A TECHNIQUE FOR MERGING STATE AND NON-STATE LINEAR REFERENCING SYSTEMS

by

ZACHARY THOMAS RYALS

A THESIS

Submitted in partial fulfillment of the requirements
for the degree of Master of Science
in the Department of Civil and Environmental Engineering
in the Graduate School of
The University of Alabama

TUSCALOOSA, ALABAMA

2011

ABSTRACT

The Wisconsin Department of Transportation (WisDOT) maintains two separate linear referencing systems (LRS) for managing state roads and local roads. The State Trunk Network (STN) only represents statewide routes such as interstates, state highways, and county highways. The Wisconsin Information System for Local Roads (WISLR) system includes all roads, but focuses on local roads. These systems are not related and some significant differences between the systems exist. Therefore, the focus of this project was to develop a technique to relate the two systems.

Merging STN and WISLR is advantageous because it allows the transfer of transportation data between the two systems. To move this data, a table, called the link_link table, was created that relates the two systems based on link IDs and lengths of links. Point data in STN (in the form of link IDs and offset distances) can be programmatically moved using the link_link table to create an equivalent table of WISLR link IDs and offset distances. The merge technique was implemented on a statewide scale. This thesis presents the link_link table design and coding methodology, quality checks for the link_link table, the results of moving data between the systems, and recommendations for future work related to this project.

DEDICATION

This thesis is dedicated to my friends and family, especially those who helped me make it through the completion of this masterpiece.

ACKNOWLEDGMENTS

First and foremost, I would like to thank God for giving me the opportunity to work on this project and for providing me with the courage and drive to persist. I'd also like to thank my friends, family, and colleagues for helping and encouraging me through throughout the course of conducting this research and writing this thesis.

I'd like to Dr. Andrew Graettinger for being a great friend to me and for pointing me in the right direction when I wasn't sure which way to go. Without him, none of this research would have been possible. I'd like to thank Ashley Tu for being someone I could turn to when I needed an ear to listen. I'd like to thank Blake Doherty, Michael Herron, Zach Wilbanks, Kenyona Pierre, and Rachel Cary for performing the data entry, as well as being awesome office-mates. I thank Brian Screws for writing code for me when I was still struggling to learn.

These people have given me with courage, strength, knowledge, and opportunity, and I am forever grateful.

CONTENTS

LIST OF TABLES

LIST OF FIGURES

Figure 2.2. Visualization of 20-27(2) conceptual model (Scarponcini, 2001) 8

LIST OF ILLUSTRATIONS

CHAPTER 1

INTRODUCTION

1.1 Introduction

Since the advancement of computers and computer software, transportation data storage and analysis methods have drastically changed. The introduction of Geographic Information Systems (GISs) has revolutionized not only the way people display and process data, but also the amount of data that can be handled at one time. GISs are a genre of software similar to computer-aided drafting (CAD) programs and database management programs. GISs allow visualization, storage, and spatial analysis of location-related data, while providing robust storage and analysis of attribute data that may or may not be directly related to the spatial data. GISs provide tools that can be applied to many different fields, especially transportation.

The past two decades have witnessed many studies, conferences, and workshops that have focused on the development of a GIS that can efficiently utilize and display transportation data (Miller and Shaw, 2001; Vonderohe et al., 1997). This type of GIS is known as GIS-T, which is designed to handle transportation-related data such as networks, pavement types, bridge locations, crash data, and other Department of Transportation (DOT) data.

Initially, state DOTs maintained data about statewide routes within their jurisdiction; these included both interstates and state roads. Local roads have, for the most part, been maintained by county and city DOTs, who essentially ignored data regarding the statewide routes. Recently,

the federal government has called on state DOTs to begin mapping and analyzing local roads in addition to the state routes to allow more complete understanding of statewide transportation data. This presented a problem for most states since local road systems had either been developed in a county-by-county or city-by-city manner or not at all.

The Wisconsin DOT (WisDOT) built a Linear Referencing System (LRS) called the State Trunk Network (STN) in the mid 90's to satisfy the business needs of WisDOT relative to statewide routes. About a decade later, a second LRS was developed called the Wisconsin Information System for Local Roads (WISLR) that focused mainly on local roads and maintained little or no data regarding the state roads. Each of these systems was designed to meet the business needs of WisDOT, which is well achieved within each system; however, since STN and WISLR were designed and implemented separately, there is little correlation or data sharing between the two systems. Therefore, data that is maintained in one system has no way of being transferred to the other system. Given that there are approximately 12,000 miles of pavement that overlap in each system, which encompasses some of WISLR and all of STN, there exists a great need to develop a bridge between the two systems that affords efficient data sharing and transferability (Graettinger et al., 2008; Graettinger et al., 2009).

Unfortunately, since these two systems were developed in a disconnected environment, there are inherent differences in the technique for drawing complex transportation features. STN describes transportation entities such as intersections in much greater detail than WISLR. STN accounts for these intersections by placing several links to represent all of the different directions and turns that can be made; however, WISLR may draw the same intersection as a single node.

This research describes a method that effectively relates the two systems without the need to interrupt business practices at WisDOT. In this join, neither system is required to adjust or

change any attribute information or spatial features (with the exception of incorrect areas that are occasionally encountered when processing the systems).

1.2 Thesis Organization

Chapter 2, Literature Review, discusses the structure and components of a LRS, illustrates the data structures of STN and WISLR, and describes previous work related to LRSs and data merging issues. Chapter 3, Methodology, describes the merge technique used for this research as well as techniques for checking the quality of the merge. This chapter also displays different methods for moving data between STN and WISLR. Chapter 4, Results, discusses the current progress of this ongoing project and the coding effort required for this merge technique. The results of moving data between the systems are also shown. Chapter 5, Conclusion and Future Work, concluding remarks about the results of the research are presented. This chapter also includes suggestions for future work to be performed on maintenance procedures and the data processing techniques (moving points between the systems).

CHAPTER 2

LITERATURE REVIEW AND BACKGROUND

A linear referencing system (LRS) is a computer-based conceptual model that typically represents road networks for the purpose of maintenance by Departments of Transportation (DOTs). A LRS stores road information in a graphical format along with the necessary attributes, such as road length and direction. This attribute data within the LRS allows a user to display transportation data by simply specifying the link ID and measure of a particular data element (i.e. data such as "Route 43, 4.2 miles", would specify a data point located 4.2 miles from the beginning of Route 43).

Sometimes referred to as linear location referencing systems, LRSs were developed to map transportation business data. The main advantage of a LRS is that the system does not require a spatial reference as do geographic datasets. Two-dimensional and 3-D data can be displayed as 1-D data that is not related to any specific location on earth (Scarponcini, 2002). A challenge faced by many large transportation agencies is the existence of multiple data formats and decentralized data analysis techniques (Kiel and Pollack, 1998), which include LRSs. Unfortunately, the LRSs of the past years evolved through business needs, and were never properly designed (Vonderohe and Hepworth, 1998). Therefore, combining multiple LRSs into a single system within an organization is difficult and complex.

2.1 Linear Referencing System Conceptual Model

A properly designed LRS is made from a set of components, data, and tools that comprise a system where the goal is to locate data along a linear feature such as a road. The LRS data model has been studied by transportation organizations with the intent of developing a generic model that encompasses all business needs of federal, state, and local DOTs. The conceptual model in Figure 2.1 shows a LRS as a compilation of three main parts: a datum, network (s), and linear referencing methods (LRMs).

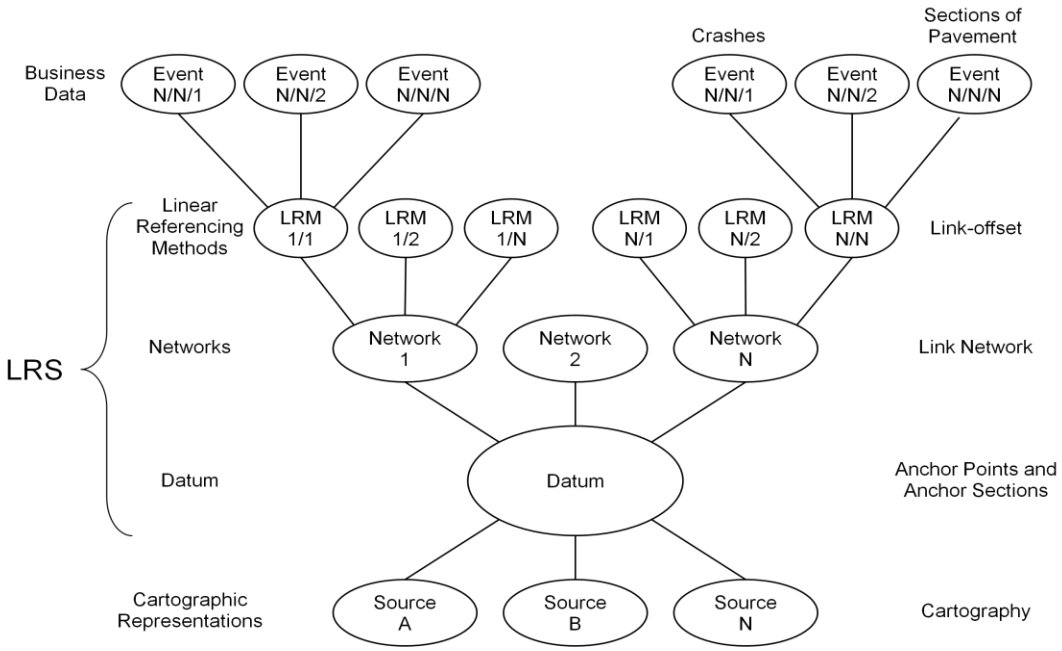

Figure 2.1. Generalized LRS model known as NCHRP 20-27(2) model (Vonderohe et al., 1997)

The datum, shown in the center of Figure 2.1, is an absolute set of anchor point and anchor sections. These anchors points and segments are related to real locations on the earth, and act as a platform for movement among the other parts of the conceptual model. Anchor points are zero-dimensional locations and require some detailed explanation of the location in the field, which can be quantitative and/or qualitative. Anchor sections are non-branching lines that are solely a connection between two anchor points. The length of the anchor section can be

calculated in the field to provide an accurate relationship between the anchor points (Vonderohe et al., 1997).

A network, just above the datum in Figure 2.1, is best described as a means for interaction and movement among point locations (Miller and Shaw, 2001). There are several types of networks, and these different types of networks can be present simultaneously through a common datum that is associated with each LRS, as shown by Network N in Figure 2.1. A common network type is a link-node system, where links are directional and act as conduits for flow, and nodes are locations where conduits meet. Vice-versa, nodes can be described as locations where flow can change, and the links simply connect certain nodes, as described in the WisDOT Location Control Management Manual (1995). The network often, but not always, describes the type of LRM that can be used in that system of features.

A LRM is a method for describing the location of transportation data on a given network. Multiple LRMs can branch from a single network, as seen above the network level in Figure 2.1 as LRM 1/N (the 1/N signifies that this is the Nth LRM on Network 1). Some common LRMs are street address; route-milepost; stationing; on/at, distance, direction; and link-offset, sometimes called reference point (Scarponcini, 2002; WisDOT LCM Manual, 1995; Graettinger et al., 2008; Graettinger et al., 2009). The link-offset method is employed by WisDOT in the STN system. This method states the directional link on which the transportation data is located, as well as the distance down link that must be traveled from the beginning of the link to the event.

Notice to the left of Figure 2.1 that the LRS does not encompass the entire model. Rather, the LRS is comprised of only the essential components and all other data either sits on top of, or

hangs off of, the LRS. These additional pieces of data are business data and cartographic representations, at the bottom and top of Figure 2.1, respectively.

Events are the visual product of processing business information through a LRS and are at the center of spatial analysis. Events can be points or lines, depending on the data format and the needs of the user. In a link-offset LRM, points will be represented by a link ID and an offset measure, while linear events will be represented by a link ID, a start measure, and an end measure (Vanderohe and Hepworth, 1998). Bridge locations and segments of pavement are tangible data events, while crash points and project reference lines are abstract data events. One important point to remember, however, is that events are generated solely through a LRS, and since the LRS is an abstract representation of reality, the locations of events will not always correspond to their actual location in the field.

Cartographic representations can change over time and show varying levels of detail depending on the specific needs of the DOT; therefore, several maps can be related to the LRS based on the virtual anchors of the linear datum. This cartography is not necessary to the functionality of the LRS, but can provide visual perspective to better understand the relationships of the network(s) and event data.

A graphic version of the levels of the conceptual LRS data model is shown in Figure 2.2. Starting from the top of Figure 2.2, there are Events located by the LRM. The LRM works by referencing the Network, the Network is located on the earth's surface by the Datum, and the cartography is overlaid onto the Datum for visualization.

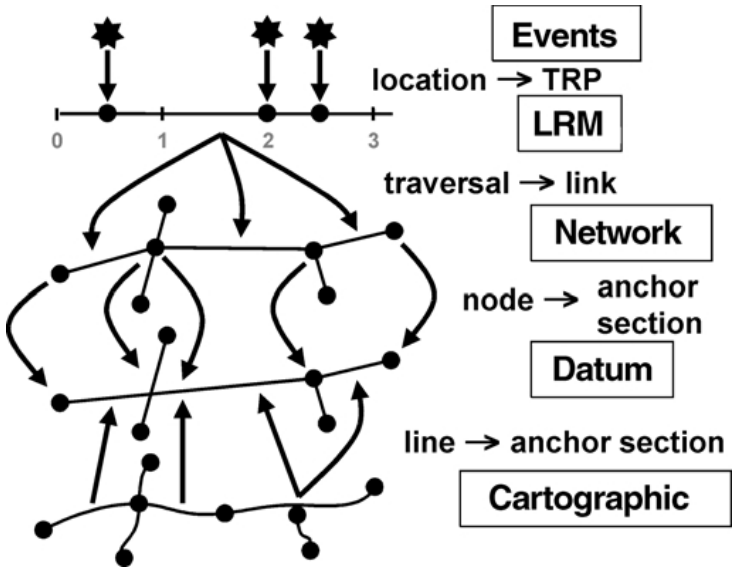

Figure 2.2. Visualization of 20-27(2) conceptual model (Scarponcini, 2001)

2.2 Background

The Wisconsin Department of Transportation (WisDOT) maintains the State Trunk Network (STN) and the Wisconsin Information System for Local Roads (WISLR) which are two separately designed LRSs that were developed independently of one another. The STN LRS was developed in the 1990s to satisfy WisDOT's business needs for maintaining Wisconsin's interstates and state roads. WISLR was developed approximately a decade later to encompass all roads, but focused on local roads. STN was fully functioning by the start of WISLR development, and suspending the utilization of STN in order to create the WISLR system would not have been practical. This separation was acceptable to WisDOT considering that each system works with different data for different users. However, each system contains some information that the other system does not, which decreases the usability of the systems on a macroscopic scale. WISLR shares approximately 12,000 miles of roadway with STN (Graettinger et al., 2008; Graettinger et al., 2009), but these LRS features are almost completely

unrelated, other than representing many of the same physical transportation features. Currently, similar business data is generated in both systems that cannot be analyzed as a whole, so the need for a merge exists.

WisDOT chose to keep STN and WISLR separate in almost all respects, even when devising a linear datum. Past literature describes two classes of datums that have been used in the development of LRSs. The first class is a datum that relates the system to no particular spatial location on the earth's surface; rather, features in the network are tied to field data positions that translate the theoretical location of the network features. The second class of datum is one that is directly tied to the spatial location as accurately as possible; this results in a network and datum that functionally synonymous (Curtin et al., 2007). STN was constructed utilizing the first class of the datum, while WISLR was built utilizing the second class.

Figure 2.3 details this problem from a conceptual standpoint. The center of the figure shows a possible design of the STN and WISLR systems, where the separate networks are related to a common linear datum. However, on the left and right side of the Figure 2.3, it is apparent that the systems are inherently different. The systems are similar in that both lack a distinct datum; rather, each system has the datum embedded in the respective network, which is to say that the network and datum are functionally equivalent (though in WISLR, the cartographic chains are part of the network, and there are no driven distances, so there is technically no functional datum). The chains at the bottom of the figure are the cartographic entities that represent the state routes and local roads. Another difference between STN and WISLR is that each system employs a different LRM (Graettinger et al., 2008; Graettinger et al., 2009).

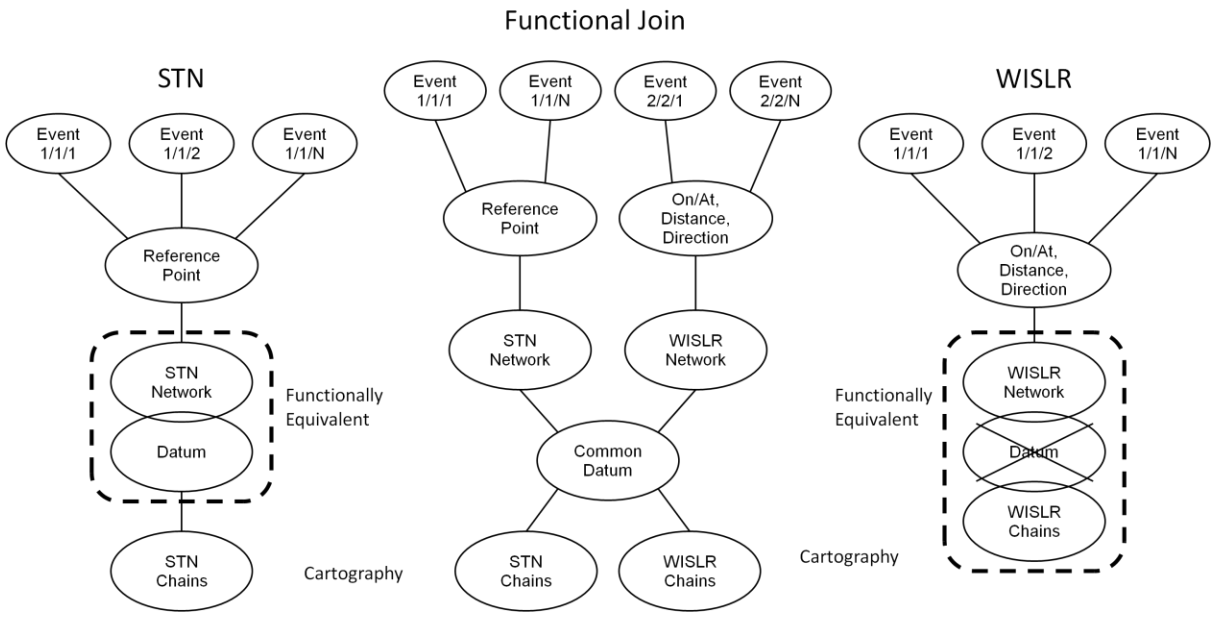

Figure 2.3. STN and WISLR conceptual model (Graettinger et al., 2008) and functional join

Relative to multiple types of networks and LRMs that can a LRS can inherit, there are also multiple options for network design that describe the precision of the LRSs ability to display transportation data. The segmentation of the network into multiple route features can occur according to whatever requirements are specified by the organization creating the LRS (route features are the mechanisms that allow the LRMs to work within the GIS software). Routes are typically created directly from the network; therefore, the design of the network directly affects the precision of the LRS. Some LRS architects choose to divide networks routes by road names or logical routes (i.e. Route 66), while others choose to divide the network into a more integrated network by dividing the network features at any location where two line features intersect (Curtin et al., 2007); the latter includes the planar embedding method of network creation, which ensures that topological integrity is maintained, and does not allow certain intersections such as bridge overpasses to provide connectivity between links (Miller and Shaw, 2001). The integrated method of network construction greatly increases precision when using the LRS to

display data. Segmented routes provide more calibration in displaying measured transportation data than long features that may span over multiple intersections. However, this approach to network design greatly increases the number of features in the system.

The WISLR system is described best by the integrated method of network design, while the STN system is similar to this method, but typically allows multiple road features to comprise one link. Because these methods result in a heavily increased number of features, the difficulty of performing a merge is increased as well.

2.3 Data Merging Issues

The solution to a merger between STN and WISLR takes a unique approach when compared to the majority of the previous methods. The problem with transportation data sharing is typically caused by the diversity of transportation data formats, which leads to inconsistencies and duplications. The major challenge is to develop a means of exchange between systems for not only the transfer of event data, but also completeness relative to updating and maintenance issues (Dueker and Butler, 2000). Some have developed scripting functions to translate the location of the data before mapping occurs (O'Neill and Harper, 1997).

Others have devised approaches to merging spatial objects in the past, using both top-down and bottom-up approaches; however, these methods require some form of aggregation of the data. The top-down approach refers to a matching technique that assumes data objects captured within the same data model are similar, so the primary matching is on object and attribute data, and then on geometry features. The bottom-up approach is reversed from the top-down in that geometry is matched first, and then by objects and attributes (Sester et al., 1998). The bottom-up approach

is the most similar to the approach described in this study; however, aggregation is not performed on the data. Data elements are separately identified and analyzed individually.

The solution to the merge problem is the construction of a relationship between topological objects of the STN and WISLR system, respectively. This relationship is defined and built upon within a hand- and custom tool-populated table that describes links in either system by ID, start measure and end measure, so that each record of the merge table relates a section of one system to a corresponding link length in the other system.

CHAPTER 3

METHODOLOGY

3.1 Introduction

The purpose of this research is to design, test, and implement a technique for merging the State Trunk Network (STN) and the Wisconsin Information System for Local Roads (WISLR), the two Linear Referencing Systems (LRSs) maintained by the Wisconsin Department of Transportation (WisDOT). STN represents state routes only, while WISLR represents all roads, but focuses on local roads. Data is maintained separately in each system; therefore, STN and WISLR must be merged to allow a more complete analysis of statewide business data.

The merge technique chosen and implemented for this project was a link-to-link method, where a section of a link in one system is matched to a section of a link in the other system. This relationship of sections of links is then stored in a table called the link_link table. This table was populated on a county-by-county basis to divide the statewide coding effort among several people. This chapter details the STN and WISLR systems, describes the methodology used for populating the link_link table, discusses issues that arose during data coding, illustrates the programs that were used to facilitate the coding process, and also describes the Quality Assurance/Quality Control (QA/QC) process for checking the table for each county, and details the process of moving data from WISLR back to STN.

3.2 STN and WISLR Systems

STN was developed in the 1990s to manage and maintain transportation information on state highways and interstates in Wisconsin. The STN network contains only information on links that represent state-managed roads. Within the past decade, the federal government has called for more complete maintenance and safety analyses that include local roads. Therefore, WISLR was developed in response to this mandate. The WISLR system contains all roads in the state; however, the focus of WISLR is on local roads. Data regarding state routes is not maintained in WISLR, because STN is the primary system used to maintain, display, and analyze state route transportation events such as crashes, construction, and other transportation data. A map of Madison, Wisconsin, showing STN and WISLR is shown in Figure 3.4.

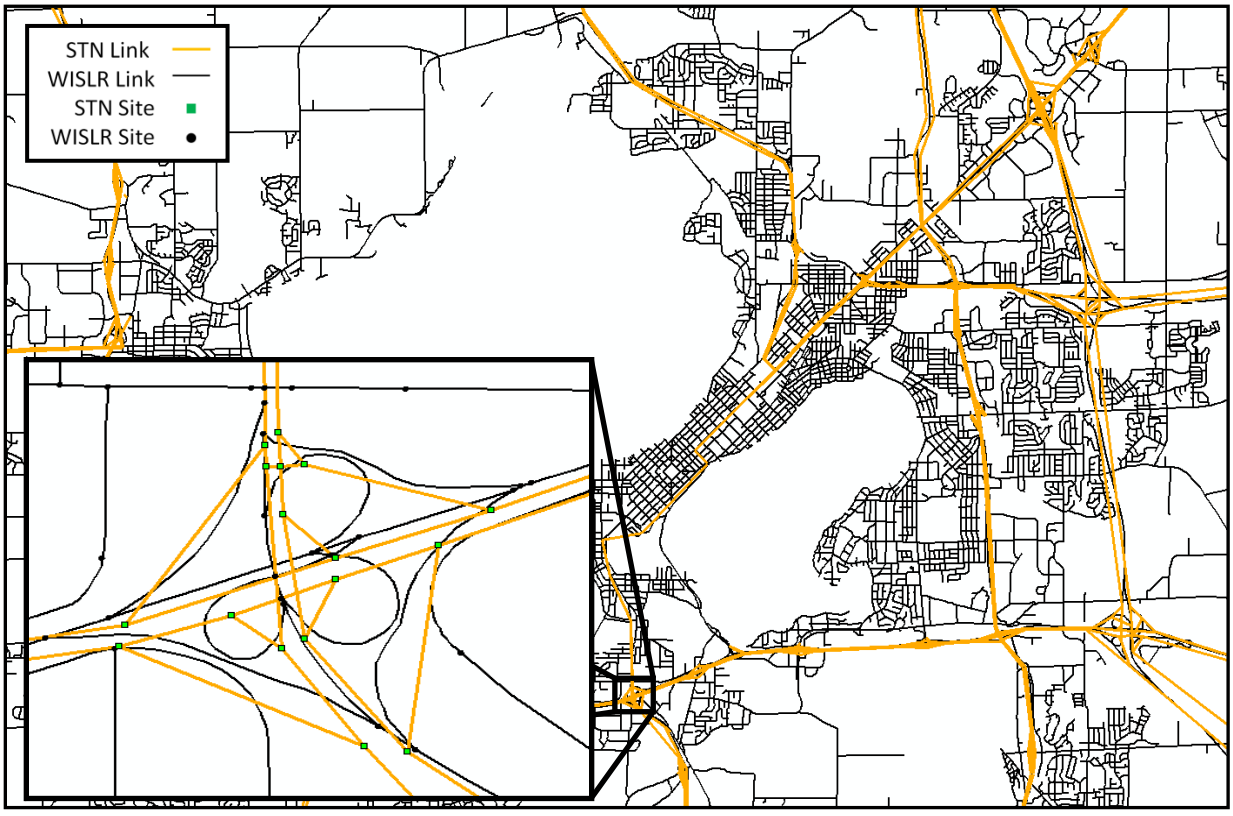

Figure 3.4. STN (orange lines) and WISLR (black lines) systems in Madison, Wisconsin

Figure 3.4 shows the STN system (orange lines) overlaid on the WISLR system (black lines). The STN system consists of links and sites (or nodes), along with cartographic chains (not shown) as well as transportation business data. Each roadway direction in STN is represented by a link that has an ID, length, from-site, and to-site. The from- and to-sites establish directionality and are stored in the attribute table; however, map directionality is a function of topology that is established when the system was drawn. STN links are stored in thousandths of a mile, although links are only measured to a precision of a hundredth of a mile (e.g., the smallest length of an STN link is ten units). Notice that the STN links do not follow the cartographic representation; rather, they are drawn from one STN sites to another as straight lines, and only represent connectivity between sites. The STN sites are shown in the inset of Figure 3.4 as green squares. Each site contains an ID number and an on/at description. The STN chains are the cartographic representation of state routes and are utilized as a visual display since straight-line links are not meant for geographic accuracy, even at nodes.

