ENHANCING THE SPATIAL RESOLUTION OF AN ACTIVE LINEAR REFERENCING SYSTEM AND CALIBRATING A FUNCTIONAL MERGE

by

MARK EDWARD SIMPSON

ANDREW J. GRAETTINGER, COMMITTEE CHAIR
ALEXANDER M. HAINEN
RANDY K. SMITH

A THESIS

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

TUSCALOOSA, ALABAMA

2016

ABSTRACT

The Wisconsin Department of Transportation (WisDOT) maintains two linear referencing systems (LRS), the State Trunk Network (STN) and the Wisconsin Information System for Local Roads (WISLR). These two systems, while they were developed at different times and independently from one another, provide WisDOT the capabilities to report Highway Safety Manual data for the entire state and local road network. Because the two systems have limited data sharing capabilities, a functional merge, called the Link-Link table, was previously developed to facilitate a relationship between STN and WISLR. While data translation through the Link-Link table is acceptable, this project seeks to enhance data transfer between these two LRSs.

Certain inherent differences exist between STN and WISLR that make accurate data movement challenging even through the link-link table, notably, the poor state route representation in WISLR and the difference in business rules that govern how roadways are represented in each system. This research developed a method to enhance the 12,000 miles of state route line work in WISLR, without disrupting the daily business activities of WisDOT, to match the STN line work.

Utilizing the improved WISLR state trunk line work, a method to calibrate the Link-Link table to reconcile the linear offset discrepancy that exists at gore points was developed. A case study evaluated Interstate 94 in Saint Croix County by locating 46 gore locations and calibrating 113 records in the Link-Link table. The calibration technique produced 209 new Link-Link

records that were used to translate 534 crashes from STN to WISLR. Results show that greater positional accuracy can be achieved when translating data through the calibrated table.

DEDICATION

Any challenging work that is worth doing requires great sacrifice. None so much as the sacrifice given by my loving wife, Gina. I am forever indebted to her for her encouragement and support throughout this humble effort. This thesis is dedicated to her.

ACKNOWLEDGMENTS

I would like to express my very great appreciation to Dr. Andrew Graettinger for giving me this opportunity. I am very grateful to him for his patient guidance, encouragement, and valuable input during this research. He has been a great mentor and friend.

I would like to thank Rebecca Russell for her leadership and hard work performing the Link-Link table update and the high quality work she produced. I am grateful to Shane Crawford and Brittany Shake for their constant assistance and willingness to share their technical knowledge when I was struggling to understand.

I would like to thank Daniel Conner, Jessica Crane, Brennan Glynn, Lora Jones, Jake Levins, Nick McDaniel, Sam Parsons, Clint Smith, Caitlyn Spann, Sidney Stratton, and Jaime Williams for their tireless work editing the WISLR system. To the staff at the Wisconsin Department of Transportation, Kelly Schieldt and Kathleen Spencer, I extend my thanks for their constant support during the WISLR editing project. I am grateful to all these people for helping me complete this work.

CONTENTS

LIST OF TABLES

LIST OF FIGURES

CHAPTER 1

INTRODUCTION

1.1 Introduction

Linear Referencing Systems (LRS) have been widely used in the transportation industry for many years. Linear Referencing is simply defined as a method for storing and managing geospatial information along a linear feature. Since transportation features can most often be characterized as linear, this simple definition allows transportation agencies who are tasked with managing roadway assets a method to easily manage roadway assets. As the nation's infrastructure has grown , the need to store more comprehensive, consistent, and accurate data related to LRS's has become increasingly paramount .

The Wisconsin Department of Transportation (WisDOT) currently manages two LRSs, the State Trunk Network (STN) and Wisconsin Information System for Local Roads (WISLR). STN was developed in the early 1990's to represent all state roads (interstates, US highways, state highways) within the state. This system was originally developed as a tabular data set with very strict business rules for measuring and collecting data, storing data, and the routine management of data updates. In 2002, WisDOT launched WISLR. As the name implies, WISLR was initially conceived to accurately store information related to local roads. Because the STN system already maintained state routes with a high level of resolution, the original development of WISLR did not include accurate state route line work data, although a general representation of the state routes was included in WISLR.

These two systems serve WisDOT well but because the systems were built in different environments, no logical way to link the basic features in each system existed; therefore, performing a task such as state wide crash analysis was not possible without a way to link the two systems. Previous research developed a functional merge that relates linear features, known as links, in STN to the corresponding link in WISLR (Ryals, 2011). This functional merge, known as the Link-Link table, allows the movement of event data between the two LRSs. The Link-Link table has been shown to be very effective at moving crash data, and has been instrumental in the development of a state wide crash map (Graettinger, Qin, Parker, & Forde, 2009).

The structure of the Link-Link table enables users to examine areas within WISLR that have a lower state route representation. The state routes having a lower resolution, or poor geometry in WISLR, either missing, incomplete or misrepresented, are recognized in the Link-Link table with established problem flags. This ensures that anytime data exist in those areas, manual procedures can be implemented to move the data in an acceptable manner. However, moving the data in these areas of ambiguity causes concessions to be made with respect to where exactly data is moved to in WISLR. For example, WISLR line work does not include rest areas. If data, such as a crash, occur in a rest area, it is recorded in the STN data (all crash events that occur along state routes are stored in STN). When that crash is translated to a WISLR link, the crash will move to the nearest mainline highway, not exactly representative of where the crash occurred. This same type of ambiguity occurs on several other roadway features.

Fundamentally, the Link-Link table must be manually updated to account for state highway changes that are reflected in either LRS (Biswas, 2014,Morrison, 2012).To date, three

Link-Link table updates have been completed since the original development. The third update, completed in 2015, relates STN and WISLR data sets through the 2013 calendar year.

Through the original development of the Link-Link table and subsequent updates, a consistent conclusion has been made that the only way to fully resolve problem flag areas is to edit the LRS with the lower resolution to match the higher resolution LRS (Morrison, 2012, Ryals, 2011,Graettinger, et. al, 2013). Since WISLR is the lower resolution LRS, the state routes within WISLR must be edited to match STN. This is not a trivial task since almost 12,000 miles of state roadways exist in Wisconsin. WisDOT headquarters, along with several other regional and local offices throughout the state, and other agencies in WI, use WISLR on a continual basis. This means that editing WISLR must be done without disruption to WisDOT's daily business activities, or any other end user's business practice.

The opportunity to embark on a project of this scale became a reasonable option when the federal government released the All Roads Network of Linear Referenced Data (ARNOLD) initiative in 2012. The ARNOLD initiative requires that states provide extensive coverage of the geospatial network of highways within their state (Winter & Cheatham, 2012). While WisDOT currently has all highways represented, the representation is divided into two separate systems. WisDOT seeks to take an initial step towards a single LRS by updating WISLR to include accurate state route data. One of the focuses of this research presents the method for updating the WISLR system to match STN's representation of the state routes without disrupting WisDOT's daily use of WISLR. Concurrent with the WISLR line work editing, the third Link-Link table update has been completed. During this update, the need for a formalized update procedure manual was recognized. Submitted as a supplement to the annual Link-Link table update, a procedure manual was developed and is included as Appendix A to this work.

The final topic of this research presents a method to reduce the linear offset discrepancy that exists in the Link-Link table. A calibration technique draws upon the well-established Link-Link table methodology and the anticipated updated WISLR data set and calibrates link relationships at known locations of inaccuracies due to differences in business rules known as gore points.

1.2 Thesis Organization

This thesis consists of five Chapters. Chapter 2, Background and Literature Review, discusses the federal guidelines related to state's highway reporting requirements. The Background will briefly discuss the conceptual model of the linear referencing system, the STN and WISLR data structures, and the previous work related to the Link-Link table. Chapter 2 discusses the difference between STN and WISLR and details this difference at a particular area known as gores points. Chapter 3, Methodology, will be broken down into two components: WISLR Editing, and Gore Point Calibration. Chapter 4, Results, will present the current progress of WISLR editing, and the gore point calibration results from a pilot study conducted in Saint Croix County. Chapter 5, Conclusion and Future Work, presents a discussion about the results of this research and also discusses Future Work related to the possibility of WisDOT moving into one unified LRS. Appendices A and B provide detailed information created during this work.

CHAPTER 2

LITERATURE REVIEW AND BACKGROUND

2.1 Introduction

On July 6, 2012, the federal government passed the Moving Ahead for Progress in the 21st Century Act (MAP-21) (Highway Safety Improvements Program, 2012). The law outlines funding for transportation programs and revamps the policy and framework that guide the growth and development of the county's transportation infrastructure. Along with funding appropriation and program updates, one of the key pieces of the law requires each state to submit a complete base map for the purpose of geolocating all public roads. This is significant because until this point, the Federal government has only required states to submit roadway data for Federal-aid roads.

Spatial data collection is not new for the Federal government. As far back as 1953, the US recognized the need to organize and maintain locational information about the various assets within the country (OMB, 1990). The OMB A-16 appoints various federal agencies with tasks associated with the collection and coordination of the various types of geographic and spatial data. The circular has undergone five revisions since the initial release, each time updated to reflect changes in technology and to further describe the components of the spatial data infrastructure and to improve the use and coordination of the data (OMB, 2010).

The United States Office of Management and Budget's Circular A-16 also assigned theme responsibilities to certain government agencies regarding data collection and reporting. The lead agency of the Transportation Data Theme is assigned to the Federal Highway

Administration (FHWA). The FHWS is responsible for managing all aspects of the nation's transportation, not only the highway system. The Transportation data theme makes the FHWA responsible for ensuring that highway transportation data is collected and available in order to support Congress in establishing authorization and appropriation legislation. One of the ways FHWA collects these data is through a program known as the Highway Performance Monitoring System (HPMS) (FHWA, 2014).

Highway Performance Monitoring System is a national program that collects all the nation's public road mileage, including a defined list of attribute data that include, but are not limited to, performance and condition data. The mileage is certified each year by the States' Governors. Each state is required to submit its transportation data in accordance with the reporting requirements specified in the HPMS Field Manual. While the states are not mandated to submit HPMS data, funding for Federal-aid roads depends on the state submission of the data.

Part of the mission statement outlined in the OMB Circular A-16 is to collect and maintain data in such a way that the efforts are not duplicated. However, a report by the US Government Accountability Office (GAO) explains that while the policies and procedures for managing the geospatial data exist, implementing the procedures have not been a priority (GAO, 2012). In fact, the GAO notes that three agencies are acquiring road data independently. This points to an inefficiency related to duplicity, which is not in keeping with a key point of the OMB A-16 purpose. The MAP-21 requirement that all states must provide a base map of all public roads within its state attempts to address this inefficiency.

The requirement outlined in MAP-21 that each state submit a base map of all public roads within its state is known internally at FHWA as the All Roads Network of Linear Referenced Data (ARNOLD). While several states have some type of linear referenced data

describing their road way network, very few have a complete data set with all roads: 40% have state and federal only, 53% have state, federal, & local roadways, 7% have all levels, including private, and about one third of the states are currently using more than one LRS (AASHTO, 2009).

The FHWA recognizes that many states use proprietary LRS data management practices and applications, so the ARNOLD initiative does not specify what type of LRS the state must use, only that a single unified map must be submitted with accurate centerline mileage along with the already established attribute data.

Historically, WisDOT has combined local and state roadway data when reporting HPMS data. The cartographic base map and the required attribute data is a merged version of STN and WISLR. The ARNOLD initiative has given WisDOT the incentive and resources to pursue enhancing the WISLR LRS to fill a known gap in the data and stream line the HPMS data reporting procedure.

2.2 Linear Referencing Systems

A large amount of business data used by transportation departments are linearly referenced. Data range from functional classification of a roadway, route name, and direction to pavement condition, bridge location, and crashes. The generic linear referencing model, known as the NCHRP 20-27 (2), provides the basic framework to relate multiple levels of information along a roadway network. Figure 2.1 shows the conceptual model broken down into three main components: a datum, a network(s), and linear referencing method. The NCHRP model introduced the linear datum. The linear datum consists of anchor points that represent real world, identifiable points, and are connected by anchor sections. The anchor sections represent actual road way segments and the measured or calculated distance between anchor points.

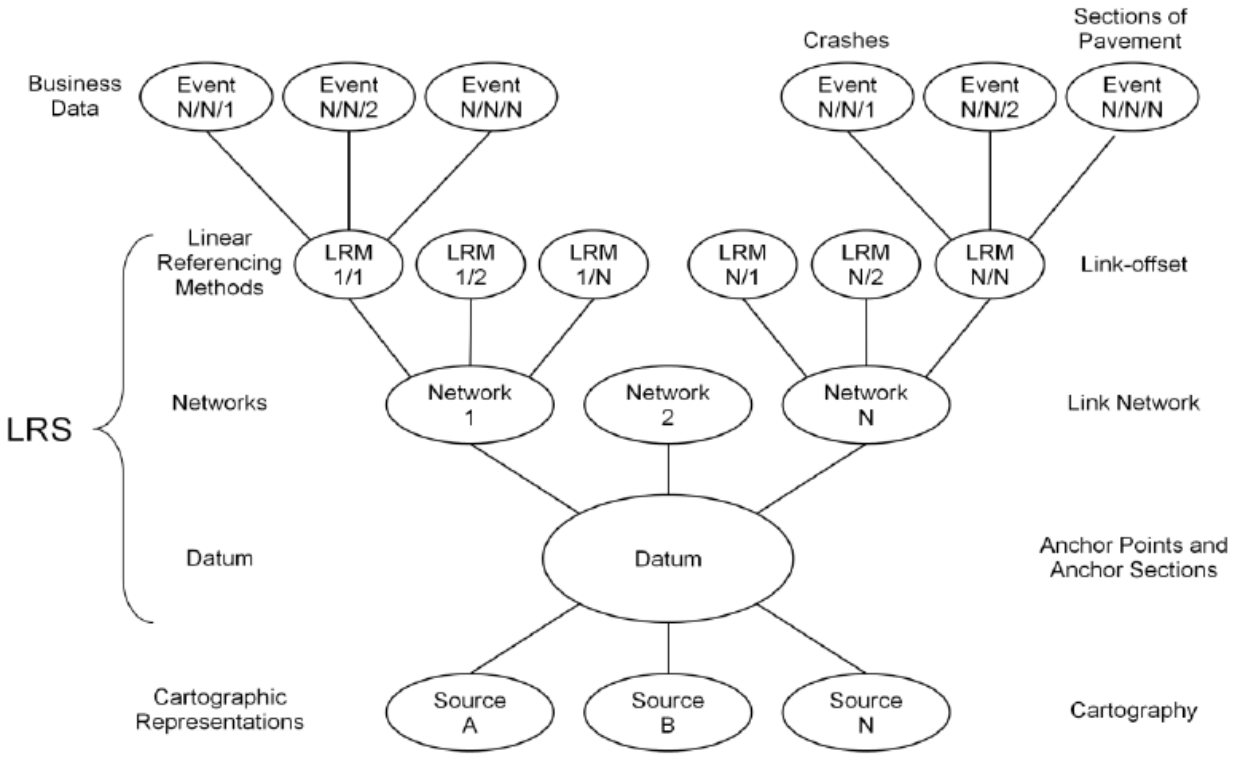


Figure 2.1. Generalized LRS model known as NCHRP20-27 (2) model (Vonderhohe, Chou, Sun, & Adams, 1997)

The network is described as the aggregation of topological objects and provides a means for movement among point locations. Different types of networks can be present in a common datum associated with an LRS, as shown in Figure 2.1 by Network N. One common network is known as a link-node system, used by both STN and WISLR. Nodes are locations where links meet and typically describe the locations where flow can change direction. Links can be described as flow conduits connecting nodes (Wisconsin Department of Transportation, 1998).

A linear referencing method (LRM) describes the method by which events are located along a network. While several linear referencing systems exist, all work on the premise that the location of a point can be determined by stating the distance and direction from a known point (Vonderhohe et al., 1997). STN and WISLR utilize a method known as link-offset, which locates event data along the direction of a link by specifying a measure along the link from the

beginning node (Vonderhohe & Hepworth, 1998). WISLR uses an additional LRM known as On/At, Distance, Direction. This LRM describes events on a route, some distance down the route, in a specified direction. An event can be located along the direction of travel and a certain distance along a link from a known starting point.

Business data are characterized by events that occur along a network. Events can be points describing physical structures, temporal events (e.g. crashes), or linear events such a pavement sections or speed limit zones.

2.3 Linear Referencing Systems used by WisDOT

WisDOT currently maintains two linear referencing systems, the State Trunk Network (STN) and the Wisconsin Information System for Local Roads (WISLR). STN was developed in the early 1990's and launched in 1993 to satisfy the business needs of WisDOT related to maintaining the nearly 12,000 miles of state designated roadways. Four years later, development of WISLR began in an effort to more efficiently maintain the local roads database and improve the dissemination of local road information to end users. With STN already in place, the primary focus of WISLR was on accurate local road representation. WisDOT chose to include state road polyline features in WISLR, however, because STN was fully functional, with detailed state route information and applications in use, very little attention was given to the state routes. Consequently, state routes in WISLR contain ambiguities, lower resolution, inconsistent and incorrect naming, and poor cartography.

While both systems are composed of links and nodes, each has a linear datum that is devised differently. The STN datum is comprised from the aggregation of the network, built from the real world measured distance and stored in such a way that features in the network are tied to the field positions that will translate the theoretical position of the network features. The

datum in WISLR is similar to STN's datum in that it is an aggregation of the network, but differs because the datum is tied directly to the cartographic location. Both datums result in the network and datum being functionally equivalent (Curtin et al., 2007), and lacking a distinct datum because the datum is embedded within the network (Graettinger et al., 2009).

The network design used by STN and WISLR can be described as segmented, that is, the network is divided at any location where two or more line features intersect. These intersections usually describe locations where traffic can change direction. Segmented routes provide more precision when displaying transportation data than features that span over multiple intersections. STN follows this convention, but the links that make up the network typically only break where two state roads intersect, omitting local road intersections. This leads to multiple roadway features on one link. In contrast, the WISLR network will divide at any location where two roads intersect, regardless of road type. As a result, the WISLR LRS contains a larger number of features to represent the same network or route compared to STN.

The STN system is made of links and nodes (or sites) to represent state roads. Each link has a unique ID, a total length (stored in hundredths of a mile), and a from- and to- site. The from- and to- sites are the end points of links that establish direction and represent points along a roadway where two or more state transportation features intersect. Figure 2.2 (a) shows the STN representation of a typical divided highway with exits. Figure 2.2 (b) highlights the abstract nature of the graphical representation of STN. The links in STN are logical constructs between two sites and the visual representation of links is solely done to aid in editing.

Figure 2.2. STN representation (orange lines) of a typical divided highway with exits, (a) close up of off ramps showing turn lanes, (b) shows a larger image of STN link representation.

The WISLR system is similar in structure to STN in that links and sites are used to represent roadways. Figure 2.3 shows the WISLR representation at the same intersection shown in Figure 2.2 (a). WISLR links contain similar attributes to STN links, and occasionally share the same from- and to- site locations, but WISLR link lengths are measured and stored in feet. Figure 2.3 also highlights the lack of resolution of WISLR compared to STN. WISLR does not represent the dual-carriageway nature of the highway and the turn lanes and crossovers at the ramp termini are missing. This lack of resolution on state routes is typical within WISLR

Figure 2.3. WISLR representation (black lines) of typical divided state highway, (a) close up of exit, (b) larger image showing cartographic nature of WISLR.

The primary difference between the two LRSs is the method used to generate the basic centerline data. WISLR data are derived by the heads-up digitation of the roadway from aerial images and the mileage is the shape length as generated by the GIS. STN data are gathered in the field in such a manner that data collection points are fully described and repeatable.

2.4 Link-Link Table.

A functional merge, known as the Link-Link table, relates links in STN to links in WISLR that describe the same section of roadway (Ryals, 2011). The table was initially conceived to facilitate the movement of crash data located in STN to WISLR so that a comprehensive state wide map of crashes could be developed. The approach used by Link-Link most closely follows a bottom-up approach, where the geometry is matched first, followed by objects and attributes (Sester et al., 1998). In contrast, a top-down approach requires object and attribute data to be similar in both systems so that the primary matching is done through some form of data aggregation, followed by geometry matching. Since STN and WISLR have been built in different environments and follow different business rules for data collection, object and attribute data matching is not possible unless geometry is matched first.

The basic structure of the Link-Link table consists of six primary columns, as shown in Table 2.1. The first three columns contain the portion of the STN link, and the last three columns describe the corresponding WISLR link(s).

Table 2.1. Six primary columns used in the Link_Link table with descriptions (Ryals, 2011)

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

A basic example of a typical Link-Link record is outlined in Figure 2.4. In the example, a one-to-many relationship is defined where one STN link is related to three WISLR links. The example also highlights the use of access points, which are defined in the next section. The basic attribute data associated with the STN link and the WISLR links are shown in Figure 2.4 (a) and (b). The attribute tables indicate direction and total offset distance of the link, in addition to the link ID. Figure 2.4 (e) is the populated Link-Link table with the relationships established.

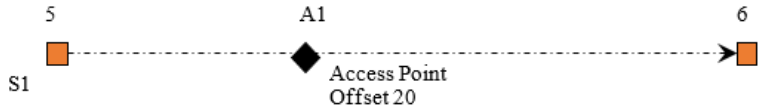
a)

STN Link ID	From Site	To Site	Distance
S1	5	6	66

b)

WISLR Link ID	From Site	To Site	Distance
W1	1	2	100
W2	2	3	50
W3	3	4	200

c)



d)

e)

STNid	STNstart	STNend	WISLRid	WISLRstart	WISLRend
S1	0	20	W1	0	100
S1	20	30	W2	0	50
S1	30	66	W3	0	200

Figure 2.4. Basic example of the Link-Link table population procedure: (a) and (b) show the attribute data for the STN link and WISLR links, respectively, (c) and d) are the graphical representation of STN and WISLR, (e) presents the populated table based on the relationship shown.

Additional columns are included in the Link-Link table to highlight specific differences that exist for any given record. Table 2.2 shows the flag columns with descriptions of each. The differences result mostly from the low resolution of WISLR with respect to state routes. For example, WISLR typically does not represent weigh-stations or rest areas on state routes, so the

record in the Link-Link table would contain a flag, letting an analyst know that special attention should be paid when translating event data in the affected area.

Table 2.2. Link-Link table problem flag columns, with descriptions (Ryals, 2011)

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

The first Link-Link table was created to relate 2009 line work in each LRS. Since the initial creation, three updates have been performed to reflect annual line work changes in both LRSs. The third, and most recent update, includes LRS data through the end of calendar year 2013. Each published table has successfully translated crash data from STN to WISLR. The first update refined crash movement from WISLR to STN (Morrison, 2012), and the second developed a method that used the Link-Link table to assign names to ramps in the WISLR system (Biswas, 2014). While the table has been shown to be acceptable for data translation, the need to enhance the lower resolution LRS, WISLR, remains. The inclusion of problem flags in the Link-Link requires an analyst to manually decide how to move data located on an affected record, often requiring data to be translated to the closest possible feature. This is acceptable in the realm that intervention by a user is necessitated, and the ambiguity is accounted for and noted. However, to use the Link-Link table to effectively move specific attribute data, like route names or linear data, and translate data to points representing actual spatial locations, WISLR must match STN's resolution on state routes. WisDOT seeks to enhance the WISLR system to

match STN as closely as possible, forming the basis of this research which seeks to edit all the state route data in WISLR.

2.5 Linear Offset Discrepancies

STN and WISLR differ in many ways that affect the accuracy of data translation between the systems. One way is related to insufficient, or incomplete, polyline resolution (Hallmark et al., 2003). The deficiencies in resolution include missing segments or improper representation of roadway features. The Link-Link table accounts for differences through the use of problem flags. Another discrepancy is related to how the data is gathered, specifically how the polylines represent roadway features and to what extent business rules define data gathering procedures. Dissimilarities in data gathering and segment representation causes differences in linear measurement along roadways. Currently, the Link-Link table accommodates for this difference in two ways: i) the use of access points ii) a ratio calculation.

As part of the data set, STN records access points that describe measured distances along routes to locations where traffic can access the roadway. These points can be any physically identifiable access point in the field, along a link and between. The access point is stored as a point occurring on a link, with an offset distance and a descriptor, usually the intersecting local road name. The STN data typically does not contain nodes where local roads intersect a state route, but WISLR links do end at each intersection with a local or county road. Therefore, the access point measure along an STN link represents a measured distance of the corresponding WISLR link. Since the Link-Link table relates each WISLR link to the STN link along a given route, access points (when available) are used in place of a ratio when more than one WISLR link exists along any given single STN link. The access points effectively act a control point,

reducing the linear error by providing smaller increments by which to control the relationship (Federal Highway Adminstration, 2001).

A ratio is used when two or more WISLR links are related to a single STN link and no access point is available. The ratio relates the length of a WISLR link as a percentage of the length of the STN link where STNpart/STNfull=WISLRpart/WISLRfull. The ratio calculation is effective at translating data between STN and WISLR when the beginning and ending nodes generally describe the same location along the road. Detailed explanation of the ratio method and an example calculation can be found in Appendix A.

The use of ratios to relate links and the inclusion of access points provide adequate calibration for accurate data translation where the line work in the two systems describe physical locations in a similar fashion, even when spatial alignment is poor. But in areas where the link describes different locations along a road, differences are much greater and cannot be fully managed by ratios or access points. The most problematic area occurs where STN and WISLR describe the junction of on and off ramps with a mainline road (Biswas, 2014). Figure 2.5 (a) shows that STN describes this point as a gore point where the pavement edges meet, and is clearly defined for data collectors to recognize. WISLR does not share this same level of strict business rules with regards to data collection. The gore point in WISLR is represented by the point where the centerline of the ramp meets the centerline of the mainline, Figure 2.5 (b). While this description of the gore point in WISLR can be thought of as business rule, the actual location of this point is determined by an editing technician through heads-up digitizing over an aerial image, which introduces interpretive error.

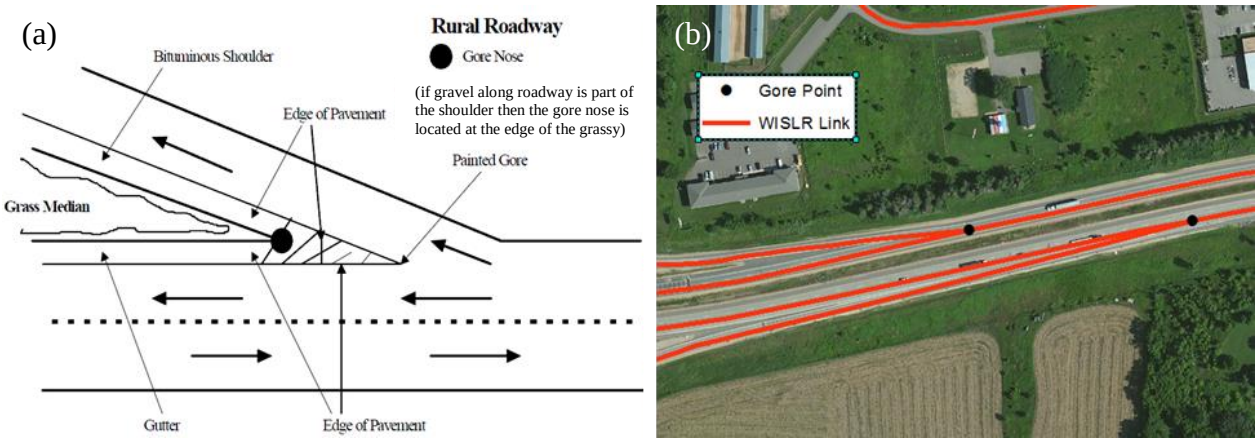


Figure 2.5. Comparison of gore point locations, (a) STN (LCM, 1996), (b) WISLR line work with gore points locations

A digitized point without a physically locatable feature also makes in field measurements difficult or even impossible to replicate. The lack of stable gore point control in the WISLR business rules is further compounded by the poor dual carriageway representation in WISLR.

The gore point conflict is currently handled in the Link-Link table through the use of a flag column. The flag column alerts an analyst that the link relationship includes a gore point and that data translation may need to be handled manually. The ability to develop a method to account for the gore point difference has been severely limited by the poor resolution in WISLR, especially on dual carriageway highways. The ARNOLD project eliminates this challenge through accurate representation of state routes in WISLR. Drawing upon the improved resolution of state routes in WISLR, and the established functional merge known as the Link-Link table, the work presented here will introduce a new method to reduce the linear offset discrepancy that occurs due to differences in gore point locations in both LRSs.

CHAPTER 3

METHODOLOGY

The goal of this research has two parts: i) to improve the resolution of the WISLR LRS to match as closely as possible the roadway network as represented by the STN system; and ii) develop a method to reconcile the linear offset discrepancy that is present at gore point locations through the use of the Link-Link table. The gore point calibration is dependent upon accurate WISLR line work and a populated Link-Link table. The following sections will outline the methodology of the WISLR editing to match the state roadway representation in STN and discuss the technique for calibrating the Link-Link table at gore points.

3.1 WISLR Enhancement

The first objective of this research focused on editing state route line work in WISLR. Because of the existence of STN, the state route links in WISLR are primarily used for visual reference and typically do not contain accurate LRS data. With the goal of providing a single LRS for all roads in Wisconsin, state route links in WISLR need to be edited to match the higher resolution representation in STN. The most common required edits in WISLR include proper dual carriageway representation, one-way ramps, turn lanes, and the general realignment of state routes to match ortho-rectified aerial photos.

3.1.1 *Remote Access*

WISLR is a fully operational system that is used daily to meet the business needs at WisDOT. Therefore, editing activities cannot interfere with ongoing WISLR operations. Editing requires remote access to WISLR through computers located at the WisDOT headquarters using the remote access platform known as BOMGAR™. WisDOT uses BOMGAR™ extensively throughout the agency because of high level security features and full customization, which gives it the capability to set user ids and permissions, thereby controlling access to certain local workstation applications, and monitoring and tracking usage. The use of BOMGAR™ for remote access for WISLR editing was established during a pilot study and is shown to provide a stable platform for remote editing.

Editing takes place on a local workstation in the WisDOT headquarters when an editor logs into a local workstation. WISLR is edited using a custom application known as Location Control Management (LCM), which was developed by WisDOT and built upon the ESRI's ArcMap platform. LCM has been used by WisDOT since WISLR's inception and has been continually maintained and enhanced with a suite of custom tools. The functionality of LCM is built entirely on the premise of providing a platform to edit the attributes of the WISLR LCM.

3.1.2 *Types of Edits*

The goal is to edit the state routes in WISLR to match, as accurately as possible, the state trunk network as modeled in the STN LRS and be cartographically aligned. In general, WISLR is incomplete in almost all aspects of state trunk data, with the exception of the basic, cartographic centerline representation of the roadways, and in most cases, the centerline alignment is inaccurate. Accommodating this level of deficiency can only be accomplished by

an editor visually inspecting every state route. For the purposes of this research, state routes are defined as Interstate Highways (IH), U.S. Highways (USH), and State Highways (STH). Several other transportation features, known as off-main line routes, are maintained at the state level and are considered part of the state trunk network. Off-main line routes include ramps, state road connectors, weigh stations, rest areas, park and rides, frontage roads, and turn lanes.

