);

        try
        {
            Type t =
Type.GetTypeFromProgID("esriDataSourcesGDB.FileGDBWorkspaceFactory");
            System.Object obj = Activator.CreateInstance(t);
            IWorkspaceFactory workspaceFactory = obj as
IWorkspaceFactory;
            IRasterWorkspaceEx rasterWorkspaceEx =
(IRasterWorkspaceEx)workspaceFactory.OpenFromFile(gdbPath, 0);
            IWorkspace2 workspace2 = (IWorkspace2)rasterWorkspaceEx;

```

```

        return
workspace2.get_NameExists(esriDatasetType.esriDTRasterDataset,
datasetName);
    }
    catch
    {
        return false;
    }
}

public static IField CreateField(FieldParameters fieldParams)
{
    IField field = new FieldClass();
    IFieldEdit fieldEdit = (IFieldEdit)field;
    fieldEdit.Length_2 = fieldParams.Length;
    // Only string fields require that you set the length.
    fieldEdit.Name_2 = fieldParams.Name;
    fieldEdit.Type_2 = fieldParams.Type;

    return field;
}

public static void BackupSearchRasterDataset(string
rasterDatasetName, IApplication app)
{
    if (!SearchRasterDatasetExists(rasterDatasetName))
        return;

    int numArchives = 1;
    string archiveFile = SearchGDB + "\\\" + rasterDatasetName + \"_\"
+ numArchives;
    while
(GeodatabaseSupport.SearchRasterDatasetExists(System.IO.Path.GetFileName
e(archiveFile)))
    {
        ++numArchives;
        archiveFile = SearchGDB + "\\\" + rasterDatasetName + \"_\" +
numArchives;
    }

    Geoprocessor GP = new Geoprocessor();
    ESRI.ArcGIS.DataManagementTools.CopyRaster copyRaster = new
ESRI.ArcGIS.DataManagementTools.CopyRaster();
    copyRaster.in_raster = SearchGDB + "\\\" + rasterDatasetName;
    copyRaster.out_rasterdataset = archiveFile;
    IGeoProcessorResult gpResult =
(IGeoProcessorResult)GP.Execute(copyRaster, null);

    // if copy is successful, delete the old file
    if (gpResult.Status ==
ESRI.ArcGIS.esriSystem.esriJobStatus.esriJobSucceeded)
    {
        ESRI.ArcGIS.DataManagementTools.Delete delete = new
ESRI.ArcGIS.DataManagementTools.Delete();
        delete.in_data = SearchGDB + "\\\" + rasterDatasetName;
        gpResult = (IGeoProcessorResult)GP.Execute(delete, null);
    }
}

```

```

        else
            return;

        // set the renderer of the copied raster to match the original
        one, and set it to not visible in the TOC
        IRasterDataset rasterDataset =
OpenGDBRasterDataset(archiveFile);
        IRasterRenderer uniqueValRasterRenderer =
MappingSupport.MakeClassifyRasterRenderer(rasterDataset);
        IRasterLayer rasterLayer =
MappingSupport.SetRasterLayerRenderer(app, rasterDataset,
uniqueValRasterRenderer);
        rasterLayer.Visible = false;
        // refresh the display
MappingSupport.Refresh(app);
        // refresh the TOC
        ((IMxDocument)app.Document).UpdateContents();
    }
}

```

A.11 Shared Code that Supports Dialog Interaction

The following code is shared across the tools and supports functionality in the dialogs:

```

class DialogSupport
{
    public static void UpdateTeamNames(ref
System.Windows.Forms.ComboBox teamNamesCB)
    {
        List<string> newNames =
GeodatabaseSupport.GetSearchDomainChoices(GeodatabaseSupport.TeamDomain
Name);
        UpdateComboBox(ref teamNamesCB, newNames);
    }

    public static void UpdateOperationalPeriods(ref
System.Windows.Forms.ComboBox operationalPeriodCB)
    {
        List<string> periods =
GeodatabaseSupport.GetSearchDomainChoices(GeodatabaseSupport.Operationa
lPeriodDomainName);
        UpdateComboBox(ref operationalPeriodCB, periods);
        operationalPeriodCB.SelectedIndex =
operationalPeriodCB.Items.Count - 2; // select the most recent
operational period
    }

    private static void UpdateComboBox(ref
System.Windows.Forms.ComboBox comboBox, List<string> newEntries)
    {
        comboBox.Items.Clear();
        foreach (string entry in newEntries)
            comboBox.Items.Add(entry);
    }
}

```

```

    public static void OperationalPeriodChanged(ref
System.Windows.Forms.ComboBox operationalPeriodCB)
    {
        if (operationalPeriodCB.SelectedIndex ==
operationalPeriodCB.Items.Count - 1)
        {
            // add another period since "Next" was selected
            GeodatabaseSupport.DomainCreationStatus status =
GeodatabaseSupport.CreateAndUpdateSearchDomain(GeodatabaseSupport.Opera
tionalPeriodDomainName, operationalPeriodCB.Items.Count.ToString());
            if (status == GeodatabaseSupport.DomainCreationStatus.FAIL)
            {
                System.Windows.Forms.MessageBox.Show("Unable to create
a new operational period.", "Period Error",
System.Windows.Forms.MessageBoxButtons.OK,
System.Windows.Forms.MessageBoxIcon.Warning);
                operationalPeriodCB.Select();
                return;
            }
            UpdateOperationalPeriods(ref operationalPeriodCB);
        }
    }

    public static void SetPanAsCurrentTool(IApplication application)
    {
        ICommandBars documentBars = application.Document.CommandBars;
        UID cmdID = new UIDClass();
        cmdID.Value = "{0830FB32-7EE6-11D0-87EC-080009EC732A}";
        ICommandItem cmdItem = documentBars.Find(cmdID, false, false);
        application.CurrentTool = cmdItem;
    }
}

```