The WISLR system, also shown in Figure 3.4, is represented by black lines as the links, and black dots as the sites (or nodes). A WISLR link contains attributes similar to STN links, and occasionally has identical from- and to-sites, though WISLR lengths are measured in feet rather than in thousandths of a mile as in STN. The key difference between the two systems is that WISLR links are a cartographic representation of the actual transportation features and show curvature of the roads in addition to providing connectivity between sites.

A zoomed-in view of a small section of the STN and WISLR systems is shown in Figure 3.5. The STN link has been identified using the ArcMap Identify tool, which displays a window showing the attributes of the selected feature. Notice in the upper portion of the Identify window, there are two features selected, shown as two rows of link IDs, but only one link is

shown on the map. This is due to the fact that there are two STN features on top of one another at this one location, and therefore, there are two possible directions of travel. Notice the attributes in the Identify window. The link ID is shown in the "RDWY_LIN_1" field, the name shortened from "RDWY_LINK_ID"; the from-site is shown in the "REF_SITE_F" field, the name shortened from "REF_SITE_FROM"; the to-site is shown in "REF_SITE_T" field, the name truncated from "REF_SITE_TO"; and the link length is displayed in the "FROM_TO_DI", truncated from "FROM_TO_DISTANCE."

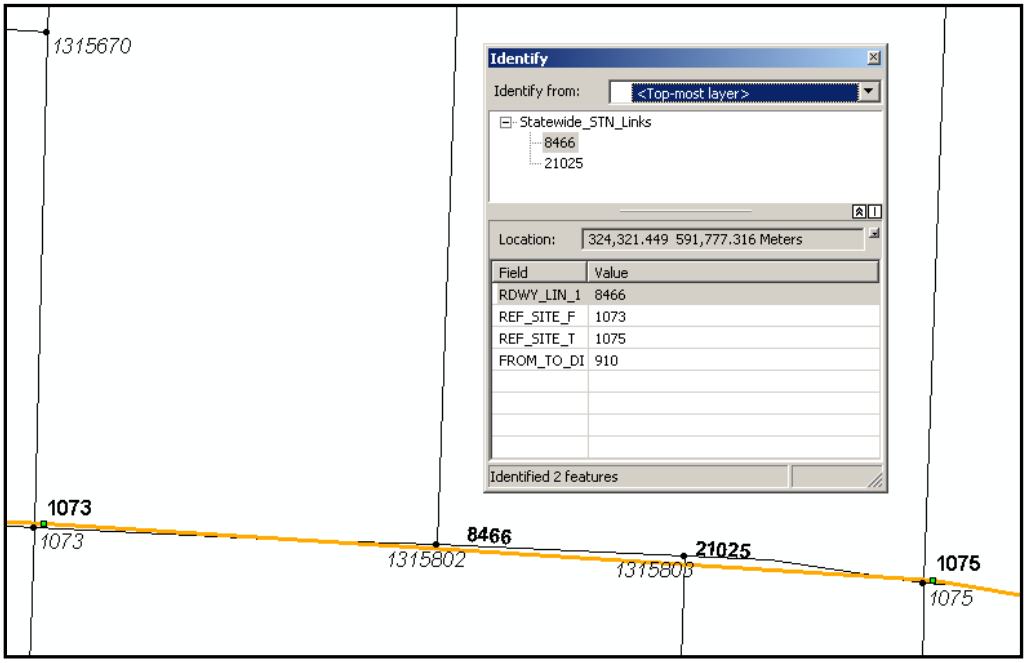

Figure 3.5. STN and WISLR have similar data structure, both containing link ID, from-site, to-site, and link length

In Figure 3.5, one STN link ID is 8466. To determine the direction of the link, the from-site is read to be 1073, indicating that this link allows travel from the left to the right of this figure. There are three full WISLR segments shown in the figure, whereas only one full STN segment is shown. This is related to the design specifications that each system follows. STN does not require a site at every location a local road intersects a state route, only where other state routes

and larger arterials intersect. WISLR, however, does require a site at every intersection, regardless of the road classification.

Contrary to the STN structure in which each direction is represented by one link, WISLR does not follow this structure. WISLR was created by processing digitized lines and inserting a node at any location where a line ended or intersected another line. This process automatically created two links for every line, regardless of whether the link represented a one-directional or two-directional street.

Another significant difference between the two systems is the resolution of complex areas such as divided highway intersections, an example of which is shown in Figure 3.6.

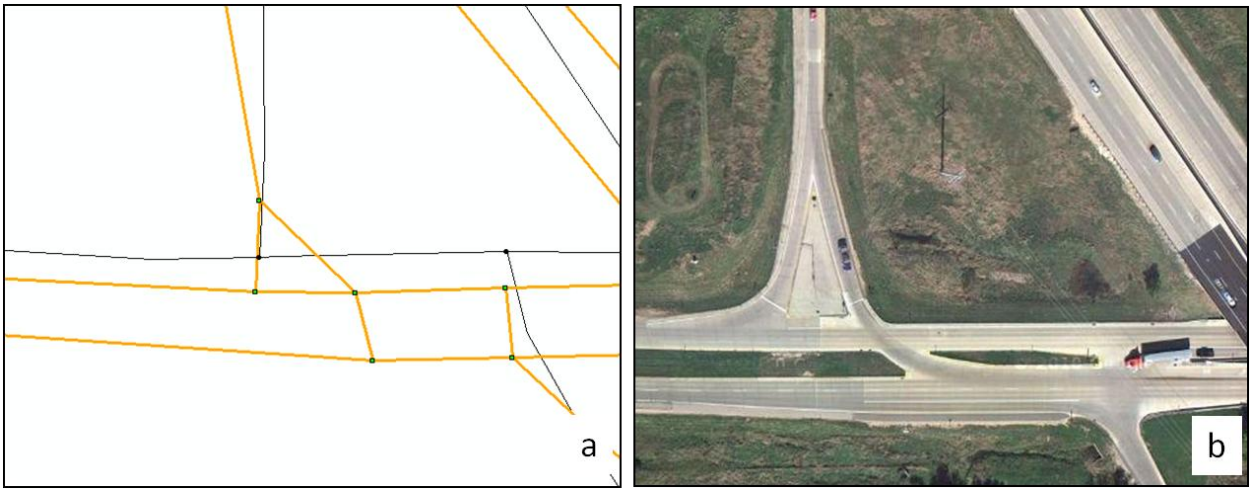

Figure 3.6. a) STN and WISLR are very different in logical design in this interchange area; b) an aerial view shows the actual structure of the intersection

There is an east-west divided highway represented in STN as two approximately parallel sets of orange links in Figure 3.6a. WISLR represents this highway as one set of overlapping black links. An aerial view of this intersection is shown in Figure 3.6b. This location is an interchange showing three on- or off-ramps. The off-ramp coming from the top of the figure at location A splits into two turn-lanes; however, WISLR does not show this split. Another extra link that is

present in STN but missing in WISLR is the vertical link in location B that connects the through-lane to the opposite side of the divided highway. This link is known as a median crossover and represents a situation in which a driver can cross the divided highway and make a left turn. WISLR represents the median crossover with a single node.

In order merge STN and WISLR along state routes, a table was designed and developed to relate the two systems. The following section will describe the table as well as illustrate the method for populating the table.

3.3 Link_link Table Design and Population

A key design strategy for the link_link table is simplicity without sacrificing functionality. As described before, the merge methodology is essentially matching sections of links in each system. Table 3.1 shows the layout of the basic link_link table. Each row in the table represents what is believed to be one segment of pavement in reality that is described in two separate LRSs. The result of the link_link table design process was a table containing six main columns.

Table 3.1. Names and descriptions of the six main link_link columns

STNid	STNstart	STNend	WISLRid	WISLRstart	WISLR end
Unique identifier for the STN link	Start measure for the STN section	End measure for the STN section	Unique identifier for the corresponding WISLR link	Corresponding start measure for the WISLR section	Corresponding end measure for the WISLR section

The first three columns of the link_link table describe a portion of the STN system, while the last three columns describe a corresponding portion of the WISLR system. To demonstrate the link_link table population procedure, an example section of roadway is presented in Figure 3.7 and coded into Table 3.2. For simplicity, only one direction is coded in the link_link table. In

Figure 3.7, the orange STN link A represents four black WISLR links a, b, c, and d. The length of each link is displayed in parentheses. STN measurements are in thousandths of a mile (or units of 5.28 feet), while WISLR measurements are in feet. Note that the STN sites are not always spatially approximate to the corresponding WISLR sites.

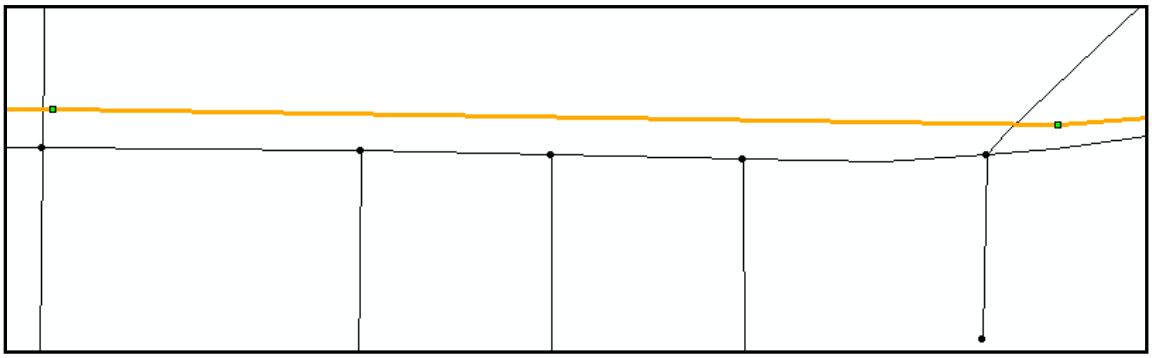

Figure 3.7. Typical link_link coding example

The STN link attribute table stores the STN link ID, which for this example is A. As shown in the top row of Table 3.2, the first STN start value is always zero, and last STN end value is the length of the STN link which is 310 units for link A in this example. The WISLR ID column can be populated by extracting the link ID from the WISLR link attribute table. Given that the entire length of the four WISLR links in this example are within the limits of the STN link, the WISLR start values will all be zero and the WISLR end values will always be the length of the WISLR link. A single record in the link_link table representing a complete WISLR link is the most common situation.

Table 3.2. Example link_link table coded using ratio calculations

STNid	STNstart	STNend	WISLRid	WISLRstart	WISLRend
A	0	103	a	0	528
A	103	165	b	0	317
A	165	227	c	0	317
A	227	310	d	0	422

After entering the values that can be determined from the attribute tables, underlined in Table 3.2, the next values to be entered are the STN start and STN end values for the rest of the cells. These values essentially represent the location of the WISLR site on the STN link. A simple ratio calculation is performed to obtain these values (where STN part/STN full = WISLR part/WISLR full).

Notice when the ratio calculation is performed for STN start and STN end values that these numbers are rounded to nearest thousandth of a mile (nearest single unit), unlike the length data found in STN, which is measured to nearest hundredth of a mile (nearest ten units).

Another set of data in STN known as Access Points, shown as black X's in Figure 3.8, can be used in coding and calibrating the link_link table. Access Points provide references to local roads and other landmarks in the form of an STN link ID and an offset. Figure 3.8 displays the same STN and WISLR links as shown in Figure 3.7, but includes Access Points. Access Points are used when possible to aid in calibration and improve reliability. These Access Points are driven distances and contain the street names in a description field as well as driven distance, shown in parentheses in Figure 3.8. This information assists in determining which Access Point corresponds to which WISLR site.

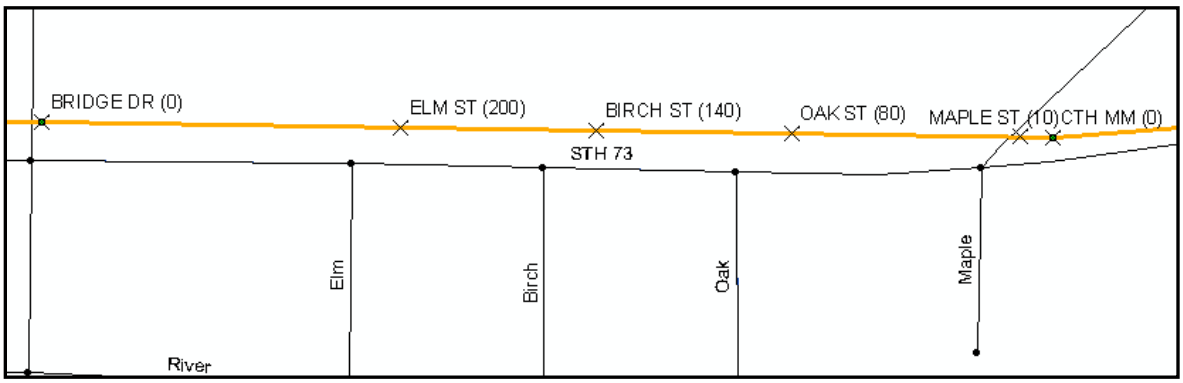

Figure 3.8. Access Points provide calibration for the link_link table

Populating the link_link table with Access Point information slightly adjusts the locations of the WISLR sites that were calculated with a ratio in Table 3.2. Table 3.3 shows the link_link table including Access Point information. Comparing the STN end location in the first row of Table 3.2 to Table 3.3, it can be seen that distance calculated by the ratio (103 in Table 3.2) is seven units less than the driven distance associated with the Access Point (110) and stored in Table 3.3. Other Access Points were within five units of the ratio values. While this does not present a major difference in the final product, Access Points increase the confidence in the data by using driven distances to determine intersection locations rather than ratio-calculated values.

Table 3.3. Example link_link table coded using Access Points information

STNid	STNstart	STNend	WISLRid	WISLRstart	WISLRend
A	0	110	a	0	528
A	110	170	b	0	317
A	170	230	c	0	317
A	230	310	d	0	422

Refer to Appendix A to view a detailed procedure for coding the link_link table with and without Access Points.

3.4 Flag Columns

Coding the link_link table is straightforward when STN and WISLR match up well, as shown in the previous example. However, as mentioned previously and shown in Figure 3.6, there can be significant differences in the representations of certain areas such as complex intersections. Another cause of these differences can be areas that WISLR does not consider part of a state road system, such as a weigh-station or a Park & Ride, which STN maintains. As shown in Table 3.4, a series of flag columns were conceived and included as a part of the link_link table.

There are five flags: turn-lane flag, median-crossover flag, gore point flag, weigh-station flag, and a problem flag. Each flag uses one column in the link_link table, except for a problem situation, which utilizes the problem flag as well as a "Comments" column. These flag situations are explained in detail in the following subsections. Refer to Appendix A to view a detailed guide for coding links that require flags.

Table 3.4. Flags columns and descriptions

Turn-lane	Median Crossover	Gore Point	Weigh-station	Problem	Comments
Turn-lanes represent straight and right-turning traffic lanes	Median crossovers are representative of the distance to cross an intersection	Gore points occur at acute-angle pavement intersections	Weigh-stations and Park & Rides are only maintained in STN	Problem flags are used to mark areas where STN or WISLR has errors	Comments about the problem flag are written in this column

3.4.1 Turn-lane Flag

Turn-lanes, are locations in STN where traffic flow splits into a through lane and right-turning lane. These links are typically not represented in WISLR; therefore, a flag is needed to mark the presence of this inconsistency. The turn-lane flag essentially represents a location where the same section of a WISLR link is coded twice in the link_link table. An example of turn-lanes is shown in Figure XX where westbound travel on STN link X splits into links A and B and southbound travel on STN link Y splits into links E and F.

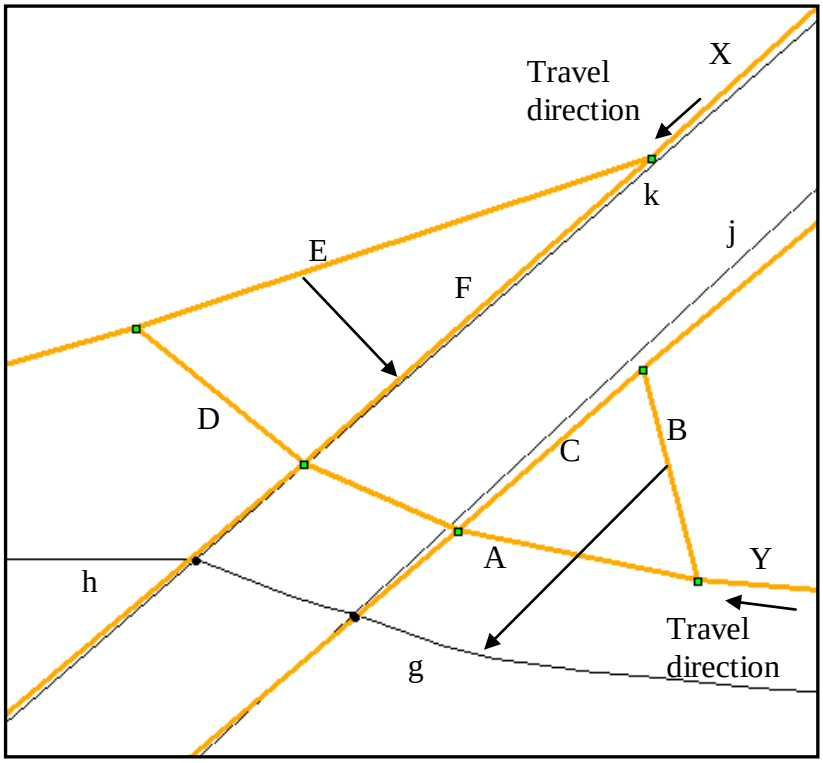

Figure 3.9. Turn-lanes must be coded to calculated sections of WISLR links

Directionality governs how a turn-lane is coded in the link_link table. In Figure 3.9, for example, STN links A and B are turn-lanes traveling on WISLR link g (right to left), so STN links A and B are coded to a length of WISLR link g while link C is coded to WISLR link j. STN links E and F are the turn-lanes when traveling down WISLR link k, and by convention, STN links E and F are both coded to WISLR link k. In the example shown in Figure 3.9, the link_link records for A, B, C, D, E and F are all given a 1 flag in the "Turn-lane" column of the link_link table.

The example shown in Figure 3.9 is where two state routes intersect, and there are turn-lanes connecting the links. A slightly different turn-lane situation appears when there is no intersecting state route, and the intersection connects on- and off-ramps with local roads. WISLR link k or j exist, but have no corresponding STN links. The turn-lane convention is

adjusted in this situation. STN links C and F are not present, and therefore STN link E must be coded along WISLR link h.

The distance of the coded WISLR segment is calculated from the length of the STN turn-lanes. Typically, these links are 10-20 thousandths of a mile. The STN link length is multiplied by 5.28 to convert the length to feet and rounded to the nearest foot. This length is used as the WISLR length in the link_link table.

3.4.2 Median-Crossover Flag

Intersections in reality are not point entities; rather, intersections cover an area that represents segments of pavement. However, logically, there are many ways to represent these intersections in a LRS. WISLR represents most intersections as nodes, while STN draws links to represents these same areas. The distance through an intersection (e.g. from the stop line to the far edge of the median) is known as a median-crossover, an example of which is shown in Figure 3.10. Median crossovers are coded to a single point in WISLR, but are often represented as a link in STN.

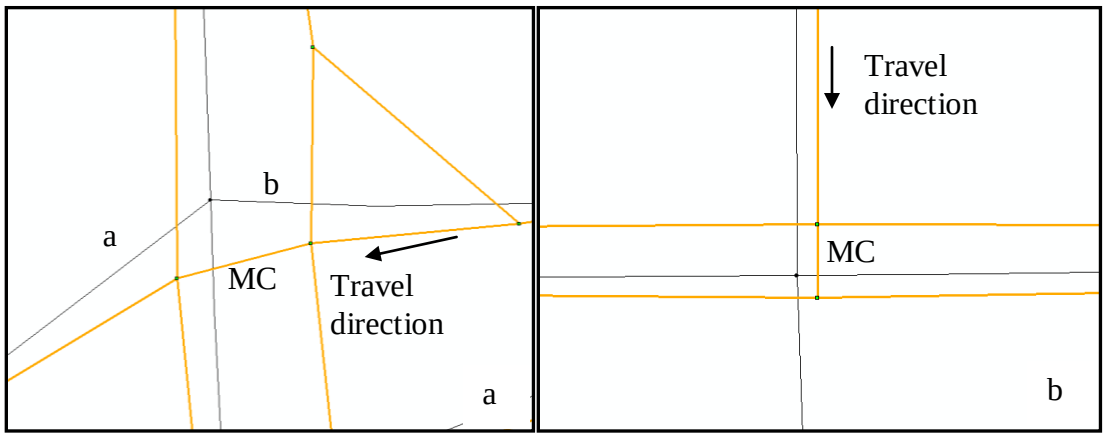

Figure 3.10. a) Median crossovers can represent through-intersections; b) Median-crossovers can also be present at "T" intersections

Median crossovers can occur when local roads or off-ramps cross a divided highway or "T" into a divided highway. In the example shown in Figure 3.10a, the median crossover shows the ability to turn left or continue straight after traversing the median crossover link (labeled as MC) from east to west. In this case, the STN link could be coded to WISLR links a or b, so a convention was created for this situation. The convention is that the median crossover should be coded to the beginning point of a WISLR link, which governs which WISLR link ID is used. The ID of the WISLR link that is not coded as the WISLRid, in this example WISLR link b, is placed in the "M" column.

The convention discussed above was conceived as a result of discussions with WisDOT. There were multiple methods that were discussed, but this convention was the ultimate result, since this method is the simplest and still retains the opposing WISLR link ID in the "M" column. However, it should be noted that some of the other methods are more logical relative to moving data between the two systems. One of the other methods involved coding half of the median crossover to a small portion of each of the WISLR links and compressing the other STN links at that intersection to a slightly smaller length of WISLR. This method is the most logical because no length of the STN median crossover is lost as in the simplest method, and a crash that occurs directly at the WISLR node would correspond to the center of the median crossover. While the implemented method is the simplest, the programming logic that processes the crash data can be modified to process the data as if the method discussed in this paragraph were used.

"T" intersections at divided highways, as shown in Figure 3.10b, are coded and flagged differently in the link_link table compared to the through-street median crossover situation previously discussed. With a "T" intersection, there are most likely two overlapping STN links, and with only one WISLR link available per STN link, the median crossovers are coded to the

respective sides of the directionally-appropriate WISLR links. A flag of 1 is placed in the "Median crossover" column, rather than a WISLR ID, as in the previous situation.

3.4.3 Gore Point Flag

On-ramps and off-ramps are typically constructed with a sharp offset angle from the main road to provide a driver a safe and easy transition between road sections at a high speed. There is a beveled section of pavement that smoothes the transition between the pavement sections called a gore point, as shown in the aerial view of a gore point in Figure 3.11. STN and WISLR representations of this gore point are overlaid on the image. The inconsistency between STN and WISLR in this case is not visual; rather, the difference is in the method of measurement of the links connecting to this gore point. WISLR measures the ramp along the centerline of the ramp and main road, whereas STN measures the ramp to the gore point as a convention.