To provide the most complete state trunk network in WISLR, this work established five primary categories for editing. Each type of enhancement is detailed in the remainder of this section.

Dual-Carriageway

For the purposes of HPMS, FHWA defines dual carriageways as a roadway where opposing traffic lanes are separated by a median width greater than four feet or with a positive barrier (FHWA, 2014). Throughout the development of the STN LRS, dual carriageways have been recorded and stored following the HPMS convention. During the initial development of WISLR, state routes were usually represented by a single cartographic feature, containing two lines and carrying two directions of travel, regardless of the type of roadway. Many of the dual-carriageways, commonly known as divided highways, are not represented by individual one-way polyline features in WISLR. Accurately modeling dual carriageways with separate travel directions allows representation of more complex transportation features further enhancing data translation from STN to WISLR.

Directionality

Travel direction along a segment of roadway is determined by the connection of sites by links. As previously mentioned, the from- and to-sites of a link establish direction. The

importance of defining travel direction accurately is needed for routing and road maintenance applications, and when translating data between STN and WISLR. The reversal in direction of a WISLR link causes events to move or map from STN to the opposite position of the corresponding WISLR link. This type of ambiguity only exists in WISLR where a single one-way link is shown. One-way links are single polyline features that are used to represent roadways with one direction of travel, and contain a single link ID. A two-way link is also a single polyline feature but contains two link IDs describing two travel directions. Two-way links typically represent undivided roadways. In WISLR, many cases exist where this type of ambiguity is present. The most common occurrence is located on ramps and median crossovers. In the event that WISLR does represent dual-carriageways, many times both travel directions are digitized in the same direction, causing one travel lane to be represented in the wrong directions. Direction reversal has also been observed on some off main line features such as ramps. Understanding and interpreting travel direction along any segment of highway is typically accomplished by studying available aerial images and through the use of Reference Point diagrams, which will be discussed later.

Basic centerline attribute data

The graphical representation of a LRS in a GIS is the collection of lines (links) with endpoints representing points along the roadway. The collections of links form the roadway network, but do not have knowledge. This lack of intelligence prevents the practical implementation of any LRM used to locate data or create routes. An additional step must be included that assigns relevant attribute data to the links. These attribute data provide identifying information which can be used in a variety of GIS applications. The physical attribute data can be changed or manipulated without affecting the underlying representation of the roadway in the

LRS. The use of attributes to describe polyline features (or links) provides a more dynamic platform to utilize the LRS in multiple applications. The most important attributes stored in WISLR are roadway category, route name, and municipal location. Apart from the spatial representation and resolution of roadway elements, the first two attributes are assigned to WISLR by matching the corresponding STN data. The resources use to define roadway attributes using STN resources will be discussed in the following section.

Road way categories are described by four main types: Interstate (IH), US highway (USH), state highway (STH), and off-main lines. The latter is further broken down into five types: Ramp, Connector, Wayside, Rest Area, and Weigh Station. These roadway categories have been used by WisDOT since the development of STN so inclusion of the same categories in WISLR is desired. The attribution of roadway category facilitates programmatic quantification of each category which can be used for reporting and during many types of analysis.

The primary source of information used to assign roadway categories and route names comes from the Reference Point (RP) system. The RP system is used by WisDOT in STN as a means of designating the linear location of features along a road segment. While STN uses the RP system as a LRM, for the purposes of WISLR editing, RP's serve as a resource to name and categorize specific features along a roadway. RP's are assigned to STN highways at landmark locations, (bridges, at grade intersections, etc.) and at termini of off-mainline features. Where RP's exist that describe at-grade intersections and a change in roadway category occurs, an RP diagram is generated. The diagram is manually generated by WisDOT personnel and is incorporated into the data management plan related to RP numbering. A typical RP diagram is shown in Figure 3.1 (a) alongside the STN links representing the roadway, Figure 3.1 (b). As seen in Figure 3.1, the RP image provides a cartographical representation of the STN links.

Figure 3.1. RP diagram (a) detailing roadway types and route names associated with STN links (b) at the interchange.

The RP diagram contains four pieces of relevant information needed to properly attribute WISLR links: a descriptor specifying mainline or off-mainline, the route name, transportation symbol indicating route type, and secondary routes, if applicable. To illustrate how RP's are used, Figure 3.2 is an annotated version of the RP diagram in Figure 3.1. The RP's point to intersections corresponding to nodes on STN links.

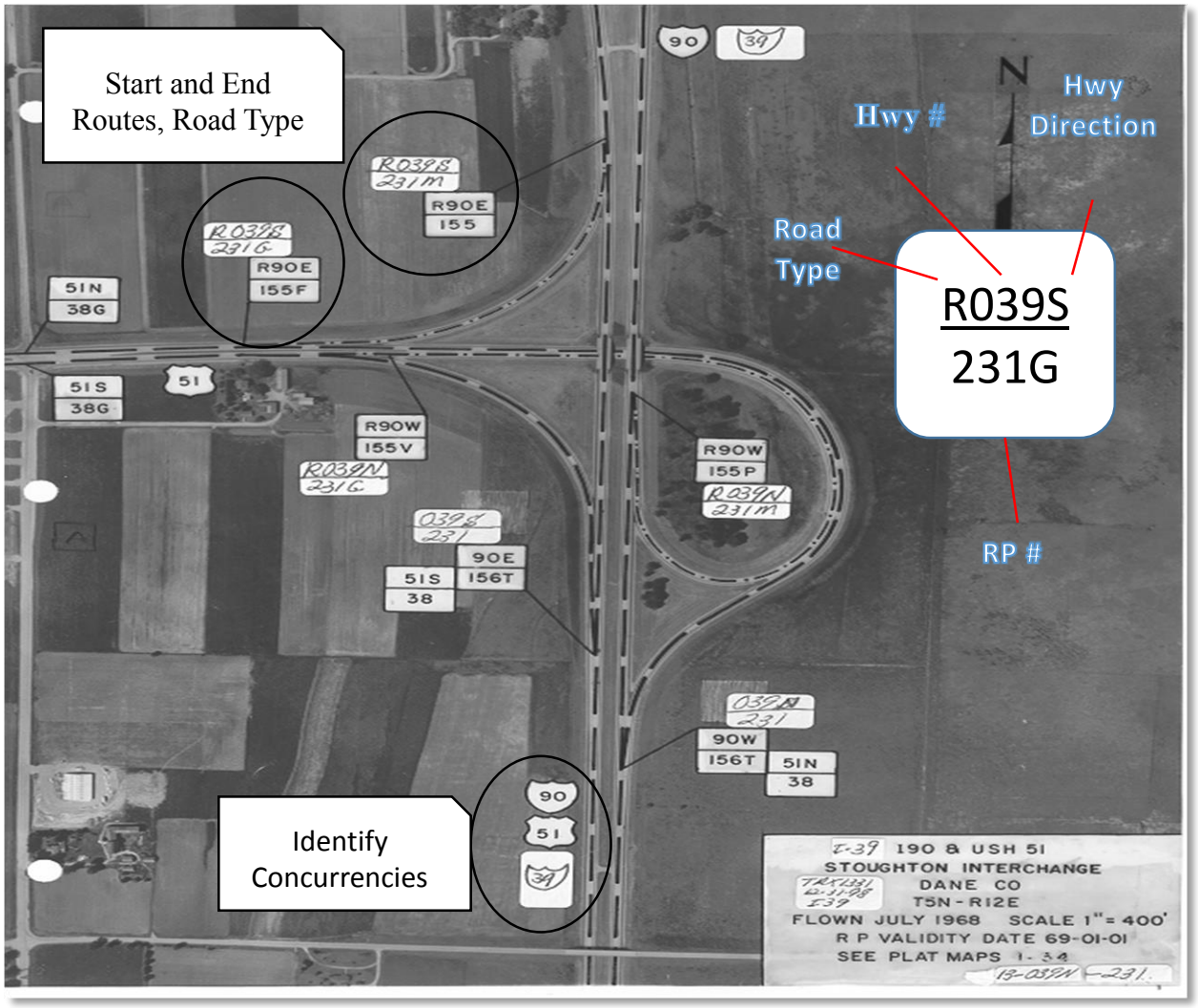


Figure 3.2. Annotated RP image describing the individual elements of the STN LRS

The RP tag is shown as a small white stamp on the diagram and contains four pieces of information, three of which are used when identifying attributes needed for WISLR. In the example shown in Figure 3.2, the highlighted RP is listed as R039S/231G. The first digit, represented by R in the example, establishes the roadway type. STN uses the following five letters to describe the type of roadways: A: Safety Rest Area, C: Connector, F: Frontage Road, R: Ramp, S: Weigh Station. If the highway type is a mainline, the position is left blank. The designation assigned by this position is used when assigning the roadway type attribute in

WISLR. The second element of the RP tag is the highway number, shown as 039 in the example. For an RP on a mainline, the number of the highway on which the feature is being represented is listed. If the RP describes an off mainline feature, the intersecting mainline highway number is used. This highway number is used for naming in WISLR. Naming WISLR links by matching STN's naming convention ensures uniformity between the LRS's, provides another level of relationship between WISLR and STN, and extends the ON-AT LRM to the state routes in WISLR, which was not previously as robust. The last position on the first line of the RP indicates highway direction. WisDOT follows the convention that all state highways proceed in an east to west or north to south direction, regardless of compass direction or vehicle travel direction. The number shown on the bottom portion of the RP tag is assigned by WisDOT and is not used for WISLR editing purposes.

The RP system provides the basis for naming and categorizing WISLR links during editing. Throughout the state of Wisconsin, many different types of roadway features and scenarios exist, so a complete naming convention was developed to encompass all known possible categories and naming scenarios that may be encountered. Table 3.1 shows the naming convention. The complete naming convention document is provided in Appendix B and serves two purposes: 1) sets the standard for detailed and consistent naming guidelines for the various state trunk network features, 2) highlights the use of RP diagrams for attributing (categorizing, naming, and direction of travel) WISLR by annotating existing diagrams with the correct name and category type and acts a guide for use by editors.

Table 3.1. Naming convention and category assignment used for WISLR editing.

Highways and Facilities	WISLR LCM Naming Method
Mainline	USH 14, STH 14, IH 39
Ramps	Ramp USH 14 to CTH MM (2)
Connectors	Connector USH 141 to CTH MM (1)
Connectors that connect two portions of the same highway	Connector USH 141
Connectors with Crossovers	Connector USH 141 to CTH MM (3)
Roundabouts – Marked as Connectors	RAB USH 141 (1)
Roundabouts – Marked as Mainline	USH 141
Frontage Road	Frontage Rd IH 43 (1)
Weigh Stations	SWEF # 19 (*1*)
Brake Check Area	Brake Check Area USH 8 (1)
Park and Ride Lots	P&R College Ave (1)
Rest Areas	Rest Area # 22 (1)
Waysides	Wayside (1)
Turn Lanes	Turn Lane USH 141 to *"insert local name"*
J-Turns	J-Turn Connector USH 141 (1)
Business Route	USH B51

Concurrent Routes

Concurrent routes, also known as secondary routes, occur when the physical roadway carries two or more different highway or routes numbers. Wisconsin allows concurrent routes to be posted on state highways, therefore managing secondary routes is necessary. To begin, the

roadway links are given a primary name, which follows the IH, USH, STH hierarchy. Within each of these categories, the lower numbered route will be used in naming. For example, in central Wisconsin, IH 39, IH 90, and IH94 are concurrent for thirty miles. The primary name given to the shared links is IH 39. The other two routes are assigned as secondary routes. Assigning a primary name provides consistency throughout the route and is also used in for display purposes.

Digitizing centerline data and creating polyline features with attributes (links) does not establish the route. Establishing the route is accomplished by sorting links, which puts links in order, traversing the cardinal direction of the route. Sorting is first performed on the primary route, followed by the secondary route(s) using a custom tool embedded within the LCM application. Assigning routes to links pieces together the network that make up the LRS and is a key component to the ON-AT LRM used by WISLR. Locating events on state routes in WISLR using ON-AT requires stable and consistent route name information to be stored, making it vulnerable to name changes, so strict standards for naming and formatting are followed.

The existence of a secondary route is not always obvious. RP diagrams provide one source of information, especially at exits and interchanges. But concurrencies on state routes can occur in places not represented by RP images, and as such, require the use of STN route data tables. These tables provide a complete list of routes, including secondary routes, and the associated links those routes follow.

Local road concurrencies are also possible, especially in urban areas. Assigning the local road name as a secondary route is needed because the WISLR base map is used by traffic operations personnel and law enforcement, and often times the local users might only know a state route by its local road name (i.e., USH 8 is marked as 14th Ave within the corporate limits).

Two separate ArcMap files are used at the local desktop by the editors to locate state and local secondary routes during editing.

Realignment of state line work

Realignment refers to the accurate cartographic representation of each roadway segment. Accurate spatial alignment is necessary because centerline WISLR link data are based on digitized line work, so differences in cartographic variations cause inaccuracies in the stored length values in WISLR. This variation leads to greater error in data translation between STN and WISLR. Performing this edit requires reshaping existing WISLR line to match aerial images. Ancillary realignment is performed where local roads intersect state roads.

3.1.3 *Workflow*

A flow chart has been developed that incorporates the type of edits needed with the functionality of the tools in LCM. The flow chart, shown in Figure 3.3, guides the logical order of edits by tying the type of edit performed to the process that must be followed based on the program rules. For example, if a particular link is drawn in the incorrect direction with respect to flow of traffic, LCM will not allow the editor to simply reverse the direction of the link. The editor must abandon the link and digitize a new link and create a new route. The flow chart is inclusive of the enhancements needed to comprehensively edit WISLR state trunk data.

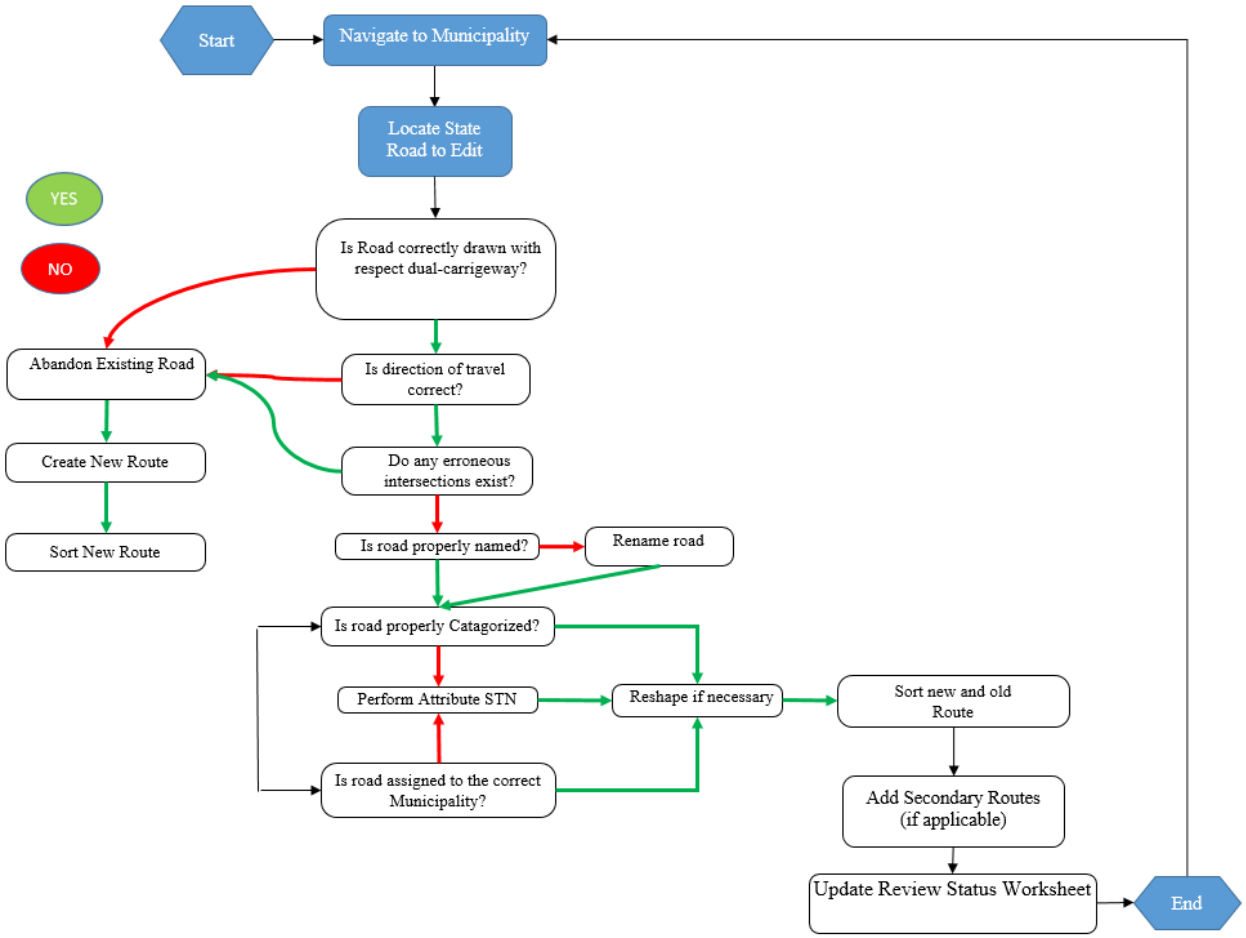


Figure 3.3. WISLR editing flow chart detailing order of edits based on program rules.

3.1.4 *WISLR Progress.*

Tracking editing progress is a multistep process. Institutional constraints at WisDOT make real-time access to the WISLR data unavailable, so a method to track progress was developed using STN links. Since the only edits to be performed are along state highways, and the fact that WISLR state trunk data is incomplete, the most useful resource for tracking edits was the use of STN link data. The STN shape data (shape data is a term that describes data that can be viewed and analyzed geospatially) and accompanying attribute tables were obtained from a designated WisDOT data repository. This data was post- processed using ESRI's ArcMap 10.3 so that all mainline and off mainline routes were represented and each STN link was attributed to

a unique municipality. Assigning STN links by municipality make it possible to select and uniquely classify links as each segment of the route in WISLR is edited.

Managing progress requires the individual editor to report the route and the municipality that the route traverses when all edits are complete. A shared file using SharePoint®-a Windows® based application, was established that allows each editor to enter route identification items as each route editing is complete. Reporting by municipality provides a standardized order to track progress and ensures complete route editing.

The completed route for each municipality is used to assign a unique identifier and date of completion of the route in ArcMap. The date information is used to track progress, schedule route editing, and forecast completion of routes.

3.1.5 *Quality Assurance and Quality Control (QA/QC)*

The quality of the WISLR line work editing is important to maintain consistency within the WISLR LRS and for the enhancement of data translation between STN and WISLR. A series of steps are performed during and after editing which are designed to identify naming and attribution errors and to ensure complete route editing.

Interpretive error is the primary cause of errors during editing operations. These errors include incorrect primary route name, wrong roadway type category, and incorrect municipality assignment. These errors are classified as interpretive due the method by which WISLR data is attributed. Editing necessarily involves the inspection of multiple resources, requiring editors to interpret the STN representation of a given route and translate that into the WISLR LRS, increasing the possibility of interpretive risk. The required step of applying a sort order to links to form a route also provides the quality assurance check for correct and consistent naming, proper municipality assignment, and roadway category type. The reason is that the sort

application uses each link's route name to put the links in order of cardinal direction through a municipality. If an error exists along a route related to naming or municipality assignment, the sort tool will not allow the links to be sorted into the route and it will alert the editor to make the necessary corrections. This step is performed during the active editing phase and provides the most robust tool for quality assurance.

Following the completion of a given route, an entry is made into the SharePoint file that will be used by WisDOT staff for further quality control. The full route traversal is displayed graphically and examined for gaps. If an editor has skipped over a municipality that a route passes through, or failed to report a municipality, WisDOT personnel inform the editor and the correction is made.

The final quality control check involves the visual inspection of routes and comparison of edited WISLR line work with STN images. This step is performed by at least two trained individuals. Errors found during this phase of quality control typically involve the omission of short off main line links, or state route intersections with local roads that are out of alignment.

3.1.6 *Conclusion*

The methodology presented in this section was developed to enhance the state trunk network resolution of WISLR to match the state trunk data in STN. This section outlined the remote connectivity, types of enhancements required to improve data translation, and the various resources used to match the LRSs. The next section will focus on one area where the enhancements to WISLR will improve data translation between STN and WISLR. Chapter 4, Results, will present the current status of WISLR enhancements.

3.2 Gore Point Calibration

The enhancements to WISLR state trunk line work outlined in the previous section will improve data translation between STN and WISLR by eliminating most, if not all, ambiguities that exist within WISLR without changing the fundamental process for populating the Link-Link table. Ambiguities most often result from the lack of representation in WISLR, forcing the inclusion of problem flags in the Link-Link table. One ambiguity that cannot be eliminated by line work editing exists around gore points, specifically the way in which the two systems represent nodes at gore points. This dissimilarity manifests itself by a large offset difference when data are translated between the two systems close to the gores. While the inclusion of complete state trunk line work in WISLR will not eliminate the linear offset difference that exists between gore points, the ability to define a method to reconcile this difference now exists. Consequently, the enhancement of state route representation in WISLR to match STN is fundamental to the methodology proposed in this section.

Gore points are commonly known as the triangular intersection where two roads merge or split, such as at an on or off ramp. STN business rules have established strict guidelines to locate gore points in the field. As a convention, STN defines the gore point as the sharp angle where two pavements intersect, usually where grass or gravel meet the pavement, ensuring consistent field measurements and replication for the purposes of mainline and ramp mileage. WISLR gore points are subjective in nature because they are established through digitized line work by a technician that places the nodes of links at the centerline merge point of the ramp and mainline. This convention is generally followed by WisDOT staff for local road representation, but rarely implemented during the state trunk line work editing for reasons outlined in previous sections. Along state routes, the pre-edited gore point locations in WISLR are inconsistent, incomplete,

and inaccurate. Often times, the dual carriageways are not accurately shown so the ramp intersections with main lines would be poorly located along the route, making it impossible to establish a standard for calibrating these areas. Consistent and complete representation of gore point locations in WISLR, although subject to human interpretation, provides the foundation to capture the STN gore point location spatially on the WISLR link and define a relationship that reconciles this difference and improves data translation. This section will detail a method to improve the linear precision of data translation at gore point affected links using enhanced WISLR link work and a populated Link-Link table.

3.2.1 *Un-calibrated Data Translation at Gore Points*

The Link-Link table provides the platform to translate data from STN to WISLR by relating links in each LRS that represent the same segment of roadway. The positional accuracy of data movement between the systems is compensated for through the use of access points or a ratio calculation. However, neither one of these methods can resolve the positional difference caused by gore point locations. In the STN LRS, links traveling from an on ramp toward an exit ramp, tend to have a much larger length measurement compared to the same WISLR link because the measurement is taken from the gravel or grass nose so the acceleration and deceleration portions of ramp are included in the mainline link measurement. Consequently, the link between an off ramp and on ramp are much shorter compared to the corresponding WISLR link. Figure 3.4 illustrates the offset difference through a de-constructed representation of an on-ramp highlighting the gore point in each system.

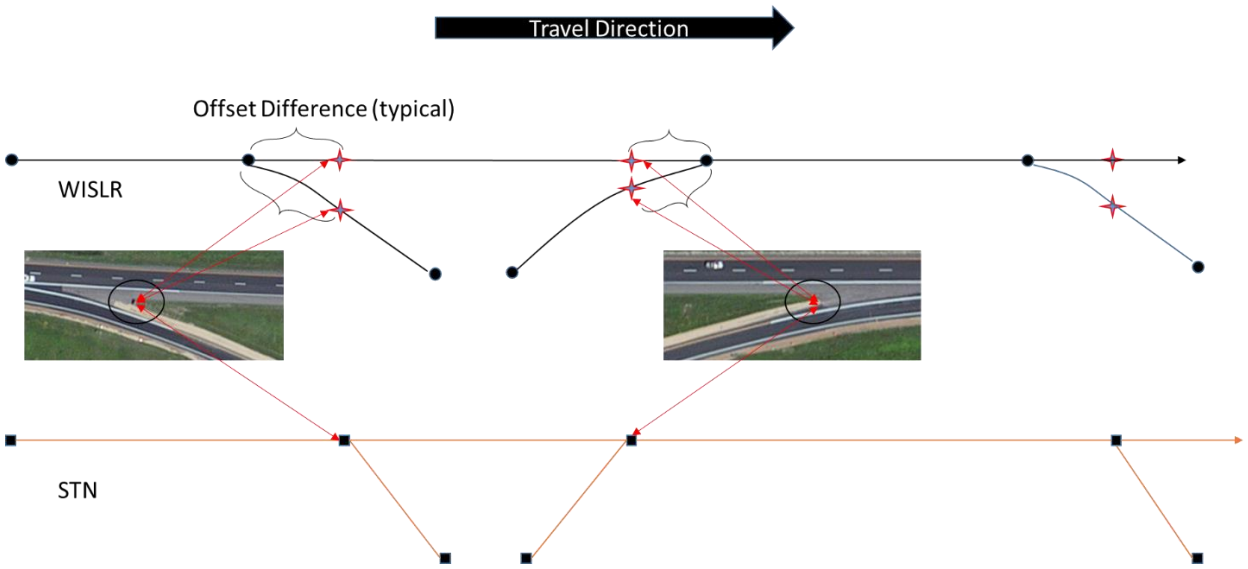


Figure 3.4 Offset difference caused by different representations of gore points in WISLR and STN

While the length differences alone do not cause positional data movement inaccuracy, the defining position of the gore point at the beginning and ending node of the link(s) within each LRS causes a large local variance close the gore points during data translation, with the greatest difference occurring at the gore point and dissipating toward the midpoint of the link relationship.

This difference is most easily shown by translating a crash event occurring close to, or at, a gore point, from STN to WISLR. Figure 3.5 illustrates the difference by translating four actual recorded crashes on STN links that occurred close to gore points to the corresponding WISLR links.

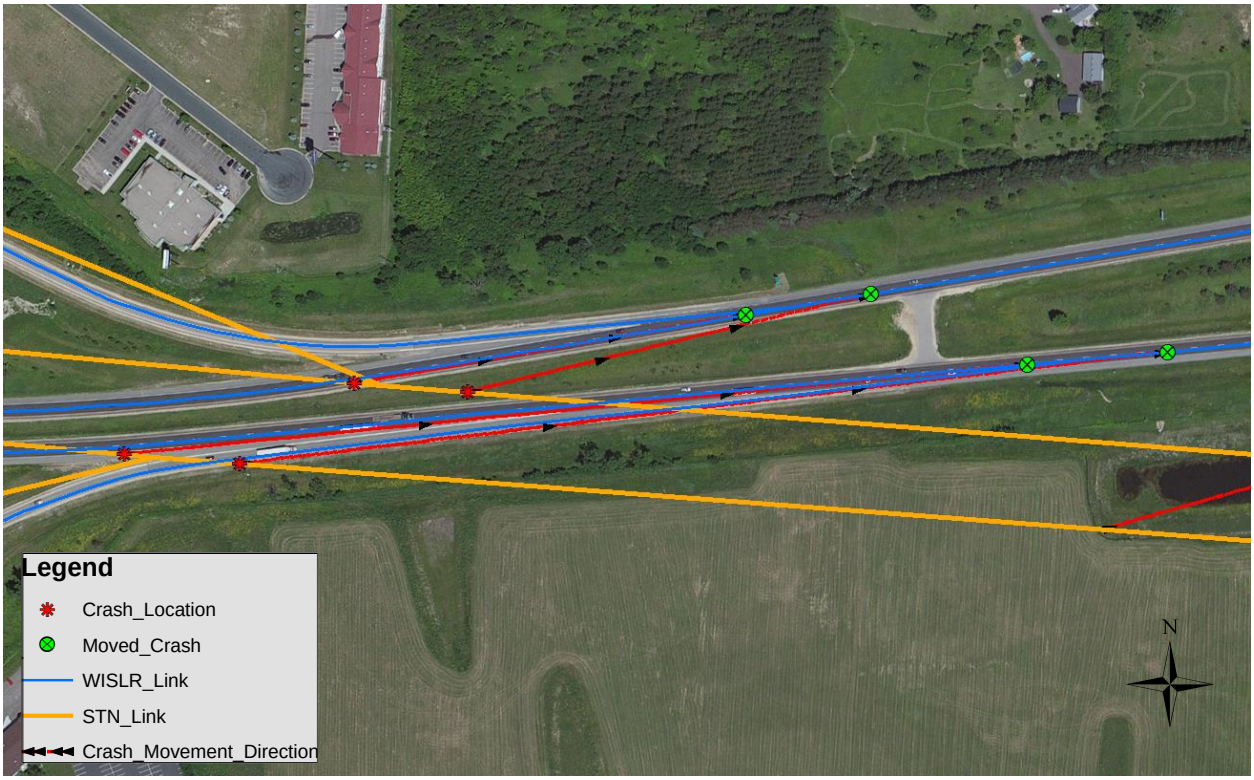


Figure 3.5 Crash data movement through an un-calibrated Link-Link table, highlighting the large variance in positional location when data is moved from STN to WISLR close to gore points

The crash data shown in Figure 3.5 was translated using the un-calibrated Link-Link table. The four crashes are recorded in STN to have occurred close to the gore point. When translating the crashes to the WISLR links in the east bound lane, the crash moves to a location 1,700 feet east of the actual crash point. On the west bound lane of the highway, the crash moves back (east) almost 750 feet before the actual location of the crash. The example in Figure 3.5 highlights data movement between mainline links, but the same discrepancy occurs along the ramp links.

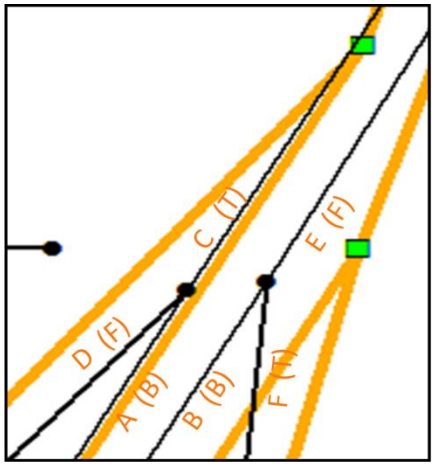
3.2.2 *Approach*

Calibrating the Link-Link table requires complete state route line work representation in WISLR and a current populated Link-Link table. The method presented here builds directly on

the established procedure which relates links in STN to corresponding links in WISLR. The three basic steps needed to calibrate the Link-Link table for gore length discrepancies are identification of gore affected links, generating a table of gore points with offset values, and writing the calibrated records. Each step will be detailed in this section.