Figure 3.11. Aerial image of a gore point alongside STN and WISLR

A gore point can be imagined as a node attribute. Each link that connects to that node is going to be affected by the existence of the gore point. Records in the link_link table associated with a gore are flagged with one of three different values: To (T), From (F), or Both (B). The "To" and

"From" values respectively indicate whether a link travels towards or away from a gore. If a gore exists on both sides of a link, the record is flagged with a "Both" in the Gore Point flag column of the link_link table. With gore points identified, an adjustment to the STN or WISLR link by some factor on the side that connects to the gore point can be made. Therefore, the flag was required to identify the location of the gore relative to the link in question.

3.4.4 Weigh-station Flag

Weigh stations, Park & Rides, and rest areas are maintained in the STN system, but are not often represented in the WISLR system. The locations of these STN links must be arbitrarily assigned to a location in WISLR. Typically, rest areas and parking lots will be drawn in STN branching from a state route that is drawn in WISLR; therefore, the STN links will be coded to a single point in WISLR. The point will be a distance down the WISLR link equal to the length of the corresponding STN link. For example, if a parking lot is drawn as a single STN link, this link will be coded to the point on WISLR off of which the STN link branched. The record for this link is given a 1 flag in the "Weigh station" column in the link_link table.

3.4.5 Problem Flag

Problematic areas are not commonplace, but occur often enough to require modification of the STN or WISLR systems to resolve. Problems arise in areas where the link_link table coder cannot determine the correct relationship between the systems. Problem areas are flagged with a "1" in the "Problem" column. These records are also given a written comment in the "Comments" column describing the problem as well as the source of the problem (STN or WISLR). The link_link records that are marked as a problem will need to be assessed by WisDOT.

3.5 Link_link Coding Tool

In order to increase the efficiency of the coding process and reduce the amount of human error present in the link_link table, a semi-automated tool was designed and developed for this project. This tool, known as the Link_link Coding Tool, significantly increased the speed of coding simple areas (i.e. where problems are not present). Through coding experiments, the efficiency of the tool was determined to be approximately three times faster than coding without the tool.

The Link_link Coding Tool is only semi-automated because the coder is required to select links from the map. The tool then populates the link IDs and performs any needed calculations.

The user interface (UI) of the Link_link Coding Tool is shown in Figure 3.12. The tool removes the need for any calculation to be performed by hand. When mp features are selected and entered into the tool, the tool will either calculate a ratio automatically, or the user can select Access Points via the tool to be used in populating the link_link table.

The Link_link Coding Tool UI shown in Figure 3.12 can be divided into a working area (top), and an output area (bottom). In the example shown in Figure 3.12, one direction of travel for one section of road has been completed and is shown in the bottom half of the UI. The opposite direction of travel is being coded and is in the top half of the UI.

The top of the UI is where the attribute data from the map features are stored until the coder chooses to enter the data into the preview section, the bottom half of the UI. The relevant STN and WISLR sites must be selected so the program knows which links pertain to which direction of travel. The “Calculate” button performs a ratio calculation when the data for both directions has been entered; however, if Access Points exist for the STN links, the Access Points will be used in lieu of a ratio. The “Calculate” button also calculates ratios around Access Points that

might be missing. The coder clicks the "Commit" button to submit the records to the link_link table.

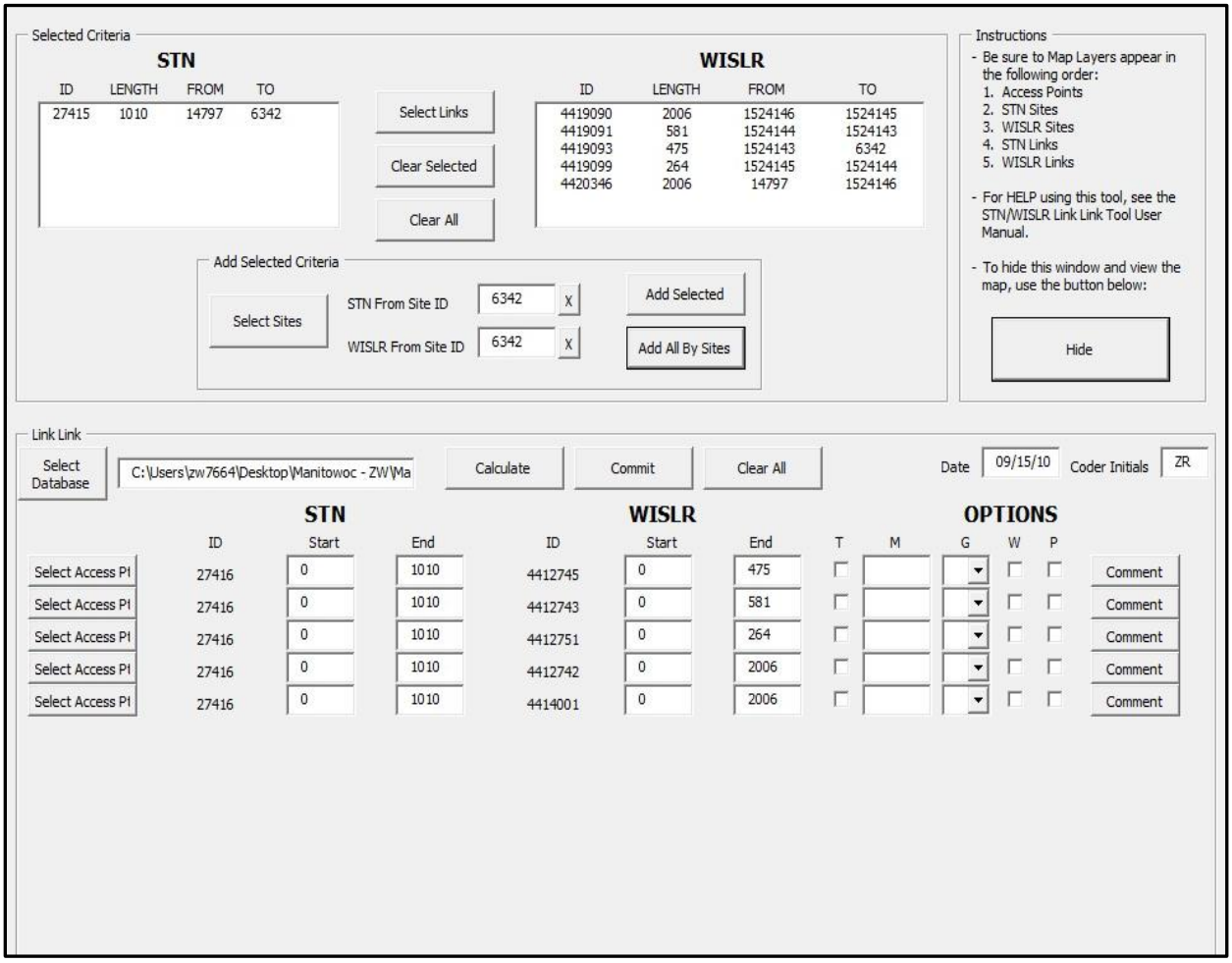

Figure 3.12. Link_link Coding Tool UI

Refer to Appendix B to view the program code for the Link_link Coding Tool.

3.6 Project-specific Programs

Three programs were written to facilitate data processing, testing, and management of the link_link table: the STN Point Generator, the Point Moving Program, and Database Merge Tool. Creating and moving points from STN to WISLR is the most efficient and complete method for checking the quality of the link_link table. Moving data from STN to WISLR, currently

performed by the Point Moving Program, ensures that the link_link table was populated correctly which is the primary objective of the LRS merge. These programs are used in the quality checking procedures that are described in the section entitled Quality Assurance/Quality Control (QA/QC). Because each county is coded individually, once a link_link table is completed and quality checked, the Database Merge Tool allows the table to be added to a single database table and prevent duplicate values from appearing in the table.

3.6.1 STN Points Generator

The STN Points Generator program creates a set of points on every STN link every 100th of a mile. This program uses an Excel spreadsheet containing the STN link IDs and link lengths for every link in a county. The program then reads each link and calculates the 100th of a mile points down each link. These points are then output to a STN points table. This table consists of a Unique ID, STN link ID, and STN link offset. These offsets are from zero to the full length of the STN link, in 100th of a mile increments. Refer to Appendix C to view the programming code for the STN Points Generator.

3.6.2 Point Moving Program

The Point Moving Program uses the STN points table, created by the STN Points Generator program, and the link_link table, created by hand coders, to generate the WISLR points table. The WISLR points table is essentially equivalent to the STN points table in that the WISLR table contains a Unique ID (the program transfers the ID from the STN points table), a WISLR link ID, and a WISLR link offset.

The program cycles through each record in the STN points table and selects the records from the link_link table with a matching STN link ID. The program reads the STN offset for the current

STN point record, and then selects the records of the link_link table contains that STN link ID and a STNstart and STNend range that encompass the STN offset value. Using the STNstart, STNend, WISLRstart, and WISLRend values, the STN offset is translated into a WISLR offset value through a simple ratio calculation.

The logic behind the STNstart and STNend range governs the WISLR link to which a point will move. The logic states the if the STN offset is greater than the STNstart and less than or equal to the STNend, then that offset is in the range and this record should be used for calculating the corresponding WISLR offset. This convention declares that if an STN point falls directly on the STNend value, the STN point will move to the end of a WISLR link. This convention does not modify the physical location of where a point will move; however, it can skew the number of crashes associated with a certain WISLR link (i.e. an analysis of the number of crashes on each WISLR link may show a particular WISLR link with multiple crashes, when in reality it is possible that those crashes should have moved to different WISLR link). Given that intersections (nodes) are each associated with more than one link, this problem is anticipated.

Once each record in the STN points table has been processed, the STN input points and the WISLR output points can be displayed in GIS along the STN and WISLR links, respectively. Using a route event tool in GIS, the visualization of these points on the two LRSs aids in quality checking and assures that the link_link table was coded correctly. Refer to Appendix D to view the program code for the Point Moving Program.

3.6.3 Database Merge Tool

The Database Merge Tool allows the primary statewide database, containing the cumulative link_link table, and the individual county link_link table to be merged. An individual county database is merged into the statewide database as part of the assembly of the statewide link_link table. The program uses a Structure Query Language (SQL) expression to omit the insertion of any records from the county link_link table that have a matching STN link ID in the statewide table. This prevents duplicate records from appearing in the statewide table. Refer to Appendix E to view the program code for the Database Merge Tool.

3.7 Quality Assurance / Quality Control (QA/QC)

One of the most important aspects of the link_link table population process is quality. WisDOT requested the utilization of a series of quality validation measures before the table is submitted. When merging two LRSs, confusion is occasionally unavoidable; therefore, QA/QC checks are an essential part of the population process.

After completing the initial population of the link_link table, a series of checks are applied by the coder. There are seven checks which include: STN Link check, WISLR Link check, Access Point check, Gore Point check, Point Moving Program Output check, WISLR Link Visual check, XY Connector Line check, and the Reversed Link Identification Tool. There is also a program that finds erroneous WISLR links. Each check is briefly described in the following sections.

3.7.1 STN Link Check

The STN system contains a finite number of graphic elements that represent the interstate and state highways that exist in Wisconsin. All STN links in the state must be coded into the

link_link table. This check is relatively simple, and only involves a table join within the GIS. The link_link table is added to the GIS from the Access database or Excel file, and is then joined to the STN links attribute table for the current county. If there is a link in the attribute table that is not represented in the link_link table, null values will appear after the join. These null values identify un-coded STN links.

3.7.2 WISLR Link Check

The WISLR system contains a relatively large number of graphical entities that represent all roads in the state. Therefore, off state route links in WISLR will not be used in the link_link. To assure that there are no mistyped IDs in the link_link table, every WISLR link ID used in the link_link table must be a valid ID. The check is performed by joining the WISLR links attribute table to the link_link table based on WISLR link ID, and then checking that there are no records in the link_link table that do not have a match in the WISLR links attribute table.

3.7.3 WISLR Link Visual Check

There is a large number of WISLR links in each county, but not all are used in the link_link table, as described previously. In order to ensure only correct WISLR links are used in coding the link_link table, a visual check is performed by displaying only the WISLR links that are used in the link_link table, so that any unnecessary links or gaps are easily visible to the coder.

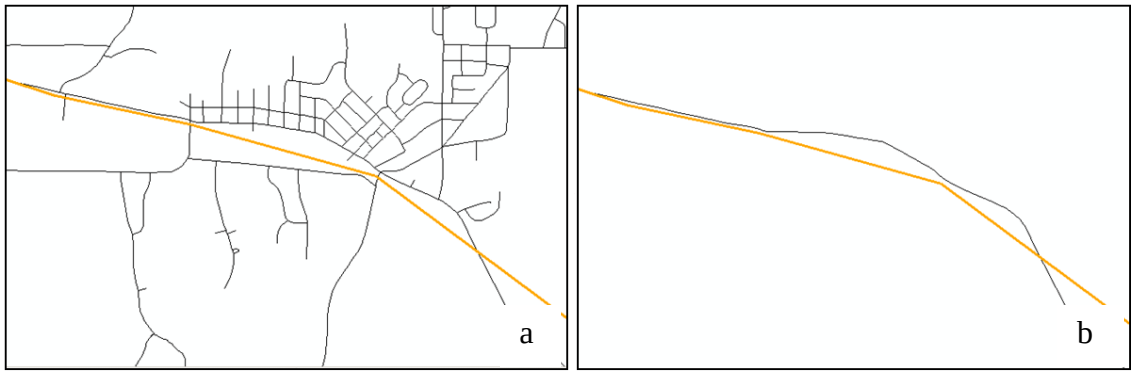

Figure 3.13. a) All WISLR links shown; b) Only WISLR in the link_link table are shown

Figure 3.13a shows a view of the STN and WISLR links before beginning this visual check. Figure 3.13b shows the same view but only the STN links and the WISLR links that are included in the link_link table are displayed. The removal of the unused links can be performed within ArcMap by loading the link_link table into the program, summarizing the WISLR ID column, then joining this summary table to the WISLR links shapefile. Any link that is not used will have null values in key columns and can be removed from the map.

3.7.4 Access Point Check

Access Points that describe an intersection in WISLR should be included in the coding of the link_link table. After the link_link coding process is complete, a check identifies all of the unused Access Points within the county. The coder then analyzes each unused Access Point to guarantee that unused points were not mistakenly overlooked.

The Access Point check is performed by creating a unique column in both the link_link table and Access Point attribute table. This unique field is the link ID concatenated with the offset distance. The tables are then joined based on this column, and all unused Access Points appear as null values in the link_link table.

3.7.5 Gore Point Check

Gore points are somewhat of a subjective aspect of coding and are analyzed on a case-by-case basis. Typically, gores at intersections are easy to spot, but can sometimes be overlooked. This check assures that the coder looks at areas that typically contain gores and verifies that all links that should be marked in the "Gore Point" column are correctly marked. Gores are flagged with either a To, From, or Both flag, depending on the location of the gore with respect to the link.

As shown in Figure 3.14, for a single direction of travel, a gore point will always have at least three associated links. Each record in the link_link table typically represents a whole WISLR link; therefore, the link_link table can be joined to the WISLR links shapefile based on WISLR link ID.

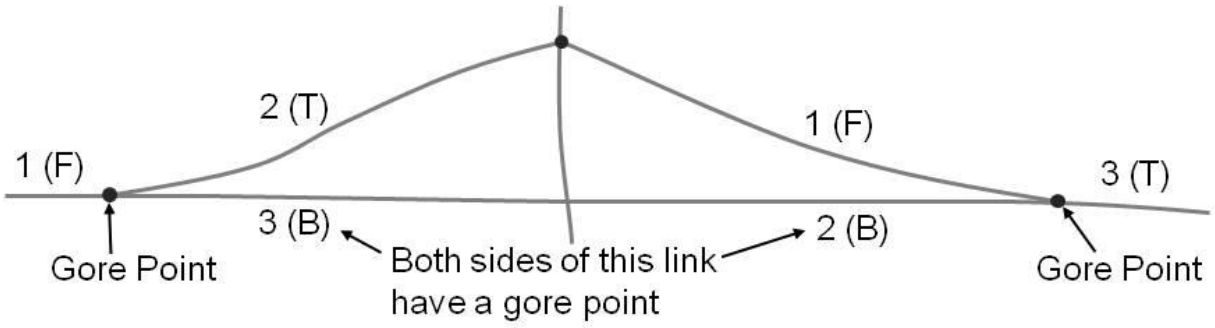

Figure 3.14. Gore point at an off-ramp

The symbology of the gore links is changed so that specific values in the "Gore Point" column are represented by different colored WISLR links, which creates a visual check that the coder can quickly evaluate and make changes as necessary.

3.7.6 Point Moving Program Output Check

The Point-Moving program, which moves data points from STN to WISLR, is executed as part of the quality control procedures. The input is an Access Database containing the link_link table

and a table of STN points. The STN Points table can be any number of points that would be on one of the links coded in the link_link table. A unique ID number is provided for each STN point before the program is executed, and this unique ID is transferred to each corresponding WISLR point. The program reads the STN point, calculates where each point moves to in WISLR, and then creates a WISLR point table that contains one point for every STN point. A check is performed to verify that the number of input and output values is the same. If these numbers differ, then one or more records in the link_link table are incorrect.

3.7.7 XY Connector Line Check

The XY Connector Line check is a visual check that illustrates the output from the Point Moving program in a way that allows mistakes and improper coding to be displayed in an obvious manner. Each input point for the program contains a unique identifier that is carried over to the output. Using this identifier as a common field, the input and output can be joined together. Coordinates are calculated using the ArcMap Geometry Calculator to provide starting and ending locations for each set of points. These locations then have a line drawn from start to end using a set of tools known as Hawths Analysis Tools for ArcGIS (Beyer, 2004). Hawths Analysis Tools is a free toolset that works within ArcMap.

An example of the connector lines when there are no problems is shown in Figure 3.15a. The lines are parallel and evenly spaced throughout an entire STN link. Figure 3.15b shows an example of a WISLR link that is drawn in reverse direction. Since the routes are created from the WISLR cartography, the links must be drawn in the direction of travel. The attribute data may be correct, but cartographically, the end of the link is the start and vice versa. This is

apparent in Figure 3.15b where the connector lines cross. The STN starting point is transferred to the WISLR end point and the STN end point is transferred to the WISLR starting point.

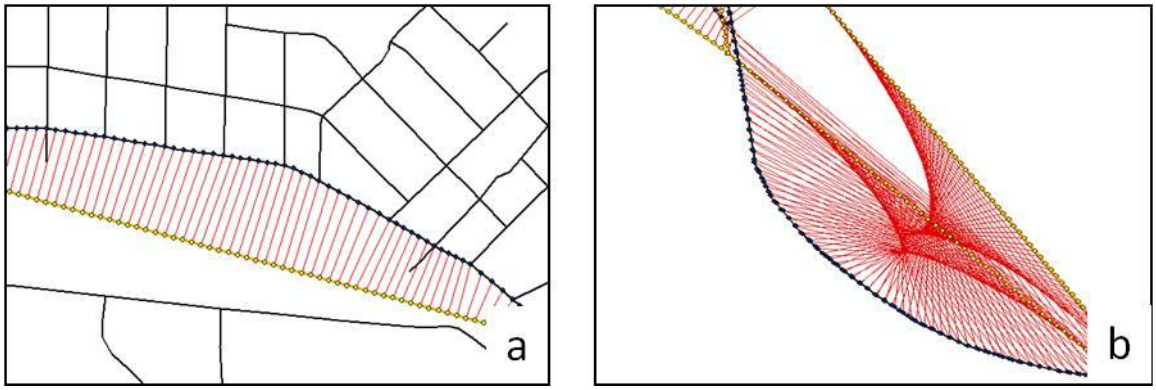

Figure 3.15. XY connector lines showing correctly and incorrectly coded links

The reversed WISLR links are only evident when this QA/QC check is performed; however, this does not constitute a problem in the link_link table, only a problem that must be addressed in the "Problem" and "Comments" column. This check is performed an average of three times during the QA/QC of the coding of a single county.

3.7.8 Reversed Link Identification Tool

This tool was developed to automatically determine if WISLR links are reversed as discussed in the previous subsection. This tool uses the cartographic lengths of the STN and WISLR links, as well as the lengths of the connector lines to find where the connector lines are not displaying in a parallel manner. Much of the information used by this program is calculated at the beginning of the XY Connector Line check, therefore it is advantageous to perform these checks simultaneously, so that problems may be discovered before the full visual check is performed.

A generic example of correct and incorrect XY lines is shown in Figure 3.16. Notice the considerable difference in lengths between the matching lines in the correct and incorrect

illustrations (where line 3 is the exception). This difference is how the tool determines which links may be reversed.

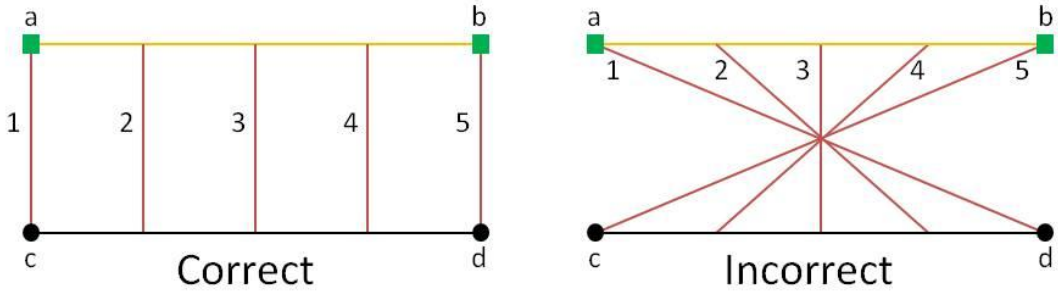

Figure 3.16. Simplified examples of correct and incorrect XY lines

Two checks are performed by the Reversed Link Identification Tool. The first check looks at the distance between the STN start and the WISLR start, which, in the example in Figure 3.16, is line 1, and compares it to the graphical length of the STN link. If the first connector line is larger than the STN link length, there could be a reversed link.

The second check method compares the length of line 1 to the distance between nodes a and d, as well as nodes c and b. If the length is approximately equal, a problem with this WISLR may exist. The WISLR link ID numbers are flagged as potentially incorrect and output to a table that can be added to ArcMap and analyzed. After making changes, the Point Moving program is executed again and the XY Connector Line check is performed again.

3.8 Moving Data back to STN from WISLR

STN and WISLR were constructed to meet separate business needs within WisDOT. Different business needs require different levels of detail. For example, STN emphasizes detail around intersections with multiple links representing turn-lanes, while WISLR may only place a node to represent the entire intersection. This causes a problem given that each system is significantly

different in some areas. In order to test the robustness of the link_link merge technique, an effort was made to move data from the WISLR system to the STN system. This is contrary to previously discussed methods of data transfer where the STN was the origin LRS and WISLR was the destination LRS.

As shown in Figure 3.17, intersections in STN, represented with orange links and green sites, are drawn differently than in WISLR, where a black node represents the entire intersection. Since intersections in WISLR are less detailed than in STN, moving a point at a WISLR intersection produces a problem: there are multiple locations in STN where a WISLR point can be located.

Two of the programs used during the project, the STN Points Generator and the Point Moving Program, were modified to move points from WISLR to STN. To account for inherent errors when moving from a detailed system (STN) to a less detailed system WISLR), turn-lane and median crossover flags were included as input into the modified Point Moving Program in order to apply a degree of confidence to the output data. The flag columns mark links where there are multiple locations to which a single point can move and therefore, the confidence level of that point decreased.

The purpose of this experiment was to determine if a data point moved from STN to WISLR, would the point move back to the same location in STN, and so on.

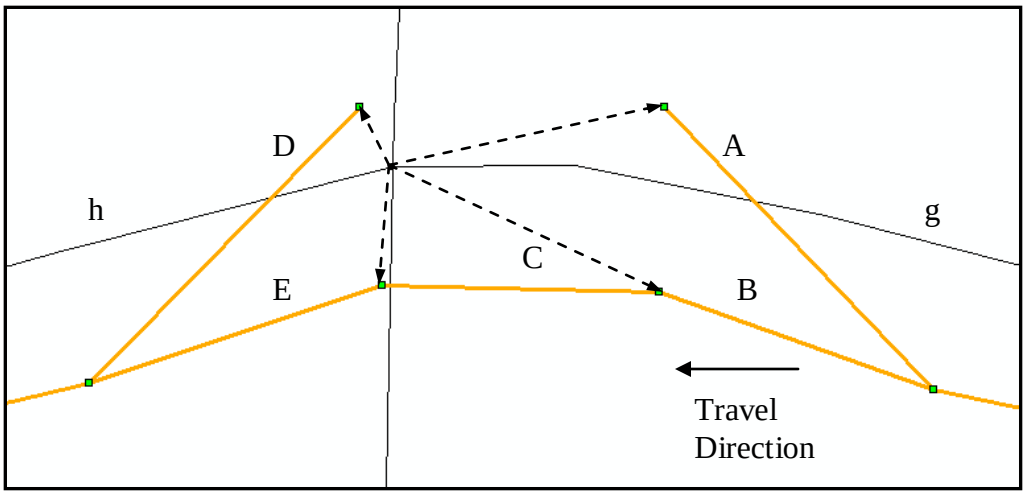

Figure 3.17. STN and WISLR intersections must be coded differently in the link_link table and can therefore lead to multiple possible locations for a WISLR point to fall in STN.

As discussed in Section 3.4, Flag Columns, the intersection in Figure 3.17 would be coded as a set of turn lanes and a median crossover. Links A and B are coded to the same section of link g, links D and E are coded to the same section of link h, and link C is coded to the node connecting link g and h.

Given that there are instances where STN links must be coded to the same section of a WISLR link, there are invariably instances where WISLR points can transfer to multiple locations in STN. For instance, a WISLR point placed at the end of link g and at the beginning of link h (i.e. at the WISLR node) can be located at either side of STN link C, to the end of link A or B, or to the beginning of link D or E, as shown by the arrows in Figure 3.17.

The median crossover flags were used to determine which location in STN to use. For this experiment, it was decided that a point in WISLR would never go to a median crossover in STN; the point would always go to the adjacent turn-lane STN link, as shown in Figure 3.18a. Figure 3.18b shows two STN points on the median crossover (labeled point 1 and point 2) move to the beginning of the associated WISLR link (labeled points 1 and 2). In Figure 3.18a, these two points will move back to the turn-lane STN link (labeled point 1 in Figure 3.18a).

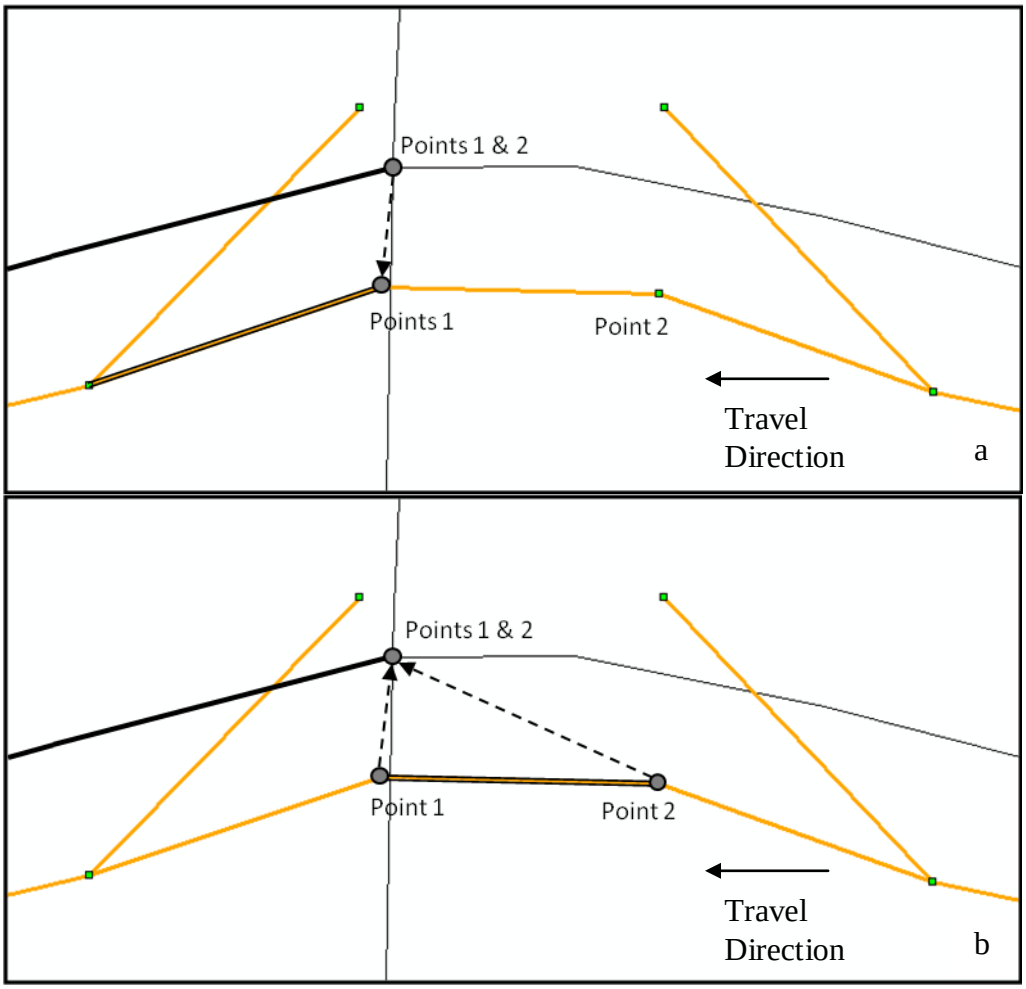

Figure 3.18. a) WISLR points will move to one location in STN, b) STN points move from a median crossover to one point in WISLR

The decision to exclude median crossovers in this experiment is justified by the fact that it is decidedly better to have ambiguous points move to one location by convention rather than have the program randomly select an STN location.

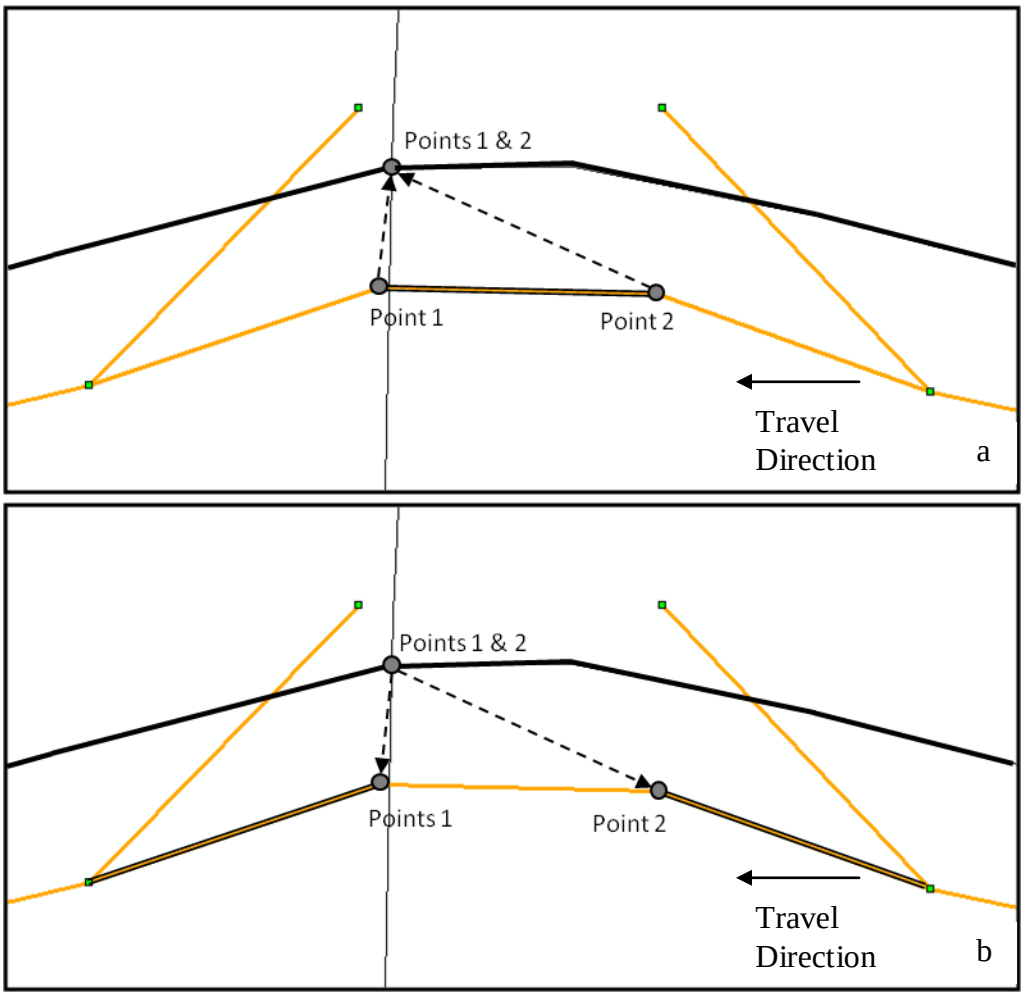

Figure 3.19. Median crossovers could possibly be handled to allow points to travel back to the same physical location

As shown in Figure 3.19a, there is another possible method for processing median crossovers that involves using the value in the median crossover (either a "1" or a WISLR link ID) to determine where the point should move from STN to WISLR. Currently, the Point Moving Program will move a point from a median crossover to the beginning of a WISLR link, as shown in Figure 3.18b. However, the Point Moving Program can be modified so that if there is a WISLR ID in the median crossover flag column, one point will be moved to the beginning of one WISLR link, and the other point will be moved to the end of the WISLR link in the median

crossover column, as shown at the top of Figure 3.19a. Using this method, the points would move back to the two separate STN turn-lane links, as shown in Figure 3.19b.

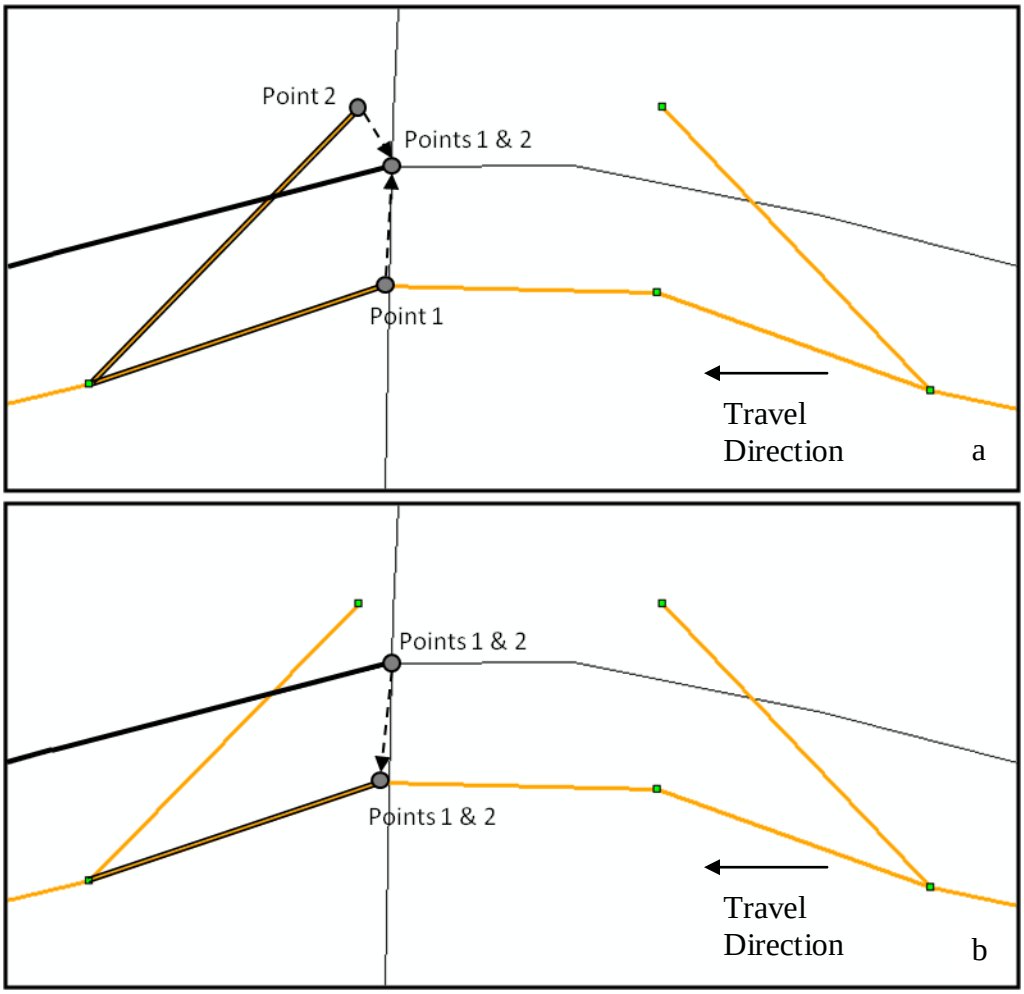

Figure 3.20. a) STN points from two turn-lane links move to one location in WISLR, however b) will only move back to one location in STN

In this experiment, turn-lane flags were also used in deciding which STN link would receive a WISLR point. As shown in Figure 3.20, a WISLR point that falls within the section of a WISLR link that is designated for two STN turn-lanes will only move back to one STN turn-lane. In the link_link table, there is no difference between the two turn-lane link records other than STN ID; therefore, there is no logical way to choose the STN link to which the point would be moved.

The program chooses the STN with the lowest ID number as the STN link that receives the point. Additionally, a less arbitrary method should be developed to handle turn-lane information.

3.9 Conclusion

This research was performed to determine if two LRSs could be merged successfully. The merge method chosen for this research was a link-to-link method, which matched portions of the links in each system together and related these sections in a table. This chapter discussed the structure of STN and WISLR, respectively, described the relationship between the two LRSs, detailed the link_link table and coding methods, and how the link_link table was quality checked. Chapter 4, Results, will describe table development progress, the results of moving data from STN to WISLR, and the results of moving data from WISLR back to STN.

CHAPTER 4

RESULTS

4.1 Statewide Link_link Coding Results

The link_link coding technique is capable of bridging the gap between the STN and WISLR LRSs. As shown in Figure 4.21, approximately 60% (18,576 coded STN links / 31,183 total STN links) of the state has been coded into the statewide link_link table so far.

The shade of each county in Figure 4.21 illustrates the number of STN links associated with each county. Notice that the majority of the counties that comprise the southeastern portion of the state are of darker shade and therefore contained more state routes (i.e. STN links). It was decided at the beginning of this project that these southern counties would be coded first, since the higher-populated counties were typically more cumbersome than counties with less-populated counties. This afforded ample opportunity to mitigate issues that were encountered and develop techniques for handling similar issues in the future.

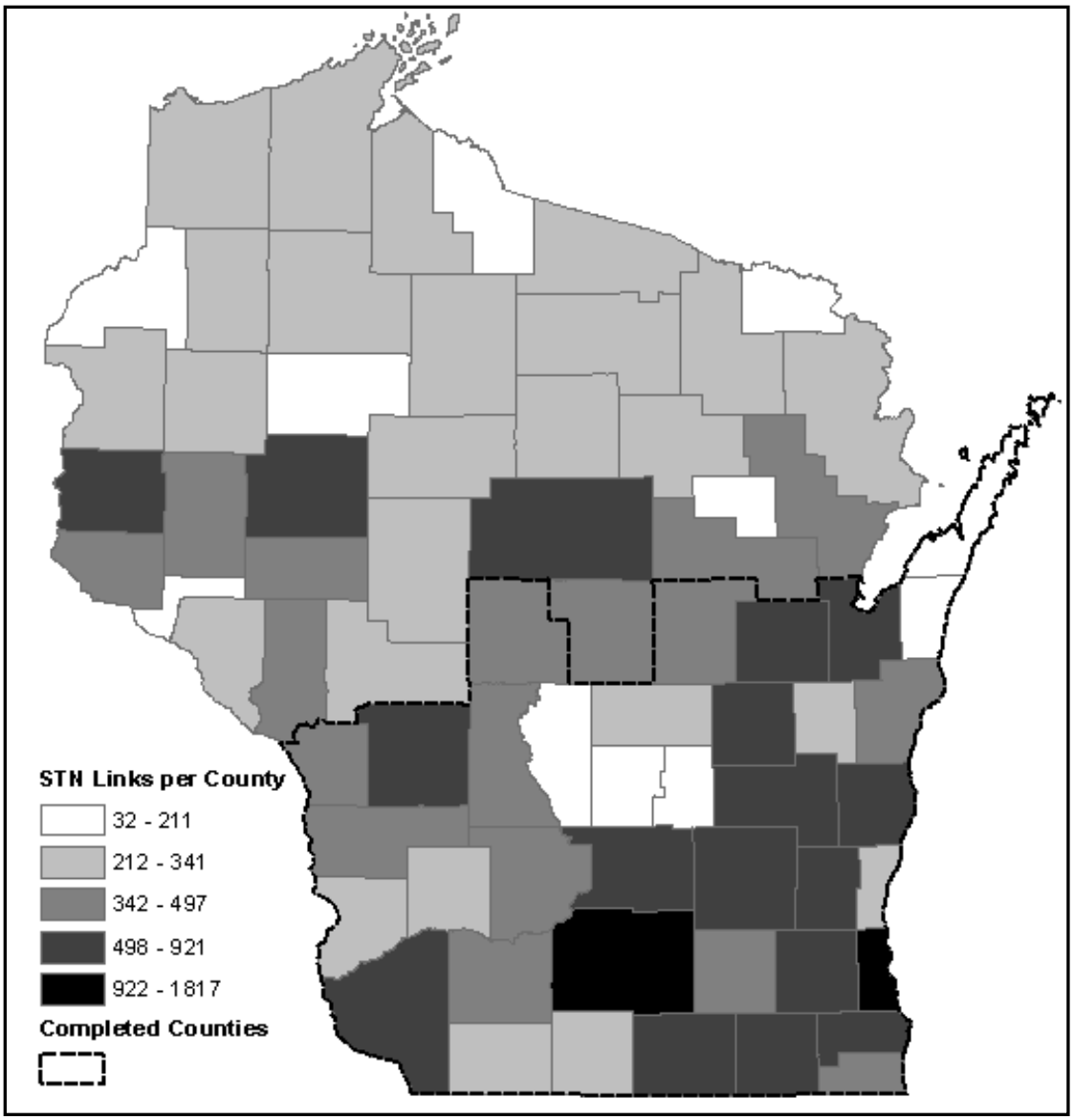

Figure 4.21. The statewide link_link table is currently 60% percent complete. The majority of the completed counties contained more links relative to the incomplete counties.

To date, the statewide link_link table contains 38 of the 72 counties in Wisconsin and contains 39,627 records. Using the Point Moving Program, real crash data supplied by WisDOT was translated from STN to WISLR. Between 2005 and 2009, there were 281,300 crashes that occurred on state routes. Of these crashes, 200,733 were moved from STN to WISLR, which accounts for 71% of the crash records for these five years. 100% of the crash records are expected to move after the completion of the statewide link_link table.

Table 4.5 shows a summary of the flag columns in the statewide link_link table so far. These flagged records indicate special situations that may require additional attention when processing data and updating each system. It should be noted that 5.8% STN links do not have a problem. A single problem can affect other links within proximity to that problem, and are therefore flagged as well.

Table 4.5. The link_link table contains multiple records containing each flag

Flag column	T	M	G	W	P
Number of records in the statewide link_link table	943	972	6010	153	2301
% of total link_link records	2.4%	2.5%	15%	0.4%	5.8%

4.2 Coding Effort

Coding the link_link table is not a "one-man" job; it requires multiple people working simultaneously to show prompt and reliable results. The time allotted for the project was 18 months. Throughout the project, there have been a total of five students who have worked to make progress towards a completed statewide link_link table. Figure 4.2 shows the approximate timeline for the statewide link_link table.

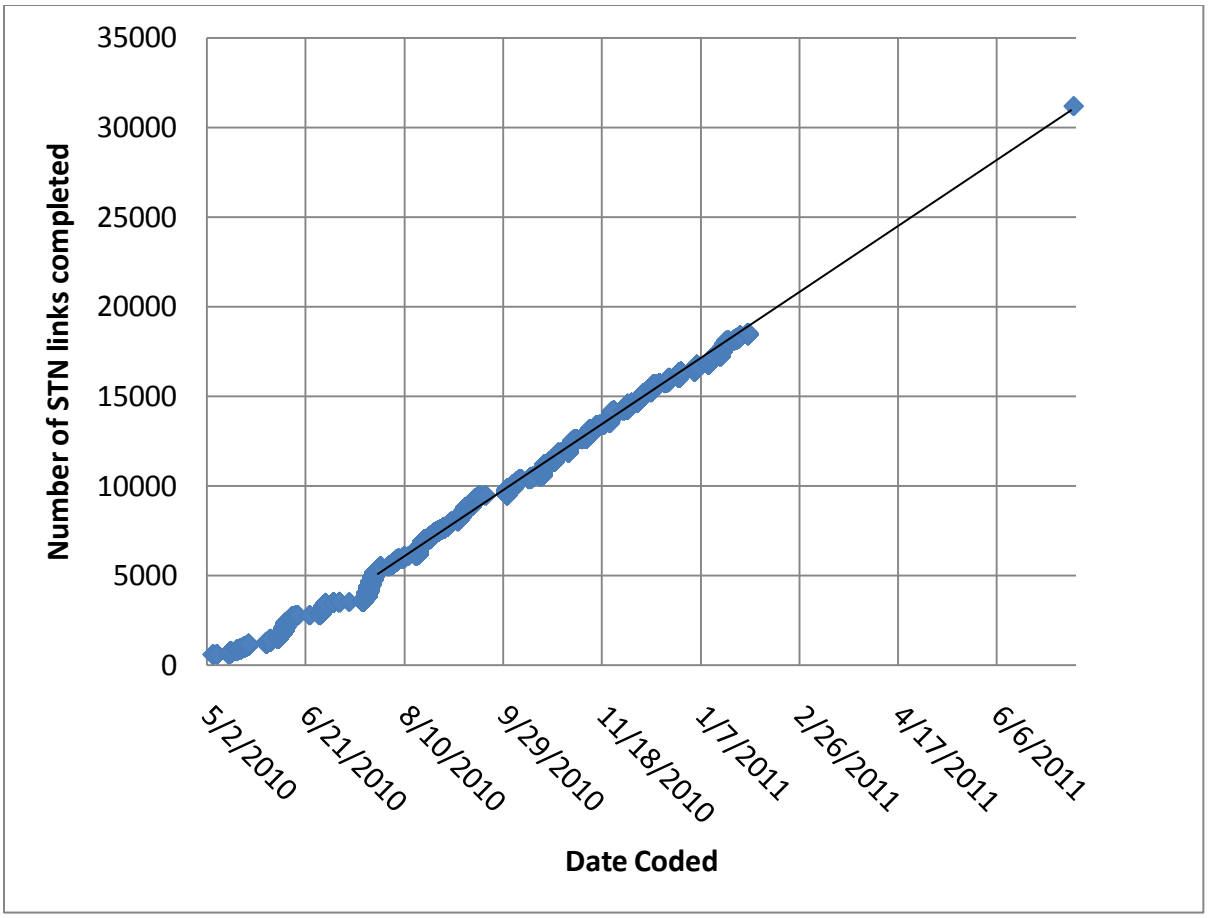

Figure 4.22. Statewide progress shown as completed STN links versus the date the link was coded

It can be seen in Figure 4.22 that the rate of completion for STN links has been steady from June 2010 to February 2011. The goal of 31,183 completed STN links can be seen at the top of the figure. A line has been extended from the current STN link completion data to the statewide goal. This line indicates that at the current rate, the project will reach completion in the time allotted for the project.

Table 4.6. STN and WISLR each contain many links and thousands of miles of roadway

LRS	Number of Links	Miles of roadway
STN	31,183	12,000
WISLR	780,877	116,500

It can be seen in Table 4.6 that the effort to code was greatly increased by the steep number of links in STN. As mentioned previously, there are currently 39,627 records in the link_link table, indicating that there are approximately 1.3 WISLR links per STN coded STN link.

4.3 Dane County Case Study

To further illustrate the robust nature of the link_link table, a case study was performed on Dane County, one of the first counties to be coded in the link_link table. First, crash data was moved from STN to WISLR. Then those same crashes were moved from WISLR back to STN. The following sections describe the results of both parts of the case study.

There are 1,520 STN links in Dane County, which accounts for approximately 445 miles of the state routes in Wisconsin. From 2005 to 2009, there were 17,740 crashes in Dane County. After executing the Point Moving Program, 17,740 (100%) crash points moved from STN to WISLR.

As discussed in Section 3.7, a technique for moving data from WISLR back to STN was developed. The 17,740 crash points that were moved from STN to WISLR in the previous portion of the case study were processed through the modified Point Moving Program. While the number of output records matched the number of input records, there were 444 points (3%) that moved to a different STN link than the orginal STN link on which the point had started.

Each discrepancy was a result of one of three situations. The first situation was where a turn-lane flag existed. An STN point moved from one STN turn-lane link and returned to the other

STN turn-lane link in that area. The second situation was where a median crossover flag existed. By convention, an STN point on the STN median crossover will never move back to the same STN median crossover link. These two situations were expected as this was a result of creating conventions for handling the flags during processing.