Identifying Gore Affected Links

Gore point locations can be thought of as node attributes. Links that connect to that node are going to be affected by the presence of the gore point. The Link-Link table recognizes this attribute by including a flag for the records in the table that are associated with a gore point, using one of three identifying values: Both (B), To (T), or From (T). The From and To flags are used to identify links that are traveling away from or towards a gore point, respectively. The Both flag means that gores exist on both ends of a link. Figure 3.6 illustrates the use of problems in the Link-Link table.



STNid	STNstart	STNend	WISLRid	WISLRstart	WISLRend	T	M	G
A	0	440	i	0	2587			B
B	0	490	ii	0	2510			B
C	0	220	iii	0	1282			T
D	0	240	iv	0	1019			F
E	0	1880	v	0	9865			F
F	0	1820	vi	0	9917			T

Figure 3.6 Example showing gore point flags used in the Link-Link table.

Gore point flags are assigned by a coder during the population of the Link-Link table and provide a means to visually locate affected links. The intent of the flag is to alert an analyst that the record contains gore point affected links and any data translation through these records should be manually inspected. Figure 3.7 shows the gore point affected links highlighted with unique symbology based on the gore point flag. This built-in functionality of the Link-Link table aids in identifying locations, or more appropriately links, which require calibration..

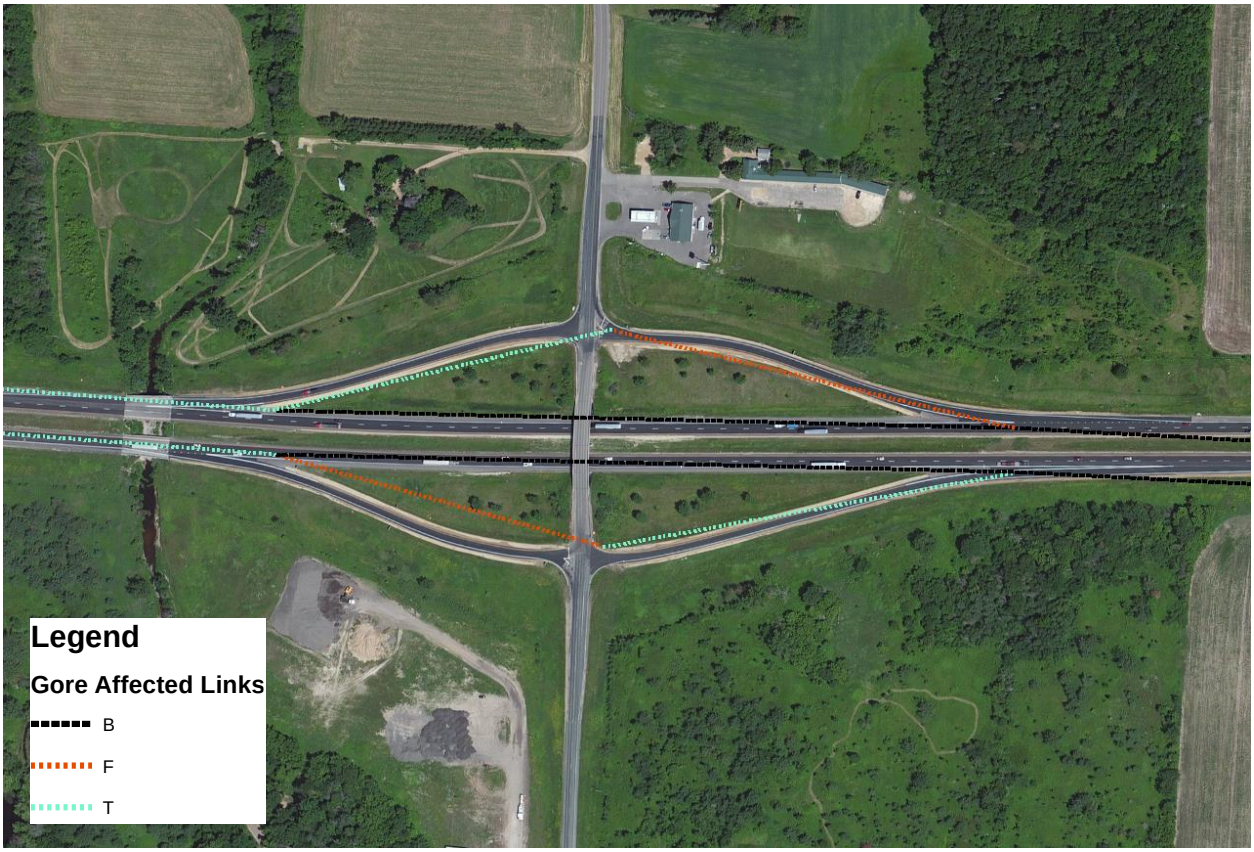


Figure 3.7 Gore point affected links shown with unique symbology generated using gore point flag columns in the Link-Link table.

Gore Point Table Generation

In order to establish a calibrated relationship between gore point affected links, a measure must be obtained that defines the locational difference. This measure is captured through a point feature data set generated within ERSI's ArcMap. Once the affected link is located, the WISLR

link is overlaid on an aerial image and the coder places a point directly on the WISLR link corresponding the gore location at a map scale of 1:250. Since gore locations, as defined by STN, are not directly in the centerline of the roadway, the location is translated laterally to the WISLR link. This process, detailed in Appendix C, produces two gore point tables, one for mainline gore points and one for ramp gore points. Figure 3.8 shows the gore point locations placed by the coder and the resulting data tables.

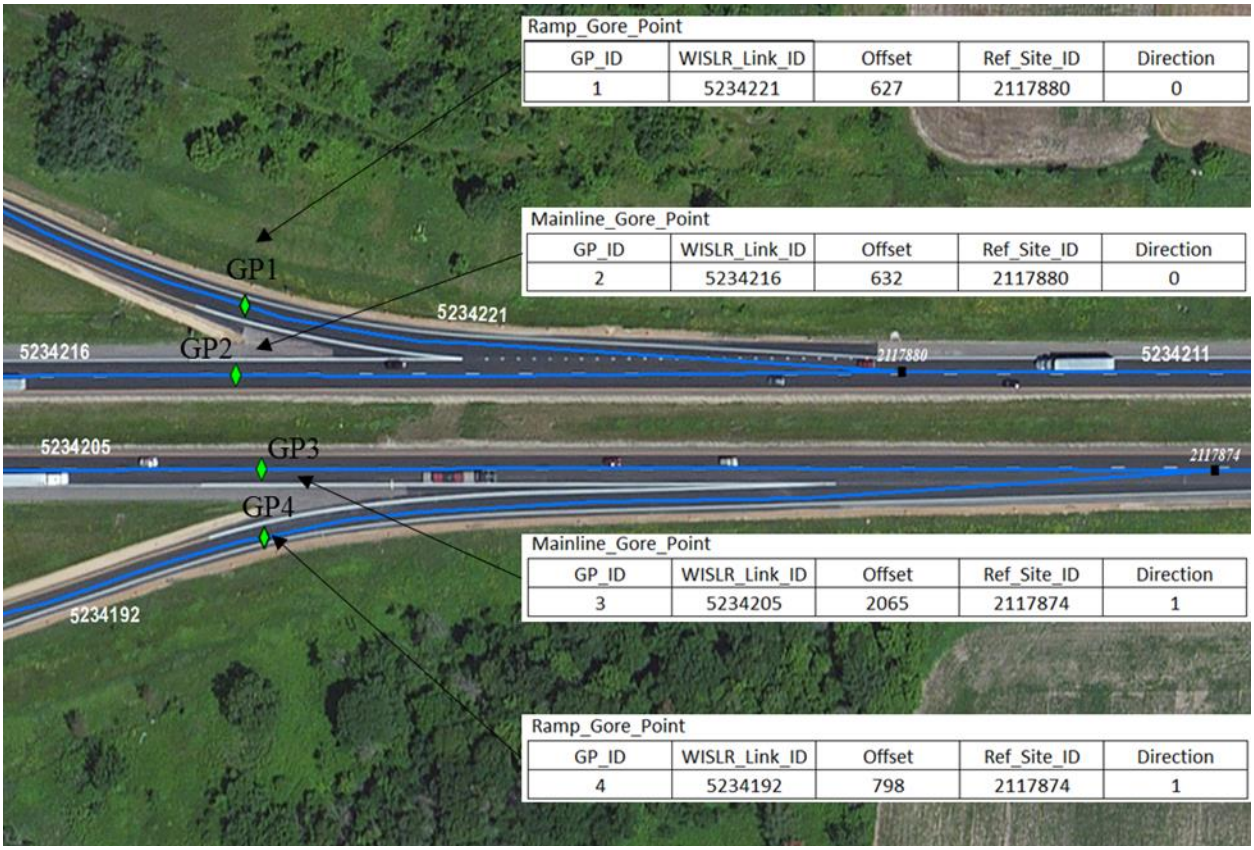


Figure 3.8 Gore points placed on WISLR links coorsponding to the STN gore point, with the generated data tables. One table for mainline gore points and one for ramp gore points.

The tables include five columns, four of which are essential to the calibration process. The pertinent elements are WISLR_Link_ID, Offset, Ref_Site_ID, and Direction. These columns are detailed below.

- WISLR_Link_ID- This columns identifies the link on which the gore point is located.

- Offset-The Offset is the measure along the WISLR link from the beginning site. This value represents the offset difference and is used directly in the calibration procedure. The value is a derived from the shape distance and is stored in feet.

- Ref_Site_ID- The Ref-Site_ID is captured during the point generating process as the nearest WISLR gore site to the gore point. In Figure 3.8, notice that for GP3 and GP4 the Ref_Site_ID is the "to" site, and for GP1 and GP2 the Ref_Site_ID is the "from" site. The nearest site is captured because in some cases the WISLR link is not continuous between gore points, that is, more than one link is present. Storing the nearest site provides the programmatic advantage of establishing the direction from the digitized gore point to the WISLR gore point without forcing an additional operation to track through links that may exist between gore points.

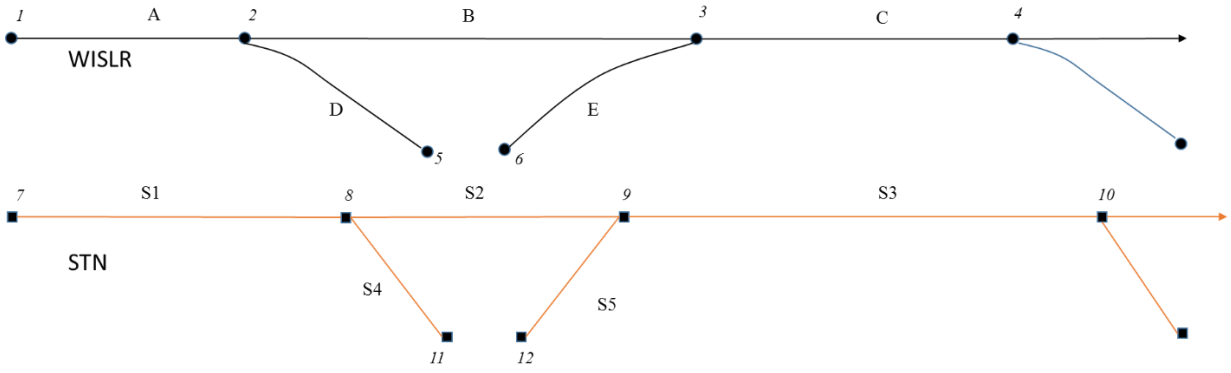
- Direction- The unique value of 1 or 0 is populated in this column to identify the gore point as an off ramp or on ramp gore point. In combination with the Ref_Site_ID, the direction indicator aids in establishing a logical order, or track, for the links to be ordered programmatically.

Two tables are produced for the purpose of identifying ramp and mainline gore points, respectively. This distinction is made to control duplicate data transfer when translating data, specifically to prevent duplicate transfer from STN to WISLR and to allow data transfer from the calibrated segments of the WISLR ramp and mainline to the STN mainline. This distinction will be apparent following further explanation of the calibrated table.

Writing Calibrated Link-Link Records

The process of writing a calibrated link relationship into the Link-Link table involves rewriting the affected record(s) to reflect the gore point offset value. This re-written record

establishes a new relationship that will serve as the basis for data translation for the affected links. Once the affected links have been identified and the gore point table generated, the steps to calibrate the records can begin. The process is most easily defined through a basic example. Figure 3.9 shows a typical gore point scenario along a divided highway with the un-calibrated Link-Link table.



Link attributes

STN Link ID	From Site	To Site	Distance
S1	7	8	130
S2	8	9	130
S3	9	10	160
S4	8	11	110
S5	12	9	100

Distance=1000's of mile

WISLR Link ID	From Site	To Site	Distance
A	1	2	500
B	2	3	1000
C	3	4	500
D	2	5	800
E	6	3	700

Distance=feet

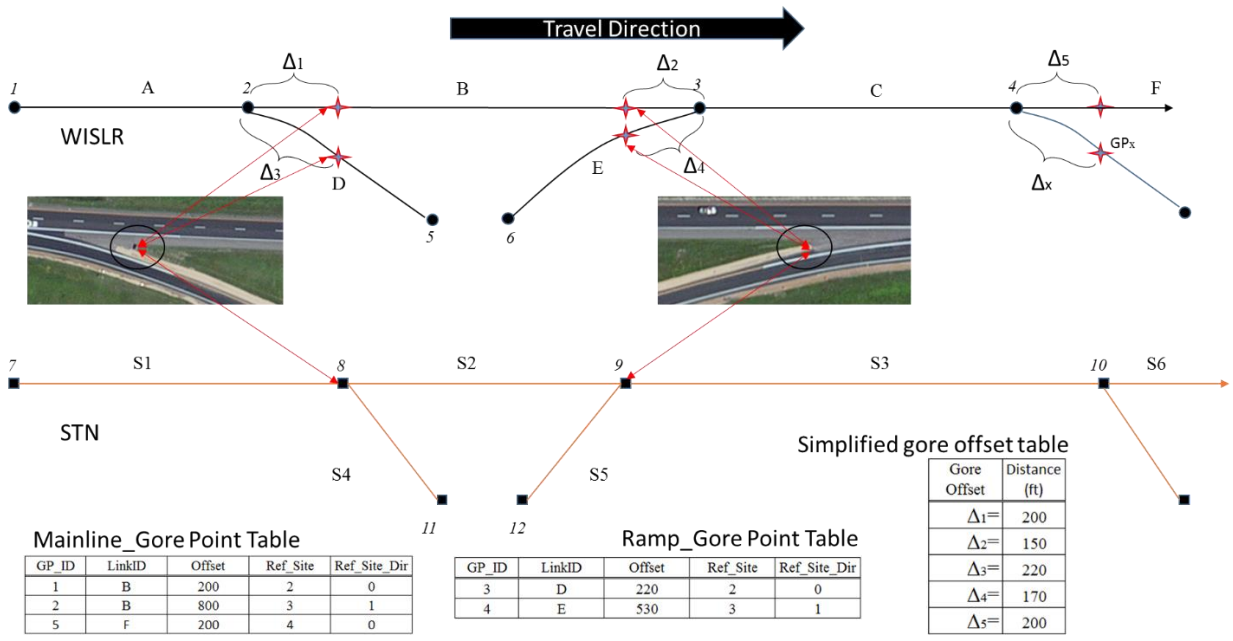
Un-Calibrated Link-Link table

STNid	STNstart	STNend	WISLRid	WISLRstart	WISLRend
S1	0	130	A	0	500
S2	0	130	B	0	1000
S3	0	160	C	0	500
S4	0	110	D	0	800
S5	0	100	E	0	700

Figure 3.9 Example of an un-calibrated Link-Link table at a gore point affected divided highway

For the example shown in Figure 3.9, the populated un-calibrated table uses the link attributes to directly populate the records.

The calibration process forces a shift in the link to link relationship based on the gore point offset value. This shift moves the gore point measure amount from one link and applies that measure to either the preceding or the following link, depending on the location of the gore point, and establishes a new record that contains the offset value and the corresponding links. The leftover measure is then used to write another record to define the next relationship along the path. To demonstrate the method, the example shown in Figure 3.9 is reintroduced in Figure 3.10 with the manually generated gore points shown on the WISLR links.



GP_ID	LinkID	Offset	Ref_Site	Ref_Site_Dir
1	B	200	2	0
2	B	800	3	1
5	F	200	4	0

GP_ID	LinkID	Offset	Ref_Site	Ref_Site_Dir
3	D	220	2	0
4	E	530	3	1

Gore Offset	Distance (ft)
Δ_1=	200
Δ_2=	150
Δ_3=	220
Δ_4=	170
Δ_5=	200

Figure 3.10 Gore point affected links with the manually generated gore points shown on the links with the generated gore point tables and a simplified gore offset table inset

Included in Figure 3.10 is a simplified gore offset table. To aid in clarity, three elements from the generated gore point table are excluded from simplified table, the WISLR_LinkID, Ref_Site_ID, and Direction. The Δ values correspond the gore offset along the link, with two exceptions. The values of Δ_2 and Δ_4 are derived as the difference between the full length of the WISLR link and the offset value captured in the generated gore point table.

Using the un-calibrated Link-Link table records and the simplified gore offset table, shown in Figures 3.09 and 3.10 respectively, the calibrated table is set up as shown in Figure 3.11.

Mainline Affected Links

STNid	STNstart	STNend	WISLRid	WISLRstart	WISLRend
S1	0	*STNPart1*	A	0	*Full Length*
S1	*STNPart1*	*Full Length*	B	0	Δ_1
S2	0	*Full Length*	B	Δ_1	*Full Lenth-* Δ_2
S3	0	*STNPart2*	B	*Full Lenth-Δ2*	*Full Length*
S3	*STNPart2*	*STNPart3*	C	0	*Full Length*
S3	*STNPart3*	*Full Length*	F	0	Δ_5

$$STNPart1 = S1 - \left[\frac{\Delta 1}{A + \Delta 1} = \frac{STNpart1}{S1}\right] \equiv S1 - \left[\frac{\Delta 1 \times S1}{A + \Delta 1}\right]$$

$$STNPart2 = \left[\frac{\Delta 2}{(C + \Delta 2 + \Delta 5)} = \frac{STNPart2}{S3}\right] \equiv \left[\frac{S3 \times \Delta 2}{(C + \Delta 2 + \Delta 5)}\right]$$

$$STNPart3 = \left[\frac{\Delta 5}{(C + \Delta 2 + \Delta 5)} = \frac{STNPart3}{S3}\right] \equiv S3 - \left[\frac{S3 \times \Delta 5}{(C + \Delta 2 + \Delta 5)}\right]$$

Note:
S1,S3,A, and C are the corresponding link distance values

Ramp Affected Links

STNid	STNstart	STNend	WISLRid	WISLRstart	WISLRend
S1	*STNPart1*	*Full Length*	D	0	Δ_3
S4	0	*Full Length*	D	Δ_3	*Full Length*
S5	0	*Full Length*	E	0	*Full Length-Δ4*
S3	0	*STNPart2*	E	*Full Length-Δ4*	*Full Length*

Figure 3.11 Calibrated Link-Link table populated with defined values. The defined values are described as STNPart1, STNPart2, and STNPart3.

The defined values of STNPart1, STNPart2 and STNPart3 are calculated using a modified version of the ratio method described previously. The modification adds the gore point offset value to the full length of the WISLR link, effectively stretching the overall WISLR link length to an equivalent STN length representation. This value is then used to populate the STNend and STNstart values in the calibrated table. The alternative to using the ratio is performing the direct conversion calculation from feet in WISLR to hundredths of mile in STN. While this would produce results similar to the ratio method for data translated close to the gore point, it would limit the calibration zone only to the area within the length discrepancy. Utilizing the ratio will evenly distribute the link relationship throughout to entire STN and WISLR link, ensuring even distribution of each link's length.

Once the calculations have been performed, the link records are populated into the table. The example is continued in Figure 3.12 with all values populated.

Mainline Affected Links

STNid	STNstart	STNend	WISLRid	WISLRstart	WISLREnd
S1	0	93	A	0	500
S1	93	130	B	0	200
S2	0	130	B	200	850
S3	0	28	B	850	1000
S3	28	*122*	C	0	*500*
S3	*122*	*160*	F	0	200

$STNPart1 = 130 - \left[\frac{130 \times 200}{(500 + 200\ )}\right] = 93$

$STNPart2 = \left[\frac{160 \times 150}{(500 + 150 + 200)}\right] = 28$

$STNPart3 = 160 - \left[\frac{160 \times 200}{(500 + 200 + 150\ )}\right] = 122$

Ramp Affected Links

STNid	STNstart	STNend	WISLRid	WISLRstart	WISLREnd
S1	93	130	D	0	220
S4	0	110	D	220	800
S5	0	100	E	0	530
S3	0	28	E	530	700

Figure 3.12 Calibrated Link-Link table records. The tables are shown divided only for clarity and are not divided within a fully populated Link-Link table

The calibrated Link-Link table shown in Figure 3.12 contains ten records, compared to five for the same links in the un-calibrated table. In practice, the new table will include ten new records. This is due to temporal data tracking within the Link-Link table. If a link in either LRS is no longer valid, meaning that it not actively used to display the roadway as it currently exists, or if changes to length measurements are shown to occur, the Link-Link record will become historic. While the link is no longer active or valid, legacy data, such as crashes, might still be associated with the link. To accommodate for line work changes that occur in either system, the Link-Link table utilizes date columns that indicate whether or not a link relationship is current (Morrison, 2012). Rather than deleting a link relationship when either link changes in the LRS, the Link-Link table update process requires the affected record to be populated with a date in the Historic date column. Inclusion of these date columns allows data to be translated through the table on both current and historic link records, as needed. In the case of calibration, an historic date will be populated in the un-calibrated link record, and the calibrated record will become the

current link relationship. In order to track the calibrated records, a column is appended to the Link-Link table called Calibrated. A value of U is entered into the historic link record, indicating the record was made historic by gore point calibration. The current calibrated record will be populated with a value of C. Uniquely identifying the un-calibrated record allows more flexibility for an analyst and potential future programmatic calibration techniques.

Calibrating a set of links at the gore points will cause a segment of length along an STN link to be duplicated in the Link-Link table. One record will contain the mainline WISLR relationship, the other will relate to the WISLR ramp portion. This is illustrated in Figure 3.12, where Link S1 from measure 93 to 130 is related to both WISLR link B in the mainline table and link D in ramp table. If this duplicated relationship is not defined, data translation along this segment will occur twice when moving from STN to WISLR. However, data movement in the reverse direction will not be affected, as the WISLR link is not duplicated in the table. The separate ramp and mainline gore point tables were generated to address this issue and a second new column called Ramp_Seg is appended to the Link-Link table. The ramp record in the Link-Link table that is affected by the gore point difference is identified and a unique value of 1 is placed into the Ramp_Seg column. This added functionality allows data transfer to be defined to specifically exclude records with a Ramp_Seg value of 1 when moving data from STN to WISLR, and allow unaffected use of the records in the reverse direction. The Ramp_Seg column and the Calibrated columns are appended to the Link-Link to prevent any disruption to current programming code.

3.2.3 *Implementation*

The research presented here calibrates a populated the Link-Link table. In order to calibrate the records, the table must first be populated according to the already established

procedure that addresses gore points with the flag column. Immediately following the process outlined in the county wide QA/QC portion of the update procedure, the coder will then execute the calibration steps. The purpose behind calibrating a properly populated table is two-fold. The quality checked, un-calibrated table provides a control to compare with the calibrated table, and the inclusion of a unique identifier for a calibrated record in the table affords analysts the option of translating data using either version of the table. It is possible that other calibration techniques are devised that do not require manual calibration at gore points, so maintaining the un-calibrated, one-to-one link-link relationship is desired.

3.2.4 *Quality Assurance and Quality Control (QA/QC)*

The Link-Link table update procedure includes four basic QA/QC steps that must be implemented to ensure the highest quality table is produced (Morrison, 2012). The research presented here does not require the development of additional quality checks, instead the repetition of three steps: STN link check, WISLR link check, and the XY Connector line check. The fourth check, which is not repeated, includes a visual gore point affected link check. Repetition of this step is unnecessary because those affected links are the focus of this work. The three checks to be implemented were established during the original Link-Link coding, but will be discussed briefly with the purpose for performing the step a second time.

STN Link Check

All STN links must be coded into the Link-Link table. Performing the process involves checking the STN links in the Link-Link table against the STN data file by joining the data sets based on Link ID. Additionally, the full length of each STN link is compared with the full length corresponding STN link in the Link-Link table. This process will identify any un-coded links and ensure the full length, without gaps or overshoots, of each STN link is represented in the

table properly. The calibration procedure is performed following this step, so it is unlikely that any STN links are un-coded. However, the full length of STN must be rechecked. Any length not recorded in the calibrated table is corrected.

WISLR Link Check

The WISLR link check ensures that all state routes in the WISLR data set are coded in the Link-Link table. The original procedure is a visual process that requires the coder to join WISLR link IDs coded in the Link-Link table to the WISLR shape file data and look for gaps along the route. Since the WISLR data contain all roads in the state, unique symbology is applied to the joined and un-joined links to aid in identification. Any links not coded in the table are examined and the table is revised. The WISLR link check is performed during the un-calibrated QA/QC steps, so checking the links in the table is not required. However, checking the full length, and gaps within links, of the WISLR links is necessary. The same reason for checking the full length of STN lengths applies to this step.

XY Connector Line Check

The final check of the coded Link-Link table involves a step known as the XY Connector Line check. This process provides a visual representation of the link-to-link relationship in the table by generating a set of lines connecting each link (STN and WISLR) along the length of each link. This step is perhaps the most powerful QA/QC step because it essentially translates data, in this case programmatically generated points, along the full length of the links. Three programs were previously developed to accomplish this: STN Points Generator, Point Mover, and XY lines. Details associated with running the XY Connector line check can be found in Appendix A and in previous work (Ryals, 2011). This process is repeated following calibration

and all adjusted areas are examined for errors. .An example comparing the un-calibrated to calibrated table using the XY Connector check is shown in Figure 3.13.

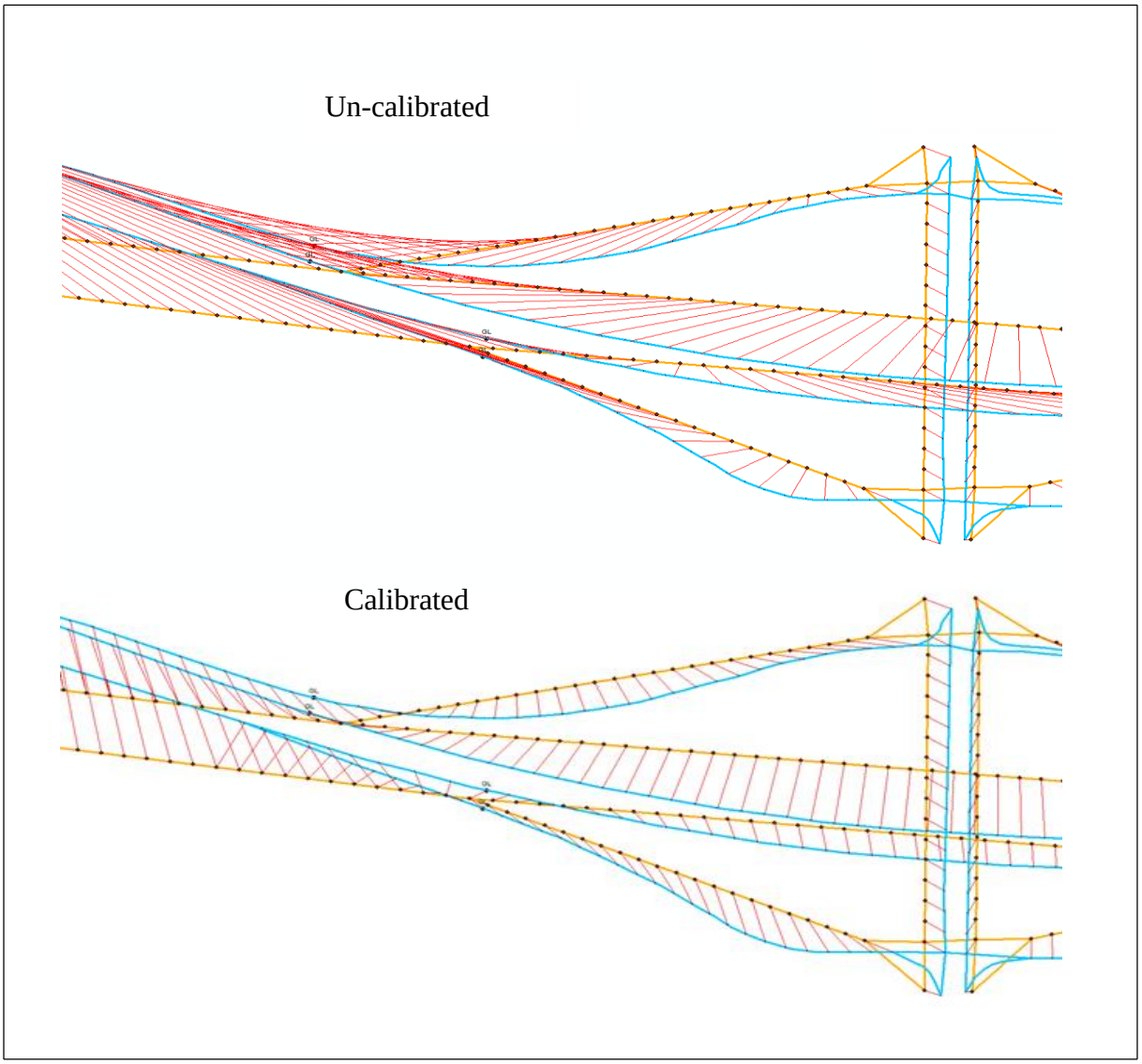


Figure 3.13 Example result of the XY Connector line check performed on un-calibrated and calibrated Link-Link tables. STN links are shown in orange, WISLR in blue, and the connector lines in red.

3.3 Conclusion

This research focused first on developing a method to enhance the state route resolution in WISLR to match STN. It was shown that WISLR can be edited without disruption to

WisDOT's daily business practices using a combination of several resources. Secondly, a method to calibrate the Link-Link table has been developed. The method presented does not alter the process by which the Link-Link table is populated, and serves to provide calibration at links that are affected by the presence of gore points. It has been found that by adding two additional columns to the Link-Link, applications that utilize the table will have greater flexibility when translating data. Chapter 4, Results, will discuss the level of effort required to edit WISLR, the current WISLR editing progress, and present the results of a case study in which Link-Link table records have been calibrated and crash data has been translated on an interstate highway in St. Croix county Wisconsin.

CHAPTER 4

RESULTS

4.1 WISLR Editing

Editing the state trunk line in WISLR was divided into three phases based on the roadway category: The order began with the focus on Interstate Highways, followed by US Highways, and finishing with State Highway. The editing order was established based primarily on the point that the IH category line work contained the least accurate representation of the actual roadways the most off main line features, such as ramps, rest areas, waysides, followed closely by US routes.