The third situation is similar to the turn-lane and median crossover situations in that there are multiple STN links to one WISLR link. However, this situation is neither a turn-lane nor a median crossover. This situation involves an interchange where one direction of road splits and the roads travel parallel for a significant distance. STN splits these roads correctly, and therefore represents this area with two links. WISLR on the other hand does not split the road at these locations, and therefore represents the same area with only one link. This causes STN points to travel to one WISLR link, but travel back to either of the STN links.

4.4 Conclusion

This chapter discussed the results of implementing the link_link table on a statewide level, described the coding effort necessary to complete the statewide link_link table, and showed the results of moving data points back and forth between STN and WISLR. The results shown are deemed successful, given that while the merge technique takes considerable time to perform, the results show a 100% data transfer between the systems for counties that have been coded.

CHAPTER 5

CONCLUSION AND FUTURE WORK

5.1 Conclusion

The results of the research presented in this thesis illustrate that the Link_link merge technique is effective for connecting two Linear Referencing Systems (LRSs). The LRSs used in this merge were the State Trunk Network (STN) and the Wisconsin Information Systems for Local Roads (WISLR), two road networks that were created by the Wisconsin Department of Transportation (WisDOT) in order to manage state routes and local roads, respectively.

The first goal of this research was to develop a simple yet robust method for merging the LRSs without interrupting business practices at WisDOT. Another goal was to expand the method by creating a process for flagging areas where STN and WISLR displayed significant differences, as well as developing a systematic approach to quality-checking the merge technique. The final goal of this research was to utilize the merge technique to move data back and forth between each system.

The results show that the link_link table and associated programs successfully meet the expectations of this research project. The link_link table relates a section of a link in STN to a section of a corresponding link in WISLR. Expanding the link_link table to incorporate differences and problems in STN and WISLR greatly increased the accuracy and reliability of the final table. The Quality Assurance/Quality Control (QA/QC) measures assure that each STN

link is correctly represented in each county table before the county table is appended to the statewide link_link table. The results of moving data between the two systems were successful, proving that the link _link method is robust enough to handle moving data between the systems.

5.2 Future Work

STN and WISLR are constantly updated relative to both cartography and attribute data because of new road construction projects and updates to the systems. Therefore, an annual version of STN and WISLR is released containing newly completed projects as well as updated map features. The link_link table is currently being coded using the updated 2009 STN and WISLR data. Given that WisDOT occasionally retires, merges, and splits links in the systems, significant differences in the link_link table will occur over time. A process for annually updating the link_link table should be developed.

Problem areas that are flagged in the link_link table should be submitted to the WisDOT employees responsible for updating the STN and WISLR systems, respectively. These problematic areas are not frequent, as described in Chapter 4, Results, but do require attention to assure complete accuracy of the link_link table throughout the state.

Data moving techniques were developed and tested in this research, but more advanced methods should be considered in the future. Logical conventions for handling data movement between the systems in areas around turn-lanes, median crossovers, and gore points should be developed and implemented in the data moving process. The flag columns provide a platform for implementing these changes.

The research performed for this thesis was extensive, but further research is recommended to perfect the link_link table. This thesis concludes with the expectation that this methodology can be implemented in other states as well.

REFERENCES

Beyer, H. L. 2004. Hawth's Analysis Tools for ArcGIS. Available at http://www.spatialecology.com/htools.

Curtin, K. M., Nicoara, G., and R. R. Arifin. 2007. A Comprehensive Process for Linear Referencing. Journal of the Urban and Regional Information Systems Association 19(2): 23-32.

Dueker, K. J., and J. A. Butler. 2000. A geographic information system framework for transportation data sharing. *Transportation Research Part C-Emerging Technologies 8(1-6): 13-36.*

Graettinger, A. J, Qin, X., Spear, G., Parker, S. T., and S. Forde. 2008. State and Non-State Network Mapping Integration. *Proceedings of the 2008 Mid-Continent Transportation Research Forum*, Madison, WI.

Graettinger, A. J., X. Qin, G. Spear, S. T. Parker, and S. Forde. 2009. "Combining State Route and Local Road Linear Referencing System Information" Journal of the Transportation Research Record, Vol. 2121, p. 152-159.

Kiel, D., and J. Pollack. 1998. Issues in adapting linear referencing systems for transportation applications: current practice and future outlook. Journal of Computing in Civil Engineering 12(2): 60-61.

Miller, H.J. and S. L Shaw. "GIS-T Data Models," Oxford UniversityPress, 2001.

O'Neill, W. A. and E. A. Harper. 1997. Linear location translation within GIS. National Cooperative Highway Research Program, Transportation Research Board, Vol. 1593, p. 55-63.

Scarponcini, P. 2002. Generalized model for linear referencing intransportation. *GeoInformatica 6(1): 35-55*.

Sester, M., Anders, K., and V. Walker. 1998. Linking objects of different spatial data sets by integration and aggregation. *GeoInformatica 2(4), 335-358*.

Vonderohe, A. P., Chou, C.L., Sun, F. and T. M. Adams. 1997. A generic data model for linear referencing systems. *NCHRP Research Results Digest 218*, National Cooperative Highway Research Program, Transportation Research Board, Washington, DC.

Vonderohe, A. P., and T. D. Hepworth. 1998. A Methodology for Design of a Linear Referencing System for Surface Transportation, Final Report. Project AT-4567,Sandia National Laboratories.

Wisconsin LCM Manual. Wisconsin Department of Transportation, Madison, WI.

APPENDIX A

This appendix will describe the process of coding the link_link table, explain the flag columns and when to use each one, and provide a step-by-step guide to performing the coding and the QA/QC for a single county. This guide was written with the expectation that the reader has a basic understanding of the ArcGIS Desktop software and the format of the link_link table.

The first step in coding records in the link_link table is to determine which links are to be coded. Refer to the STN and WISLR links shown in the Figure A.1. These links will be used for this coding example. It should also be noted that there are two STN links in Figure A.1, representing both directions of the same section of road.

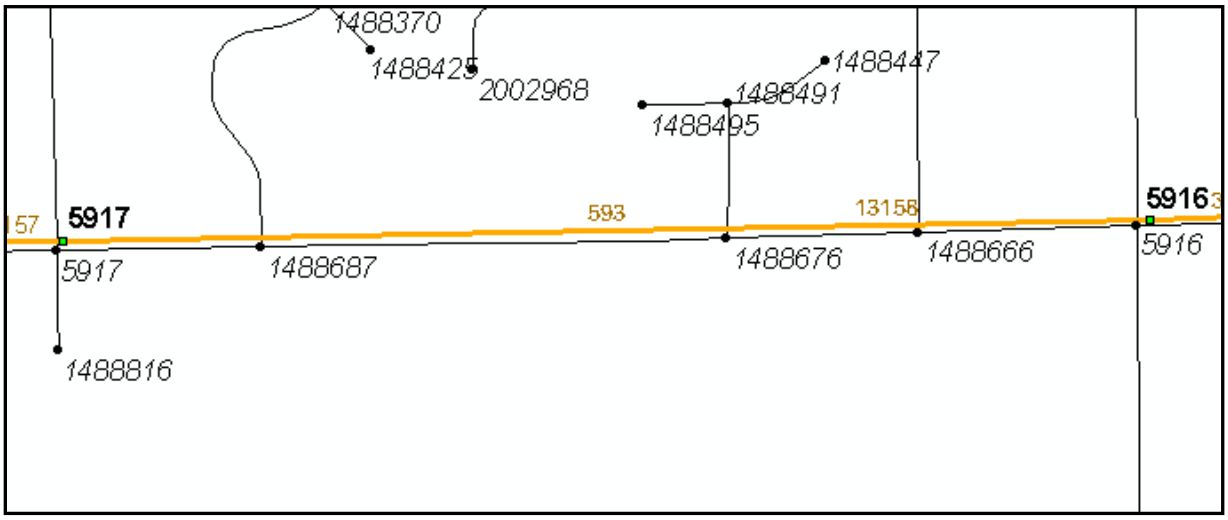

Figure A.1. STN and WISLR sections for link_link coding example

The STN links, WISLR links, STN sites, WISLR sites, STN chains, Access Points, and the county names shapefiles should be added to the ArcMap document. It is beneficial to display the

labels for the STN and WISLR links and sites to aid in determining the direction of each link being coded.

To start entering data into the link_link table, the table must be prepared. The table should follow the structure of Table A.1. In Figure A.1, notice that there are four WISLR links representing the same section of road that STN represents with only one link. This means that for each STN link in this example, there will be four WISLR links. In this example, each WISLR also represents a record in the link_link table; therefore, each direction of travel will require four records in the link_link table.

Table A.1. Link_link table prior to adding STN and WISLR data

STNid	STNstart	STNend	WISLRid	WISLRstart	WISLRend

Display the attributes of the STN links using the Identify tool. Choose which direction to code first. The first STN link to be coded is STN link 593. Using the Identify tool (not shown), it can be seen that the REF_SITE_F value for link 593 is 5917. This is the ID number of the STN from site. From the map, it can be seen that STN site 5917 is on the left side of Figure A.1. The REF_SITE_T value is 5916, on the right side of Figure A.1, indicating that the STN link travels from left to right. Given that there will be four records for this STN link, the first column of four records can be coded as shown in Table A.2. The first STN start and last STN end can be entered into the table at this point as well. The start of the first STN portion is always zero, and the STN end of the last STN portion is always the full length of the STN link. In this example, the STN link length is 1010.

Table A.2. The first column of the link_link table is populated for one direction as well as the STN link start and end values

STNid	STNstart	STNend	WISLRid	WISLRstart	WISLRend
593	0				
593					
593					
593		1010			

Using the Identify tool, the WISLR link IDs can be obtained. The WISLR site labels are used to determine the direction of each WISLR link. The first WISLR link to be coded into the table is the WISLR link with the corresponding STN site ID (5917). It is a convention to code the WISLR links following the direction of the STN link (e.g. from left to right). The first WISLR link to be coded into the table is link 4195671, followed by the link 4196038, the second in the line. The third WISLR link to be coded is link 4196039, followed by link 4196030. The link_link table should be populated as shown in Table A.3.

Table A.3. The link_link table has been populated with WISLR IDs

STNid	STNstart	STNend	WISLRid	WISLRstart	WISLRend
593	0		4195671		
593			4196038		
593			4196039		
593		1010	4196030		

The WISLR start and WISLR end for each record can be entered by using the data obtained from the Identify tool for each link. The lengths for the WISLR links are as follows: 1003, 2270, 950, and 1056, as shown in Table A.4.

Table A.4. The link_link table has now been coded with WISLR start and end values

STNid	STNstart	STNend	WISLRid	WISLRstart	WISLRend
593	0		4195671	0	1003
593			4196038	0	2270
593			4196039	0	950
593		1010	4196030	0	1056

The link_link table has now been coded with information gathered from the attribute data. The next step is to check for Access Points. For this example, there are no Access Points present, so a ratio will have to be calculated to determine the remaining STN start and STN end values.

The ratio formula for the first STN end value is as follows:

STN end = STN start (WISLR part / WISLR full) * STN length

STN end = 0 + (1003 / (1003 + 2270 + 950 + 1056)) * 1010

STN end = 192

The STN end is rounded to the nearest whole number, which equals 192. This value is entered into both the STN end and the STN start of the next record, indicating that the first portion ends at 192, and the next portion starts at 192, as shown in Table A.5.

Table A.5. The first STN end value is calculated using a ratio formula

STNid	STNstart	STNend	WISLRid	WISLRstart	WISLRend
593	0	192	4195671	0	1003
593	192		4196038	0	2270
593			4196039	0	950
593		1010	4196030	0	1056

This step is repeated for the next two records as well. Remember that in the second ratio calculation the STN start value is now 192, not zero as in the first ratio formula. Table A.6 shows the results of the hand-coding procedure with each STN start and STN end calculated using the ratio. Remember that this is only for one direction.

Table A.6. The link_link table has now been coded for one STN link

STNid	STNstart	STNend	WISLRid	WISLRstart	WISLRend
593	0	192	4195671	0	1003
593	192	626	4196038	0	2270
593	626	808	4196039	0	950
593	808	1010	4196030	0	1056

Now that one direction has been coded, the next direction can be coded without the use of the ratio formula. Note that if the STN links are not the same length, such as on a divided highway, the ratio formula must be used regardless.

Add four records to the link_link table. Obtain the opposite STN link ID and place it into the link_link table, along with the first STN start and last STN end values, as shown in Table A.7.

Table A.7. The opposite STN link is now to be coded into the link_link table

STNid	STNstart	STNend	WISLRid	WISLRstart	WISLRend
593	0	192	4195671	0	1003
593	192	626	4196038	0	2270
593	626	808	4196039	0	950
593	808	1010	4196030	0	1056
13158	0				
13158					
13158					
13158		1010			

Obtain the WISLR IDs using the Identify tool for each of the four WISLR links that represent the STN link being coded. Use the WISLR sites to ensure the correct direction WISLR link is being coded. Table A.8 shows the WISLR IDs, WISLR starts and WISLR ends in the link_link table.

Table A.8. The link_link table now contains the WISLR IDs, WISLR starts, and WISLR ends for the current direction

STNid	STNstart	STNend	WISLRid	WISLRstart	WISLRend
593	0	192	4195671	0	1003
593	192	626	4196038	0	2270
593	626	808	4196039	0	950
593	808	1010	4196030	0	1056
13158	0		4204876	0	1056
13158			4204885	0	950
13158			4204884	0	2270
13158		1010	4204517	0	1003

Now that the attribute information has been entered for these STN and WISLR links, the remaining STN start and STN end values can now be calculated by subtracting the STN start and end values of the previous direction from the full length of the current STN link. The formula is as follows:

1st STN end = STN length – last STN start

1st STN end = 1010 – 808

1st STN end = 202

Working from the first row to the second and so on for the current STN link, the values of the last row to next-to-last row and so on, can be subtracted from the full STN length. The rest of the calculations are as follows:

$$1010 – 626 = 384$$

$$1010 – 192 = 818$$

These values are entered into the link_link table, as shown in Table A.9.

Table A.9. The final link_link table for the STN links in this example

STNid	STNstart	STNend	WISLRid	WISLRstart	WISLRend
593	0	192	4195671	0	1003
593	192	626	4196038	0	2270
593	626	808	4196039	0	950
593	808	1010	4196030	0	1056
13158	0	202	4204876	0	1056
13158	202	384	4204885	0	950
13158	384	818	4204884	0	2270
13158	818	1010	4204517	0	1003

The records in the link_link table for the STN links in this example have been hand-coded using ratio calculations. The following example is of the same STN and WISLR links, only in this example, there are Access Points available.

The STN and WISLR links to be coded are shown in Figure A.2. Notice that there are now Access Points along the STN links. The labels for the Access Points display the STN link ID, the road name, and the offset of the Access Point. The labels for the WISLR links are also shown. The STN and WISLR sites labels have been turned off for this figure.

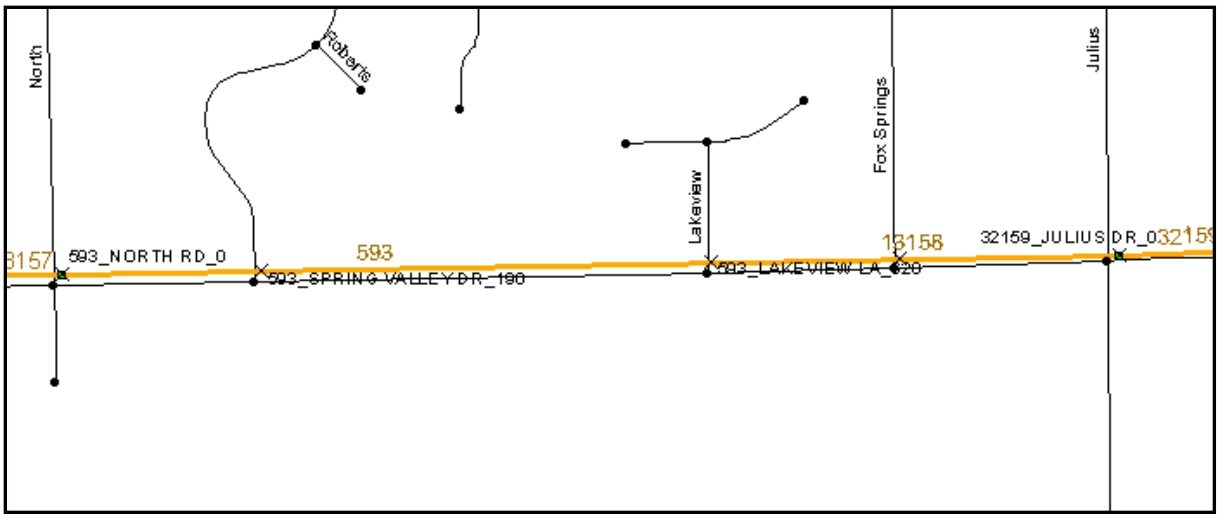

Figure A.2. Access Points are now available to aid in coding the link_link table

The link_link table should be coded with attributes exactly as in the previous example as shown in Table A.4; only in this example, the ratio calculation will not be used to obtain the STN start and STN ends. The STN start and end values will be obtained from the Access Points.

Access Points are typically on one direction in STN, not both directions; therefore, it is crucial to use the correct link with which to use the Access Points in the link_link table. It can be seen in Figure A.2 that the labels for the Access Points specify STN link 593. The link_link table for link 593 should be coded as in the previous example. Going from left to right, the first Access Point is on Spring Valley Drive (the label is not showing in the figure, but the first road adjoining the state route is Spring Valley Drive) at an offset of 190. The second Access Point is at Lakeview Lane at an offset of 620. The third and final Access Point label is not showing in the Figure A.2 (labels will sometimes not appear due to a small display area); however, the attributes state that the Access Point is on Fox Springs Drive at an offset of 800. All of the Access Points for this STN link are present and valid, so the offset information for these Access Points can now be entered into the link_link table, as shown in Table A.10.

Table A.10. One direction of the link_link table coded using Access Points

STNid	STNstart	STNend	WISLRid	WISLRstart	WISLRend
593	0	190	4195671	0	1003
593	190	620	4196038	0	2270
593	620	800	4196039	0	950
593	800	1010	4196030	0	1056

To obtain the STN start and STN end for the opposite direction, simply subtract the STN starts and ends from the full STN length as in the previous example. The full link_link table, coded using Access Points, is shown in Table A.11.

Table A.11. The link_link table for both STN links was coded using Access Points

STNid	STNstart	STNend	WISLRid	WISLRstart	WISLRend
593	0	190	4195671	0	1003
593	190	620	4196038	0	2270
593	620	800	4196039	0	950
593	800	1010	4196030	0	1056
13158	0	210	4204876	0	1056
13158	210	390	4204885	0	950
13158	390	820	4204884	0	2270
13158	820	1010	4204517	0	1003

The examples shown in this appendix were typical of actual situations encountered when coding the link_link table and should be used as a guides to coding the link_link table.

APPENDIX B

The following Visual Basic for Applications (VBA) code handles the Link_link Coding Tool that resides in ArcMap. The user selects map features per the instructions from the program and the program calculates and commits the link_link records to a database chosen by the user. The Link_link Coding Tool UI is shown in Figure 3.12.

ThisDocument:

```
'This Sub runs on click of the LinkLink Command Button.  It looks for any selected
features on the map and applies them accordingly
Private Sub LinkLink_Click()
    'ArcObjects
    Dim pMxDoc As IMxDocument
    Set pMxDoc = ThisDocument

    Dim pMap As IMap
    Set pMap = pMxDoc.FocusMap

    Dim pFeat As IFeature

    Dim temp As String

    On Error GoTo Err_ReadLayers

    'STN LINKS:  Read the STN Link Layer (lyr at position 3)
    Dim pFeatureLayerSTN As IFeatureLayer
    Dim pFeatSelSTN As IFeatureSelection
    Dim pFeatCursorSTN As IFeatureCursor

    Set pFeatureLayerSTN = pMap.Layer(3)
    Set pFeatSelSTN = pFeatureLayerSTN
    pFeatSelSTN.SelectionSet.Search Nothing, False, pFeatCursorSTN

    temp = ""

    Set pFeat = pFeatCursorSTN.NextFeature

    'Loop through all of the selected STN Links and add them to the STN list box at the
top of the form
    Do Until pFeat Is Nothing
        If isDuplicateStn(pFeat.Value(pFeat.Fields.FindField("RDWY_LIN_1"))) = False Then
```

```
            frmMain.lstSTN.AddItem pFeat.Value(pFeat.Fields.FindField("RDWY_LIN_1"))
            frmMain.lstSTN.List(frmMain.lstSTN.ListCount - 1, 1) =
pFeat.Value(pFeat.Fields.FindField("FROM_TO_DI"))
            frmMain.lstSTN.List(frmMain.lstSTN.ListCount - 1, 2) =
pFeat.Value(pFeat.Fields.FindField("REF_SITE_F"))
            frmMain.lstSTN.List(frmMain.lstSTN.ListCount - 1, 3) =
pFeat.Value(pFeat.Fields.FindField("REF_SITE_T"))
        End If
        Set pFeat = pFeatCursorSTN.NextFeature
    Loop

    'STN SITES:  Read the STN Sites Layer (lyr at position 1)
    Set pFeatureLayerSTN = pMap.Layer(1)
    Set pFeatSelSTN = pFeatureLayerSTN
    pFeatSelSTN.SelectionSet.Search Nothing, False, pFeatCursorSTN

    temp = ""

    'If more than one STN site was selected, throw up a MsgBox with an error
    If pFeatSelSTN.SelectionSet.Count > 1 Then
        MsgBox "Please choose only 1 STN Site.", vbInformation, "Error"
    'Else, add the site to the STN site box
    Else
        Set pFeat = pFeatCursorSTN.NextFeature
        If pFeat Is Nothing Then
        Else
            frmMain.txtStnFSite.Text = pFeat.Value(pFeat.Fields.FindField("REF_SITE_1"))
        End If
    End If

    'WISLR LINKS:  Read the WISLR Links Layer (lyr at position 4)
    Dim pFeatureLayerWISLR As IFeatureLayer
    Dim pFeatSelWISLR As IFeatureSelection
    Dim pFeatCursorWISLR As IFeatureCursor

    Set pFeatureLayerWISLR = pMap.Layer(4)
    Set pFeatSelWISLR = pFeatureLayerWISLR
    pFeatSelWISLR.SelectionSet.Search Nothing, False, pFeatCursorWISLR

    temp = ""

    Set pFeat = pFeatCursorWISLR.NextFeature

    'Loop through all of the selected WISLR Links and add them to the WISLR list box at
the top of the form
    Do Until pFeat Is Nothing
        If isDuplicateWislr(pFeat.Value(pFeat.Fields.FindField("RDWY_LINK_"))) = False
Then
            frmMain.lstWISLR.AddItem pFeat.Value(pFeat.Fields.FindField("RDWY_LINK_"))
            frmMain.lstWISLR.List(frmMain.lstWISLR.ListCount - 1, 1) =
pFeat.Value(pFeat.Fields.FindField("LCM_FROM_T"))
            frmMain.lstWISLR.List(frmMain.lstWISLR.ListCount - 1, 2) =
pFeat.Value(pFeat.Fields.FindField("REF_SITE_F"))
            frmMain.lstWISLR.List(frmMain.lstWISLR.ListCount - 1, 3) =
pFeat.Value(pFeat.Fields.FindField("REF_SITE_T"))
        End If
        Set pFeat = pFeatCursorWISLR.NextFeature
    Loop
```

```
    'WISLR SITES:  Read the WISLR Sites Layer (lyr at position 2)
    Set pFeatureLayerWISLR = pMap.Layer(2)
    Set pFeatSelWISLR = pFeatureLayerWISLR
    pFeatSelWISLR.SelectionSet.Search Nothing, False, pFeatCursorWISLR

    'If more than one STN site was selected, throw up a MsgBox with an error
    If pFeatSelWISLR.SelectionSet.Count > 1 Then
        MsgBox "Please choose only 1 WISLR Site.", vbInformation, "Error"
    'Else, add the site to the STN site box
    Else
        Set pFeat = pFeatCursorWISLR.NextFeature
        If pFeat Is Nothing Then
        Else
            frmMain.txtWislrFSite.Text =
pFeat.Value(pFeat.Fields.FindField("REF_SITE_I"))
        End If
    End If

    'ACCESS POINTS:  Read the Access Points Layer (lyr at position 0)
    Dim pFeatureLayerACCPT As IFeatureLayer
    Dim pFeatSelACCPT As IFeatureSelection
    Dim pFeatCursorACCPT As IFeatureCursor

    Set pFeatureLayerACCPT = pMap.Layer(0)
    Set pFeatSelACCPT = pFeatureLayerACCPT
    pFeatSelACCPT.SelectionSet.Search Nothing, False, pFeatCursorACCPT

    'If more than one Access Point was selected, throw up a MsgBox with an error
    If pFeatSelACCPT.SelectionSet.Count > 1 Then
        MsgBox "More than 1 Access Point was chosen; be advised errors may occur during
calculation.", vbInformation, "Error"
    'Else, add the Access Point offset to the current row End value and the next row's
start value
    End If
    'Else If
        'Get the selected Access Point
        Set pFeat = pFeatCursorACCPT.NextFeature
        'Iterate throuh the STN Links in the LinkLink section and get the End value of
the last one added.  This
        'value will always be the entire length of the STN Link that is being worked
with.
        Dim iter As Integer
        iter = 1
        Dim stnLength As String
        stnLength = ""
        While iter <= 20
            If iter = 20 Then
                If frmMain.Controls("txtStnEnd" & iter).Text <> "" Then
                    stnLength = frmMain.Controls("txtStnEnd" & iter).Text
                End If
            ElseIf frmMain.Controls("txtStnEnd" & iter).Text <> "" And
frmMain.Controls("txtStnEnd" & (iter + 1)) = "" Then
                stnLength = frmMain.Controls("txtStnEnd" & iter).Text
                iter = 21
            End If
            iter = iter + 1
        Wend
```

```
        'If no Access Point was selected, we need to make sure any End value text boxes
that were locked are unlocked
        If pFeat Is Nothing Then
            'Go through each added link; if it is locked, make sure it has had an Access
Point added (End value is not
            'equal to STN Link's length).  If it hasn't had an Access Point added unlock
it.
            For i = 1 To 20
                If frmMain.Controls("txtStnEnd" & i).Locked = True And
frmMain.Controls("txtStnEnd" & i).Text = stnLength Then
                        frmMain.Controls("txtStnEnd" & i).Locked = False
                End If
            Next i
        Else
            iter = 1
            While iter <= 20
                'Find the locked Stn End value text box.  This is the one that is
awaiting an Access Point
                If frmMain.Controls("txtStnEnd" & iter).Locked = True And
frmMain.Controls("txtStnEnd" & iter).Text = stnLength Then
                    'Check that the Access Point's associated link Id is the same as the
link that we're dealing with
                    If CLng(frmMain.Controls("lblStnId" & iter).Caption) =
pFeat.Value(pFeat.Fields.FindField("RWLK_ID")) Then
                        'Check that we aren't trying to add an Access Point to the last
row of a link direction
                        If frmMain.Controls("lblStnId" & iter).Caption =
frmMain.Controls("lblStnId" & (iter + 1)).Caption Then
                            frmMain.Controls("txtStnEnd" & iter).Text =
pFeat.Value(pFeat.Fields.FindField("ACSI_PT__1"))
                            frmMain.Controls("txtStnStart" & (iter + 1)).Text =
pFeat.Value(pFeat.Fields.FindField("ACSI_PT__1"))
                        Else
                            MsgBox "Cannot add access point to the last row in a
direction of links.", vbInformation
                            For i = 1 To 20
                                If frmMain.Controls("txtStnEnd" & i).Locked = True And
frmMain.Controls("txtStnEnd" & i).Text = stnLength Then
                                    frmMain.Controls("txtStnEnd" & i).Locked = False
                                End If
                            Next i
                        End If
                    Else
                        MsgBox "The selected Access Point does not match the
corresponding link.", vbExclamation
                        For i = 1 To 20
                            If frmMain.Controls("txtStnEnd" & i).Locked = True And
frmMain.Controls("txtStnEnd" & i).Text = stnLength Then
                                frmMain.Controls("txtStnEnd" & i).Locked = False
                            End If
                        Next i
                    End If
                    iter = 21
                End If
                iter = iter + 1
            Wend
        End If
    'End If
```

```
    'Unhide Form
    frmMain.Show

    Exit Sub

Err_ReadLayers:
    MsgBox "Feature Input Error:  " & Error, vbCritical

End Sub

'This function ensures that the same STN Link is not added more than once to the STN list
box
Private Function isDuplicateStn(ByVal id As String) As Boolean
    isDuplicateStn = False
    For i = 0 To frmMain.lstSTN.ListCount - 1
        If frmMain.lstSTN.List(i, 0) = id Then
            isDuplicateStn = True
        End If
    Next i
End Function

'This function ensures that the same WISLR Link is not added more than once to the WISLR
list box
Private Function isDuplicateWislr(ByVal id As String) As Boolean
    isDuplicateWislr = False
    For i = 0 To frmMain.lstWISLR.ListCount - 1
        If frmMain.lstWISLR.List(i, 0) = id Then
            isDuplicateWislr = True
        End If
    Next i
End Function
```

FrmMain Code:

```
Private Sub btnCalculate_Click()
    Dim StnLinkId As String
    Dim STNLinkLength As Integer
    Dim stnNewLength As Double
    Dim stnNewend As Double
    Dim iter As Integer
    Dim cnt As Integer
    Dim sumWISLR As Long
    Dim isAccPts As Boolean

    Dim StnId As Integer
    StnId = 0
    Dim sect As Integer
    sect = 0
    Dim P As Integer
    Dim LockedSE As Double
    Dim LockedSS As Double
    Dim rate As Double
    Dim NewSE As Integer
    Dim qt As Integer
```

```
    On Error GoTo Err_Calculate

    isAccPts = False
    iter = 1

    'Loop through all valid Stn Link Ids and check if any of their End value text boxes
are locked
    While Me.Controls("lblStnId" & iter) <> "" And iter <= 20
        'If a locked End value text box is found, we assume an Access Point was added.
Set the AccPts bool to True
        If Me.Controls("txtStnEnd" & iter).Locked = True Then
            isAccPts = True
        End If
        iter = iter + 1
    Wend

    'If we have Access Points present, go to a Sub that will ensure that the direction of
links with Access Points
    'appears first in the LinkLink section.  It must appear first since the second
direction of links Start/End values
    'are based off of the Start/End values from the first direction of links.
    If isAccPts Then
        Call OrderAccPts(iter - 1)
    End If
    qt = 1

    For qt = 1 To (iter - 1)
        If CInt(Me.Controls("txtSTNstart" & qt).Text) <> 0 Then
            Me.Controls("txtSTNstart" & qt).Locked = True
        End If
    Next

    StnLinkId = Me.Controls("lblStnId1").Caption

    If StnLinkId = "" Then
        MsgBox "Please add links before calculating.", vbInformation
        Exit Sub
    End If
    STNLinkLength = CInt(Me.Controls("txtStnEnd" & ((iter - 1) / 2)))
    'StnLinkLength = CInt(Me.Controls("txtStnEnd1")
    iter = 1
    cnt = 0
    sumWISLR = 0
    'Get the number of links in a direction (cnt) and the sum of the Wislr link lengths
(sumWislr)

    While Me.Controls("lblStnId" & iter).Caption = StnLinkId And iter <= 20

        cnt = cnt + 1
        'sumWislr = sumWislr + CInt(Me.Controls("txtWislrEnd" & iter))
        iter = iter + 1

    Wend

    answer = vbYes
    'Both directions of links must be in the LinkLink section for the calculations to be
accurate.  If the other
```

```
    'direction of links is not found, throw up a warning to the user
    If Me.Controls("lblStnId" & (cnt + 1)) = "" Then
        answer = MsgBox("It is recommended that links for both directions are calculated
at the same time for the sake of accuracy.  Proceed?", vbQuestion + vbYesNo, "Warning")
    End If

    If answer = vbYes Then
        'Do not calculate starts/ends for 1-to-1's
        If cnt = 1 Then
            MsgBox "There are not enough links to calculate distances.", vbInformation
        Else

            For iter = 1 To cnt
            If Me.Controls("txtSTNstart" & iter).Locked = False And
Me.Controls("txtSTNend" & iter).Locked = False Then
                sect = 0
                P = iter
                Do Until Me.Controls("txtSTNEnd" & P).Locked = True Or
Me.Controls("lblSTNid" & P) <> StnLinkId
                    sect = sect + 1
                    sumWISLR = sumWISLR + CInt(Me.Controls("txtWISLRend" & P).Text)
                    P = P + 1
                Loop
                LockedSS = 0
                LockedSE = CDbl(Me.Controls("txtSTNend" & P).Text)

                MsgBox (P)

                If Me.Controls("txtSTNend" & P).Locked = True Then
                sumWISLR = sumWISLR + CInt(Me.Controls("txtWISLRend" & (sect +
iter)).Text)
                End If

                MsgBox (sumWISLR & ",   " & (sect + iter))

                For P = iter To (sect + iter)
                    If Me.Controls("txtSTNend" & P).Locked = False Then
                        rate = (CDbl(Me.Controls("txtWISLRend" & P).Text) / sumWISLR) *
(LockedSE - LockedSS)
                        rate = Round(rate, 0)
                        NewSE = CInt(Me.Controls("txtSTNstart" & P).Text) + rate
                        Me.Controls("txtSTNstart" & (P + 1)).Text = NewSE
                        Me.Controls("txtSTNstart" & (P + 1)).Locked = True
                        Me.Controls("txtSTNend" & P).Text = NewSE
                        Me.Controls("txtSTNend" & P).Locked = True
                    ElseIf Me.Controls("txtSTNend" & P).Locked = True Then
                    End If
                Next

            ElseIf Me.Controls("txtSTNstart" & iter).Locked = True And
Me.Controls("txtSTNend" & iter).Locked = False Then
                LockedSS = CInt(Me.Controls("txtSTNstart" & iter).Text)
                sect = 0
                P = iter
                sumWISLR = 0
                Do Until Me.Controls("txtSTNend" & P).Locked = True Or P = cnt
                    sect = sect + 1
                    sumWISLR = sumWISLR + CInt(Me.Controls("txtWISLRend" & P).Text)
```

```
                P = P + 1
            Loop
            sumWISLR = sumWISLR + CInt(Me.Controls("txtWISLRend" & (sect +
iter)).Text)
            LockedSE = CInt(Me.Controls("txtSTNend" & (sect + iter)).Text)
            For P = iter To (sect + iter)
                If Me.Controls("txtSTNend" & P).Locked = False And P <> cnt Then
                    rate = (CDbl(Me.Controls("txtWISLRend" & P).Text) / sumWISLR) *
(LockedSE - LockedSS)
                    rate = Round(rate, 0)
                    NewSE = CInt(Me.Controls("txtSTNstart" & P).Text) + rate
                    Me.Controls("txtSTNstart" & (P + 1)).Text = NewSE
                    Me.Controls("txtSTNstart" & (P + 1)).Locked = True
                    Me.Controls("txtSTNend" & P).Text = NewSE
                    Me.Controls("txtSTNend" & P).Locked = True
                ElseIf Me.Controls("txtSTNend" & P).Locked = True Then
                End If
            Next
        ElseIf Me.Controls("txtSTNstart" & iter).Locked = False And
Me.Controls("txtSTNend" & iter).Locked = True Then

        ElseIf Me.Controls("txtSTNstart" & iter).Locked = True And
Me.Controls("txtSTNEnd" & iter).Locked = True Then

        End If

    If CInt(Me.Controls("txtSTNend" & iter).Text) > STNLinkLength Then
    Me.Controls("txtSTNend" & iter).Text = STNLinkLength
    End If

    Next

        btnCalculate.Enabled = False
    End If
End If

   'Start calculations on the other direction of links.  Check again and throw a warning
if the other
        'direction of links is not found
        StnLinkId = Me.Controls("lblStnId" & (cnt + 1))
        If StnLinkId = "" Then
            MsgBox "Please be advised that calculating only one direction of links
can result in inaccurate start and end values the other direction.", vbExclamation,
"Warning!"
        'For the other direction, our calculations aren't ratio based, but simply
mirror the starts and ends of
        'the previous direction to avoid rounding differences and ensure conistentcy
of lengths for both directions
        Else
            STNLinkLength = CInt(Me.Controls("txtStnEnd" & (cnt + 1)))
            stnNewLength = 0
            stnNewend = 0
            iter = cnt + 1
            cnt = cnt * 2


            'Go through the other direction of links
            While iter <= cnt
```

```
                    If Me.Controls("lblStnId" & iter).Caption <> StnLinkId Then
                        MsgBox "Unexpected link found.", vbInformation
                    Else

                        Me.Controls("txtStnStart" & iter).Text = stnNewend

                        If (iter <> cnt) Then
                            'Get the length of the current STN's mirrored link and apply
it
                            stnNewLength = CInt(Me.Controls("txtStnEnd" & (cnt - iter +
1))) - CInt(Me.Controls("txtStnStart" & (cnt - iter + 1)))
                            stnNewLength = Round(stnNewLength, 0)
                            stnNewend = stnNewLength + CInt(Me.Controls("txtStnStart" &
iter))
                            Me.Controls("txtStnEnd" & iter).Text = stnNewend

                        End If
                    End If
                    iter = iter + 1
                Wend
            End If
   Exit Sub
Err_Calculate:
    MsgBox "Calculate Error: " & Error

End Sub


'Calculate Start and End values while accounting for Access Points
Private Sub OrderAccPts(ByVal linkCnt As Integer)

    Dim dir1HasAccPts As Boolean
    Dim dir2HasAccPts As Boolean
    dir1HasAccPts = False
    dir2HasAccPts = False

    On Error GoTo Err_OrderAccPts

    If Me.Controls("lblStnId" & 1) = Me.Controls("lblStnId" & ((linkCnt / 2) + 1)) Then
        'Do Nothing.  The rest of the Calculate Sub will throw warnings for having only
one direction of links.
    Else
        For i = 1 To linkCnt
            'Check which direction of links that Access Points are applied
            If Me.Controls("txtStnEnd" & i).Locked = True Then
                If i <= (linkCnt / 2) Then
                    dir1HasAccPts = True
                Else
                    dir2HasAccPts = True
                End If
            End If
        Next i

        'Throw an error if Access Points are found on both directions of links
        If dir1HasAccPts = True And dir2HasAccPts = True Then
            MsgBox "For valid calculations, only one direction of STN Links should have
Access Points.", vbInformation
        ElseIf dir1HasAccPts = True And dir2HasAccPts = False Then
```

```
            'The direction of STN links with Access Points is already on top, do nothing.
        ElseIf dir1HasAccPts = False And dir2HasAccPts = True Then
            'The direction of STN links with Access Points is on bottom.  Now we need to
put it on top.

            'Declare the array for holding link row data
            Dim myLinkRows() As LinkRow
            ReDim myLinkRows(1 To linkCnt)

            'Declare a row for getting the data from each link row
            Dim aRow As LinkRow

            'Loop through each row and add it to the array
            For i = 1 To linkCnt
                Set aRow = New LinkRow
                aRow.StnId = Me.Controls("lblStnId" & i).Caption
                aRow.StnStart = Me.Controls("txtStnStart" & i).Text
                aRow.StnEnd = Me.Controls("txtStnEnd" & i).Text
                If Me.Controls("txtStnEnd" & i).Locked = True Then
                    aRow.AccPt = True
                Else
                    aRow.AccPt = False
                End If
                aRow.WislrId = Me.Controls("lblWislrId" & i).Caption
                aRow.WislrStart = Me.Controls("txtWislrStart" & i).Text
                aRow.WislrEnd = Me.Controls("txtWislrEnd" & i).Text
                If Me.Controls("chkT" & i).Value = -1 Then
                    aRow.T = True
                Else
                    aRow.T = False
                End If
                aRow.M = Me.Controls("txtM" & i).Text
                aRow.G = Me.Controls("comboG" & i).ListIndex
                If Me.Controls("chkW" & i).Value = -1 Then
                    aRow.W = True
                Else
                    aRow.W = False
                End If
                If Me.Controls("chkP" & i).Value = -1 Then
                    aRow.P = True
                Else
                    aRow.P = False
                End If
                aRow.Cmnt = Me.Controls("txtHiddenComment" & i).Text
                Set myLinkRows(i) = aRow
            Next i

            'Now that all of the data for each row is saved in an array, reverse the data
based on direction and
            'apply it to the correct row
            For i = 1 To (linkCnt / 2)

                Dim newId As Integer
                newId = (linkCnt / 2) + i

                Dim newRow As LinkRow
                Set newRow = myLinkRows(newId)
```

```
    Me.Controls("lblStnId" & i).Caption = newRow.StnId
    Me.Controls("txtStnStart" & i).Text = newRow.StnStart
    Me.Controls("txtStnEnd" & i).Text = newRow.StnEnd
    If newRow.AccPt = True Then
        Me.Controls("txtStnEnd" & i).Locked = True
    Else
        Me.Controls("txtStnEnd" & i).Locked = False
    End If
    Me.Controls("lblWislrId" & i).Caption = newRow.WislrId
    Me.Controls("txtWislrStart" & i).Text = newRow.WislrStart
    Me.Controls("txtWislrEnd" & i).Text = newRow.WislrEnd
    If newRow.T = True Then
        Me.Controls("chkT" & i).Value = -1
    Else
        Me.Controls("chkT" & i).Value = 0
    End If
    Me.Controls("txtM" & i).Text = newRow.M
    Me.Controls("comboG" & i).ListIndex = newRow.G
    If newRow.W = True Then
        Me.Controls("chkW" & i).Value = -1
    Else
        Me.Controls("chkW" & i).Value = 0
    End If
    If newRow.P = True Then
        Me.Controls("chkP" & i).Value = -1
    Else
        Me.Controls("chkP" & i).Value = 0
    End If
    Me.Controls("txtHiddenComment" & i).Text = newRow.Cmnt
Next i

For i = (linkCnt / 2) + 1 To linkCnt

    newId = i - (linkCnt / 2)

    Set newRow = myLinkRows(newId)

    Me.Controls("lblStnId" & i).Caption = newRow.StnId
    Me.Controls("txtStnStart" & i).Text = newRow.StnStart
    Me.Controls("txtStnEnd" & i).Text = newRow.StnEnd
    If newRow.AccPt = True Then
        Me.Controls("txtStnEnd" & i).Locked = True
    Else
        Me.Controls("txtStnEnd" & i).Locked = False
    End If
    Me.Controls("lblWislrId" & i).Caption = newRow.WislrId
    Me.Controls("txtWislrStart" & i).Text = newRow.WislrStart
    Me.Controls("txtWislrEnd" & i).Text = newRow.WislrEnd
    If newRow.T = True Then
        Me.Controls("chkT" & i).Value = -1
    Else
        Me.Controls("chkT" & i).Value = 0
    End If
    Me.Controls("txtM" & i).Text = newRow.M
    Me.Controls("comboG" & i).ListIndex = newRow.G
    If newRow.W = True Then
        Me.Controls("chkW" & i).Value = -1
    Else
```

```
                    Me.Controls("chkW" & i).Value = 0
                End If
                If newRow.P = True Then
                    Me.Controls("chkP" & i).Value = -1
                Else
                    Me.Controls("chkP" & i).Value = 0
                End If
                Me.Controls("txtHiddenComment" & i).Text = newRow.Cmnt
            Next i

        Else
            'Neither direction of links has Access Points, do nothing.
        End If

    End If

     Exit Sub

Err_OrderAccPts:
    MsgBox "Order Access Points Error: " & Error, vbCritical

End Sub


Private Sub btnCommit_Click()
    Dim strQuery As String
    Dim myCn As ADODB.Connection
    Dim isRows As Boolean
    Set myCn = New ADODB.Connection


    myCn.ConnectionString = "Provider=Microsoft.JET.OLEDB.4.0;" _
        & "Data Source=" & txtDatabase.Text & ";Persist Security Info=False;"

    On Error GoTo Err_DB
    myCn.Open

    isRows = False
    If Me.Controls("txtDateMod").Text = "" Then
        MsgBox ("There is no date added in the DateMod box.")
        Exit Sub
    ElseIf Me.Controls("txtCounty").Text = "" Then
        MsgBox ("There is no county entered into the County box.")
        Exit Sub
    ElseIf Me.Controls("txtCoder").Text = "" Then
        MsgBox ("There are no coder initals entered into the Coder box.")
        Exit Sub
    End If

    'Loop through the 20 rows in the LinkLink section
    For i = 1 To 20
        'If an empty row is found, exit the loop
        If Me.Controls("lblStnId" & i).Caption = "" Then
            i = 21
        Else
            'Contruct the query string by adding to it each field from the current row
```

```
            strQuery = "INSERT INTO
[link_link](STNid,STNstart,STNend,WISLRid,WISLRstart,WISLRend,T,M,G,W,P,Coder,DateMod,Com
ments,County) VALUES"
            strQuery = strQuery & "(" & Me.Controls("lblStnId" & i) & "," &
Me.Controls("txtStnStart" & i) & "," _
                    & Me.Controls("txtStnEnd" & i) & "," & Me.Controls("lblWislrId" & i)
& "," _
                    & Me.Controls("txtWislrStart" & i) & "," & Me.Controls("txtWislrEnd"
& i) & ","

            If (Me.Controls("chkT" & i).Value = -1) Then
                strQuery = strQuery & "'1',"
            Else
                strQuery = strQuery & "Null,"
            End If

            If (Me.Controls("txtM" & i).Text = "") Then
                strQuery = strQuery & "Null,"
            Else
                strQuery = strQuery & "'" & Me.Controls("txtM" & i).Text & "',"
            End If

            Select Case Me.Controls("comboG" & i).Value

                Case ""
                    strQuery = strQuery & "Null,"
                Case "T"
                    strQuery = strQuery & "'T',"
                Case "F"
                    strQuery = strQuery & "'F',"
                Case "B"
                    strQuery = strQuery & "'B',"
                Case Else
                    strQuery = strQuery & "Null,"

            End Select

            If (Me.Controls("chkW" & i).Value = -1) Then
                strQuery = strQuery & "'1',"
            Else
                strQuery = strQuery & "Null,"
            End If

            If (Me.Controls("chkP" & i).Value = -1) Then
                strQuery = strQuery & "'1',"
            Else
                strQuery = strQuery & "Null,"
            End If

            If (Me.Controls("txtCoder").Text = "") Then
                strQuery = strQuery & "Null,"
            Else
                strQuery = strQuery & "'" & Me.Controls("txtCoder").Text & "',"
            End If

            strQuery = strQuery & "'" & Me.Controls("txtDateMod").Text & "',"

            If (Me.Controls("txtHiddenComment" & i).Text = "") Then
```

```
                strQuery = strQuery & "Null,"
            Else
                strQuery = strQuery & "'" & Me.Controls("txtHiddenComment" & i).Text &
"',"
            End If

            strQuery = strQuery & "'" & Me.Controls("txtCounty").Text & "')"

            'Execute the query
            myCn.Execute strQuery

            strQuery = ""
            isRows = True
        End If
    Next i

    myCn.Close


    If isRows = True Then
        MsgBox "Rows Succesfully Added!", vbInformation
        btnCommit.Enabled = False
    Else
        MsgBox "No Rows Found!", vbInformation
    End If

    Exit Sub
Err_DB:
    MsgBox "Database Error:  " & Error & " Please make sure a valid database is
selected.", vbCritical
End Sub
```

APPENDIX C

The STN Points Generator program was written in VB.NET using Microsoft Visual Studio 2010. The code is presented in this appendix. Figure C.1 shows the UI for the STN Points Generator program. The user selects whether the points are being generated for STN or WISLR. If STN is chosen, the increment for the points will be 10 units; if WISLR is chosen, the increment will be 5.28 feet unless specified otherwise by the user. One required input is an Excel table with the link ID and link length in the first two columns of the table. The other required input is the Access Database to which the program will output the generated points table that will be used for QA/QC. If STN is chosen, the user must select a county from the combobox, otherwise every STN link in the table will be processed.

The output table contains a Unique ID, STN (or WISLR) link ID, and link offset.