Interstate highways make up 782 miles of roadway in Wisconsin, accounting for approximately 6% of the total state mileage. Editing began on September 18th, 2014 with two editors and took eight months to complete. During the first three months of editing, logistical issues with editing were resolved, such as remote connectivity into the WisDOT workstations and establishing procedures to address technical problems encountered during editing. During this time, many of the technical aspects discussed in Chapter 3 were developed and the training of additional editors took place. Editing interstate routes was the most challenging phase of the project mainly due to the inherent complexity of interstate interchanges and the large number of off-main line facilities located along interstates requiring detailed understanding and use of the RP images to ensure accurate categorizing (road type) of links, naming, and direction.. In total, 43% of off mainline features, such as ramps, connectors, and frontage roads, occur along interstates.

Immediately following the completion of IH routes on May 15th, 2015, US highway editing began. U.S. Highways make up 4800 miles (38% of total state mileage) of state roadway and include 35% of off mainline features.. Editing U.S. highways took place during peak work availability time and proceeded very rapidly. A total of six editors working throughout the summer months completed US routes on August 18th, 2015.

State highway editing began in August, 2015 and has progress to 48% completion. The state highways account for the majority of state maintained highway mileage in Wisconsin, totaling 7845 miles, or 67% of total state mileage.

Thus far, 65% of state routes have been edited in WISLR. The progress status is summarized in Table 4.1 and shown graphically in Figure 4.1.

Table 4.1 WISLR editing completion based on total mileage.

Roadway Type	Total Mileage	Edited Mileage	% Complete	Remaining Mileage
Interstate Highways	743	743	100%	0
U.S. Highways	2637	2637	100%	0
State Highways	7845	3766	48%	4079
Off-Mainline Routes	540	531	98%	9
Total	11765	7677	65%	4088

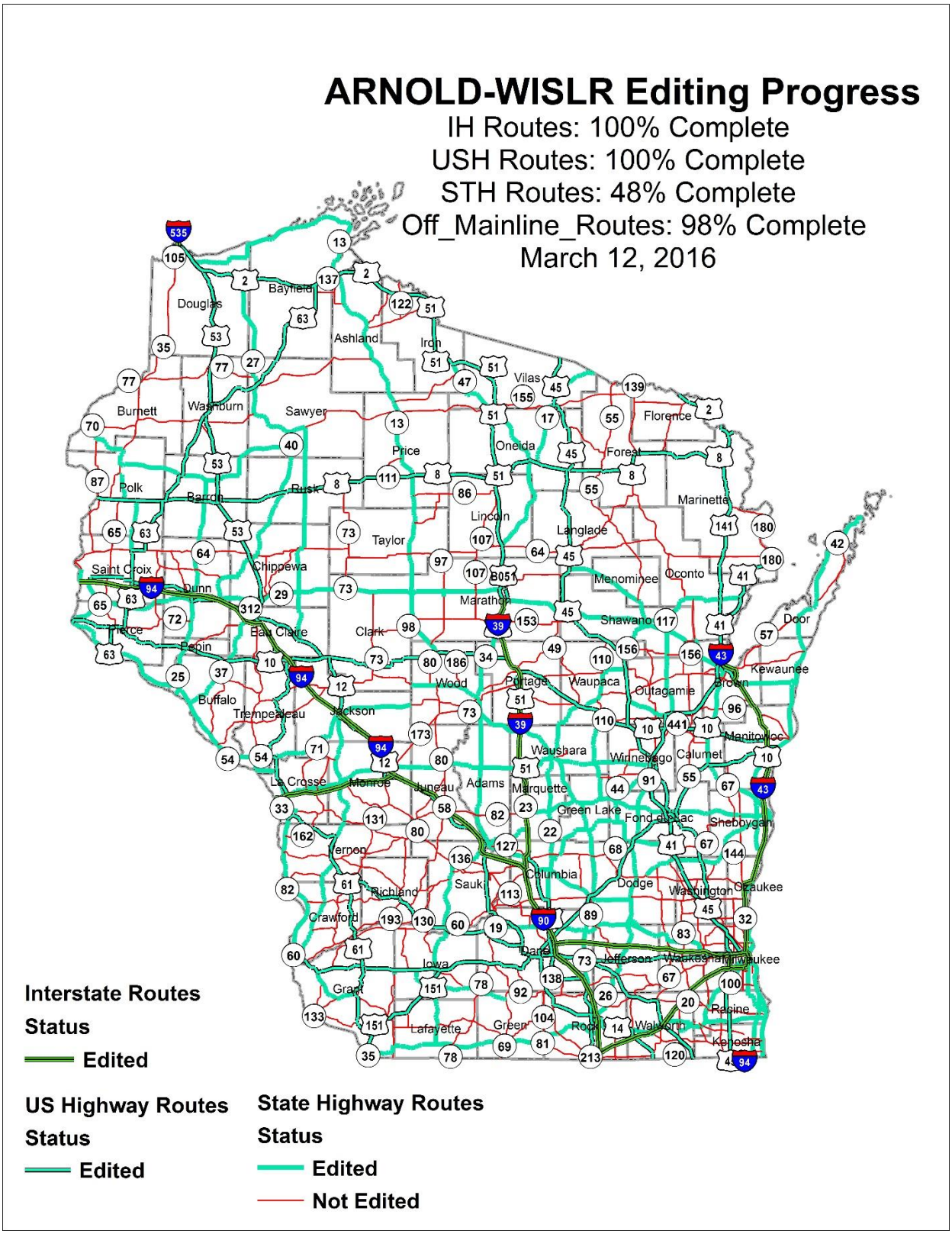


Figure 4.1 WISLR state route editing completion status at time of publication

Currently, nine editors are working on the WISLR editing project, and it is anticipated that five editors will work full time through the summer months of 2016. Projected completion of all state highway edits is September 2016. Figure 4.2 shows progress throughout the entire editing period and a projection completion date of September 2016.

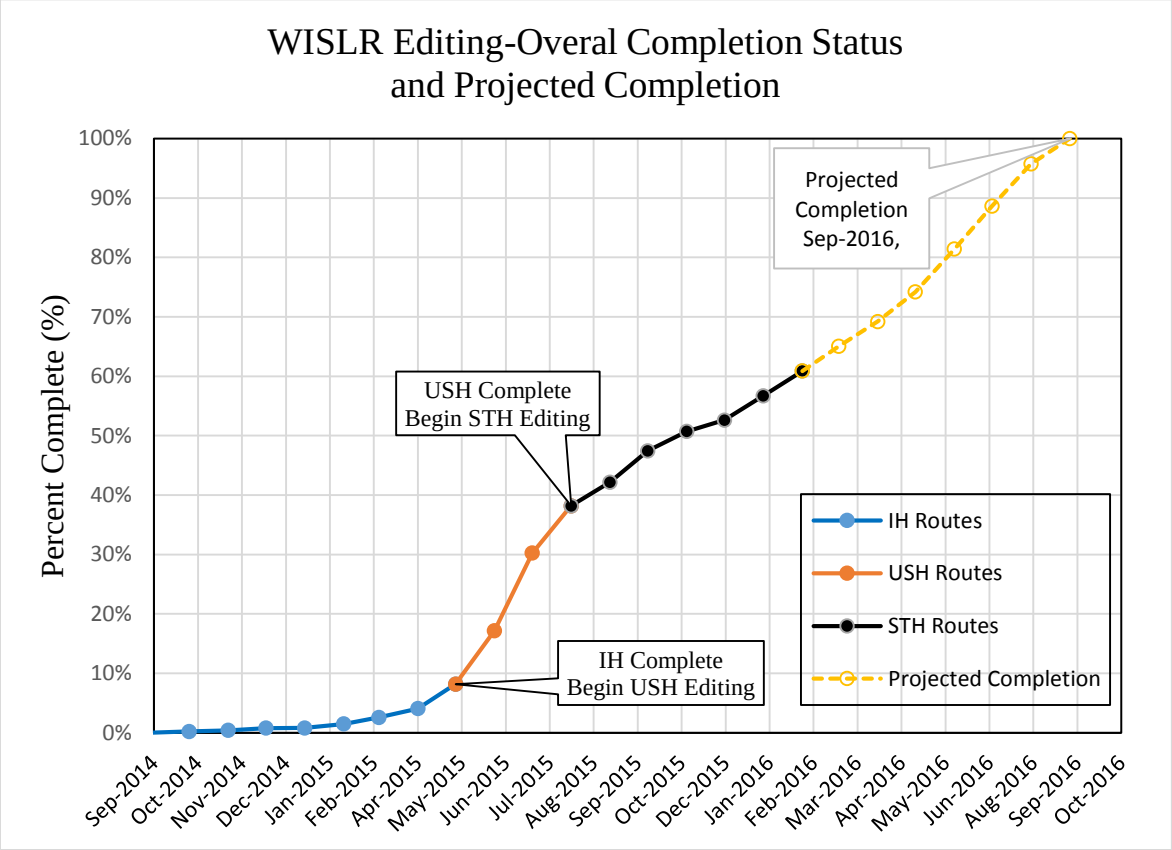


Figure 4.2 WISLR editing progress from project begin to the projected completion date of September, 2016.

4.2 Gore Point Calibration

To illustrate the effectiveness of the calibrated Link-Link table, a case study was performed on Interstate 94 in Saint Croix County. All state route line work located in Saint Croix County was edited in WISLR during a previous pilot study. Additionally, the updated line work generated during the pilot study were included in the most recent Link-Link table update.

These two elements, updated and accurate WISLR line work and a populated Link-Link table reflecting the new line work, provide the basis for calibration.

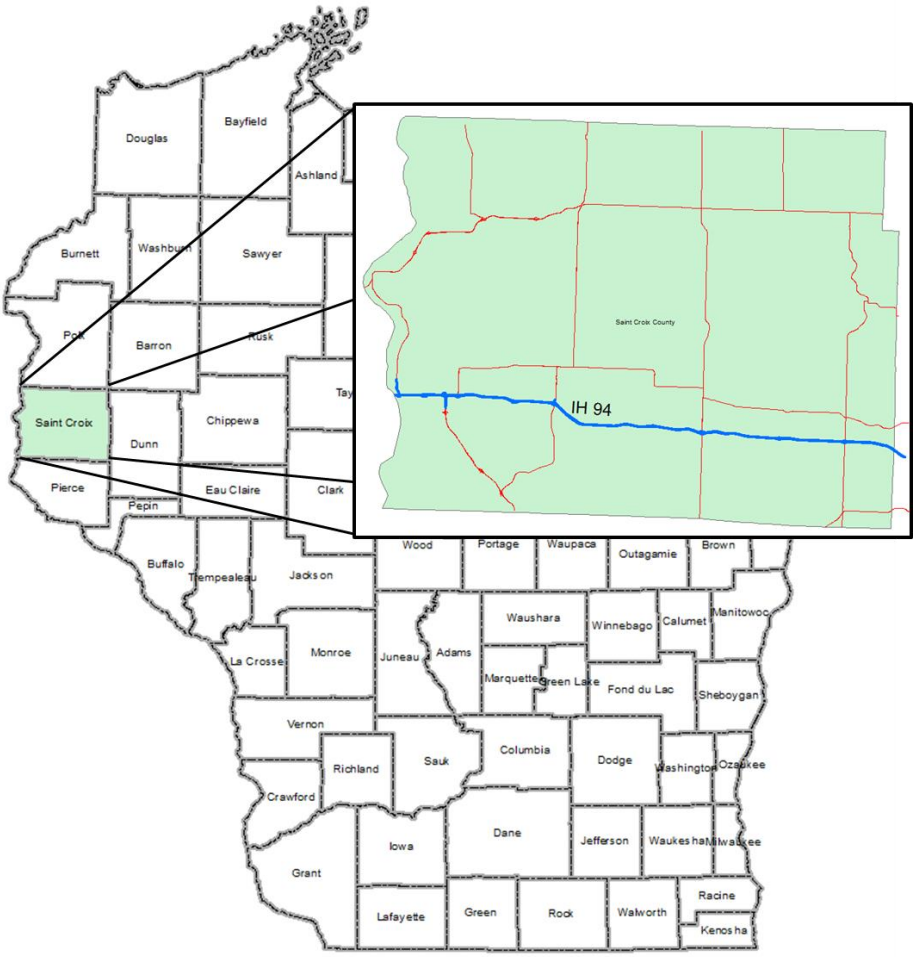


Figure 4.3 Vicinity map showing the location IH 94 used in the case study for this research.

Interstate 94 begins at the Minnesota state line and traverses the entire county, shown in Figure 4.3. This study focused only on gore points along IH 94 for two reasons: 1) IH 94 in Saint Croix county has ten exits, including two interchanges and one rest area, providing a representative sample size large enough to make several Link-Link table record changes, and 2) numerous crash data were recorded on the interstate during the time span included in the Link-Link table update, both close to gore points and scattered between gore points, allowing adequate comparison of un-calibrated data translation to a calibrated Link-Link table to be made.

The case study generated two gore point offset tables, mainline and off main line gore points, on the IH94 route, capturing 46 gore points in each. An analysis determined that the average mainline gore offset difference, 911 feet, was shown to be the greatest at on ramp gore locations, on both the mainline and off mainline links. This finding is logical considering that the acceleration lane is typically longer than the deceleration lane on a ramp. Table 4.2 shows a summary of the gore point differences. The gore point table placed gore points on 92 links, affecting 113 records in the Link-Link table. Following the calibration procedure, 209 new records were generated

Table 4.2. Summary table of gore point offset differences calculated from the gore point tables

Gore Point Location	Mean (ft)	Min (ft)	Max (ft)	Std Dev
Mainline at Off Ramp	575	13	1001	209
Mainline at On Ramp	900	110	1424	372
Off Main Line Off Ramp	603	290	994	167
Off Mainline On Ramp	921	108	1426	334
On Ramp	911			
Off Ramp	589			

To illustrate the improvement to locational accuracy data transfer gained through a calibrated Link-Link table, crash data has been translated from STN to WISLR. Crash data was obtained for the period from January 1st, 2012 through December 31st, 2013. This period was chosen because the most recent Link-Link table update was conducted to reflect LRS changes during this span. During this time period, the WISLR editing pilot study was performed and several new link relationships have been established in the Link-Link table that reflect the enhanced WISLR line work. In total 534 crashes were recorded along the Saint Croix County segment of IH 94, including ramps. Crash data was first translated first through the un-calibrated table, and next using the calibrated table. In both operations, all crashes successfully moved

from STN to WISLR. Crashes were moved through the calibrated Link-Link table, recognizing the duplicated STN link segment record, by ignoring the record in the table with Ramp_Seg identifier.

A comparison was performed that identified which WISLR link each crash translated to using the un-calibrated table versus the calibrated table. The comparison showed that 63 crashes translated to a different WISLR link following calibration. This indicates that almost 12% of the crashes in this data set occurred within the gore point offset difference area, on either the ramp or the mainline. For the 470 crashes that translated to the same WISLR link, an analysis was performed which compared the translated WISLR location of each crash through the calibrated and un-calibrated tables. The comparison showed the average change in the translated location was 444 feet, while 116 crashes moved to the same location. The box plot in Figure 4.4 shows the distribution of the translated differences.

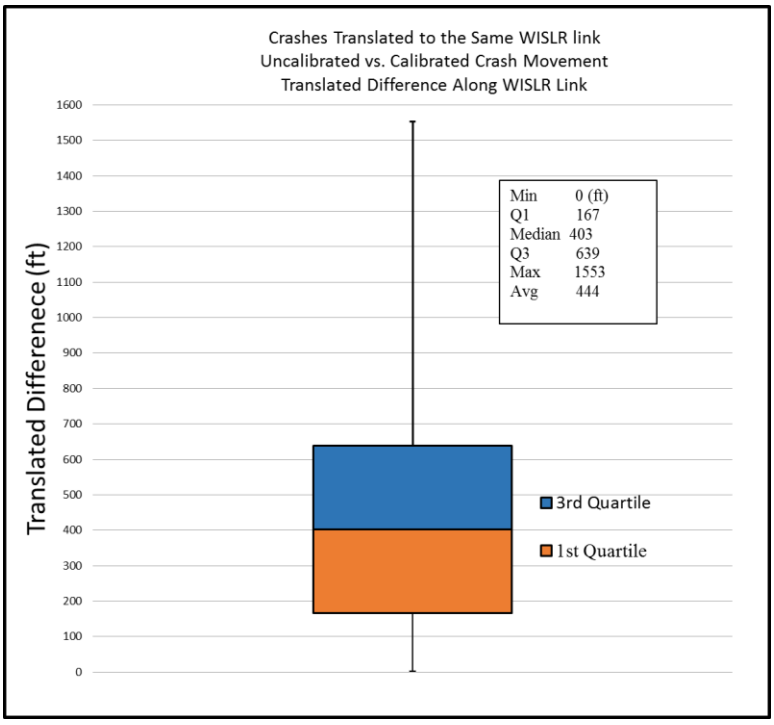


Figure 4.4 Difference in crash offset movement along WISLR length using the calibrated vs un-calibrated Link-Link table

The level effort required to calibrate the Link-Link records in this case study was found to be about fifteen hours. This amount of time was considerably more than the amount of time required to update the Link-Link table in Saint Croix County to reflect the period changes. Statewide, the current version of the Link-Link table contains 7,854 records with gore point flags. Assuming a similar amount of time (15 hours per 113 links) is required to calibrate the full Link-Link table, this technique would add approximately two and a half months additional time to the next table update, assuming five coders working twenty hours per week.

CHAPTER 5

CONCLUSION AND FUTURE WORK

5.1 Results

The results of this research show that an active Linear Referencing System (LRS) can be enhanced to match the resolution another active LRS, in spite of their differing business rules and graphical representations. The Wisconsin Information System for Local Roads, (WISLR), is used by the Wisconsin Department of Transportation (WisDOT) to manage all local roads within the state. The State Trunk Network is a second LRS managed by WisDOT and includes state routes only. The WISLR LRS is the focus of the enhancement.

A functional merge, known as the Link-Link table, relates graphical features in each LRS and provides a platform to translate data between the two. The second focus of this research shows that the enhanced WISLR line can be used to calibrate the Link-Link table to account for linear offset discrepancies that occur in areas where the link measures are different, namely at gore point locations.

5.1.1 *WISLR Enhancements*

The first objective of this research focused on editing state route line work in WISLR to match STN. This work developed standardized naming conventions so route names in WISLR are consistent with route names stored in STN. This was accomplished through the development of the naming convention guide. The guide sets the standard for detailed and consistent naming guidelines for the various state trunk network features and details the use of reference point

diagrams, maintained in STN, for attributing (categorizing, naming, and direction of travel) WISLR. The ongoing objective to edit WISLR state trunk line work has been successful.

5.1.2 *Gore Point Calibration*

A second goal of this thesis was to develop a method to calibrate the Link-Link table where significant linear offset discrepancies are exist, namely at gore points. This was accomplished by identifying the affected links in each LRS, capturing the offset value difference at gore points, and writing new records into the Link-Link to account for the difference. The results show improvements to locational translation accuracy when actual data is moved from STN to WISLR. This work proposed the addition of two columns to the Link-Link table that add allow data translation to be controlled for calibrated records.

5.2 Future Work

Fundamentally, the Link-Link must be updated to include changes that occur in either LRS. Currently, the table requires manual coding to reflect the updates. This manual coding process is based only on geometry, as no other means to relate the systems exist. The WISLR line work enhancements may provide another level of relationship embedded within the data, the route names. Programmatic techniques should be considered that will relate STN to WISLR based on route name.

The method presented in this work to calibrate the Link-Link table was accomplished manually. While the process is viable and provides greater translational accuracy, this hand coding process is inefficient and should be automated.

REFERENCES

American Association of State Highway and Transportation Officials, AASHTO. (2009). *Synthesis of State Practices in Developing Linear Referencing Systems.* NCHRP Project 08-36, Task 80.

Biswas, S. K. (2014). *Movement of Data Between Two Linear Referencing Systems of Different Resolution.* Tuscaloosa, AL: Unpublished Master of Science Degree.

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

Federal Highway Adminstration. (2001). *Implementation of GIS Based Highway Safety Analysis: Bridging the GAP*. Washington, DC: Task Report, Publication No. FHWA-RD-01-039.

FHWA. (2014). *Highway Performance Safety Manual.* Washington, DC.

GAO. (2012). *GEOSPATIAL INFORMATION OMB and Agencies Need to Make Coordination a Priority to Reduce Duplication.* A Report to the Committe on Homeland Security and Governmental Affairs, U.S. Senate, Washington, DC.

Graettinger, A. J., Morrison, A. L., Parker, S. T., Forde, S., & Qin, X. (2013). Translating Transportation Data Between Dissimilar-Resolution Linear Referencing Systems. *Journal of the Tranportation Research Record, 2399*, 103-111.

Graettinger, A. J., Qin, X., Parker, S. T., & Forde, S. (2009). Combining State Route and Local Road Linear Referencing System Information. *Journal of the Transportation Research Record, 2121*, 152-159.

Hallmark, S. L., Schuman, W. G., Kadolph, S. W., & Souleyrette, R. (2003). Integration of Spatial Point Features with Linear Referencing Methods. *Journal of the Transportation Research Record, 1836*, 102-110.

Highway Safety Improvements Program. (2012). 23 USC 148.

Morrison, A. L. (2012). *Updating and Expanding a Multi Resolution Linear Referencing System Functional Merge.* Tuscaloosa, AL: Unpublished Master of Science Degree.

OMB. (1990). *Circular No. A-16 Revised Coordination of Surveying, Mapping, and Related Spatial Data Activities.* Washington, DC.

OMB. (2010). Geospatial Line of Business OMB Circular A-16 Supplemental Guidance. Washington, DC: U.S. Government Printing Office.

Ryals, Z. T. (2011). *A technique for Merging State and Non-state Linear Referencing Systems.* The University of Alabama. Tuscaloosa, AL: Unpublished Master of Science Degree.

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

Vonderhohe, A., & Hepworth, T. (1998). *A methodology for design of linear referencing system for surface transportation, final report.* Sandia National Laboratories.

Vonderhohe, A., Chou, C., Sun, F., & Adams, T. (1997). A Generic Data Model for Linear Referencing Systems. *NCHRP Research Results Digest 218*.

Winter, D. R., & Cheatham, J. (2012). *Geospatial Network for All Public Roads.* Retrieved 2015, from https://www.fhwa.dot.gov/policyinformation/hpms/arnold.pdf

Wisconsin Department of Transportation. (1998). *Wisconsin Location Control Management Manual*.

APPENDIX A-LINK-LINK TABLE POPULATION PROCEDURE

The document contained in this appendix is titled Link-Link Table Update Procedure Manual. The manual is a compilation of work related to previous research (Biswas, 2014, Morrison, 2012,Ryals, 2011). It is intended to serve as a step-by-step user's guide which lists all steps and processes necessary to produce an updated Link-Link table for the STN and WISLR LRSs. Two programs in this guide, Date Population and Date Check, have been rewritten in C#.

CONTENTS

I. Link-Link Table Update Procedure Manual

The following manual provides all steps necessary to update the Link-Link table.

1.1. Principle Phases

II. Data Acquisition

III. Data Preparation

IV. Table Update and Population

V. Quality Assurance/Quality Control: Per County

VI. County Tables Merge

VII. Statewide Quality Assurance/Quality Control

VIII. Mapping Crashes

IX. Finalize Table for Submission

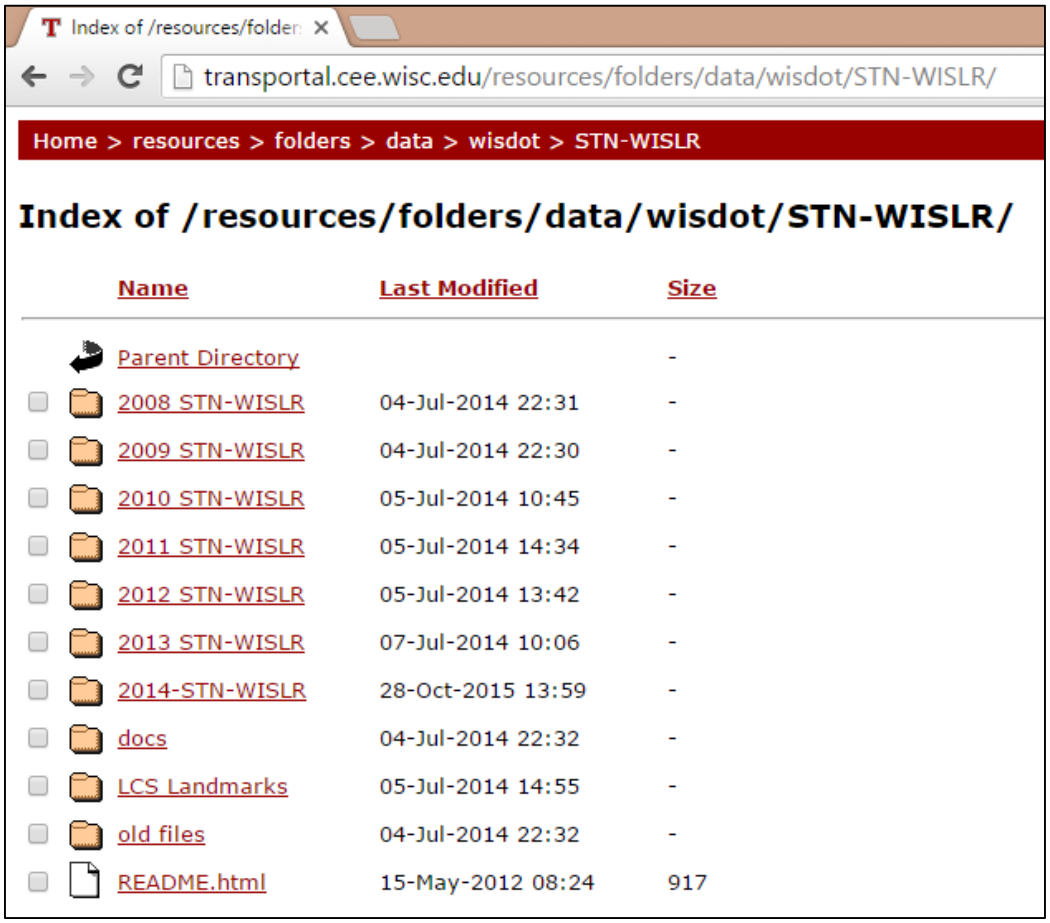
X. Appendix: Programs Used

II. Data Acquisition

To begin a Link-Link table update, data must be obtained. Wisconsin Traffic Operations and Safety Laboratory (TOPS) hosts the website The WisTransportal System which serves as a data repository for the required data tables and shape files needed to perform an update. The website is password protected, please contact the TOPS IT Program Manager Dr. Steven Parker to obtain credentials at http://transportal.cee.wisc.edu/contact.html.

2.1. Required Data

The data needed for an update include data tables and shape files. All files should be accessed and downloaded from the WisTransportal System. The image below shows the file path taken to obtain the data.



Download two sets of data, one set for the certification year update and a second set for the previous update year. For example, if you are performing a Link-Link update to include changes to the LRSs that occurred during the 2014 certification year, you must obtain the 2014 data and the 2013 data set. In this example, the 2014 data is called “new” and the 2013 data set is called “old”.

After selecting a folder in the STN-WISLR directory, the following files appear:

To download STN data:

- Select STN stoclcs-contains all required shape data.
- Select STN LCS Tables and STN LCS General-contain all data tables.

To download WISLR data:

- Select WISLR ArcSDE-contains all required shape data.
- Select WISLR Oracle-contains all data tables.

2.1.1. Refined date tables

The refined tables provide a complete list of link IDs and the date which the link became current and the date which the link became historic. The refined table is generated through a query that returns the earliest valid current date and the earliest valid historic date by examining dates in the route-link tables. Obtain two "Refined Date" tables (one for STN, one for WISLR) from TOPS by contacting Dr. Steven Parker.

2.1.2. Link-Link table

The most recent complete Link-Link table must be located and used as the starting point for the current update. The most recent complete Link-Link table can be obtained from the

WisTransportal site, by contacting Dr. Steven Parker, or from the current UA principal investigator.

III. Data Preparation

Data preparation is divided into three steps:

1. Generate current link data set
2. Identify changes to the LRS during the update period.
3. Create a map package.

3.1. Generate Current Link Data Set

In order to identify changes that occurred during a specified update period, a current link data set must be generated. For example, if you are creating a 2014 Link-Link table, you want a list of links that are current as of 12/31/2014. This is most easily accomplished using the refined tables.

1. From refined table, in column "Min_DT_RTE_LINK_CURR" select records with dates earlier than the last day of the update year.

   For example: if updating Link-Link table through 2014, then select record with dates earlier than 1/01/2015.

2. From refined table, in column "MAX_LCM_DT_HSTL", select records with dates that are null, newer or equal to the dates beyond when the update will be performed.

   For example: if updating the Link-Link table for data through 2014, then select records with dates that are null, or 01/01/2015 or later.

3.2. Generate Current Link Coverage (shape) File

Create a shape file for STN and WISLR containing only current links using the current link list generated in Section 3.1. The current link coverage file will be used to identify changes in both LRS through a series of joins.

1. For STN: locate shape file, stoclcs\rwdy_link. Join column "RDWY_LINK" in shape file with "RDWY_LINK_ID" in refined table. Export joined records to new shape file. Include Link_ID, From Site ID, To Site ID, Display Name, and the date fields from the refined tables..

2. For WISLR: locate shape file, WISLR_ArcSDE.mdb\DT_RDWY_LINK_LINE. Join columns in the same manner as the STN join. Export joined data in the same manner as the STN export.

3.3. Generate Current Reference Site Table

Reference sites identify the end points of links. Sites are used to determine direction that links travel. This is essential when populating a Link-Link record. To generate a current Reference Site table, do the following:

1. For STN: locate shape file stoclcs\ref_site. Filter dates in the column, LCM_DT_CURR to exclude any date newer than the last day of the year being updated. In column, LCM_DT_HSTL, keep only records with dates newer than the last day of the year being updated. Export selected records to new shape file called "STN_Current_Ref_Sites.

2. For WISLR: locate shape file WISLR_ArcSDE.mdb\DT_REF_SITE_PT. Filter dates, in column LCM_CURR_DT, to exclude any date newer than the last day of the year being updated. In column LCM_HSTL_DT, keep only dates newer than the last day of the year being updated. Export to new shape file called "WISLR_Current_Ref_Sites".

3.4. Identify Changes to Data

Using ESRI ArcMap, changes that occur to STN and WISLR during the update can be identified. Two joins must be performed in "opposite" directions so that links that have been added and links that have been deleted can be identified.

1. Join "old" line work data (data used in updating the most recent Link-Link table) to current line work data (data processed in section 3.2) based on link ID.

   For example: if updating the Link-Link table through 2014, join 2013 rdwy_link shape file to the processed current rdwy_link shape file from Section 3.2. Follow similar path for WISLR as outlined in Section 3.2

   a. Select the records that do not join. This identifies new records, i.e., records that have been added since the "old" data was published.

   b. Export selected records to a new shape file "NEW (STN or WISLR) Links".