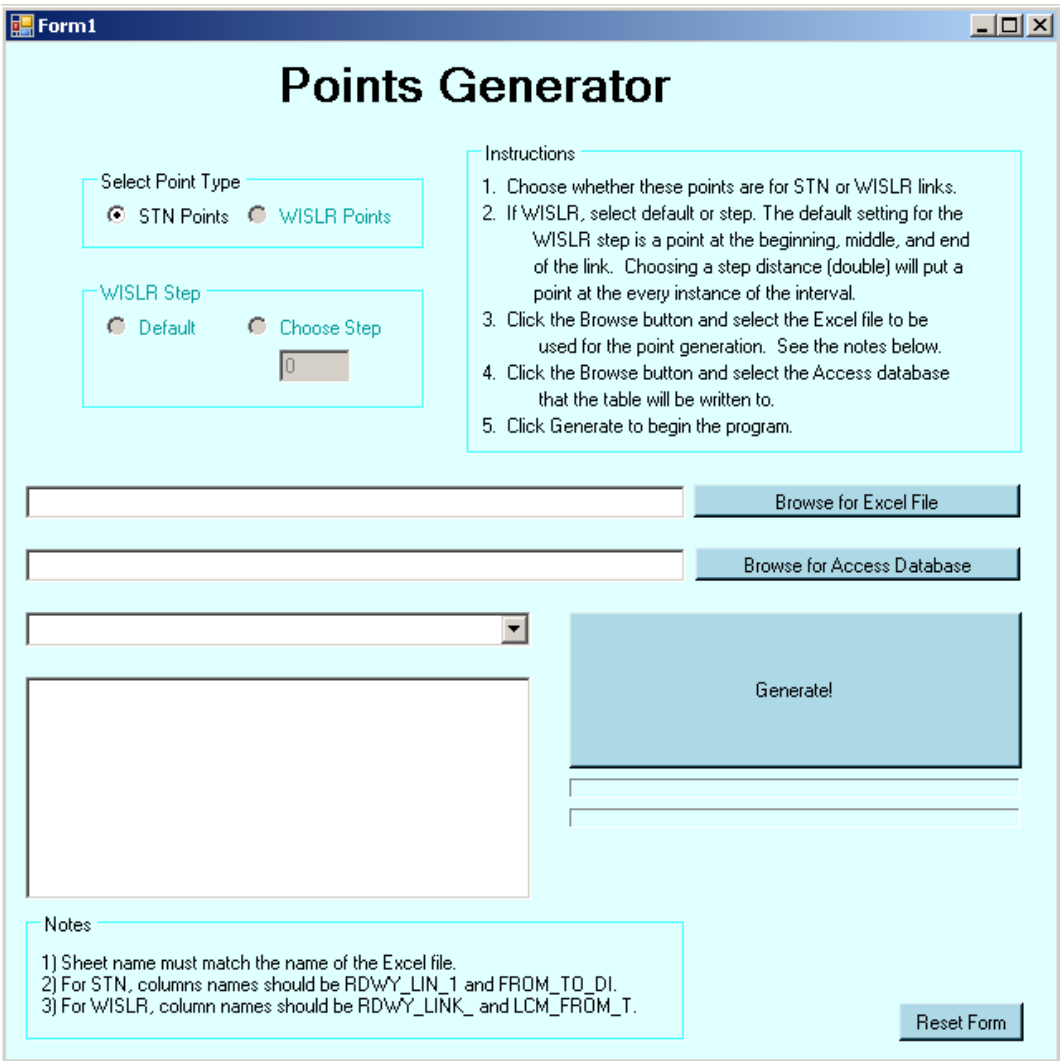

Figure C.1. STN Points Generator UI

Form1 Code:

```
Option Explicit On
Imports System
Imports System.IO
Imports System.Data
Imports System.Data.SqlClient
Imports System.Collections.Generic
Imports System.Text
Imports System.Math
Imports System.Data.OleDb

Public Class Form1
    Dim mdb_file_name As String
    Dim UserStep As Double = 0

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

```
rdoSTN.Checked = False
rdoWISLR.Checked = False
btnBrowseXL.Enabled = False
btnBrowseACC.Enabled = False
btnGenerate.Enabled = False
cmboCounty.Enabled = False
GroupBox2.Enabled = False
txtStep.Enabled = False
txtFilePath.Enabled = False
txtaccessfilepath.Enabled = False
txtStep.Text = 0

cmboCounty.Items.Add("All Wisconsin Counties")
cmboCounty.Items.Add("Adams")
cmboCounty.Items.Add("Ashland")
cmboCounty.Items.Add("Barron")
cmboCounty.Items.Add("Bayfield")
cmboCounty.Items.Add("Brown")
cmboCounty.Items.Add("Buffalo")
cmboCounty.Items.Add("Burnett")
cmboCounty.Items.Add("Calumet")
cmboCounty.Items.Add("Chippewa")
cmboCounty.Items.Add("Clark")
cmboCounty.Items.Add("Columbia")
cmboCounty.Items.Add("Crawford")
cmboCounty.Items.Add("Dane")
cmboCounty.Items.Add("Dodge")
cmboCounty.Items.Add("Door")
cmboCounty.Items.Add("Douglas")
cmboCounty.Items.Add("Dunn")
cmboCounty.Items.Add("Eau Claire")
cmboCounty.Items.Add("Florence")
cmboCounty.Items.Add("Fond du Lac")
cmboCounty.Items.Add("Forest")
cmboCounty.Items.Add("Grant")
cmboCounty.Items.Add("Green")
cmboCounty.Items.Add("Green Lake")
cmboCounty.Items.Add("Iowa")
cmboCounty.Items.Add("Iron")
cmboCounty.Items.Add("Jackson")
cmboCounty.Items.Add("Jefferson")
cmboCounty.Items.Add("Juneau")
cmboCounty.Items.Add("Kenosha")
cmboCounty.Items.Add("Kewaunee")
cmboCounty.Items.Add("La Crosse")
cmboCounty.Items.Add("Lafayette")
cmboCounty.Items.Add("Langlade")
cmboCounty.Items.Add("Lincoln")
cmboCounty.Items.Add("Manitowoc")
cmboCounty.Items.Add("Marathon")
cmboCounty.Items.Add("Marinette")
cmboCounty.Items.Add("Marquette")
cmboCounty.Items.Add("Menominee")
cmboCounty.Items.Add("Milwaukee")
cmboCounty.Items.Add("Monroe")
cmboCounty.Items.Add("Oconto")
cmboCounty.Items.Add("Oneida")
```

```
        cmboCounty.Items.Add("Outagamie")
        cmboCounty.Items.Add("Ozaukee")
        cmboCounty.Items.Add("Pepin")
        cmboCounty.Items.Add("Pierce")
        cmboCounty.Items.Add("Polk")
        cmboCounty.Items.Add("Portage")
        cmboCounty.Items.Add("Price")
        cmboCounty.Items.Add("Racine")
        cmboCounty.Items.Add("Richland")
        cmboCounty.Items.Add("Rock")
        cmboCounty.Items.Add("Rusk")
        cmboCounty.Items.Add("Saint Croix")
        cmboCounty.Items.Add("Sauk")
        cmboCounty.Items.Add("Sawyer")
        cmboCounty.Items.Add("Shawano")
        cmboCounty.Items.Add("Sheboygan")
        cmboCounty.Items.Add("Taylor")
        cmboCounty.Items.Add("Trempealeau")
        cmboCounty.Items.Add("Vernon")
        cmboCounty.Items.Add("Vilas")
        cmboCounty.Items.Add("Walworth")
        cmboCounty.Items.Add("Washburn")
        cmboCounty.Items.Add("Washington")
        cmboCounty.Items.Add("Waukesha")
        cmboCounty.Items.Add("Waupaca")
        cmboCounty.Items.Add("Waushara")
        cmboCounty.Items.Add("Winnebago")
        cmboCounty.Items.Add("Wood")

    End Sub
    Private Sub rdoSTN_Click(ByVal sender As System.Object, ByVal e As System.EventArgs)
Handles rdoSTN.Click

        rdoSTN.Checked = True
        btnBrowseXL.Enabled = True
        btnBrowseACC.Enabled = True
        btnGenerate.Enabled = True
        cmboCounty.Enabled = True
        GroupBox2.Enabled = False
        rdoWISLR.Enabled = False
        txtFilePath.Enabled = True
        txtaccessfilepath.Enabled = True

    End Sub
    Private Sub rdoWISLR_Click(ByVal sender As System.Object, ByVal e As
System.EventArgs) Handles rdoWISLR.Click

        rdoWISLR.Checked = True
        rdoDefault.Enabled = True
        rdoWISLRStep.Enabled = True
        btnBrowseXL.Enabled = False
        btnBrowseACC.Enabled = False
        btnGenerate.Enabled = False
        GroupBox2.Enabled = True
        rdoSTN.Enabled = False
        txtFilePath.Enabled = False
        txtaccessfilepath.Enabled = False
    End Sub
```

```
    Private Sub btnBrowseXL_Click(ByVal sender As System.Object, ByVal e As
System.EventArgs) Handles btnBrowseXL.Click
        OpenFileDialog1.Title = "Please select a file..."
        OpenFileDialog1.InitialDirectory = "C:\"
        OpenFileDialog1.Filter = "Excel Files|*.xls;*.xlsx;*.csv"
        OpenFileDialog1.FileName = ""

        If OpenFileDialog1.ShowDialog() = Windows.Forms.DialogResult.OK Then
            txtFilePath.Text = OpenFileDialog1.FileName
        End If
    End Sub

    Private Sub btnBrowseACC_Click(ByVal sender As System.Object, ByVal e As
System.EventArgs) Handles btnBrowseACC.Click

        OpenFileDialog2.Title = "Please select a file..."
        OpenFileDialog2.InitialDirectory = "C:\"
        OpenFileDialog2.Filter = "Excel Files|*.mdb;*.accdb"
        OpenFileDialog2.FileName = ""

        If OpenFileDialog2.ShowDialog() = Windows.Forms.DialogResult.OK Then
            txtaccessfilepath.Text = OpenFileDialog2.FileName
        End If

    End Sub



    Private Sub btnGenerate_Click(ByVal sender As System.Object, ByVal e As
System.EventArgs) Handles btnGenerate.Click

        Me.Enabled = False
        txtFilePath.Enabled = True
        mdb_file_name = txtaccessfilepath.Text.ToString
        txtStep.Enabled = False
        If rdoDefault.Checked = True Then
            UserStep = 0
        Else
            UserStep = txtStep.Text
        End If

        If rdoSTN.Checked = True Then

            Dim cn As New OleDbConnection("Provider=Microsoft.Jet.OLEDB.4.0;Data Source="
& mdb_file_name)
            Dim ssql As String
            ssql = "DROP TABLE STN_points"
            Dim cmd As New OleDbCommand(ssql, cn)
            cn.Open()
            If DoesTableExist("STN_points") = True Then
                Dim buttonYes As DialogResult
                buttonYes = MsgBox("A table entitled 'STN_Points' already exists in the
selected database.  Do you want the program to delete this table for you?  Cuz it can do
that.", MsgBoxStyle.YesNo, "Table Exists")
                If buttonYes = Windows.Forms.DialogResult.Yes Then
                    Dim buttonSure As DialogResult
```

```
                    buttonSure = MsgBox("Are you sure?", MsgBoxStyle.YesNo, "Delete
Table")
                    If buttonSure = Windows.Forms.DialogResult.Yes Then
                        cmd.ExecuteNonQuery()
                    Else
                        Exit Sub
                    End If
                Else
                End If
            Else
            End If
            cn.Close()

            Try
                Dim filePath As String = txtFilePath.Text.ToString
                'Get fileName out of filePath i.e. "Grant_STN_Links" from
"C:\Grant_STN_Links.xlsx"
                Dim fileName As String = pathToFile(filePath)
                Dim searchCriteria As String = cmboCounty.SelectedItem

                'Create Data Table and connection string
                Dim myDataTable As New DataTable
                Dim myDataRow As DataRow
                Dim queryString As String = ""
                Dim myConnString As String = "Provider=Microsoft.ACE.OLEDB.12.0;Data
Source=" & filePath & _
                    ";Extended Properties=" & Chr(34) & "Excel 12.0 Xml;HDR=YES;IMEX=1" &
Chr(34) & ";"

                'Create the SQL statement
                If searchCriteria = "" Then
                    cmboCounty.Enabled = True
                    MsgBox("No county is selected.  Please select a county, stupid.",
MsgBoxStyle.Exclamation, "No county selected")
                    Exit Sub
                ElseIf searchCriteria = "*All Counties" Then
                    queryString = "SELECT RDWY_LIN_1,FROM_TO_DI FROM [" & fileName & "$]"
                Else
                    queryString = "SELECT RDWY_LIN_1,FROM_TO_DI FROM [" & fileName & "$]
" & _
                                  "WHERE CTY_NAME='" & cmboCounty.SelectedItem.ToString &
"'"
                End If

                'Populate a Data Table with the information returned from the query
                Dim myAdapter As New OleDb.OleDbDataAdapter(queryString, myConnString)
                myAdapter.Fill(myDataTable)

                'Add points to new text file
                Dim uniqueID As Double = 1
                Dim currentID, j, currentDist As Double
                Dim sql As String
                Dim table_name1 As String = "STN_Points"
                Dim conn1 As New OleDbConnection("Provider=Microsoft.Jet.OLEDB.4.0;Data
Source=" & mdb_file_name)
                sql = "CREATE TABLE " & table_name1 & " (Unique_ID double, Link_ID
double, Link_Offset double)"
                Dim cmd3 As New OleDbCommand(sql, conn1)
```

```
                Dim rows As Integer
                Dim sumRows As Long = myDataTable.Compute("SUM(FROM_TO_DI)", "")
                rows = myDataTable.Rows.Count

                conn1.Open()
                Try
                    cmd3.ExecuteNonQuery()
                Catch ex As OleDb.OleDbException
                    MessageBox.Show(ex.Message, "OleDb Exception")
                    Exit Sub
                Catch ex As Exception
                    MessageBox.Show(ex.Message, "General Exception")
                    Exit Sub
                End Try
                conn1.Close()

                Dim conn2 As New OleDbConnection("Provider=Microsoft.Jet.OLEDB.4.0;Data
Source= " & mdb_file_name)
                Dim sql2 As String
                'sql2 = "INSERT INTO " & table_name1 & "(Unique_ID, Link_ID, Link_Offset)
VALUES (" & uniqueID & "," & currentID & "," & j & ")"
                'Dim cmdUpdate1 As New OleDbCommand("UPDATE " & table_name1, conn2)
                conn2.Open()
                For Each myDataRow In myDataTable.Rows
                    currentID = myDataRow(0)
                    currentDist = myDataRow(1)
                    For j = 0 To currentDist Step 10
                        sql2 = "INSERT INTO " & table_name1 & "(Unique_ID, Link_ID,
Link_Offset) VALUES (" & uniqueID & "," & currentID & "," & j & ")"
                        Dim cmd4 As New OleDbCommand(sql2, conn2)
                        cmd4.ExecuteNonQuery()

                        ListBox1.Items.Add(uniqueID.ToString & ", " & currentID.ToString
& ", " & j.ToString)
                        ListBox1.TopIndex = ListBox1.Items.Count - 1
                        ProgressBar1.Value = (j / currentDist) * 100
                        uniqueID = uniqueID + 1
                    Next
                    ProgressBar2.Value = (uniqueID / ((sumRows / 10) + rows)) * 100
                Next
                conn2.Close()

                'Close file
                Dim button As DialogResult
                button = MsgBox("Success! Your file was saved to " + mdb_file_name + ".
Do you want to view the results in the listbox?", MsgBoxStyle.YesNo, "Success!")
                If button = Windows.Forms.DialogResult.Yes Then
                Else

                    Me.Close()
                End If
                txtFilePath.Text = ""
                txtaccessfilepath.Text = ""
            Catch ex As Exception
                MsgBox("There was an error processing your request.",
MsgBoxStyle.Critical, "Error")
            End Try
        End If
```

```
        If rdoWISLR.Checked = True Then

            Dim cn As New OleDbConnection("Provider=Microsoft.Jet.OLEDB.4.0;Data Source="
& mdb_file_name)
            Dim sql3 As String = "DROP TABLE WISLR_points"
            Dim cmd As New OleDbCommand(sql3, cn)
            cn.Open()
            If DoesTableExist("WISLR_points") = True Then
                Dim buttonYes As DialogResult
                buttonYes = MsgBox("A table entitled 'WISLR_Points' already exists in the
selected database.  Do you want the program to delete this table for you?  Cuz it can do
that.", MsgBoxStyle.YesNo, "Table Exists")
                If buttonYes = Windows.Forms.DialogResult.Yes Then
                    Dim buttonSure As DialogResult
                    buttonSure = MsgBox("Are you sure?", MsgBoxStyle.YesNo, "Delete
Table")
                    If buttonSure = Windows.Forms.DialogResult.Yes Then
                        cmd.ExecuteNonQuery()
                    Else
                    End If
                Else
                End If
            Else
            End If
            cn.Close()

            Try

                Dim WISLRfilePath As String = txtFilePath.Text.ToString
                'Get fileName out of filePath i.e. "Grant_STN_Links" from
"C:\Grant_STN_Links.xlsx"
                Dim WISLRfileName As String = pathToFile(WISLRfilePath)
                Dim searchCriteria As String = cmboCounty.SelectedItem

                'Create Data Table and connection string
                Dim myDataTable1 As New DataTable
                Dim myDataRow1 As DataRow
                Dim queryString1 As String = ""
                Dim myConnString1 As String = "Provider=Microsoft.ACE.OLEDB.12.0;Data
Source=" & WISLRfilePath & _
                    ";Extended Properties=" & Chr(34) & "Excel 12.0 Xml;HDR=YES;IMEX=1" &
Chr(34) & ";"

                queryString1 = "SELECT RDWY_LINK_, LCM_FROM_T FROM [" & WISLRfileName &
"$]"

                'Populate a Data Table with the information returned from the query
                Dim myAdapter1 As New OleDb.OleDbDataAdapter(queryString1, myConnString1)
                myAdapter1.Fill(myDataTable1)

                'Add points to new text file
                Dim uniqueID As Long = 1
                Dim currentID, j, k, currentDist As Double
                Dim sql4 As String
                Dim table_name2 As String = "WISLR_Points"
                Dim cn3 As New OleDbConnection("Provider=Microsoft.Jet.OLEDB.4.0;Data
Source= " & mdb_file_name)
```

```
                Dim cmd3 As New OleDbCommand("CREATE TABLE " & table_name2 & " (Unique_ID
double, Link_ID double, Link_Offset double)", cn3)
                Dim rows As Integer
                Dim sumRows As Double = myDataTable1.Compute("SUM(LCM_FROM_T)", "")
                rows = myDataTable1.Rows.Count

                cn3.Open()
                Try
                    cmd3.ExecuteNonQuery()
                Catch ex As OleDb.OleDbException
                    MessageBox.Show(ex.Message, "OleDb Exception")
                    Exit Sub
                Catch ex As Exception
                    MessageBox.Show(ex.Message, "General Exception")
                    Exit Sub
                End Try
                cn3.Close()

                Dim cn4 As New OleDbConnection("Provider=Microsoft.Jet.OLEDB.4.0;Data
Source= " & mdb_file_name)
                Dim sql5 As String
                cn4.Open()

                For Each myDataRow1 In myDataTable1.Rows
                    currentID = myDataRow1(0)
                    currentDist = myDataRow1(1)

                    For j = 0 To currentDist Step (currentDist / 2)
                        sql4 = "INSERT INTO " & table_name2 & "(Unique_ID, Link_ID,
Link_Offset) VALUES (" & uniqueID & "," & currentID & "," & j & ")"
                        Dim cmd4 As New OleDbCommand(sql4, cn4)
                        cmd4.ExecuteNonQuery()

                        ListBox1.Items.Add(uniqueID & ", " & currentID & ", " & j)
                        ListBox1.TopIndex = ListBox1.Items.Count - 1
                        ProgressBar1.Value = (j / currentDist) * 100
                        uniqueID += 1
                    Next
                    If UserStep = 0 Then
                    Else
                        For k = UserStep To currentDist Step UserStep
                            sql5 = "INSERT INTO " & table_name2 & "(Unique_ID, Link_ID,
Link_Offset) VALUES (" & uniqueID & "," & currentID & "," & trunc(k, 2) & ")"
                            Dim cmd5 As New OleDbCommand(sql5, cn4)
                            cmd5.ExecuteNonQuery()

                            ListBox1.Items.Add(uniqueID & ", " & currentID & ", " &
trunc(k, 2))
                            ListBox1.TopIndex = ListBox1.Items.Count - 1
                            ProgressBar1.Value = (trunc(k, 2) / currentDist) * 100
                            uniqueID += 1
                        Next
                    End If
                    If UserStep <> 0 And uniqueID < ((3 * rows) + (sumRows / UserStep))
Then
                        ProgressBar2.Value = (uniqueID / ((3 * rows) + (sumRows /
UserStep))) * 100
                    ElseIf uniqueID < (rows * 3) Then
```

```
                        ProgressBar2.Value = (uniqueID / (3 * rows)) * 100
                    Else
                    End If
                Next
                cn4.Close()
                'Close file
                Dim button As DialogResult
                button = MsgBox("Success! Your file was saved to " + mdb_file_name + ".
Do you want to view the results in the listbox?", MsgBoxStyle.YesNo, "Success!")
                If button = Windows.Forms.DialogResult.Yes Then
                Else

                    Me.Close()
                End If
                txtFilePath.Text = ""
                txtaccessfilepath.Text = ""
            Catch ex As Exception
                MsgBox("There was an error processing your request.",
MsgBoxStyle.Critical, "Error")
            End Try
        End If
        Me.Enabled = True
    End Sub

    Private Function pathToFile(ByVal path As String) As String
        Dim fl As String = Nothing
        Dim flArray() As String
        Dim c As Char = Nothing
        Dim count As Integer = path.Length - 1
        c = path.Chars(count)
        While c <> "\"
            fl = String.Concat(c, fl)
            count = count - 1
            c = path.Chars(count)
        End While
        flArray = fl.Split(".")
        Return flArray(0)
    End Function

    Private Function fileToCounty(ByVal file As String) As String
        Dim county As String
        Dim fileSplit() As String
        Dim count As Integer = 1
        fileSplit = file.Split("_")
        county = fileSplit(0)
        While fileSplit(count) <> "STN"
            county = county + "_" + fileSplit(count)
            count += 1
        End While
        Return county
    End Function

    Private Sub releaseObject(ByVal obj As Object)
        Try
            System.Runtime.InteropServices.Marshal.ReleaseComObject(obj)
            obj = Nothing
        Catch ex As Exception
            obj = Nothing
```

```
        Finally
            GC.Collect()
        End Try
    End Sub

    Private Sub rdoWISLRStep_Click(ByVal sender As System.Object, ByVal e As
System.EventArgs) Handles rdoWISLRStep.Click

        rdoWISLRStep.Checked = True
        txtStep.Enabled = True
        btnBrowseXL.Enabled = True
        btnBrowseACC.Enabled = True
        btnGenerate.Enabled = True
        txtFilePath.Enabled = True
        txtaccessfilepath.Enabled = True
    End Sub

    Public Function trunc(ByVal number As Double, ByVal digits As Integer) As Double
        Return ((Truncate(number * Pow(10, digits))) / Pow(10, digits))
    End Function

    Private Sub btnReset_Click(ByVal sender As System.Object, ByVal e As
System.EventArgs) Handles btnReset.Click
        'Resets the form to original state
        rdoSTN.Enabled = True
        rdoSTN.Checked = False
        rdoWISLR.Enabled = True
        rdoWISLR.Checked = False
        rdoDefault.Enabled = False
        rdoDefault.Checked = False
        rdoWISLRStep.Enabled = False
        rdoWISLRStep.Checked = False
        txtStep.Clear()
        txtStep.Text = 0
        txtStep.Enabled = False
        txtFilePath.Clear()
        ListBox1.Items.Clear()
        cmboCounty.ResetText()
        txtFilePath.Enabled = False
        txtaccessfilepath.Enabled = False
        cmboCounty.Enabled = False
        btnBrowseXL.Enabled = False
        btnBrowseACC.Enabled = False
        btnGenerate.Enabled = False
        ProgressBar1.Value = 0
        ProgressBar2.Value = 0
    End Sub

    Public Function DoesTableExist(ByVal table_name As String) As Boolean
        Dim dbconn As New OleDbConnection("Provider=Microsoft.Jet.OLEDB.4.0;Data Source=
" & mdb_file_name)
        dbconn.Open()
        Dim restrictions(3) As String
        restrictions(2) = table_name
        Dim dbTable As DataTable = dbconn.GetSchema("Tables", restrictions)
        If dbTable.Rows.Count = 0 Then
            DoesTableExist = False
        Else
```

```
            DoesTableExist = True
        End If
        dbTable.Dispose()
        dbconn.Close()
        dbconn.Dispose()
    End Function

    Private Sub rdoDefault_Click(ByVal sender As System.Object, ByVal e As
System.EventArgs) Handles rdoDefault.Click

        txtStep.Enabled = False
        btnBrowseXL.Enabled = True
        btnBrowseACC.Enabled = True
        btnGenerate.Enabled = True
        txtFilePath.Enabled = True
        txtaccessfilepath.Enabled = True
    End Sub
End Class
```

APPENDIX D

The Point Moving Program VB.NET code is presented in this appendix. The program requires input from the user in the form of an Access Database that contains a link_link table and STN Points table, both properly structured. This program should executed only after the STN Points Generator has been executed. The UI for the Point Moving Program is shown in Figure D1. The program is designed to handle QA/QC points or crash data (crash data is known as RP crashes and contains a different Unique ID name than the QA/QC points).

The output of this program is a Unique ID (that matches the Unique ID from the input table), a WISLR link ID, and WISLR link offset.

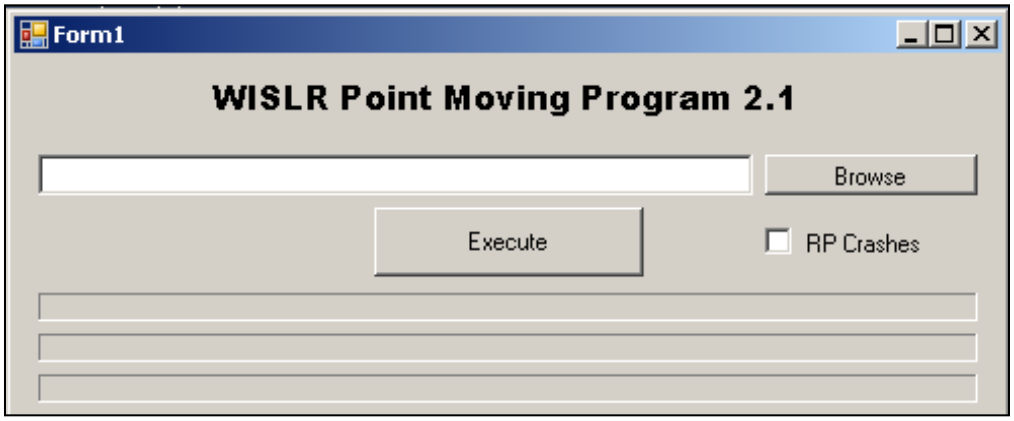

Figure D 1. Point Moving Program UI

It should be noted that the code for this program can be modified to convert WISLR points to STN points by reversing the ratio calculation performed by the program.

Form1 Code:

```
Option Explicit On
Imports System.Data.OleDb
Imports System
Imports System.IO
Imports System.Math
Imports System.Text

Public Class Form1
    Inherits System.Windows.Forms.Form
    Private STN_Table, link_link_file As String
    Dim RPid, Link_ID, Link_Offset As Double
    Dim STNid, STNstart, STNend, WISLRid, WISLRstart, WISLRend As Double
    Dim UnmatchedSTN(,) As Double

    Friend WithEvents Button5 As System.Windows.Forms.Button
    Const epsilon As Double = 0.000001

    Private Sub btnBrowse_Click(ByVal sender As System.Object, ByVal e As
System.EventArgs) Handles btnBrowse.Click
        With OpenFileDialog1
            .Filter = "mdb Files(*.)|*.mdb"
            .CheckFileExists = True
            .CheckPathExists = True
            .RestoreDirectory = True
            If .ShowDialog = DialogResult.OK Then
                txtfilepath.Text = .FileName
                link_link_file = txtfilepath.Text
            Else
                MsgBox("You didn't select your link_link file, or any file for that
matter.")
            End If
        End With
    End Sub

    Private Sub tableCheck(ByVal link_link_file As String)
        'This was to check the tables to see if there are any WISLR links the link_link
table that are not found in the WISLR points table,
        'but I don't think this will work because not every WISLR link is included in the
link_link table and not every WISLR link in the link_link table will have a point on it.
        Dim cn As New OleDbConnection("Provider=Microsoft.Jet.OLEDB.4.0;Data Source=" &
link_link_file)
        Dim cmd1 As New OleDbCommand("DROP TABLE WISLR_out", cn)
        Dim cmd2 As New OleDbCommand("DROP TABLE WISLR_Final", cn)
        Dim cmd3 As New OleDbCommand("DROP TABLE Unmatched_STN_Links", cn)
        cn.Open()
        If DoesTableExist("WISLR_out") = True Then
            Dim buttonYes As DialogResult
            buttonYes = MsgBox("A table entitled 'WISLR_Points' already exists in the
selected database.  Do you want to remove this table?", MsgBoxStyle.YesNo, "Table
Exists")
            If buttonYes = Windows.Forms.DialogResult.Yes Then
                Dim buttonSure As DialogResult
                buttonSure = MsgBox("Are you sure?", MsgBoxStyle.YesNo, "Delete Table")
                If buttonSure = Windows.Forms.DialogResult.Yes Then
                    cmd1.ExecuteNonQuery()
```

```
                Else
                    Exit Sub
                End If
            Else
                Exit Sub
            End If
        End If
        If DoesTableExist("WISLR_final") = True Then
            Dim buttonYes As DialogResult
            buttonYes = MsgBox("A table entitled 'WISLR_Final' already exists in the
selected database.  Do you want to remove this table?", MsgBoxStyle.YesNo, "Table
Exists")
            If buttonYes = Windows.Forms.DialogResult.Yes Then
                Dim buttonSure As DialogResult
                buttonSure = MsgBox("Are you sure?", MsgBoxStyle.YesNo, "Delete Table")
                If buttonSure = Windows.Forms.DialogResult.Yes Then
                    cmd2.ExecuteNonQuery()
                Else
                    Exit Sub
                End If
            Else
                Exit Sub
            End If
        End If
        If DoesTableExist("Unmatched_STN_Links") = True Then
            Dim buttonYes As DialogResult
            buttonYes = MsgBox("A table entitled 'Unmatched_STN_Links' already exists in
the selected database.  Do you want to remove this table?", MsgBoxStyle.YesNo, "Table
Exists")
            If buttonYes = Windows.Forms.DialogResult.Yes Then
                Dim buttonSure As DialogResult
                buttonSure = MsgBox("Are you sure?", MsgBoxStyle.YesNo, "Delete Table")
                If buttonSure = Windows.Forms.DialogResult.Yes Then
                    cmd3.ExecuteNonQuery()
                Else
                    Exit Sub
                End If
            Else
                Exit Sub
            End If
        End If
        cn.Close()
    End Sub

    Private Sub delStuff(ByVal link_link_file As String)
        ' gets the values of any missing STN links in the link_link table
        Dim cn As OleDbConnection
        Dim cmd2 As OleDbCommand
        Dim dr2 As OleDbDataReader
        Dim q As Integer
        cn = New OleDbConnection("Provider=Microsoft.Jet.OLEDB.4.0;Data Source=" &
link_link_file)

        cn.Open()

        If chkRP.Checked = True Then
```

```
            cmd2 = New OleDbCommand("SELECT distinct * FROM (SELECT distinct LINKID FROM
STN_Points left outer join link_link on stn_points.LINKID= link_link.stnid )WHERE stnid
is Null ", cn)
        Else
            cmd2 = New OleDbCommand("SELECT distinct * FROM (SELECT distinct link_id FROM
STN_Points left outer join link_link on stn_points.link_ID= link_link.stnid )WHERE stnid
is Null ", cn)
        End If
        dr2 = cmd2.ExecuteReader

        While dr2.Read()
            ReDim UnmatchedSTN(q, 1)
            UnmatchedSTN(q, 1) = dr2(0)
            q = q + 1
        End While

        dr2.Close()
        cn.Close()
        Application.DoEvents()

    End Sub

    Private Sub unmatchedSTN_table(ByVal link_link_file As String)
        'creates a table in the access database that holds the values of the missing STN
links
        If UnmatchedSTN Is Nothing Then
            Exit Sub
        Else
            Dim cn As OleDbConnection
            cn = New OleDbConnection("Provider=Microsoft.Jet.OLEDB.4.0;Data Source=" &
link_link_file)
            Dim table_name As String
            table_name = "Unmatched_STN_Links"
            Dim cmd As New OleDb.OleDbCommand("CREATE TABLE " & table_name & "(Link_ID
integer)", cn)

            cn.Open()
            Try
                cmd.ExecuteNonQuery()
            Catch ex As OleDb.OleDbException
                MessageBox.Show(ex.Message, "OleDbException")
                Exit Sub
            Catch ex As Exception
                MessageBox.Show(ex.Message, "GeneralException")
                Exit Sub
            End Try
            cn.Close()

            Dim cn1 As OleDbConnection
            cn1 = New OleDbConnection("Provider=Microsoft.Jet.OLEDB.4.0;Data Source=" &
link_link_file)
            cn1.Open()
            Dim n, numpoints As Integer

            numpoints = UnmatchedSTN.GetLength(0) - 1

            For n = 0 To numpoints
```

```
                'Debug.WriteLine(table_name)
                Dim sql As String = "INSERT INTO " & table_name & "(Link_ID) VALUES (" &
UnmatchedSTN(n, 1) & ")"
                Dim cmd1 As New OleDbCommand(sql, cn1)
                Dim records As Integer = cmd1.ExecuteNonQuery()
                ProgressBar2.Value = (n / numpoints) * 100
            Next
            cn1.Close()
            Application.DoEvents()
        End If
    End Sub
    Private Sub Read_Calculate(ByVal link_link_file As String)

        Dim rate, Unique_ID, WISLR_Link, WISLR_Offset As Double
        Dim i As Integer = 0
        Dim currentrecord As String
        Dim STN_String As String

        Dim STNPointsDataTable As New DataTable
        Dim STNPointsDataRow As DataRow
        Dim LinkDataTable As New DataTable
        Dim LinkDataRow As DataRow
        Dim finaldatatable As New DataTable
        Dim finaldatarow As DataRow
        Dim AccdColumn As DataColumn = New DataColumn("ACCDNMBR")
        AccdColumn.DataType = System.Type.GetType("System.String")
        finaldatatable.Columns.Add(AccdColumn)
        Dim LinkIDColumn As DataColumn = New DataColumn("Link_ID")
        LinkIDColumn.DataType = System.Type.GetType("System.String")
        finaldatatable.Columns.Add(LinkIDColumn)
        Dim OffsetColumn As DataColumn = New DataColumn("Link_Offset")
        OffsetColumn.DataType = System.Type.GetType("System.String")
        finaldatatable.Columns.Add(OffsetColumn)

        'Connects to, selects the records for, and fills the datatables to be used for
this part of the program
        Dim cn As OleDbConnection
        cn = New OleDbConnection("Provider=microsoft.jet.OLEDB.4.0;Data source= " &
link_link_file)

        STN_String = "SELECT * FROM STN_Points"
        currentrecord = "SELECT STNid,STNstart,STNend,WISLRid,WISLRstart,WISLRend FROM
Link_link"

        Dim STNadapter As New OleDb.OleDbDataAdapter(STN_String, cn)
        STNadapter.Fill(STNPointsDataTable)

        Dim rows As Integer = STNPointsDataTable.Rows.Count
        Dim linkadapter As New OleDb.OleDbDataAdapter(currentrecord, cn)
        linkadapter.Fill(LinkDataTable)
        Dim linkrows As Integer = LinkDataTable.Rows.Count
        Dim criteria As String

        Dim cn2 As OleDbConnection
        cn2 = New OleDbConnection("Provider=Microsoft.Jet.OLEDB.4.0;Data Source=" &
link_link_file)
        Dim table_name As String
        table_name = "WISLR_final"
```

```
        Dim cmd As New OleDb.OleDbCommand("CREATE TABLE " & table_name & "(Unique_ID
double, WISLR_Link double, WISLR_Offset double)", cn2)

        cn2.Open()
        Try
            cmd.ExecuteNonQuery()
        Catch ex As OleDb.OleDbException
            MessageBox.Show(ex.Message, "OleDbException")
            Exit Sub
        Catch ex As Exception
            MessageBox.Show(ex.Message, "GeneralException")
            Exit Sub
        End Try
        cn2.Close()



        For Each STNPointsDataRow In STNPointsDataTable.Rows 'Cycles through the entire
set of RP crash records one at a time, selects matching records from
            'the link_link table and runs calculations on them
            RPid = STNPointsDataRow(0)
            Link_ID = STNPointsDataRow(1)
            Link_Offset = STNPointsDataRow(2)

            criteria = "STNid = " & Link_ID 'to select rows from the link_link table with
matching link_IDs

            Dim selrows As Integer = LinkDataTable.Select(criteria).Length
            If selrows = 0 Then 'Writes any records to a separate CSV file that have a
link_ID not in the link_link table

            Else
                'Cycles through the selected set of records from the link_link table;
find which meets the criteria in the IF statement
                For Each LinkDataRow In LinkDataTable.Select(criteria)
                    STNid = LinkDataRow(0)
                    STNstart = LinkDataRow(1)
                    STNend = LinkDataRow(2)
                    WISLRid = LinkDataRow(3)
                    WISLRstart = LinkDataRow(4)
                    WISLRend = LinkDataRow(5)

                    'Performs math on any offset that falls within the correct range
                    If Link_Offset = 0 And STNstart = 0 Then
                        rate = (WISLRend - WISLRstart) / (STNend - STNstart)
                        WISLR_Offset = (rate * (Link_Offset - STNstart)) + WISLRstart
                        WISLR_Link = WISLRid
                        Unique_ID = RPid
                        Dim NewRow As DataRow = finaldatatable.NewRow() 'creates a new
row in the final datatable with the new values
                        NewRow("ACCDNMBR") = Unique_ID
                        NewRow("Link_ID") = WISLR_Link
                        NewRow("Link_Offset") = trunc(WISLR_Offset, 3)
                        finaldatatable.Rows.Add(NewRow)
                    ElseIf Link_Offset > STNstart And Link_Offset <= STNend Then
                        rate = (WISLRend - WISLRstart) / (STNend - STNstart)
                        WISLR_Offset = (rate * (Link_Offset - STNstart)) + WISLRstart
                        WISLR_Link = WISLRid
```

```
                        Unique_ID = RPid
                        Dim NewRow As DataRow = finaldatatable.NewRow() 'creates a new
row in the final datatable with the new values
                        NewRow("ACCDNMBR") = Unique_ID
                        NewRow("Link_ID") = WISLR_Link
                        NewRow("Link_Offset") = trunc(WISLR_Offset, 3)
                        finaldatatable.Rows.Add(NewRow)
                    End If
                Next
            End If
            i = i + 1
            ProgressBar1.Value = (i / rows) * 100
        Next

        i = 0
        Dim cn3 As OleDbConnection
        cn3 = New OleDbConnection("Provider=Microsoft.Jet.OLEDB.4.0;Data Source=" &
link_link_file)

        cn3.Open()
        'Writes the values in the corrected finaldatatable to a CSV file
        Dim final_ID, Final_Link, Final_Offset As String
        Dim finalcount As Integer = finaldatatable.Rows.Count
        For Each finaldatarow In finaldatatable.Rows
            final_ID = finaldatarow(0)
            Final_Link = finaldatarow(1)
            Final_Offset = finaldatarow(2)

            Dim sql As String = "INSERT INTO " & table_name & "(Unique_ID, WISLR_Link,
WISLR_Offset) VALUES " & "('" & final_ID & "' , '" & Final_Link & "', '" & Final_Offset &
"')"
            Dim cmd1 As New OleDbCommand(sql, cn3)
            Dim records As Integer = cmd1.ExecuteNonQuery()
            i = i + 1
            ProgressBar3.Value = (i / finalcount) * 100
        Next
        cn3.Close()

    End Sub
    Public Function DoesTableExist(ByVal table_name As String) As Boolean
        Dim dbconn As New OleDbConnection("Provider=Microsoft.Jet.OLEDB.4.0;Data Source=
" & link_link_file)
        dbconn.Open()
        Dim restrictions(3) As String
        restrictions(2) = table_name
        Dim dbTable As DataTable = dbconn.GetSchema("Tables", restrictions)
        If dbTable.Rows.Count = 0 Then
            DoesTableExist = False
        Else
            DoesTableExist = True
        End If
        dbTable.Dispose()
        dbconn.Close()
        dbconn.Dispose()
    End Function
    Private Sub btnExecute_Click(ByVal sender As System.Object, ByVal e As
System.EventArgs) Handles btnExecute.Click
```

```
        Me.Enabled = False
        tableCheck(link_link_file)
        delStuff(link_link_file)
        unmatchedSTN_table(link_link_file)
        Read_Calculate(link_link_file)

        MsgBox("Done. A table entitled WISLR_Final has been created to the Access
database.  Check the table 'Unmatched_STN_Links' to see which points/crashes did not
match a link in the link_link table.")

        Me.Close()

    End Sub
    Public Function trunc(ByVal number As Double, ByVal digits As Integer) As Double
        Return ((Truncate(number * Pow(10, digits))) / Pow(10, digits))
    End Function
End Class
```

APPENDIX E

The Database Merge program was designed aid in merging a county link_link table into the statewide link_link table without allowing duplicate STN link IDs to enter the statewide table. The input for the program is the Access Database containing the county link_link table and the Access Database containing the statewide link_link table. The UI for the Database Merge program is shown in Figure E1. There is no output from this program; after executing the program, the statewide link_link table will contain the records from the county link_link table, given that the STN link was not already in the statewide link_link table.

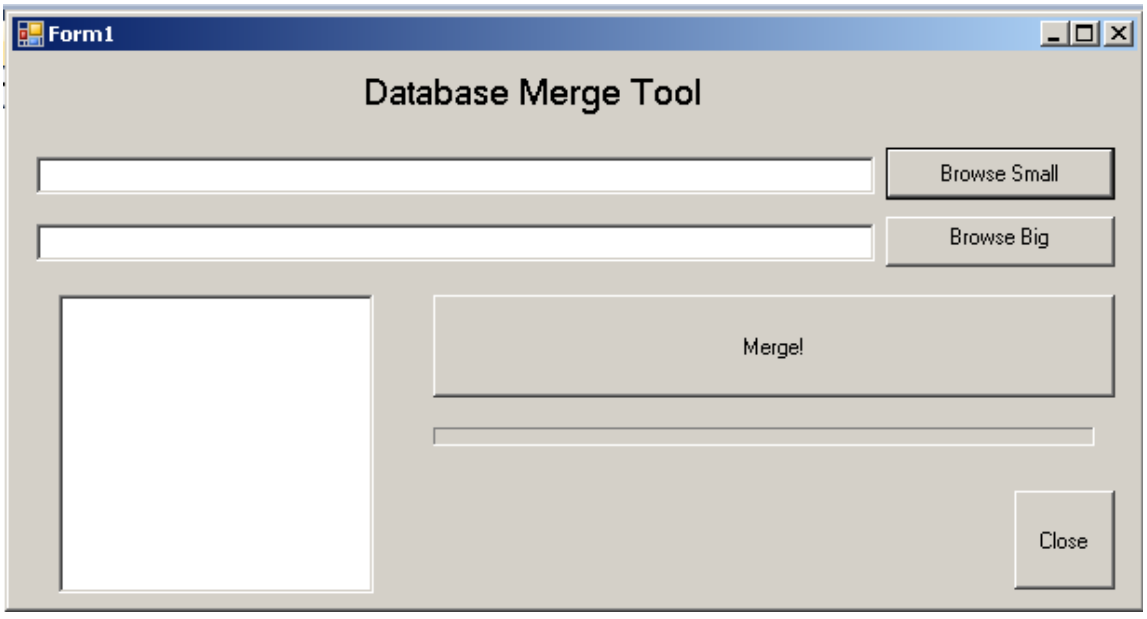

Figure E.1. Database Merge Tool UI

Form1 Code:

```
Imports System.Data.OleDb
Imports System.Text
Imports System
Imports System.IO
Public Class Form1

    Dim link_link_small As String
    Dim link_link_big As String

    Private Sub btnBrowse1_Click(ByVal sender As System.Object, ByVal e As
System.EventArgs) Handles btnBrowse1.Click
        With OpenFileDialog1
            .InitialDirectory = "C:\Users\Zach\Desktop\Link_link Folders"
            .Filter = "mdb Files(*.)|*.mdb"
            .CheckFileExists = True
            .CheckPathExists = True
            .RestoreDirectory = True
            If .ShowDialog = DialogResult.OK Then
                txtfilepath1.Text = .FileName
                link_link_small = txtfilepath1.Text
            Else
                MsgBox("You didn't select your sinngle county link_link file, or any file
for that matter.")
            End If
        End With
    End Sub

    Private Sub btnBrowse2_Click(ByVal sender As System.Object, ByVal e As
System.EventArgs) Handles btnBrowse2.Click
        With OpenFileDialog2
            .InitialDirectory = Environment.SpecialFolder.Desktop
            .Filter = "mdb Files(*.)|*.mdb"
            .CheckFileExists = True
            .CheckPathExists = True
            .RestoreDirectory = True
            If .ShowDialog = DialogResult.OK Then
                txtfilepath2.Text = .FileName
                link_link_big = txtfilepath2.Text
            Else
                MsgBox("You didn't select the multi-county link_link file, or any file
for that matter.")
            End If
        End With
    End Sub
    Private Sub btnMerge_Click(ByVal sender As System.Object, ByVal e As
System.EventArgs) Handles btnMerge.Click

        btnBrowse1.Enabled = False
        btnBrowse2.Enabled = False
        txtfilepath1.Enabled = False
        txtfilepath2.Enabled = False
        btnClose.Enabled = False
        btnMerge.Enabled = False
```

```
        Dim x As Integer = 0
        Dim cnbig As New OleDbConnection("Provider=Microsoft.Jet.OLEDB.4.0;Data Source="
& link_link_big)
        Dim cnsmall As New OleDbConnection("Provider=Microsoft.Jet.OLEDB.4.0;Data
Source=" & link_link_small)
        Dim Selbig As String = "SELECT STNid FROM Link_Link GROUP BY STNid"
        Dim dtbig As New DataTable
        Dim drbig As DataRow
        Dim dabig As New OleDbDataAdapter(Selbig, cnbig)
        dabig.Fill(dtbig)
        Dim records As Integer = dtbig.Rows.Count

        Dim tempTable As String = "Temp"
        Dim Createcmd As New OleDbCommand("CREATE TABLE " & tempTable & "(STNid double)",
cnsmall)

        cnsmall.Open()
        Try
            Createcmd.ExecuteNonQuery()
        Catch ex As OleDb.OleDbException
            MessageBox.Show(ex.Message, "OleDbException")
            Exit Sub
        Catch ex As Exception
            MessageBox.Show(ex.Message, "GeneralException")
            Exit Sub
        End Try
        cnsmall.Close()

        cnsmall.Open()
        For Each drbig In dtbig.Rows
            Dim sql As String = "INSERT INTO " & tempTable & " (STNid) VALUES ('" &
drbig(0) & "')"
            Dim cmd As New OleDbCommand(sql, cnsmall)
            cmd.ExecuteNonQuery()
            x = x + 1
            ProgressBar1.Value = (x / records) * 100
        Next
        cnsmall.Close()

        x = 0

        Dim dtsmall As New DataTable
        Dim drsmall As DataRow
        Dim Selsmall As String = "SELECT * FROM Link_Link LEFT JOIN " & tempTable & " on
link_link.STNid=" & tempTable & ".STNid WHERE " & tempTable & ".STNid is null"
        Dim cnsmall1 As New OleDbConnection("Provider=Microsoft.Jet.OLEDB.4.0;Data
Source=" & link_link_small)
        Dim cmdsmall1 As New OleDbCommand
        Dim dasmall As New OleDbDataAdapter(Selsmall, cnsmall1)
        dasmall.Fill(dtsmall)
        Dim recordlist As Integer = dtsmall.Rows.Count


        Dim dtList As New DataTable
        Dim drList As DataRow
        Dim SelList As String = "SELECT * FROM Link_Link LEFT JOIN " & tempTable & " on
link_link.STNid=" & tempTable & ".STNid WHERE " & tempTable & ".STNid is not null"
```

```
        Dim cnlist As New OleDbConnection("Provider=Microsoft.Jet.OLEDB.4.0;Data Source="
& link_link_small)
        Dim cmdlist As New OleDbCommand
        Dim dalist As New OleDbDataAdapter(SelList, cnlist)
        dalist.Fill(dtList)
        Dim list As Integer = dtList.Rows.Count

        For Each drList In dtList.Rows
            lstDuplicates.Items.Add(drList(0))
        Next
        Label1.Visible = True
        Label3.Visible = True
        Label3.Text = list.ToString
        Me.Refresh()

        Dim cndrop As New OleDbConnection("Provider=Microsoft.Jet.OLEDB.4.0;Data Source="
& link_link_small)
        Dim cmddrop As New OleDbCommand("DROP TABLE " & tempTable, cndrop)
        cndrop.Open()
        Try
            cmddrop.ExecuteNonQuery()
        Catch ex As OleDb.OleDbException
            MessageBox.Show(ex.Message, "OleDbException")
            Exit Sub
        Catch ex As Exception
            MessageBox.Show(ex.Message, "GeneralException")
            Exit Sub
        End Try
        cndrop.Close()

        Dim insertCN As New OleDbConnection("Provider=Microsoft.Jet.OLEDB.4.0;Data
Source=" & link_link_big)
        insertCN.Open()

        For Each drsmall In dtsmall.Rows
            Dim InsertString As String = "INSERT INTO Link_link (STNid, STNstart, STNend,
WISLRid, WISLRstart, WISLRend, " & _
                                         "T, M, G, W, P, Coder, DateMod, Comments,
County) VALUES " & "('" & drsmall(0) & "','" & drsmall(1) & "','" & _
                                         drsmall(2) & "','" & drsmall(3) & "','" &
drsmall(4) & "','" & drsmall(5) & "','" & drsmall(6) & "','" & drsmall(7) & "','" & _
                                         drsmall(8) & "','" & drsmall(9) & "','" &
drsmall(10) & "','" & drsmall(11) & "','" & drsmall(12) & "','" & drsmall(13) & "','" &
drsmall(14) & "')"
            Dim insertcmd As New OleDbCommand(InsertString, insertCN)
            Dim recordadd As Integer = insertcmd.ExecuteNonQuery()
            x = x + 1
            ProgressBar1.Value = (x / recordlist) * 100
        Next
        insertCN.Close()

        btnClose.Enabled = True
        Me.Refresh()

    End Sub

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

```
        Me.Close()
    End Sub

    Private Sub Form1_Load(ByVal sender As System.Object, ByVal e As System.EventArgs)
Handles MyBase.Load
        Label1.Visible = False
        Label3.Visible = False
    End Sub
End Class
```