2. Join current line work data to "old" based on link ID.

   a. Select records that do not join. This identifies deleted records, i.e., records that have been deleted since the "old" data was published.

   b. Export selected records to a new shape file "Deleted (STN or WISLR) Links.

3.4.1. Access points

Access points are used extensively throughout the Link-Link table coding process as a method to relate an STN offset measure to a full WISLR link length. Access points are continually updated in the data set by WisDOT. Access points are either deleted, edited or added to the data as needed and do not contain date information. Because no date information exists, changes to

access points must be identified using a different approach. The following steps should be used to identify STN links with changes to access points.

1. Create a "Concat" column within both the "new" and "old" access point shape files (stoclcs\LCS_General.mdb\DV_ACSI_PT).

2. Populate this field with a concatenation of the link ID (RWLK_ID), the access point # (ASCI_INTS), and its offset value (ASCI_PT1).

3. Join "new" to "old" and "old" to "new" in a similar fashion as Section 3.4. Again, select un-joined records in each step. Combine both tables to produce a full list of links affected by access point changes (deleted and new). Export this as a .dbf file named "AP_Affected_STN_Links"

4. Join the AP_Affected_STN_Links.dbf table to the complete STN link shape file based on link ID, keeping only joined records. Export joined records into new shape file named "AP_Affected STN_Links_year".

A helpful hint to identify access points along an STN link is to locate the access point on the STN link. This is accomplished by first creating a route using the Create Route tool in ArcMap. After creating the route, the access point in the table generated earlier in this section can be placed on the STN route by using the Locate Features Along Route tool.

3.4.2. Buffering WISLR links

The WISLR data represents all roads throughout the state. It is not necessary, nor is it desirable, to retain the entire data set during the update since the Link-Link table only relates WISLR links that describe state routes to the corresponding STN links. The WISLR shape file should be reduced by buffering the WISLR links along the STN chain shape file (stoclcs\stn_chain). The buffering process should be performed for both the New and Deleted WISLR Link shape file created in Section 3.4. In general, the buffer process should extend at least 500m from the stn_chain features to ensure that all state WISLR links are captured. The buffered shape files should be exported as "WISLR (NEW or OLD) Links Buffered".

3.5. Creating a Map Package

Map packages make it easy to share complete map documents with others. A map package contains all the data referenced by the layers it contains, packaged into one portable file. This is useful because it ensures that all preprocessed data and shape files for the entire state are available to each Link-Link coder every time a new county is readied for update. The detailed steps for creating a map package can be found under ArcGIS Help.

The relevant shape files needed are:

1. STN_rdwy_links-this should be the entire link shape data and is used as a reference only.

2. STN AP_Affected_Links_Scaled_Down-this shape file should have the access points located along the link.
3. STN_Current_Sites
4. STN_Current_Rdwy_Links
5. STN_New_Links
6. AP_Current_on_STN_Links
7. STN_Relevent_Deleted_Links
8. WISLR_Current_Rdwy_Links_Buffered
9. WISLR_New_Links_Buffered
10. WISLR_Deleted_Links_Buffered
11. WISLR DT_ST_PRTE_OVLY_LINE-contains additional data including road names.
12. WISLR_Sites_Buffered_to_STN_Chains
13. Statewide_County_Outlines
14. Aerial image- world imagery can be loaded from a base map or by using the GIS Server

End of Phase: Produces Statewide Map Package to be used for the update process.

IV. Table Update and Population

The following sections describe the process to update the Link-Link Table on a county by county basis.

4.1. Creating County Maps:

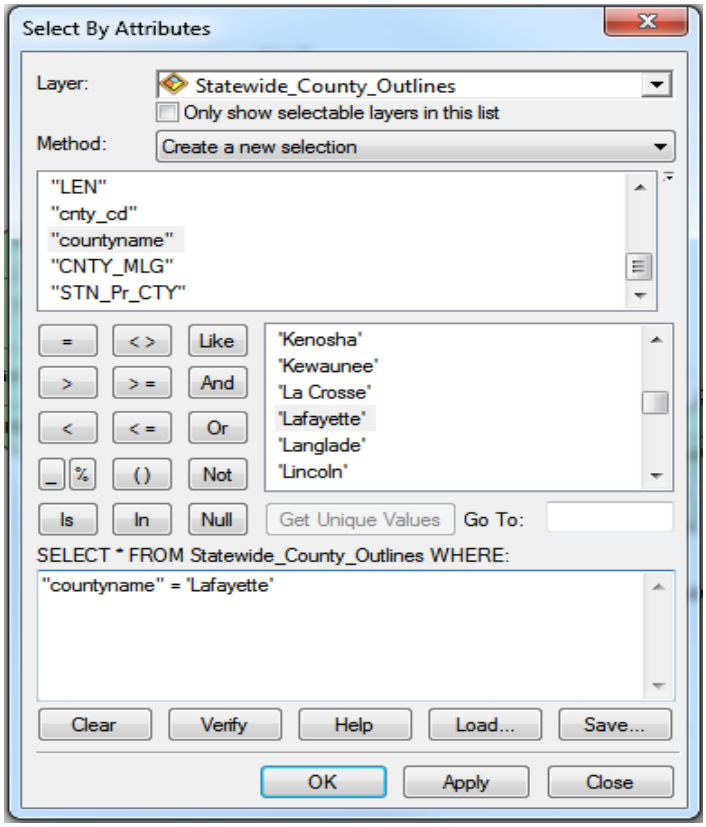
The following process will create a map of an individual county containing all link shapefiles necessary to update the Link-Link Table. These county maps contain visual representations of the changes to be made in the Table.

1. Create a Folder with the County Name of the County you are going to build.

2. Open the State_Wide_Map Package.

3. Click Selection on the Tool bar and choose select by attributes.

4. On the Select By Attributes table pop-up as shown below:
    a. Change layer to Statewide_County_Outlines.
    b. Create a query that selects only the specific county that you are creating.
        - EX: “countyname” = ‘Lafayette’
    c. Click apply. The county outline should now be highlighted on the state wide map.



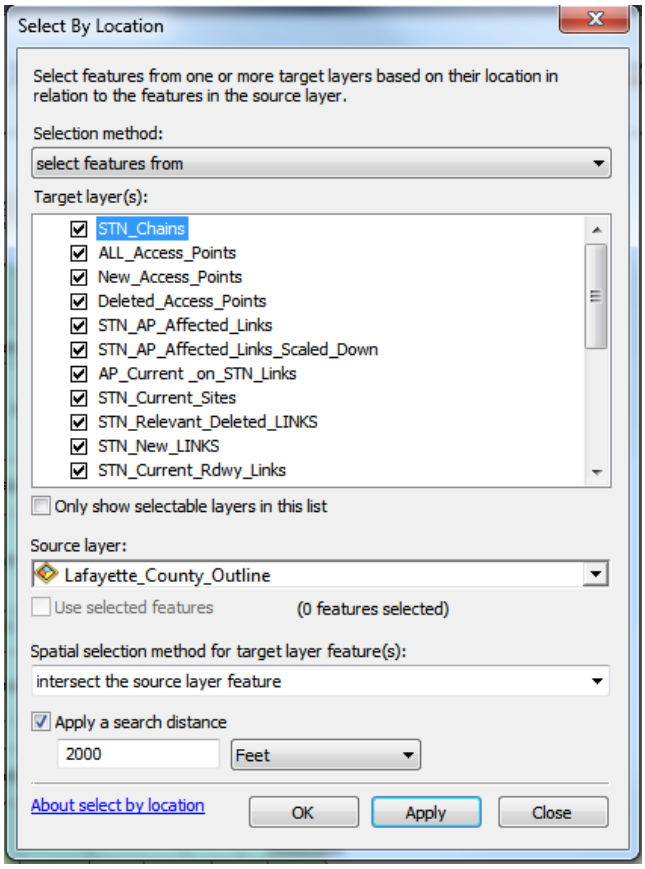
5. Export the selected feature to a new shape file.
    a. Right click on “Statewide_County_Outlines”
    b. Select Data.
    c. Click Export Data.
    d. Export: Selected features. Use the same coordinate system as: this layer’s source data.
    e. Output feature class:
        i. Select county folder with the appropriate name.
        ii. Save shape file as Countyname_County_Outline and add to current map.
            a. Ex: Lafayette_County_Outline

6. Under the Selection tab on the toolbar, choose “Clear Selected Features”.

7. Click Selection on the tool bar and choose select by location.

8. On the Select By Location pop-up:
    a. Select features from all target layers except for county outline and state outline.

b. Check to make sure that the source layer says your specific county outline (the shapefile that was just added to the map).
c. Apply a search distance of 2000 ft.
d. Hit okay.



9. Export data from each layer individually using the exact naming convention as used in the original.
    a. Right click on layer to export.
    b. Select Data.
    c. Click Export Data.
    d. Export: selected features.
    e. Output feature class:
        i. Select county folder with the appropriate name.
        ii. Save shape file EXACTLY as named on the original state map.
        iii. Do not add to current map.
        iv. Repeat this process for each shape file other than County_Outline and Statewide_County_Outlines and World Imagery.

10. Open a new ArcMap document and pull in each shape file from the county folder that was just created.

a. Pull in all relevant shape files needed for the editing process:
    - AP_Current_on_STN_Link
    - STN_Current_Sites
    - WISLR_Sites_Buffered_to_STN_Chains
    - STN_New_Links
    - STN_Relevant_Deleted_Links
    - STN_AP_Affected_Links_Scaled_Down
    - STN_Current_Rdwy_Links
    - WISLR_New_Links_Buffered
    - WISLR_Deleted_Links_Buffered
    - WISLR_Current_Rdwy_Links_Buffered
    - "Countyname"_County_Outline
b. Save new ArcMap document as "County_Map" in county folder.

11. The County Map is now ready to be used for the update process.

4.2. Creating a Corresponding County Link-Link table

The following describes the process to create a Link-Link table containing only the records within and surrounding the county being edited.

1. Open most recent complete statewide Link-Link Table in Excel.

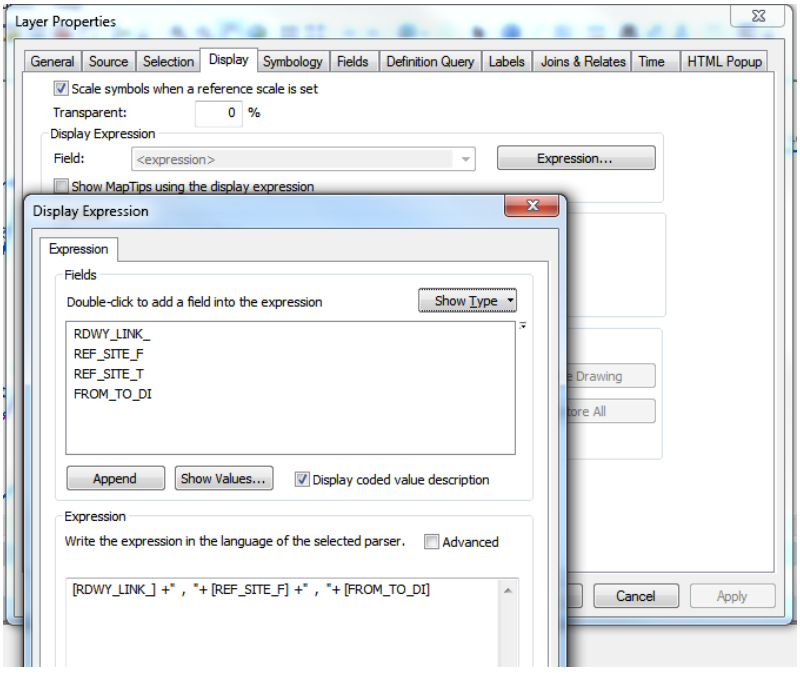
2. In Excel select the "Filter" feature under the Home tab for the header row.
    a. Under "County" column select only the county to be edited and all counties surrounding the borders.
    b. Select only these records from the table.

3. Copy and paste these entries into a blank excel Table renamed "Countyname Link-Link".
    a. Separate these entries in the new table by repeating the "Filter" feature described above.
        i. Select the counties that border the edit county on the South and West sides.
            1. Move these records to a new sheet in the Table; rename sheet accordingly "SW border".
        ii. Select the counties that border the edit county on the North and East sides.
            1. Move these records to a separate sheet in the Table; rename sheet accordingly, "NE border".
        iii. Remaining records in first sheet should only be those lying in the county being edited; rename accordingly, "Countyname".
    b. This is to be used when doing QAQC checks to see the links that are flagged as issues are actually in border links coded to the surrounding counties.

4. The county Link-Link table is now prepared to be edited as part of the update process.

4.3. Update Process

1. Using ArcMap, edit the new county map package so as to make the coding process easier.
   The following settings are recommended:
   a. To edit each shapefile as suggested above, first right click on the shapefile, and select "Properties."
      i. Under the "Display" tab, in the Display Expression section, select "Expression".
         1. For route links fill out the expression to display the Link ID, the Reference Site From, and the Offset Distance as shown in the figure below.
         2. For access points display the Link ID, the Intersection Name, and the Offset Distance.
         3. For route events display the Reference Site number.



      ii. Under the "Labels" tab, change label field expression to read link or site IDs, and route or intersection names to help when visualizing the roadways in ArcMap.
         1. Change the colors for each label to reflect the respective color of the shapefiles themselves.
         2. Hit OK.
      iii. Right click shapefile and select "Label Features" to turn on these labels.
   b. Edit each shapefile to a unique identifying color to make for easier visualization when using multiple shapefiles at once.

i. Click on each shapefile's symbol to pull up the "Symbol Selector".
    1. Change symbol to "Arrow at End" for route links to display travel direction.
    2. Change symbol to unique shapes for access point shapefile to separate it from other reference points.
    3. Assign unique colors to each shapefile.
    4. Hit OK.

2. Proceed to update the county using the process in order as described below.

4.3.1. Deleted Links

Remove the link sections that have become historic, or 'deleted', in the STN and WISLR networks first.

3. Create a copy of the county Link-Link table (excluding the bordering county link tabs in the table) and save as a comma delimited, .csv file in the "County Map" folder as "county.csv".
    a. Add copy of county Link-Link to County Map in ArcGIS.

4. Join county Link-Link table with STN_Relevant_Deleted_Links to isolate the records in the shapefile which are existent in county Link-Link Table.
    a. Right click on STN_Relevant_Deleted_Links.
        1. Based on: Rdwy_Link_
        2. From: STN_Relevant_Deleted_Links
        3. Attribute: STNid
        4. Keep only matching records
    ii. This will result in a shapefile that contains only the roadway links that are in the Link-Link Table to be deleted. (It will exclude the deleted roadway links that are outside of the Link-Link Table table). Also optional to export this list as a new shapefile in the county map folder.
    b. Use this list of STN link IDs to find each corresponding record in the county Link-Link table. (May help to sort both the list and county Link-Link table in ascending order based on WISLR link ids). Also optional to export this.

5. In the county Link-Link table in excel, as each corresponding record(s) is found, insert the day's date into the "Record_Historic" column.

6. Repeat this process for the WISLR deleted link sections.
    a. Remove the join from the county Link-Link table with the STN deleted shapefile.
    b. Join county Link-Link table with WISLR_Deleted_Links_Buffered.
        i. Right click on WISLR_Deleted_Links_Buffered.

1. Based on: Rdwy_Link_
2. From: WISLR_Deleted_Links_Buffered
3. Attribute: WISLRid
4. Keep only matching records

ii. This will result in a shapefile that contains only the roadway links that are in the Link-Link Table to be deleted. (It will exclude the deleted roadway links that are outside of the Link-Link Table table).

c. Use this list of WISLR link IDs to find each corresponding record in the county Link-Link table. (May help to sort both the list and county Link-Link table in ascending order based on WISLR link ids).

7. In the county Link-Link table in excel, as each corresponding record(s) is found, insert the day's date into the "Record_Historic" column. This record is now considered to be historic.

8. The county Link-Link table should now be up to date on the deleted link sections and their corresponding historic records.

4.3.2. New Links

Continue the update process by analyzing the links that have been newly added. Update the table for both new STN, and new WISLR links simultaneously by following the roadway and making all appropriate changes along the way. In this way, in areas where there are both new STN and new WISLR links all updates can be made at one time. This will increase the efficiency of the update process.

A Link-Link Coding Tutorial PowerPoint can be found in the Appendix to provide supplemental information for the coding process.

9. Remove all previous joins and turn off all deleted link shapefiles.

10. Turn on the following layers:
    1. STN_New_Links
    2. WISLR_New_Links

ii. Zoom out to include the entire county outline to visualize all the addition changes needing to be made in the Table for that county.

iii. When beginning to edit, zoom in to any particular area to start making changes in the Table.
    1. Follow along this roadway section till all new links in the area are added to the Link-Link table then zoom back out and find another area of new links within the county.
    2. Repeat this process until it is ensured that all new STN links have been added.

3. The WISLR_New_Links shapefile will contain links that are local and do not follow along the STN network and therefore should be neglected. **<u>However,</u>** do an additional visual check on all these links to ensure that no new WISLR link has been missed.

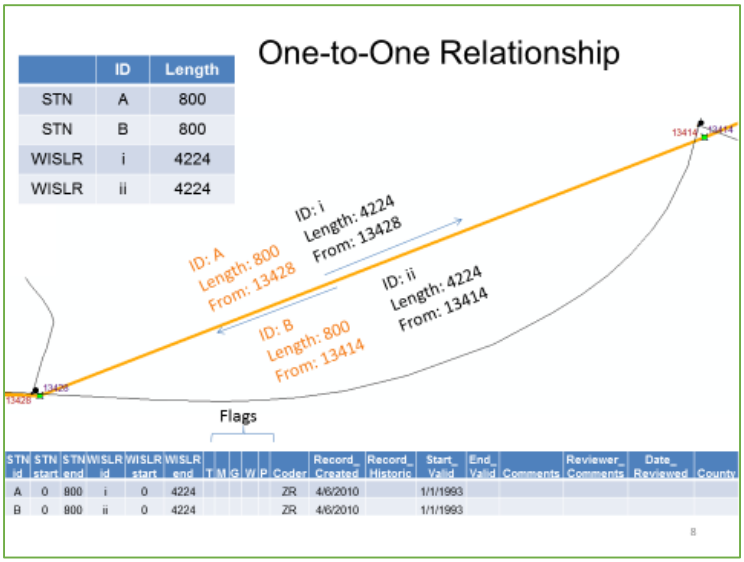
11. Once focused on a particular roadway section, turn on remaining shapefiles needed:
    1. STN_Current_Rdwy_Links
    2. WISLR_Current_Rdwy_Links_Buffered
    3. STN_Current_Sites
    4. WISLR_Sites_Buffered_to_STN_Links
    5. AP_Current_on_STN_Links

    ii. The STN/WISLR current shapefiles provide the corresponding WISLR/STN roadway section to code the new link to.

    iii. The STN and WISLR sites correspond to the links to provide what direction each link is traveling in.

    iv. The access points (AP) are added to provide accurate driven distances to use for the offset distance when coding multiple new links to a single link on the opposing network.

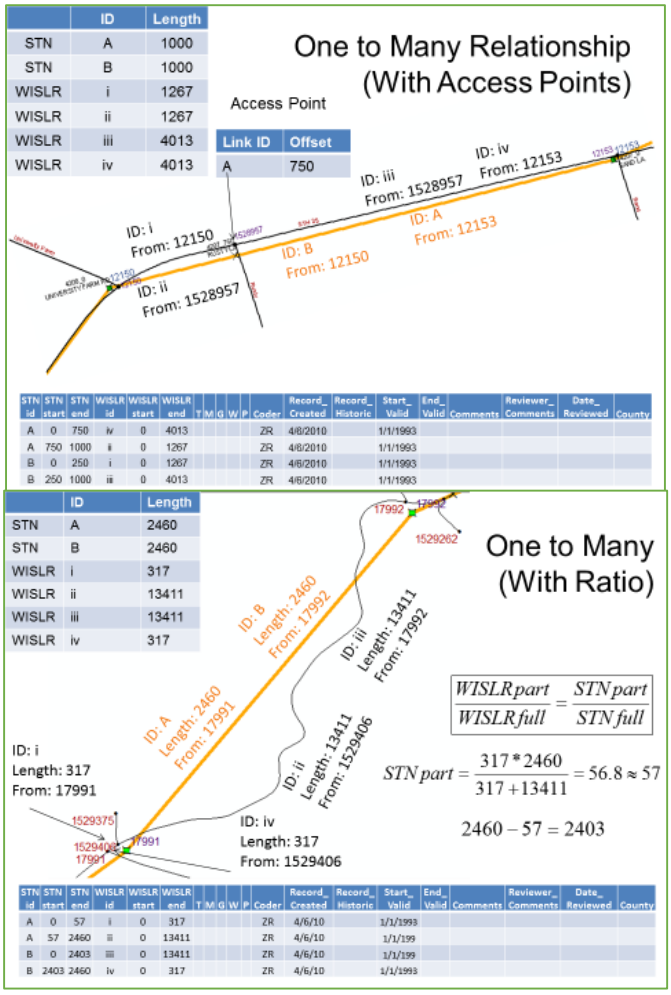
12. To add a record for a corresponding new link, take the county Link-Link table in excel and add a new row.
    a. For new links having a one to one relationship (as in one STN link to one WISLR link) fill out the new row in excel as shown below:
        1. Make note of which direction each STN and WISLR link are traveling in.
        2. Code the new STN/WISLR full length to the corresponding existing WISLR/STN full length (respectively).
        3. Add your, the coders, initials under the “Coder” column.
        4. Add the day’s date in the “Record Created” column.



b. For new links having a one to many relationship (as in one STN link to many WISLR links, or vice versa) fill out the new row in excel as shown below:
    1. Make note of which direction each STN and WISLR link are traveling and be sure to code the links traveling in the same direction together.
    2. **MUST** use the AP offset distance when available as the STN (or WISLR) break length when coding to multiple WISLR's (or STN's).
    3. If no APs are available, use the following ratio for the break lengths:

$$\frac{WISLR\ part}{WISLR\ full} = \frac{STN\ part}{STN\ full}$$

    4. Add the coder's initial under the "Coder" column.
    5. Add the day's date under the "Record Created" column.



13. For roadways or intersections that exhibit unique attributes on one network but are not represented on the opposite network as series of Flag Columns are used to identify them in the Link-Link table. The following table below shows the 5 problem columns used in the Link-Link table.

Flag Columns in the Link_link Table

New Column Description	Flag Column Label	Value
Turn-Lane Overlap	T	1
Median Cross-Over	M	ID or 1
Gore Point	G	T, F, or B
Weigh Stations, Park & Rides	W	1
Problem	P	1

a. Follow the Link-Link Coding Tutorial when coding any links exhibiting these qualities. The tutorial (provided in the Appendix) contains visual examples of both the roadway intersections and the Link-Link table to assist with the coding process.

b. For any link sections that were not mentioned in the tutorial but contain issues preventing proper coding, place a "1" in the problem column, "P" and add a corresponding comment in the "Comment" column. The guidelines for these comments can be found in the tutorial in the Appendix.
    i. Common problem not mentioned in the tutorial is the instance where a link in one of the two systems is represented in the wrong direction.
        1. Code the links as if both were traveling in same direction, mark a 1 in the problem flag column and comment accordingly.

14. For Roundabouts: (newly added roundabouts are represented differently on STN and WISLR)
    a. Where sections of the roundabout are represented in STN and <u>not</u> WISLR,
        i. Flag and code that link section as a Median Crossover.
    b. For all links in roundabout:
        i. Place a "1" in the "P" column for all newly added roundabouts and add "Roundabout" in the "Comment" column.
15. Once the particular roadway section is complete, continue to update for new links by zooming back out and find another area of new links within the county.
    a. Repeat this process until it is ensured that all new STN links have been added.
    b. The WISLR_New_Links shapefile will contain links that are local and do not follow along the STN network and therefore should be neglected. **<u>However,</u>** do an additional visual check on all these links to ensure that no new WISLR links that need to be coded have been missed.

4.3.3. Access Point Affected Links

Finish the update process by analyzing the STN roadway links that have had changes made to the Access Points (APs) which lie on those links. These changes affect the coding lengths already present in the table for STN to WISLR and subsequently need to be changed and updated.

16. Before beginning, make sure the following shapefiles are turned on:
    1. ST_AP_Affected_Links_Scaled_Down
    2. AP_Current_on_STN_Links
    3. STN_New_Links
    4. WISLR_New_Links
    5. WISLR_Current_Links
    6. STN_Current_Sites
    7. WISLR_Current_Sites (optional)
    a. As before, zoom out to entire county outline and visualize all AP affected links in the file.
17. Pinpoint a roadway section to be edited and zoom in.
18. Manually look at each link and follow along the roadway section:

a. If AP Affected STN link is also layered with a new STN Or a new WISLR link:
    i. Ignore the link, the AP change has already been accounted for.
b. If no new links are also present:
    i. Find the corresponding record(s) in the county Link-Link table in excel.
        1. Verify that the STN length in the Link-Link table is NOT the same as the offset distance provided by the AP_Current_on_STN_Links shapefile.
            a. If STN length = AP offset, do not make any changes to the table.
            b. If STN length is different from the AP offset,
                i. Mark existing record(s) in Link-Link table as historic by placing the day's date in the "Record Historic" column and add "AP Affected Link" in the "Comment" column.
                ii. Create a new record(s) almost identical to the previously existing record but replace the STN length with the new AP offset distance.

19. Repeat this process until all STN AP Affected links have been accounted for and edited appropriately.

V. County Quality Assurance and Quality Control

The following sections describe the steps to be taken after each county is fully updated as described above; a quality assurance and quality check is to be performed. This process ensures that any mistakes made in the editing process are corrected before moving on to update the next county.

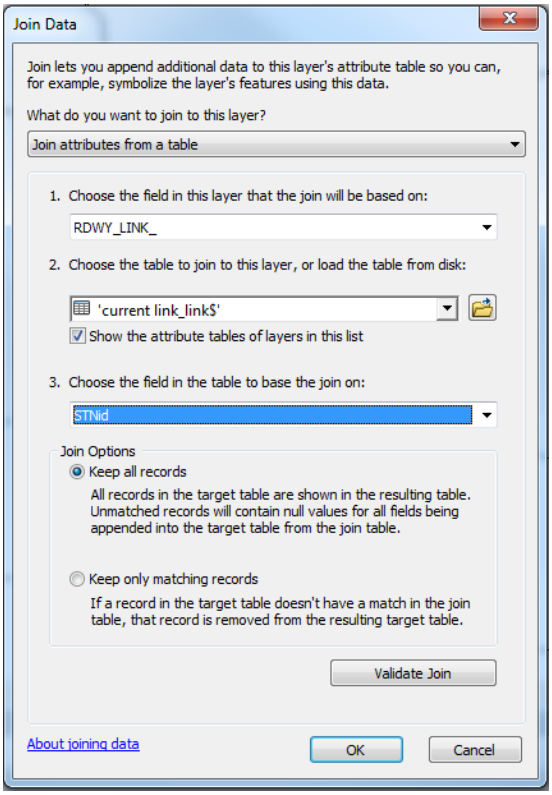
- Before beginning the QAQC procedure for the county, create a new folder within this county folder called "County QA_QC". This will allow you to export shapefiles and other tables to this folder later in the process.

5.1. STN Link Check

The following procedure describes the steps needed to ensure that all current STN roadway links are present in the county Link-Link table, and that there are no links present in the county table that are not existent in the current STN roadway links shapefile.

1. Open the county Link-Link table in excel that has been edited.
    a. Sort the "Historic" field so that only current records are showing
    b. Copy the entries without a historic date to a new sheet to create a current Link-Link table for the county.
2. Open the county map in ArcMap and add data to the county map using the yellow folder with a black plus sign on the toolbar.
    a. Select and add the current Link-Link table from the county folder.
3. Turn on the STN current roadway shapefile for the visual check.
4. Join the current Link-Link table with the STN current roadway shapefile and look for NULL values in STNid column
        i. Right click on STN_Current_Rdwy_Links shapefile and select "Joins and Relates"
        ii. Join current table with STN current rdwy links
            1. Based on: Rdwy_link_
            2. From table: "current Link-Link"
            3. Attribute: STNid
            4. Keep all records
        iii. Export joined shapefile into "County QAQC" folder for future reference if needed.

It should look like:



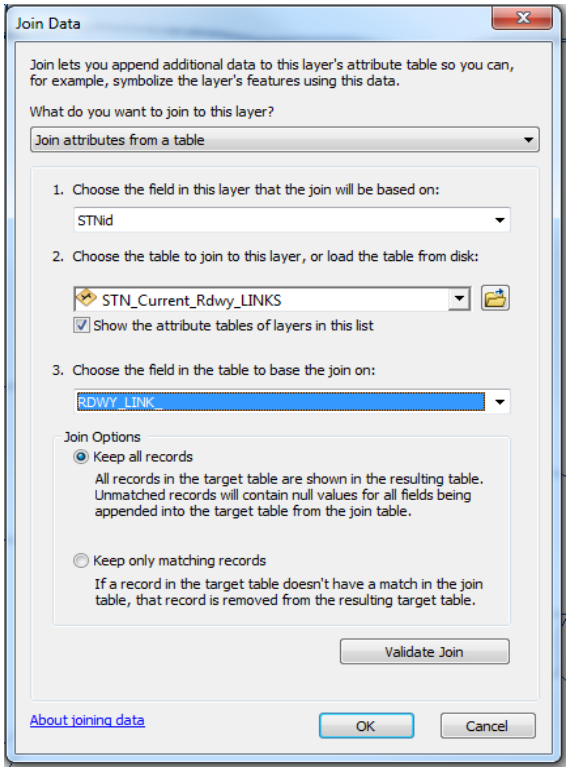
5. Select any values returned with "<NULL>" in the STNid column. This will return link IDs that are in the current STN shapefile but NOT in the county Link-Link table.
    a. Open the attribute table for STN_Current_Rdwy_Links.
        i. Choose "Select by Attributes" option on the toolbar.
            1. Where: "STNid IS <Null>"
            2. This will highlight any links in the shapefile that are missing from the table.
    b. For each <Null> value, visually check to identify the issue and fix any missing links accordingly in the county Link-Link table.
        i. For the missing links that lie on the north and east county borders use the "NE border" tab in the county Link-Link table to search and find the missing records.
            1. Check these links to ensure that no deleted or new links were also missed.
        ii. For the missing links that lie on the south and west borders it is not necessary to add them to the table as they will be included in those neighboring counties.

iii. For any other remaining missing links, correct the Link-Link table accordingly.

6. Remove the join by right clicking on the STN_Current_Rdwy_Links shapefile and selecting "Remove Join".

7. Next, join the STN_Current_Rdwy_LINKS shapefile to the current link_link table.
   a. Right click on the current link_link table and click join.
      1. Based on: STNid
      2. From: STN_Current_Rdwy_Links.shp
      3. Attribute: Rdwy_Link_
      4. Keep all records
   b. Export the new table into the "County QAQC" folder.

   The join should look like:



8. Any values that returned <Null> in Rdwy_Link_ need to be checked. This returns all links that are present in the county Link-Link table but are not present in the STN current roadway shapefile.
   a. Open joined table in ArcMap.
      i. Choose "Select by Attributes" on the table's toolbar:

1. Where: “Rdwy_Link_” IS <Null>

b. Visually check each flagged IDs in the table and fix the table accordingly.
   i. Null values generally resulted from typos and from links in bordering counties.

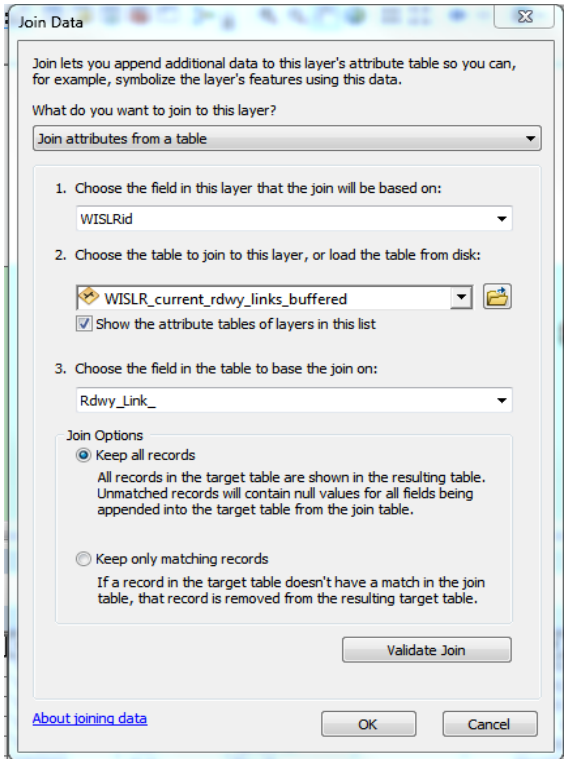
9. Remove the join by right clicking on the county current Link-Link table and selecting “Remove Join”.

5.2. WISLR Link Check

The following procedure describes the steps needed to ensure that there are no WISLR links present in the county table that are not existent in the current WISLR roadway links shapefile.

10. Next, join the WISLR_Current_Rdwy_Links_Buffered shapefile to the current link_link table.
    a. Right click on the current link_link table and click join.
       1. Based on: WISLRid
       2. From: WISLR_Current_Rdwy_Links_Buffered.shp
       3. Attribute: Rdwy_Link_
       4. Keep all records
    b. Export the new table into the “County QA/QC” folder.

The join should look like:



11. Any values that returned <Null> in Rdwy_Link_ need to be checked. This returns all links that are present in the county Link-Link table but are not present in the WISLR current roadway shapefile.
    a. Open the joined table in ArcMap.
        i. Choose "Select by Attributes" on the table's toolbar
            1. Where: "Rdwy_Link_" IS <Null>
    b. Visually check each flagged IDs in the table and fix the table accordingly.
        i. Null values generally resulted from typos and from links in bordering counties.

12. Remove the join by right clicking on the county current Link-Link table and selecting "Remove Join".

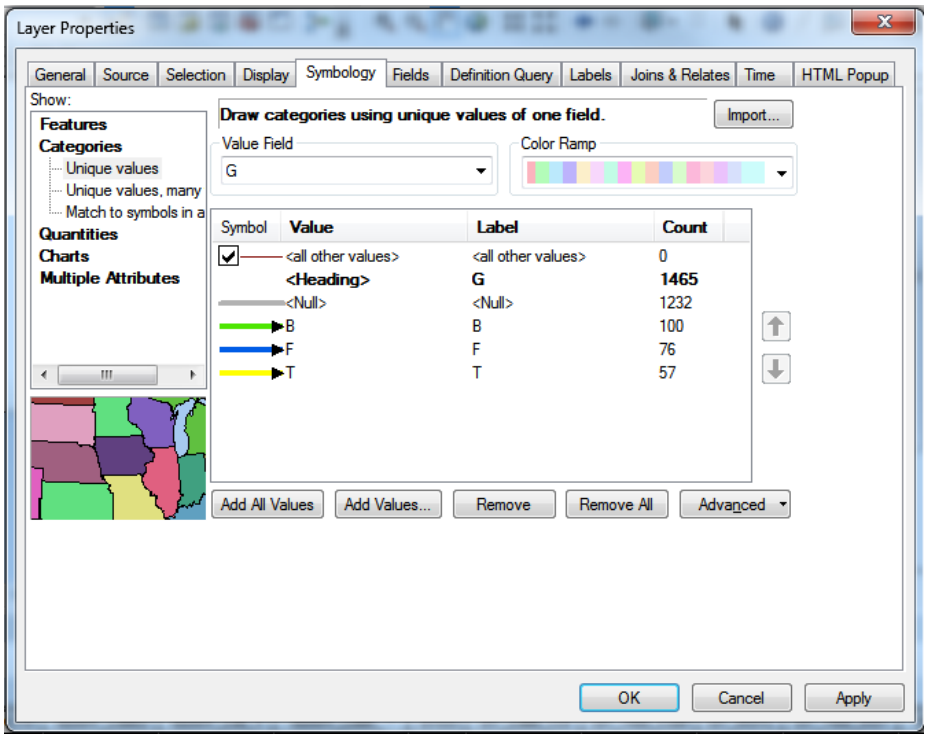
5.3. Gore Point Check

The following lists the process to ensure that all intersections involving gore points have been coded correctly in the "G" column on the county Link-Link table.

13. Turn off the STN current roadway shapefile and turn on the WISLR current roadway shapefile instead.

14. Join the current Link-Link table to the WISLR current roadway shapefile. This should join the gore point attributes to the Link-Link table.
    a. Right click on WISLR_Current_Rdwy_Links_Buffered and select join:
        1. Based on: Rdwy_Link_
        2. From: "current Link-Link table"
        3. Attribute: WISLRid
        4. Keep matching records only.

15. Right click on WISLR_Current_Rdwy_Links_Buffered and select "Properties".
    a. Select the "Symbology" tab and choose "Categories" on the left hand menu.
        i. Value Field: "G"
        ii. Hit "Add All Values"
        iii. Set the symbology, as shown below, for the gore point field to show T, F, & B in different colors so as to perform a visual checks.



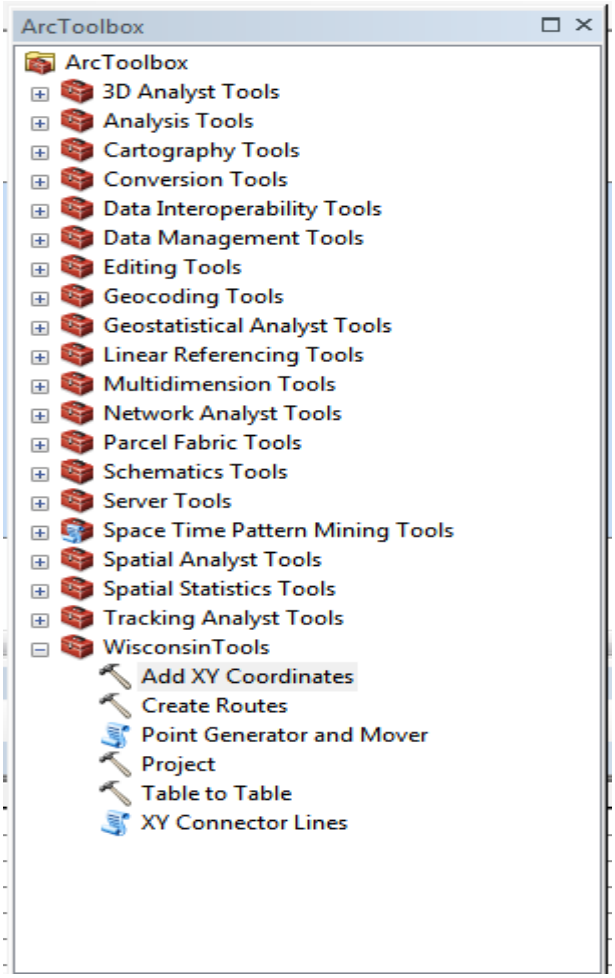
16. Look at each intersection visually and ensure that the gore points are coded correctly.

17. Remove all remaining joins before proceeding to the next check.

5.4. XY Connector Link Check

This list provides the procedure necessary to perform the XY Connector check that involves running a simulation of point data translation from STN to WISLR. This check ensures that any errors made during the coding process are caught within each county.

18. In ArcMap, click the ArcToolbox Icon. In the ArcToolbox window shown below, right click and select “Add Toolbox”.
    a. Select previously created “Wisconsin Tools.tbx” which should contain all programs necessary for the XY Connector program.
        i. Right click on Wisconsin Tools and select “Add” and “Script” to add the Point Generator and Mover Program, and the XY Connector Lines Program. (Program codes provided in Appendix)
        ii. Repeat and select “Add” and “Tools” to add the remaining ArcGIS tools used:
            1. “Create Routes”
            2. “Project (Data Management)”
            3. “Add XY Coordinates”
            4. “Table to Table (Conversion)”



19. Right click on ArcToolbox at top and select "Save Settings" and "To Default". This will keep the Wisconsin Tools toolbox in place whenever using ArcMap.
    **These steps only need to be done one time. Once the Wisconsin Tools toolbox is added and saved it will remain on that computer for all other county and statewide maps opened.*

20. In the county folder, create a new folder, "County WDir".

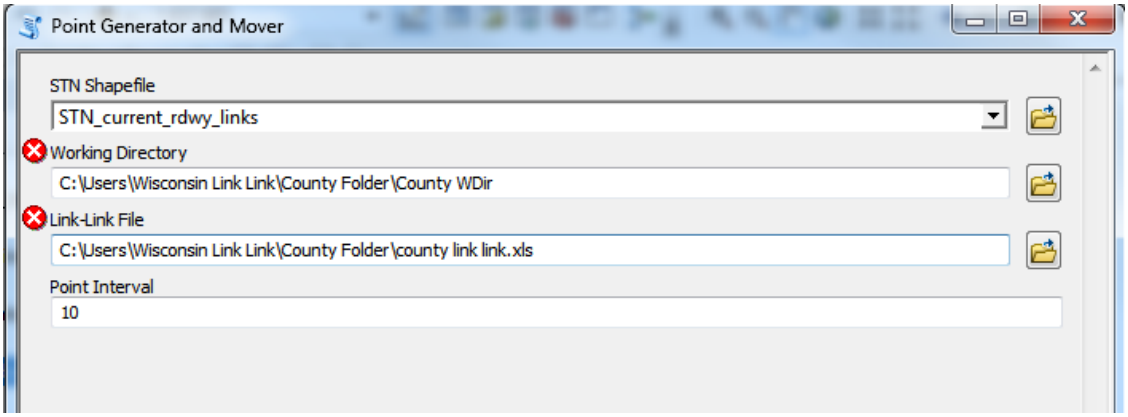
21. Take the most up to date county Link-Link table and save a copy of only the current records for the county.
    a. Change the tab in excel from "Sheet 1" to "link_link".
    b. Save the file itself "county Link-Link".

    *Note: The programs will not run properly if this naming convention is not kept for each county.

22. In ArcMap, expand "Wisconsin Tools" and select the "Point Generator and Mover" program. A pop-up should appear. Fill in the prompts accordingly:
    1. STN Shapefile: STN_Current_Rdwy_Links
    2. Working Directory: County WDir
    3. Link-Link File: county Link-Link.xls

4. Point Interval: 10
5. Select Okay

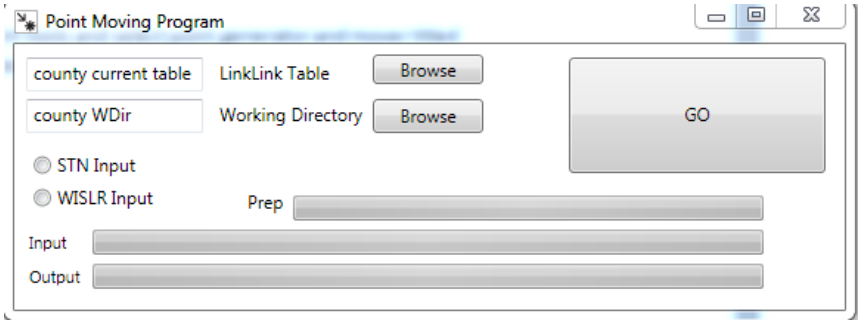
The prompt should look like this:



Once the program finishes running, there will be no noticeable differences on the map other than two tables that should appear in the table of contents.

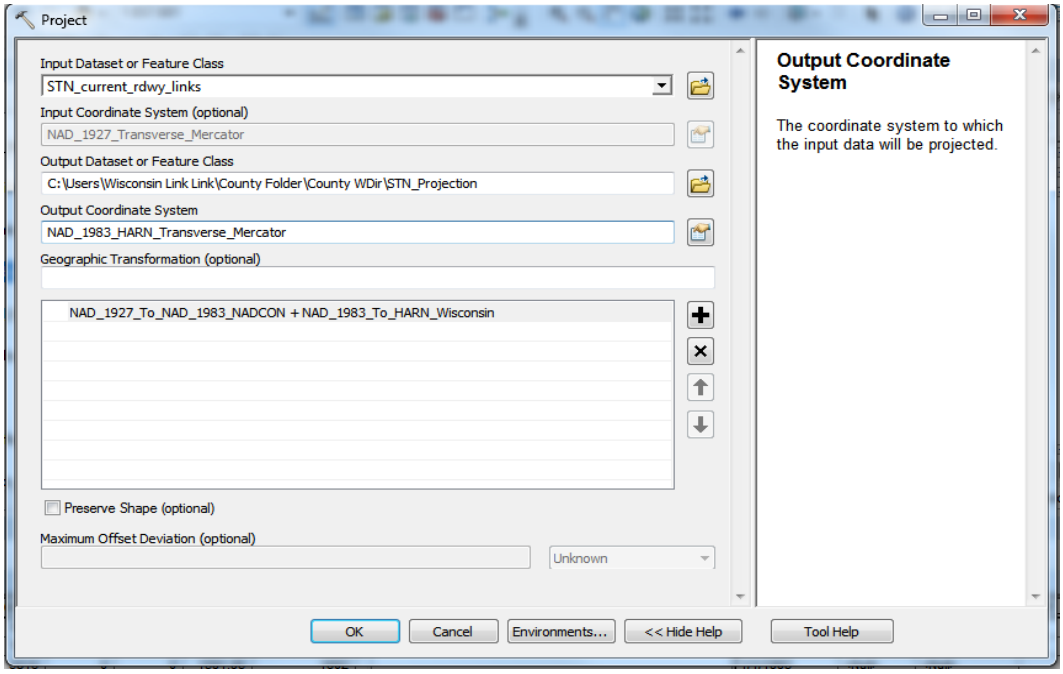
23. Using the program code provided in the Appendix, install the "Point Moving" program outside of ArcMap. (Once installed, not necessary to repeat this step)

24. Run the Point Moving program and fill in the prompts as shown below.
    1. LinkLink Table: county Link-Link
    2. Working Directory: County WDir
    3. Select: STN Input
    4. Hit Go and wait for completion of output

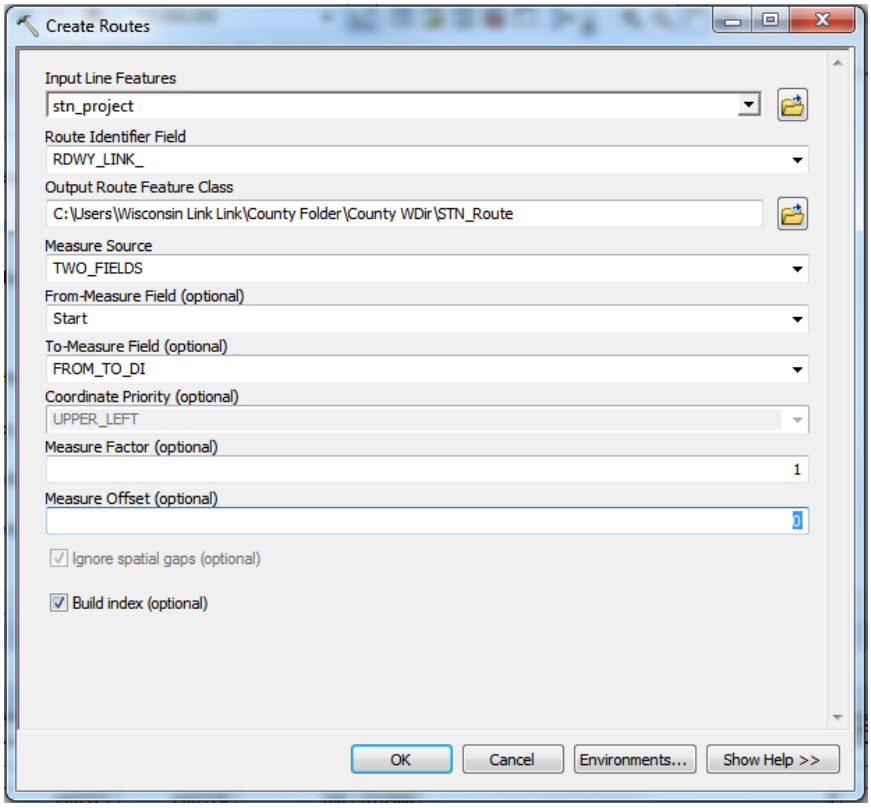


25. Once completed, look under "Wisconsin Tools" in ArcMap for "Project (Data Management)"
    a. Fill the prompts in as shown below. This step is only done once to project STN to WISLR.
        1. Input Dataset: STN_Current_Rdwy_Links
        2. Output Dataset: "STN_Projection"

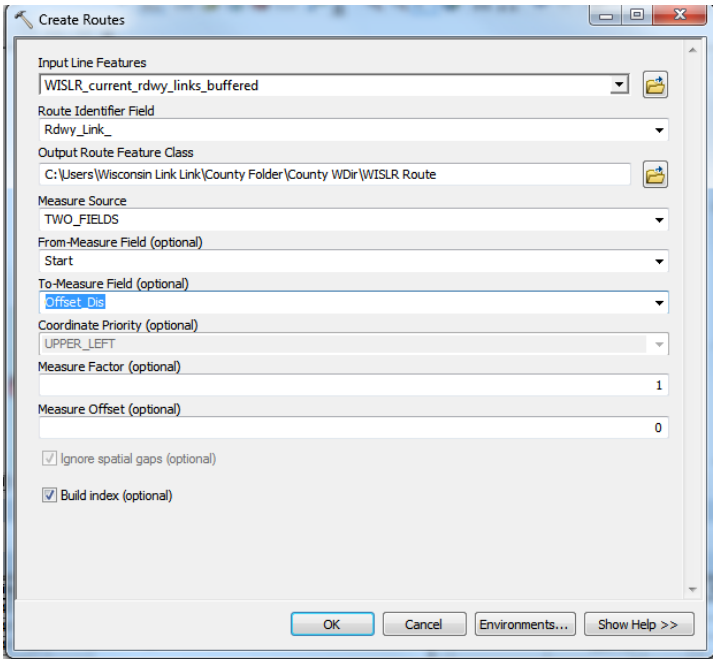
3. Output Coordinate System: NAD_1983_Harn_Transverse_Mercator
4. Geographic Transformation: **WISCONSIN** (before selecting OK, ensure that the transformation is to Wisconsin and not Michigan)



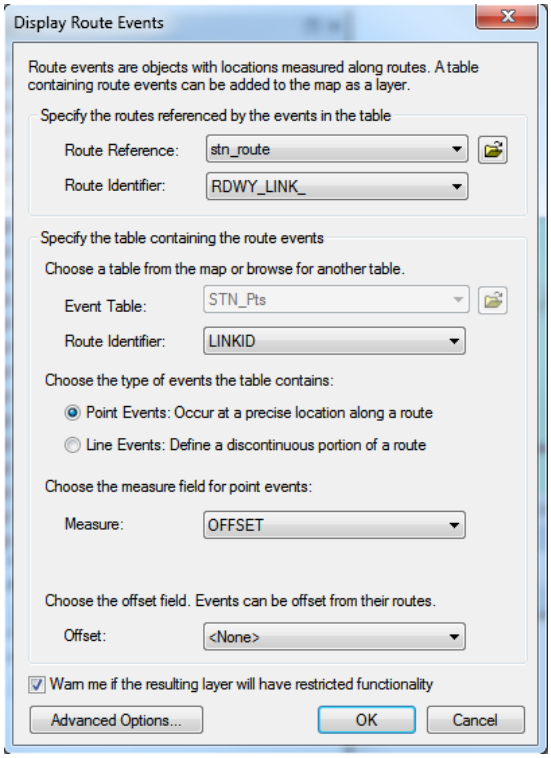
26. Next, select the "Create Routes" added under "Wisconsin Tools" toolbox. Use this tool for both STN and WISLR to result in 2 new shapefiles, STN_Route and WISLR_Route.
    a. For STN: Follow the prompt depicted.
        i. Input Line: STN_Projection
        ii. Route Identifier: Rdwy_Link_
        iii. Output Route: "STN_Route"
        iv. Measure Source: TWO_FIELDS
        v. From: Start
        vi. To: From_To_Di



b. For WISLR:
    i. Open attribute table for WISLR_Current_Rdwy_Links_Buffered and select “Add Field”.
        1. Add Field: “Start”; long integer
    ii. Select the “Create Routes” tool and follow the prompt as shown below.
        1. Input Line: WISLR_Current_Rdwy_Links_Buffered
        2. Route Identifier: Rdwy_Link_
        3. Output Route: “WISLR_Route”
        4. Measure Source: TWO_FIELDS
        5. From: Start
        6. To: Offset_Dis

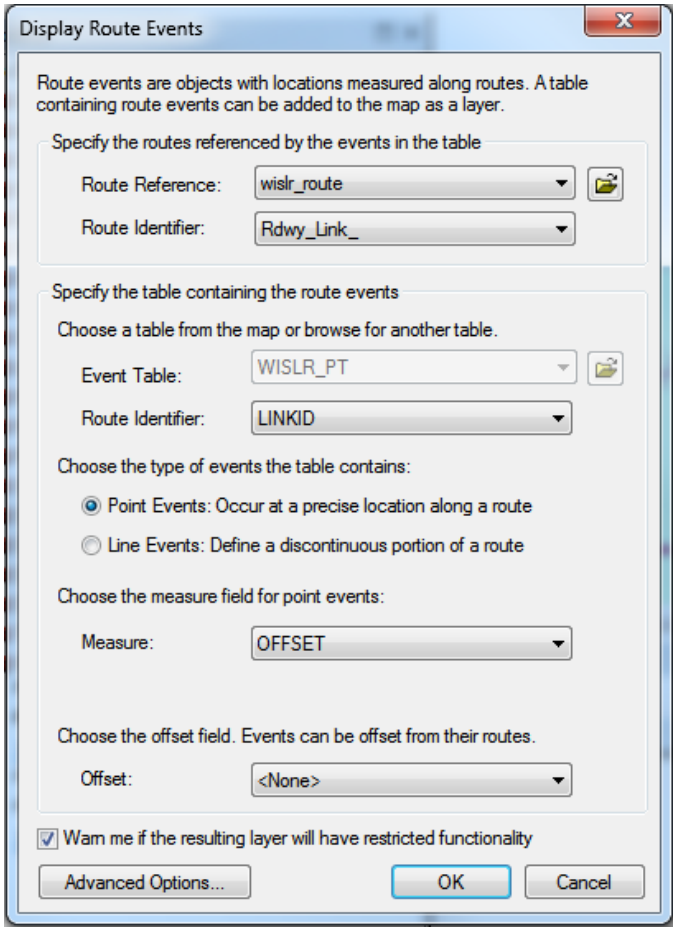


27. Next, add the "STN_Pts.dbf" and the "WISLR_PT.dbf" from the point generator and point moving program to the county map.
    a. Right click on the STN_Pts table and choose "Display Route Events".
        i. Follow the prompt below for the correct inputs to use.



ii. Export data from the point event shapefile and save it as “STN Points” in County WDir. Add it as a layer in the map. Turn off or remove the original layer.

b. Right click on the WISLR_PT table and select “Display Route Events”
   i. Follow the prompt below for the correct inputs to use. Note that these are different than the previous step

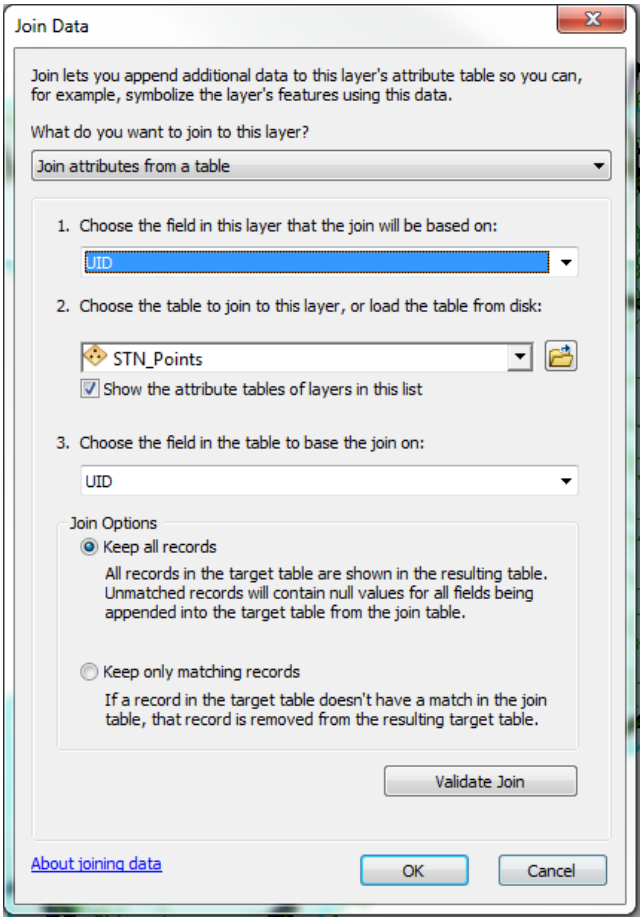


   ii. Export data from the point event shapefile and save it as “WISLR Points” in County WDir. Add it as a layer in the map. Turn off or remove the original layer.

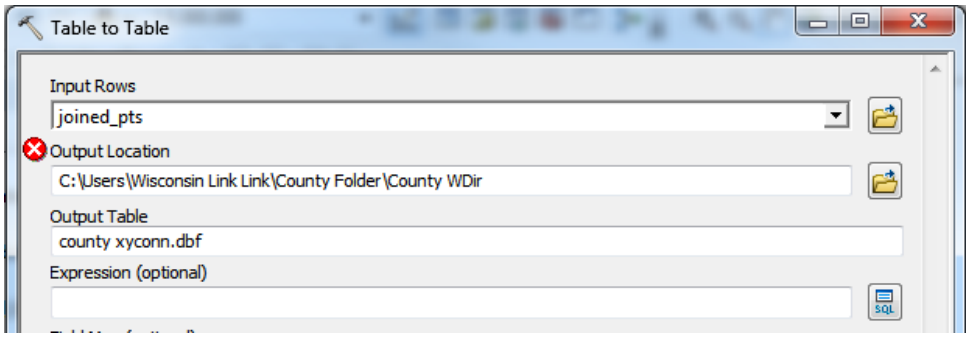
28. Under the “Wisconsin Tools” toolbox select “Add XY Coordinates”. Use the tool to add geographic coordinates for both the WISLR Points shapefile and for the STN Points shapefile.

29. After the coordinates are generated, right click on the WISLR Points shapefile and go to layer properties. Only perform this step for WISLR points.
    a. Select definition query.
       i. Create a query of: “ point_x > 0”
       ii. Hit okay.

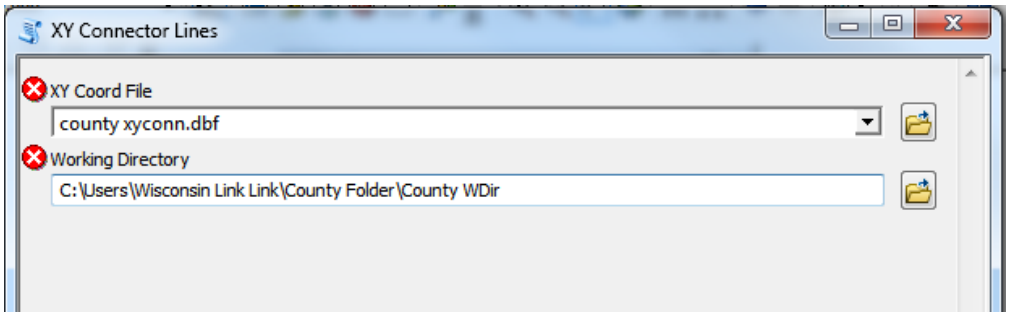
30. Join the WISLR Points with the STN Points based on UID.
    a. Right click on WISLR points and join data so as to reflect the prompt below.
        i. Based on: UID
        ii. From: STN_Points
        iii. Attribute: UID
        iv. Keep all records
    b. Export data as "Joined Points" shapefile to County WDir.



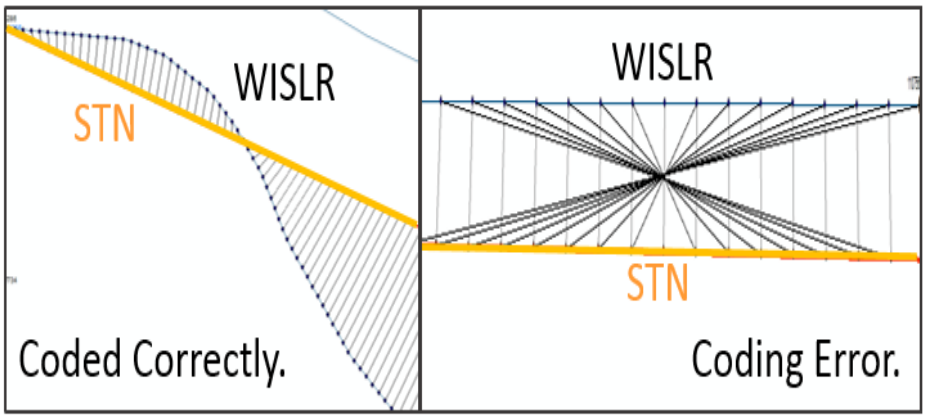
31. Select the "Table to Table" conversion under "Wisconsin Tools".
    a. Follow the prompt for the table to table conversion shown below:
        i. Input Rows: Joined_Points.shp
        ii. Output Location: County WDir
        iii. Output Table; County XYconn.dbf



32. Finally, select "Wisconsin Tools" and then "XY Connector Lines".
    a. Follow the prompt as shown:
        iii. XY Coord File: "County XYConn.dbf"
        iv. Working Directory: CountyWDir
        v. Select okay.



    b. Visually check for any discrepancies in the way that the lines are drawn. Examples of correctly drawn and incorrectly drawn lines are shown below. The lines that are drawn incorrectly are a result of errors in coding the Table. Perform this visual examination along all state routes in the county and fix the Link-Link table for any errors found.



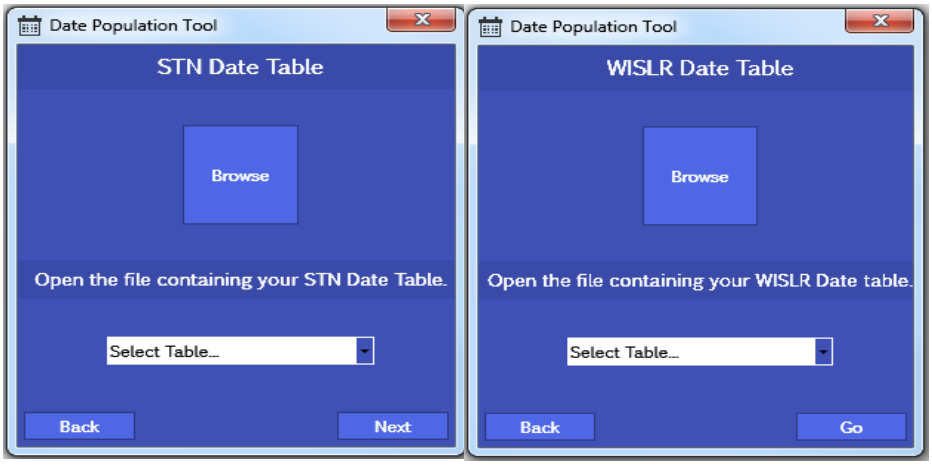
5.5. Date Population

The following procedure populates the Start_Valid and End_Valid columns for all records in the Link-Link table.

33. Convert the County Link-Link table from an excel file to an access Table naming the table "link_link table".

34. Run the Date Population Tool and follow the prompts.
    a. Choose the County Link-Link Table and select the link_link file.

    b. Select the Table containing the link IDs and their corresponding Created and Historic dates and select the tables for both STN and for WISLR.



    c. Hit "Go" and wait for program to complete date population of table.
        i. Any errors that occurred in the program are output in a text file and the link IDs and respective records in the table are examined and fixed if necessary.

Notes: Be aware of issues encountered with the date population program when the Record_Created and Record_Historic columns are not in the correct Date/Time format.

VI. Table Merge

Once all counties are updated and checked for accuracy, the separate county tables are to be merged into one statewide table using the following procedure.

1. Add all counties FINAL Link-Link table into a single access Table, each county having its own table within the Table under "*countyname*".

2. Merge the counties into a single table using the UNION function in Access SQL.
    a. As a result of the number of county tables must perform this function 3 times.
        i. First time merges half the counties, A through P, into a single table.
        ii. Second time merges the remaining half of counties P through Z, into a single table.
        iii. Third time merges the two previous merged tables into one single complete table with all counties.
            1. When running the last query to merge the two half Tables make a table from the query, "ALL", which can later be exported to excel.
    b. Using the UNION function in access also removes EXACT duplicates when merging the tables (meaning duplicates having exact same entries for ALL columns in the table).

The written code used when performing this Table merge can be found in the Appendix of this procedure manual.

Notes: Be aware of issues with the DB merges when the columns in each of the two tables did not have the same data type, such as "Text vs Memo" or "Text vs Date/Time". It is very important to make sure the tables are in the same format before performing any of the QAQC procedures.

VII. Statewide Quality Assurance and Quality Control

The following sections describe the process of performing an additional quality assurance check on the statewide Table. All of the following steps involve using both the statewide Link-Link table and the statewide map package.

7.1. Duplicity Check

This process identifies and removes any duplicated records containing current links.

1. Perform duplicity check on only the current link records because the historic records have no effect on the procedures later carried out. Compare STNid+WISLRid as there should be only ONE record with exact match.
    a. Export the statewide Link-Link table to Excel to edit these duplicates.
        i. Concatenate the columns desired: STNid, WISLRid
        In column U: ***"=A2&D2"*** =STNidWISLRid ; copy through all records.

        ii. In column V use the statement:
        "=IF(COUNTIF(U2:U2,U2)>1, "Duplicate", "")

        This will insert a duplicate statement into each concat statement that is repeated

    b. Select only records containing "Duplicate" flag and copy and save into a new table called "Duplicates.xlsx".
    c. In ArcGIS, open "Statewide" map package and add data: "Duplicates".
        i. Join duplicate table with STN current rdwy links.
            1. Based on: Rdwy_link_
            2. From table: "Duplicates"
            3. Attribute: STNid
            4. Keep matching records only
        ii. Export joined records to a shapefile: "duplicate_STN_links" and add to map.
            1. Turn on layer and visually check each duplicate link and decide which record to delete based on which county it lies more within.
                a. Decide based on: where majority of link lies, or if on border go by north/west convention for border duplicated links.
                b. Also decide based on link with proper distance coding for duplicate records with different STNstart/end values between them.

2. Delete or mark historic or adjust unwanted record from the statewide Link-Link table in excel and record which record is being deleted in a separate table/file for future reference.

7.2. Link Checks

This process checks to ensure that there are no missing current STN or WISLR roadway links from the statewide Link-Link table.

2. STN Link check:
    a. Once all duplicates are removed, export working statewide Link-Link table into excel
        i. Filter out only current records and copy into a new sheet or excel file.
        ii. Add this "Statewide_working_current" table into Statewide Map in ArcGIS
    b. Join current link table with STN current rdwy shapefile and look for NULL values in STNid column; (repeat process as in County QAQC STN Link Check).
        i. Join current statewide table with STN current rdwy links.
            1. Based on: Rdwy_link_
            2. From table: "Statewide_working_current"
            3. Attribute: STNid
            4. Keep all records
        ii. This will return link IDs that are in the current STN shapefile but NOT in the table.
    c. Fix any missing links accordingly in the complete statewide Link-Link table.

3. WISLR link check:
    a. Remove join previously placed on STN current rdwy shapefile.
    b. Join WISLR_Current_Rdwy_Links_Buffered shapefile to current link table and select links where Rdwy_link_ in WISLR shapefile is NULL; (repeat process as in County QAQC STN Link Check).
        i. This will return link IDs that are in the table but NOT in the shapefile.
    c. Fix any flagged links accordingly in the complete statewide Link-Link table.

7.3. Discontinuity Check

This procedure list checks to ensure that all links present in the statewide Link-Link table, both STN and WISLR, are not discontinuous or missing any length. Embedded in this list is an additional check of the flag columns to identify any remaining coding errors.

4. Discontinuity check

a. Take the most up to date complete statewide Link-Link table and, in Excel, filter so that only current records show.
    i. Sort in Descending Order: 1) STNend 2) STNstart 3) STNid in that order.
    ii. Re-sort to perform the WISLR checks in same order (for WISLR).
b. Run the following IF statements:
    i. STN Discontinuity + Start check:

        [112]

        =IF(STNid2=STNid1,IF(STNstart2=STNend1,"",IF(OR(T1=1,T2 =1, M1>=1, M2>=1, W1=1, W2=1, P1=1, P2=1), “”, "Discontinuity")), IF(STNstart2=0,””,IF(AND(STNstart2=STNend2, OR(M2>=1, W2=1, P2=1)),””,”Not Start at 0”)))

        ***Make sure to start formula in second row of Link-Link table and not the very first.*

    ii. STN Start/End comparison and Flag Column Check:

        [2]

        =IF(start<end, “”,IF(start>end, “wrong start/end”, IF(AND(start=end, OR(M>=1, P=1, W=1)), “”, “Missing Flag”)))

    iii. Turn Lane Check:

        [0]

        =IF(T<>1, “”, IF(start=end, “Issue in T”, “”))

    iv. Re-sort the records in descending order: 1)WISLRend 2)WISLRstart 3) WISLRid in that order.

    v. WISLR Discontinuity + Start Check:

        [44]

        =IF(STNid2=STNid1,IF(STNstart2=STNend1,"",IF(OR(T1=1,T2 =1, M1>=1, M2>=1, W1=1, W2=1, P1=1, P2=1), “”,

"Discontinuity")),IF(STNstart2=0,"",IF(AND(STNstart2=STNend2,

OR(M2>=1, W2=1, P2=1)),"","Not Start at 0")))

vi. WISLR Start/End comparison and Flag Column Check **[0]**

=IF(start<end, "",IF(start>end, "wrong start/end",

IF(AND(start=end, OR(M>=1, P=1, W=1)), "", "Missing Flag")))

vii. Turn Lane Check:
[15]

=IF(T<>1, "", IF(start=end, "Issue in T", ""))

c. For STN: Once the results are computed, filter to show only the records containing wrong start, discontinuity, or flagged values and insert into a new table, "Discontinuity.xls".
   i. Add the new table to the Statewide Map package.
   ii. Join discontinuity table with STN current rdwy links.
      1. Based on: Rdwy_link_
      2. From table: "Discontinuity"
      3. Attribute: STNid
      4. Keep matching records only
   iii. Select the matching records and export to a new shapefile containing those flagged STN ids, as "STN_Discontinuity.shp" and add to the map.
   iv. Check each STN link and fix record accordingly in most up to date Link-Link table
d. For WISLR: same steps as for STN but applies to those records with flagged WISLR discontinuity and are to be joined to the WISLR current link shapefile based on Rdwy_link_ ID and WISLRid.
   i. Select the matching records and export to a new shapefile containing those flagged WISLR links named "WISLR_Discontinuity.shp" and add to the map.
   ii. Check each WISLR link visually and fix record accordingly in up to date Link-Link table.

5. STN/WISLR Full Length Check

a. Compare the highest STNend/WISLR end for each link ID to the LCM_Offset_Distance to make sure the full length of link is coded:
    i. In excel:

        [54]

        =IF(STNid2=STNid1,"",IF(STNend1<>Offset,"Wrong_Full_length",""))

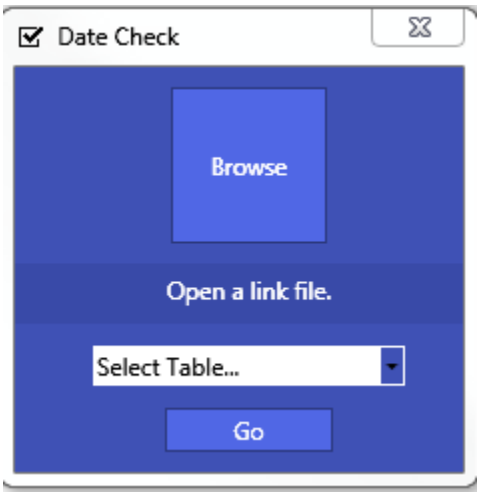
b. Filter to only show the flagged values and insert these into a new table and save. Add this new table to the Statewide Map package and join with the STN_link shapefile.
    i. Select the matching records and export to a new shapefile containing those flagged STN ids as "Length_Errors.shp" and add to the map.
    ii. Check each STN link and fix record accordingly in most up to date Link-Link table
c. Repeat for all records to compare the WISLR full lengths.

6. Convert this edited statewide table from excel into an access Table for the final quality checks to be performed.

7.4. Date Check

This process performs the last quality check of the Link-Link table by comparing all of the date columns for each record in the Link-Link table based on both STN and WISLR record date tables.

7. Date Column Check:
    a. Before performing this final check, rerun the Date Population program on the corrected Statewide Table to account for any changes made to the table through the last QAQC procedures.
        i. Select and Run the Date Population Program:
            1. For Link-Link table: "Statewide Link-Link" table
            2. STN Date: select Table table that contains STN id and respective created/historic dates.
            3. WISLR Date: select Table table that contains WISLR id and respective created/historic dates.
    b. Perform the final date column check using the computer program created by the University of Alabama (code provided in Appendix)
        i. Open "Date Check Program"
            1. Select statewide Link-Link access Table
                a. Choose statewide link_link table
            2. Select "Go"



3. Program provides output of any errors that were returned, where program checks for any record having:
    a. No Start Valid or Record Created date
    b. No Record Historic for End Valid
    c. Start Valid more recent than End Valid
    d. Start Valid more recent than Record Created
    e. Start Valid more recent than Record Historic
4. Edit the records flagged accordingly,
    a. Records missing Record Historic date are to be checked visually and fixed accordingly.
    b. Records having a Start Valid more recent than Record Created are to be identified, checked visually, and fixed accordingly.
    c. Records having a Start Valid more recent than End Valid are to be identified but not corrected in the table as all these records are marked historic and have no effect on the function of the Link-Link Table.

8. All quality assurance and quality checks are now complete. Convert the final up to date statewide Link-Link table to an access Table file named “Link-Link” for trial crash mapping.

VIII. Mapping Crashes

After the Statewide QAQC procedures are completed the Table table is then ensured to be complete and accurate and a trial mapping of the crash point data can be completed. The steps taken to obtain and apply the crash data are defined below.

8.1. Required Data

Crash data for the update period must be obtained from TOPS, Dr. Steven Parker. The crash data will contain the accident ID, date of the accident, STN Link ID, and STN Offset. The raw data will have the columns listed shown in the figure below. Locate the complete Link-Link table generated during the current update period.

	A	B	C	D	E	F	G	H	I
1	ACCDNMBR	ACCDDATE	LINKID	LKOFFSET	COUNTY	CNTYCODE	MUNITYPE	MUNICIPALI	MUNICODE
2	130600304	6/3/2013	1229	1.4	ADAMS	1	C	ADAMS	152
3	131104718	11/16/2013	1229	1.64	ADAMS	1	C	ADAMS	152
4	120706713	7/29/2012	1229	1.78	ADAMS	1	C	ADAMS	152
5	120602803	6/14/2012	1229	1.78	ADAMS	1	C	ADAMS	152
6	120105010	1/20/2012	1230	0	ADAMS	1	C	ADAMS	152
7	130307961	3/28/2013	1230	0	ADAMS	1	C	ADAMS	152
8	121102438	11/9/2012	1230	0.05	ADAMS	1	C	ADAMS	152

In the figure above, notice that LKOFFSET is stored in 100's of a mile. This will need to be converted and stored as an expanded number.

For example: ACCDNMBER 130600304 should be converted to a LKOFFSET value of 1400.

Columns E through I are not needed to perform crash movement from STN to WISLR.

8.2. Crash Movement

The steps required to move translate crash locations to the corresponding WISLR link follow the same steps used in Section 5.4 for the XY connector line procedure, with a two modifications. It is not necessary to perform the first step which generates the STN_Pts table. This is because the crash points are essentially an STN _Pts table because the crashes contain an STN link ID and an offset, which is all that is needed run the PointMovingProgram (PMP.) The PMP requires the column headings in the STN_Pts table to be named according to the output from the Point Generator, so edits must be made to the column headings in the raw crash table. In addition, the Point Generator assigns a unique ID (UID) to each STN point, so these must be generated in the crash table. The figure below shows a prepared crash table and it should be named STN_Pts in the working directory. This ensures that the PMP will recognize the table for use in generating the WISLR_Pts table. Do not proceed beyond step 25 in Section 5.4 because it is not necessary to create the XY connector lines.

	A	B	C	D	E
1	ACCDNMBR	ACCDDATE	LINKID	OFFSET	UID
2	120506557	5/24/2012	3	0	0
3	120705186	7/24/2012	3	0	1
4	121004019	10/18/2012	3	0	2
5	121209484	12/31/2012	3	0	3
6	130100997	1/6/2013	3	0	4
7	130202674	2/6/2013	3	0	5
8	130400772	4/4/2013	3	0	6
9	130406548	4/26/2013	3	0	7
10	130803271	8/15/2013	3	0	8

Once the STN_Pts table has been generated, launch the PMP to generate the WISLR_Pts table, following the steps outlined in Section 5.4. A point to consider when moving crashes is that some crashes are assigned to STN links associated with historic records in the Link-Link table. When the PMP is launched against the current records in the Link-Link table only, the historic crashes will not move to a WISLR link. Therefore, an additional step will be required once the initial WISLR_Pts table is generated to identify historic crash records. The output from the PMP are the translated crash points from STN to WISLR.

8.3. Crash Record Movement Quality Assurance and Quality Control

Once the WISLR_Pts table is generated, QA/QC must be performed to identify unmoved crashes, duplicated crashes, and crashes that moved to historic links. The steps required to identify the previous three errors are listed here.

1. In ArcMap create a file geodatabase containing the STN_Pts and WISLR_Pts tables. This allows a one-to-many join to be performed.

2. Join the two tables based on UID by launching the Make Table Query tool.

3. Summarize on accident ID.

4. Identify duplicates. Export both duplicate accident IDs and non-duplicate accident IDs.

    a. Join to STN_PTs to get Link ID and Offset

5. Join STN_Pts to non-duplicated accident ID table to identify which accident IDs did not move from STN to WISLR. Select null values and export as unmoved crashes.

6. Summarize on Link-ID

7. Join summarized table to Link-Link table to identify problem.

Re-run PMP on identified historic crashes and repeat steps 1-7 until all crashes are accounted for.

IX. Finalizing the Table

After the crash event data are moved and mapped between the two Tables, any errors or missing crashes can be identified and analyzed as described here.

1. Identify the errors resulting from the trial crash data mapping. Possible reasons preventing crashes from moving:
    a. Mapped to Historic Links: Crashes moved to link sections that became historic during the update period.
    b. Mapped to Problem Flagged Links: Crashes that result in duplicate crash points being mapped.
    c. Other Mapping Errors

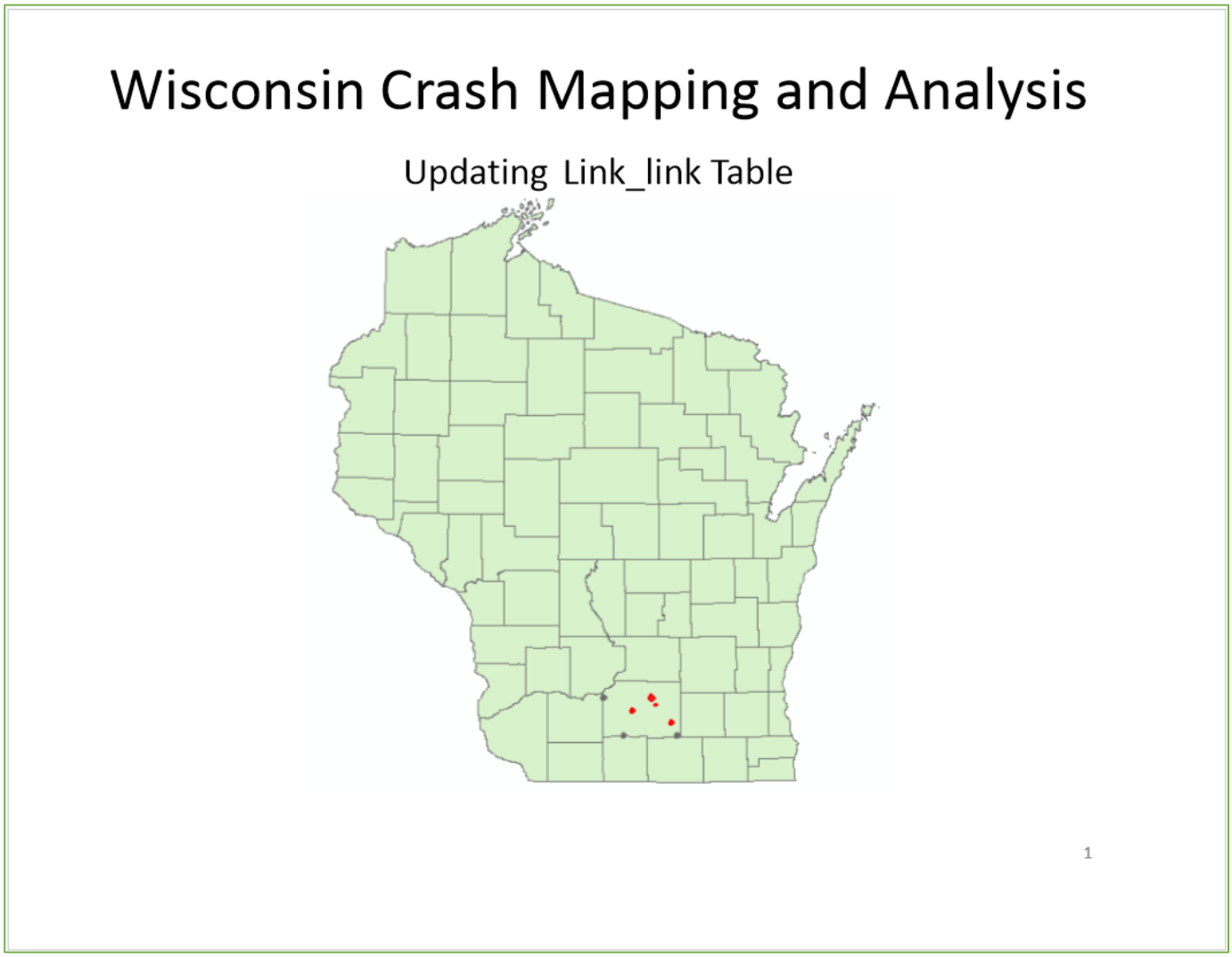
2. Visually check all the links with errors (other than mapped to historic links) and identify if problems are a result of the Link-Link table and fix the issues accordingly.

3. Create a table containing the accident crash ID's with problems and with the respective link IDs.

4. Create a final access Table for submittal:
    a. Final Statewide Link-Link table:
        i. Name: "Link-Link"
            1. Contains same columns as throughout the update process
    b. STN Accident Events:
        i. Name: "STN_Points"
            1. Contains 3 columns:
                a. UID: Unique Accident ID
                b. Link_ID: STN ID
                c. Link_Offset: Offset Distance
                d. Comment: Fill in for all accidents with mapping issues.
    c. WISLR Accident Events:
        i. Name: "WISLR_Points"
            1. Contains 3 columns:
                a. UID: Unique Accident ID
                b. Link_ID: WISLR ID
                c. Link_Offset: Offset Distance
                d. Comment: Fill in for all accidents with mapping issues.

X. Appendix: Link-Link Coding Tutorial

Following is a copy of a power point tutorial that is to be used as a reference when updating the Link-Link Table. The tutorial contains information describing how to code for problem flags and for other difficult areas in the roadway networks.



STN and WISLR

WisDOT is maintaining two linear referencing system for road network of Wisconsin:

1. **State Trunk Network (STN) :** Represents state roads by straight line and node (sites) at the ends of each line.

2. **Wisconsin Information System of Local Roads (WISLR) :** Represents local roads with the line and nodes at the ends of each line. WISLR shows the cartographic locations of the roads.

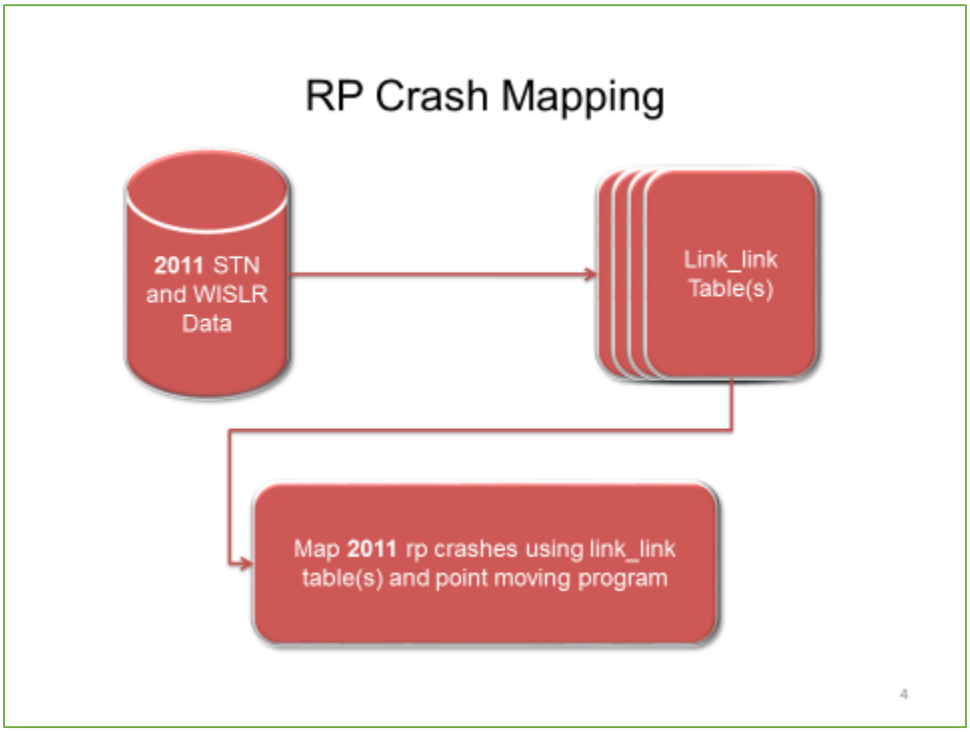
2

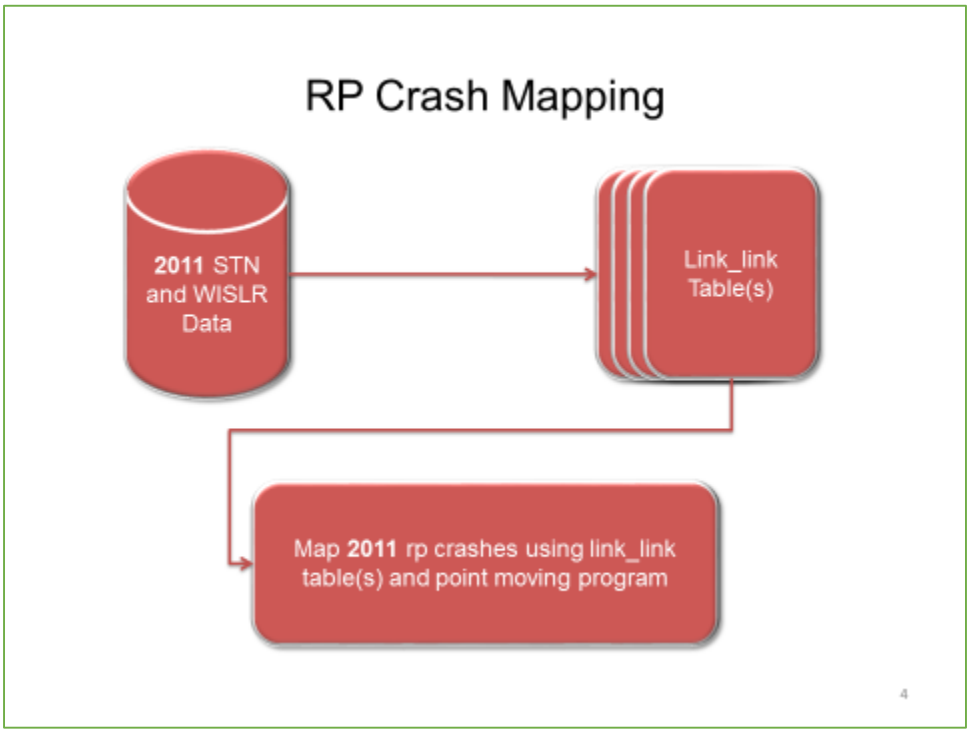
GOAL

To produce a technique and Statewide data to move location based information between STN and WISLR.

To track all locations where definitional discrepancies between the STN and WISLR exist such as: turn-lane overlaps, median crossovers, gore point links, park & rides and weigh stations, and problem areas.

3


RP Crash Mapping
2011 STN and WISLR Data
Link_link Table(s)
Map 2011 rp crashes using link_link table(s) and point moving program
4


RP Crash Mapping
2011 STN and WISLR Data
Link_link Table(s)
Map 2011 rp crashes using link_link table(s) and point moving program
4

Original Link_link Columns

Column Name	Relevant Field	Example Values
STNid	RDWY_LINK_ID	9297, 32568
STNstart	No relevant field	0 (can be greater than zero one-to-many relationship)
STNend	FROM_TO_DIS	1060, 500 (depends on ratio or access points)
WISLRid	RDWY_LINK	4425294, 9297
WISLRstart	No relevant field	0 (can be greater than zero one-to-many relationship)
WISLRend	LCM_FROM_TO_DIS	317, 9398 (depends on link length or ratio)

6

Date Columns

Record_Created : The date of entering new record

Record_Historic : The date from when the record is considered historic according to link_link table .

Start_Valid : The date from when relationship between STN and associated WISLR link starts. The most recent date between current dates of STN and associated WISLR link according to route-link table should be Start_Valid.

End_Valid : The date from when the relationship between STN and WISLR expires. The older date between historic dates of STN link and associated WISLR link is End_Valid. If only one of the links (STN or associated WISLR) has historic date, End_Valid should be that date.

- Record Created and Record Historic date should be filled by coder.
- Start_Valid and End_Valid date columns will be populated programmatically once the coding is done.

7

Record_Created and Record_Historic

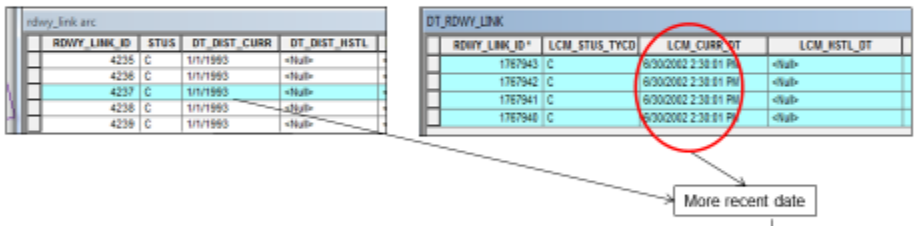
- The record of STN link **A** (WISLRid i) is expired from Link_link table on 4/12/2012 and Record_Historic date of STNid A is 4/12/2012
- A new record of STN link **B** (WISLR id ii) is entered in the place of STN link **A** on the same date and the Record_Created date for STN link **B** is 4/12/2012

STN id	STN start	STN end	WISLR id	WISLR start	WISLR end	T	M	G	W	P	Coder	Record_ Created	Record_ Historic
A	0	420	I	0	2270						ZR	6/1/2010	4/12/2012
B	0	430	ii	0	2800						LM	4/12/2012	

8

Start_Valid date

- Relationship between STN and WISLR links changes year to year.
- New relationship (new STN or new WISLR) starts
- Old relationship (Deleted STN or Deleted WISLR) ends or changes
- Start_Valid and End_Valid date represents the time span when a relationship between STN link and WISLR link is valid.
- Start_Valid : The more recent date between current dates of STN link and associated WISLR link.



STN id	STN start	STN end	WISLR id	WISLR start	WISLR end	T	M	G	W	P	Coder	Record_ Created	Record_ Historic	Start_ Valid	End_ Valid
4237	0	460	1767943	0	2534						SLF	5/24/2012		6/30/2002	
4237	460	800	1767942	0	1584						SLF	5/24/2012		6/30/2002	
4237	800	810	1767941	0	264						SLF	5/24/2012		6/30/2002	
4237	810	870	1767940	0	264						SLF	5/24/2012		6/30/2002	

End_Valid date

End_Valid : The older date between historic dates of STN link and associated WISLR link. If only one of the links (STN and associated WISLR) has historic date, End_Valid should be that date.

STN route-link table

T_RDWY_RTE_LINK

RDWY_LINK_ID *	DT_RTE_LINK_CURR	LCM_DT_HSTL	LCM_STUS
44101	1/19/2000	1/19/2000	H
44102	1/19/2000	1/19/2000	H
44103	1/19/2000	1/19/2000	H
44103	1/19/2000	12/17/2010	H
44103	1/19/2000	1/19/2000	H

WISLR route-link table

DT_RDWY_RTE_LINK

RDWY_LINK_ID *	LCM_STUS_TYCD	LCM_CURR_DT	LCM_HSTL_DT
4513617	H	7/6/2008 8:06:05 AM	3/2/2011 4:23:32 PM
4513617	H	8/18/2002 8:47:13 AM	7/6/2008 8:06:05 AM
4513618	H	8/18/2002 8:47:13 AM	3/3/2011 11:45:29 AM
4513619	C	8/18/2002 8:47:13 AM	<Null>
4513619	C	8/18/2002 8:47:13 AM	<Null>

Older date

STN id	STN start	STN end	WISLR id	WISLR start	WISLR end	T	M	G	W	P	Coder	Record_ Created	Record_ Historic	Start_ Valid	End_ Valid
44103	0	60	4513618	0	370			B			MH	8/30/2010	6/11/2012	8/18/2002	12/17/2010

10

One-to-One Relationship

	ID	Length
STN	A	800
STN	B	800
WISLR	i	4224
WISLR	ii	4224

Flags

STN id	STN start	STN end	WISLR id	WISLR start	WISLR end	T	M	G	W	P	Coder	Record_ Created	Record_ Historic	Start_ Valid	End_ Valid	Comments	Reviewer_ Comments	Date_ Reviewed	County
A	0	800	i	0	4224						ZR	4/6/2010		1/1/1993					
B	0	800	ii	0	4224						ZR	4/6/2010		1/1/1993					

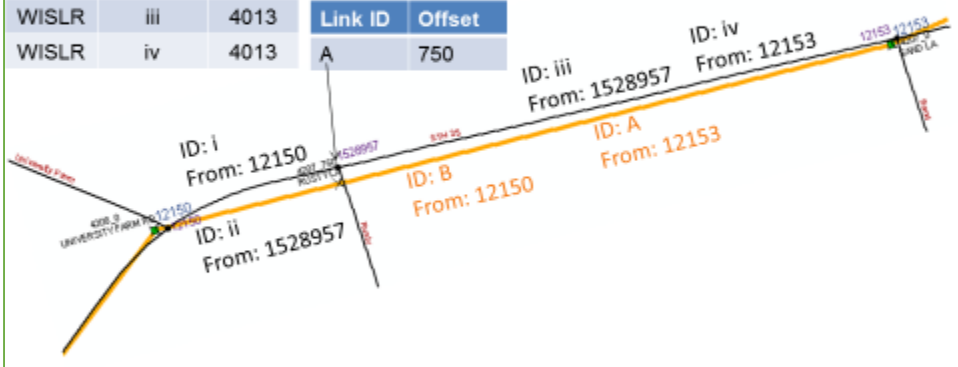
11

	ID	Length
STN	A	1000
STN	B	1000
WISLR	i	1267
WISLR	ii	1267
WISLR	iii	4013
WISLR	iv	4013

One to Many Relationship (With Access Points)

Access Point

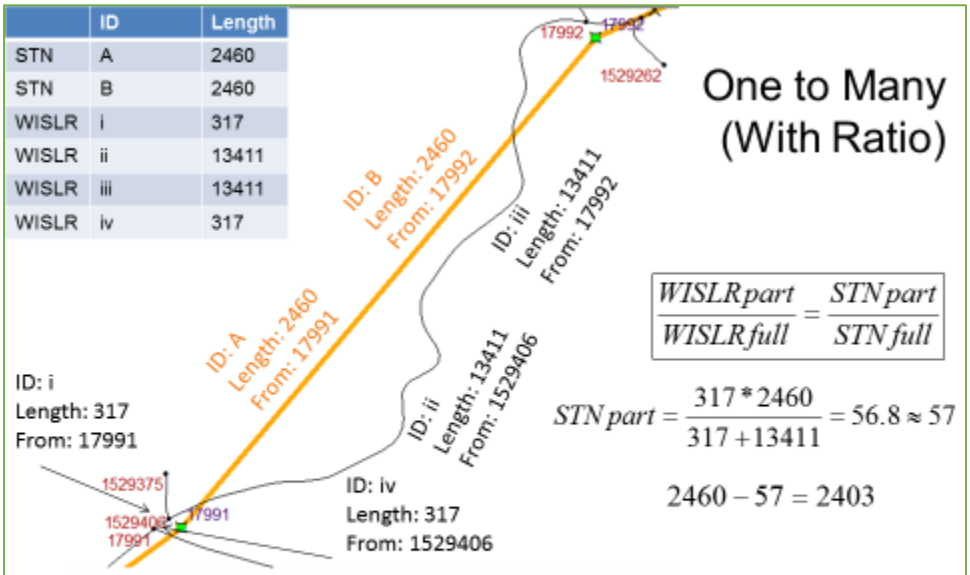
Link ID	Offset
A	750



STN id	STN start	STN end	WISLR id	WISLR start	WISLR end	T	M	G	W	P	Coder	Record_ Created	Record_ Historic	Start_ Valid	End_ Valid	Comments	Reviewer_ Comments	Date_ Reviewed	County
A	0	750	iv	0	4013						ZR	4/6/2010		1/1/1993					
A	750	1000	ii	0	1267						ZR	4/6/2010		1/1/1993					
B	0	250	i	0	1267						ZR	4/6/2010		1/1/1993					
B	250	1000	iii	0	4013						ZR	4/6/2010		1/1/1993					

	ID	Length
STN	A	2460
STN	B	2460
WISLR	i	317
WISLR	ii	13411
WISLR	iii	13411
WISLR	iv	317

One to Many (With Ratio)



$$\frac{WISLR\,part}{WISLR\,full} = \frac{STN\,part}{STN\,full}$$

$$STN\,part = \frac{317 * 2460}{317 + 13411} = 56.8 \approx 57$$

$$2460 - 57 = 2403$$

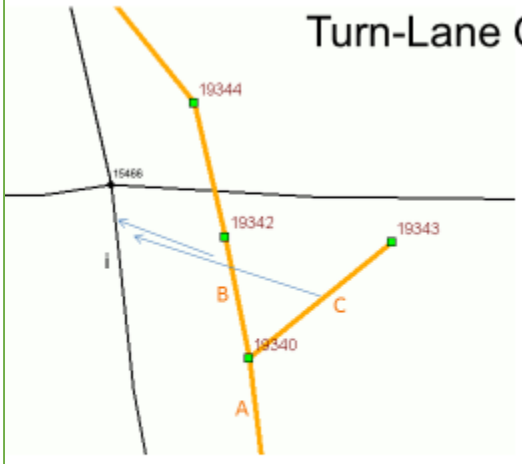
STN id	STN start	STN end	WISLR id	WISLR start	WISLR end	T	M	G	W	P	Coder	Record_ Created	Record_ Historic	Start_ Valid	End_ Valid	Comments	Reviewer_ Comments	Date_ Reviewed	County
A	0	57	i	0	317						ZR	4/6/10		1/1/1993					
A	57	2460	ii	0	13411						ZR	4/6/10		1/1/199					
B	0	2403	iii	0	13411						ZR	4/6/10		1/1/199					
B	2403	2460	iv	0	317						ZR	4/6/10		1/1/1993					

Flag Columns in the Link_link Table

New Column Description	Flag Column Label	Value
Turn-Lane Overlap	T	1
Median Cross-Over	M	ID or 1
Gore Point	G	T, F, or B
Weigh Stations, Park & Rides	W	1
Problem	P	1
Data entry personnel	Coder	ZR
Record created date	Record_Created	Date
Record historic date	Record_Historic	Date
Start date	Start_Valid	Date
End date	End_Valid	Date
Comments for problems flag	Comments	Text

14

Turn-Lane Overlap



•All turn-lanes should be flagged in the "T" column as 1

•The length of the STN link should be subtracted from the WISLR length

$$20 * 5.28 = 106$$
$$1320 - 106 = 1214$$

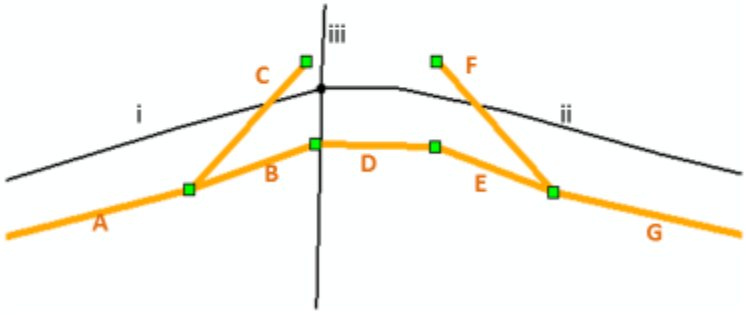
STN id	STN start	STN end	WISLR id	WISLR start	WISLR end	T	M	G	W	P	Coder	Record_ Created	Record_ Historic	Start_ Valid	End_ Valid	Comments	Reviewer_ Comments	Date_ Reviewed	County
A	0	200	i	0	1214						ZR	4/6/10		1/1/1993					
B	0	20	i	1214	1320	1					ZR	4/6/10		1/1/1993					
C	0	30	i	1214	1320	1					ZR	4/6/10		1/1/1993					

Both turn-lanes are flagged
Both turn-lanes are overlapped on same portion of WISLR

15

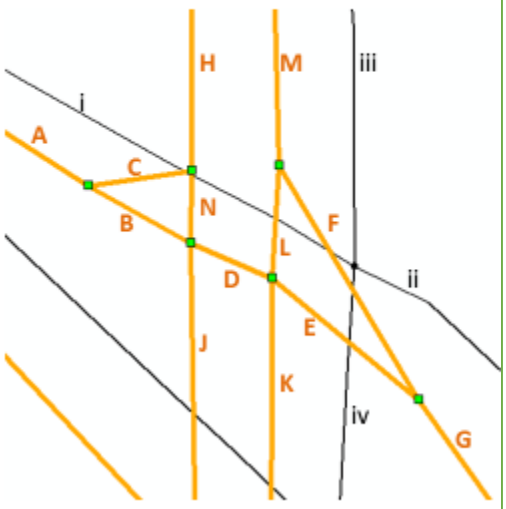
Turn-Lane on Single STN

- One state route intersects a local road
- STN links can only be coded to WISLR representing state routes
- STN links A, B & C will be coded to WISLR link **i**, and STN links E, F & G will be coded to WISLR link **ii**
- Both C & B and E & F are coded with 1's in "**T**" column
- D is coded as a median cross over



Turn-Lanes on 2 Intersecting STN Highways

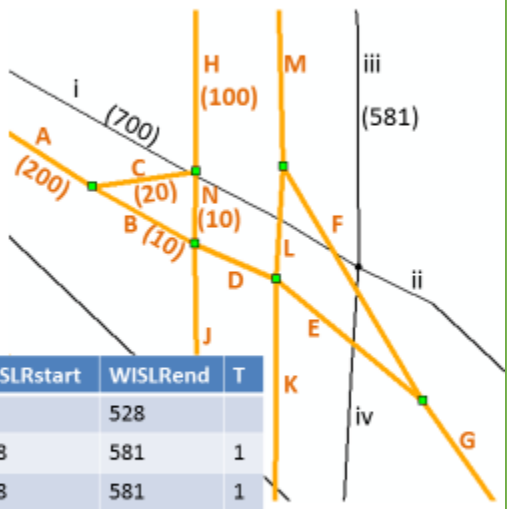
- Two state highways intersect
- STN link C is actually a turn-lane when travelling down link H, so C and N should be coded to WISLR link **iii**
- STN link B will be coded on WISLR link **i**
- STN links E & F are both turn-lanes and are coded to WISLR **ii**
- STN link L is coded to WISLR link **iii**
- STN links C, B, & N are all given 1's in the "T" column, as well as F, L, and E



Turn-Lanes on 2 Intersecting STN Highways

$$10 * 5.28 \approx 53$$

$$581 - 53 = 528$$

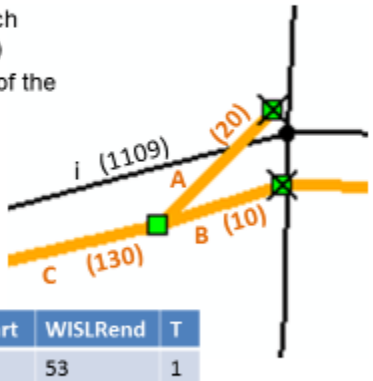


STNid	STNstart	STNend	WISLRid	WISLRstart	WISLRend	T
H	0	100	iii	0	528	
C	0	20	iii	528	581	1
N	0	10	iii	528	581	1
B	0	10	i	0	53	1
A	0	200	i	53	700	

18

Turn-lane Length Difference

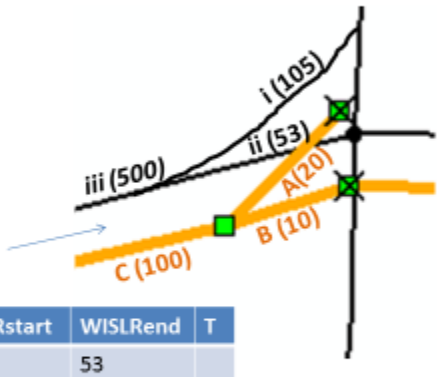
- Map the turn lanes to the same WISLR link
- Confusion when turn lane lengths differ (which STN length should be used against WISLR?)
- Solution: code both STN links to the shorter of the two in WISLR (see table below)
- Both turn-lanes are coded with a "1" in the T column



STNid	STNstart	STNend	WISLRid	WISLRstart	WISLRend	T
A	0	20	i	0	53	1
B	0	10	i	0	53	1
C	0	130	i	53	1109	

Turn-Lanes : One to one relationship

- If there is one to one relationship "T" should be null
- Sometimes turn lanes are showed in WISLR and there is one to one relationship.
- "T" column should be populated only when two STN links representing turn lane, overlap on one WISLR link .



STNid	STNstart	STNend	WISLRid	WISLRstart	WISLRend	T
A	0	20	I	0	53	
B	0	10	Ii	528	105	
C	0	100	iii	528	500	

20

Median Cross-Over

These links have only one direction

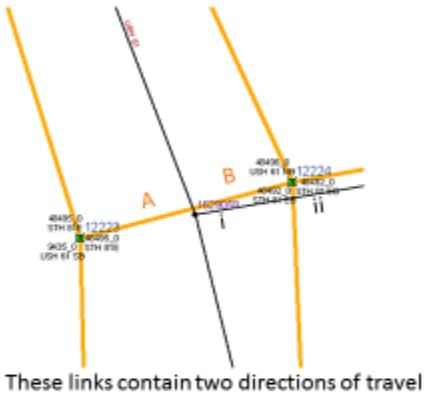
- Median cross-overs are not represented in WISLR
- Coded as one record per direction, instead of two
- Placed on start of one WISLR link, in flag column, specify WISLR id of other

STNid	STNstart	STNend	WISLRid	WISLRstart	WISLRend	T	M
A	0	10	ii	0	0		i

•By convention, we code the first WISLR link (0, 0), and place second WISLR link id in M column
•If there is an opposite direction present, this should be coded in the same manner

21

Median Cross-Over



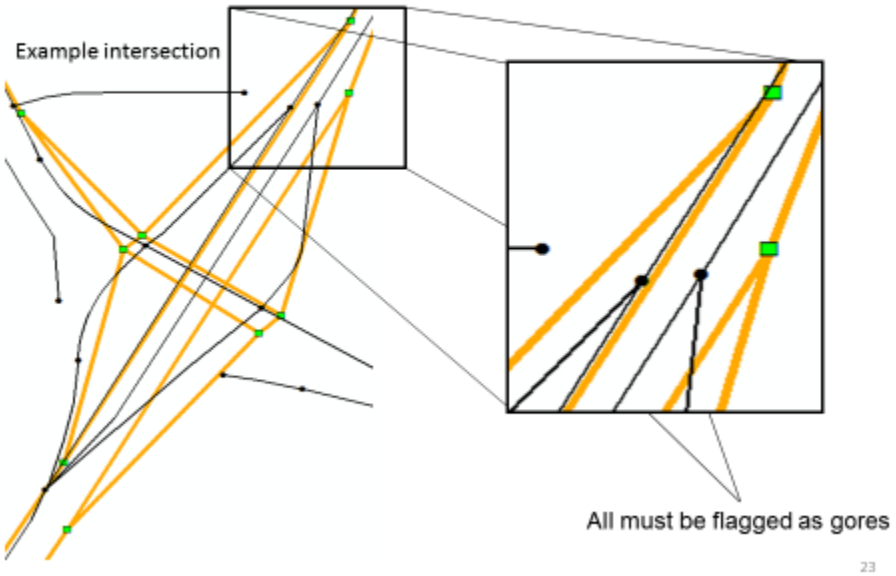
These links contain two directions of travel

- If median cross-over has no opposite WISLR link associated, a 1 is put in the M column
- Both directions must be coded, and since the opposite direction ends at the median crossover, the median cross-over link is coded using the full WISLR length

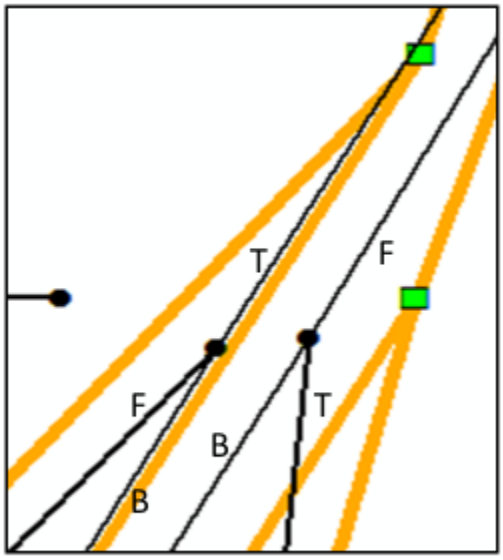
STNid	STNstart	STNend	WISLRid	WISLRstart	WISLRend	T	M
A	0	10	i	0	0		1
B	0	10	ii	524	524		1

22

Gore Points



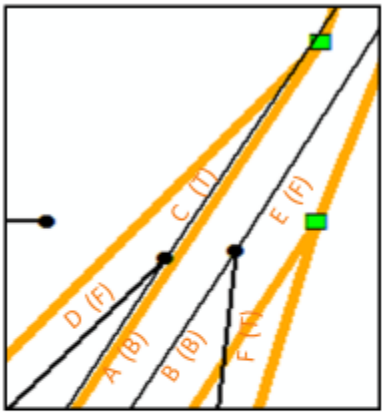
23



Gore Points

- T (To) – End of the link is a gore
- F (From) – Beginning of link is a gore
- B (Both) – Both beginning and end of link are gores

24



Gore Points

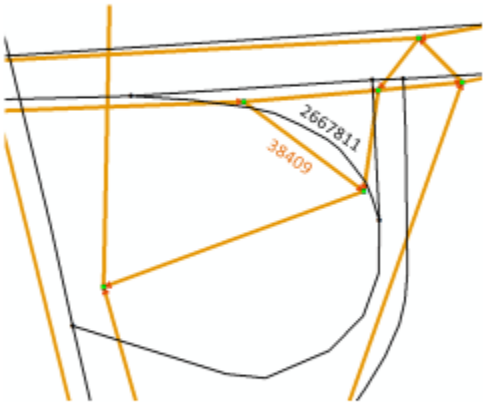
- Each site at a gore point will have three gore records associated with it in the link_link table
- Site will have either two T's, two F's, or any combination involving B's
- Gore point flag is for record in the link_link table

STNid	STNstart	STNend	WISLRid	WISLRstart	WISLRend	T	M	G
A	0	440	i	0	2587			B
B	0	490	ii	0	2510			B
C	0	220	iii	0	1282			T
D	0	240	iv	0	1019			F
E	0	1880	v	0	9865			F
F	0	1820	vi	0	9917			T

Confusion with Gore Point

26

Confusion with Gore Point



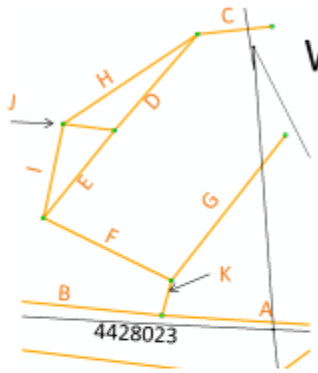
- Is it necessary to flag for all gore points ?
- If $\frac{WISLRlength}{STNlength} \approx 5.28$

marking in G column is not necessary

$$\frac{370}{70} \approx 5.28$$

STNid	STNstart	STNend	WISLRid	WISLRstart	WISLRend	T	M	G
38409	0	70	2667811	0	370			

27



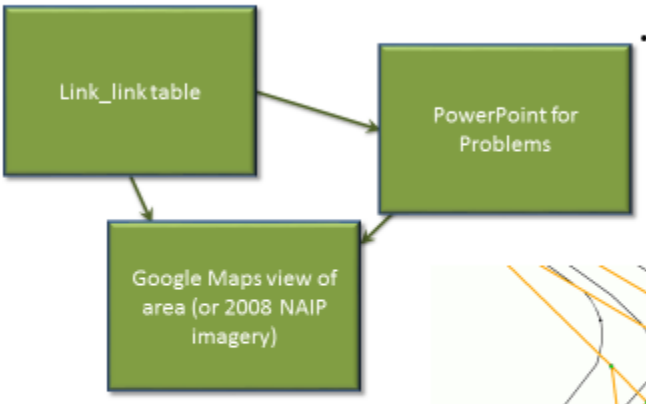
Weigh Stations, Park & Rides, Waysides

Weigh Station are not represented in WISLR

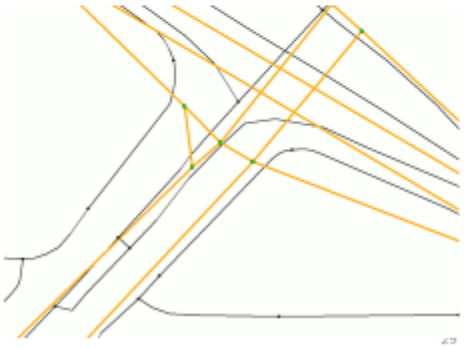
All weigh station links are placed at one location in WISLR and flagged

STNid	STNstart	STNend	WISLRid	WISLRstart	WISLRend	T	M	G	W
A	0	50	i	0	232				
B	0	280	i	232	1531				
C	0	30	i	232	232				1
D	0	30	i	232	232				1
E	0	30	i	232	232				1
F	0	50	i	232	232				1
G	0	100	i	232	232				1
H	0	50	i	232	232				1
I	0	30	i	232	232				1
J	0	10	i	232	232				1
K	0	20	i	232	232				1

Problem Flag



- Problems are typically areas where the coder cannot figure out a clear way to code the link_link table



Problem Flag

Possible erroneous WISLR links drawn in this area; total WISLR length of i & ii is more than twice the length of the STN link

STN Id	STN start	STN end	WISLR id	WISLR start	WISLR end	T	M	G	W	P	Coder	Date Mod.	Comments
A	0	20	i	0	264					1	ZR	4-6-10	WISLR link i, ii appear to be drawn inaccurately; total WISLR length is more than twice the equivalent STN length
A	20	150	ii	0	1742			T		1	ZR	4-6-10	"

Problem Flag Comments

- Comments should be written as:
  - -"Access Point STN ....." problem at access point where STN name or offset suspected to be wrong (common)
  - -"STN ...." problem with the drawn STN link or data
  - -"WISLR ...." problem with the drawn WISLR link or data
- This allows sorting of the comments and problem records by category or Access Points, STN, & WISLR

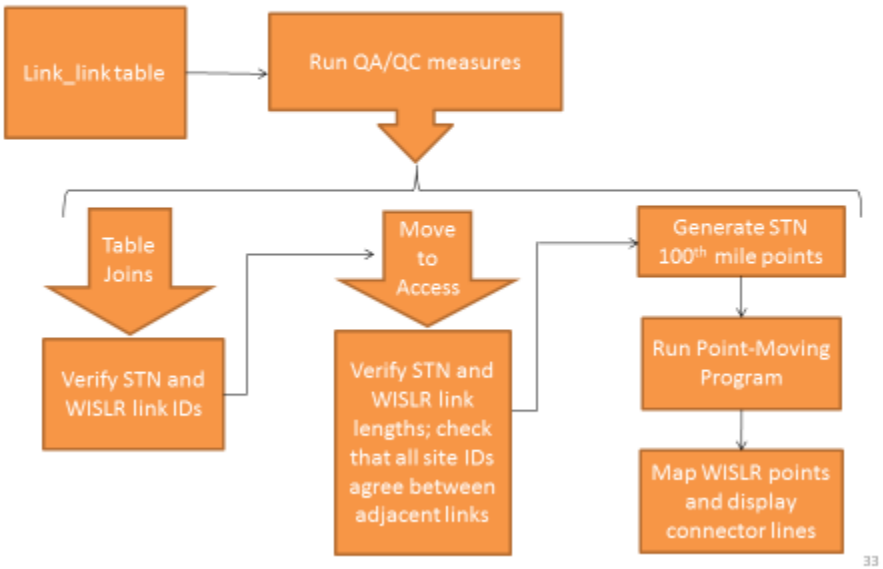
31

Ratio/Access Point Decisions

- **In the event that an STN Access Point is not labeled the same as a WISLR local road, but the access point appears to match an intersecting WISLR link, the following procedure will be used:**
  - A ratio will be performed to determine the location of the intersection along the STN link
  - If that ratio value is within ± 10 units of the Access Point offset, then the Access Point is assumed to represent the intersection
  - The Access Point Offset will be used
  - This record is then flagged with a "1" in the P column
  - If the ratio is outside the ± 10 unit range, the ratio value is used rather than the Access Point offset

In this case, Opals ratio was calculated to be within 10 units of the offset given in the Valley LA Access Point, so that Access Point offset is used in the link_link table and a problem flag is indicated

Post Processing



XI. Appendix: Point Generator and Mover Program

```
import os
import subprocess

###I/O VARIABLES###

inputSHP = arcpy.GetParameter(0)
WorkDir = arcpy.GetParameterAsText(1)
llFile = arcpy.GetParameterAsText(2)
interval = float (arcpy.GetParameterAsText(3))
#STN_PTS = arcpy.SetParameter(4, STN_PTS)
#WISLR_PTS = arcpy.SetParameterAsText(5, newtable)

###################

summarized =
arcpy.Statistics_analysis(inputSHP,WorkDir+"//STN_SUMD.DBF",[["FROM_TO_DI","SUM"]
],"RDWY_LINK_")
arcpy.AddField_management(summarized,"LINKID","DOUBLE")
arcpy.AddField_management(summarized,"OFFSET","DOUBLE")
arcpy.AddField_management(summarized,"UID","DOUBLE")
fields = arcpy.ListFields(summarized)
arcpy.CalculateField_management(summarized,"LINKID","!"+fields[1].name+"!","PYTHON")
arcpy.CalculateField_management(summarized,"OFFSET","!"+fields[3].name+"!","PYTHON")
STN_PTS = arcpy.CreateTable_management(WorkDir,"STN_Pts.dbf",summarized)

links = arcpy.SearchCursor(summarized)
Cursor = arcpy.InsertCursor(STN_PTS)
row = Cursor.newRow()
uid = 0
```

```
for link in links:
   n = 0
   while n < link.SUM_FROM_T:
      row.setValue("OFFSET",n)
      row.setValue("LINKID",link.RDWY_LINK_)
      row.setValue("UID",uid)
      uid += 1
      n = n + interval
      Cursor.insertRow(row)
   row.setValue("OFFSET", link.OFFSET)
   row.setValue("LINKID",link.LINKID)
   row.setValue("UID",uid)
   uid += 1
   Cursor.insertRow(row)

arcpy.SetParameter(4,STN_PTS)

## subprocess.check_call(["PointMove.exe","\"" + llFile + "\"","0",WorkDir])
## newtable = arcpy.CopyRows_management(WorkDir + "\\WISLR_PT.DBF",WorkDir +
"\\WISLRPTS.dbf")

##arcpy.SetParameterAsText(5,newtable)
```

XII. Appendix: Point Moving Program

```
import os
import subprocess

llFile = arcpy.GetParameterAsText(0)
workingDir = arcpy.GetParameterAsText(1)

subprocess.check_call(["PointMove.exe",llFile,"0",workingDir])
newtable = arcpy.CopyRows_management(workingDir + "\\WISLR_PT.DBF",workingDir +
"\\WISLRPTS.dbf")

arcpy.SetParameterAsText(2,newtable)
```

XIII. Appendix: XY Connector Program

```
import arcpy, sys, os

XYTable = arcpy.GetParameterAsText(0)
workingDir = arcpy.GetParameterAsText(1)

## Create ConnectorLine Feature Class ##
XYLines =
arcpy.CreateFeatureclass_management(workingDir,"XYLines.shp","POLYLINE");
in_rows = arcpy.SearchCursor(XYTable)
point = arcpy.Point()
array = arcpy.Array()
featureList = []
cursor = arcpy.InsertCursor(XYLines)
feat = cursor.newRow()
for in_row in in_rows:
    point.X = in_row.POINT_X
    point.Y = in_row.POINT_Y
    array.add(point)
    point.X = in_row.POINT_X_1
    point.Y = in_row.POINT_Y_1
    array.add(point)
    polyline = arcpy.Polyline(array)
    array.removeAll()
    featureList.append(polyline)
    feat.shape = polyline
    cursor.insertRow(feat)
arcpy.SetParameterAsText(2, XYLines)
##
# >>> in_rows = arcpy.SearchCursor(r"C:\\Users\\jdcannon\\Documents\\xycon.dbf")
```

```
# >>> point = arcpy.Point()
# >>> array = arcpy.Array()
# >>> featureList = []
# >>> cursor = arcpy.InsertCursor(r"C:\\arcgis\Testfc2.shp")
# >>> feat = cursor.newRow()
# >>> for in_row in in_rows:
# ...     point.X = in_row.STN_X
# ...     point.Y = in_row.STN_Y
# ...     array.add(point)
# ...     point.X = in_row.WISLR_X
# ...     point.Y = in_row.WISLR_Y
# ...     array.add(point)
# ...     polyline = arcpy.Polyline(array)
# ...     array.removeAll()
# ...     featureList.append(polyline)
# ...     feat.shape = polyline
# ...     cursor.insertRow(feat)
##
```

XIV. Appendix: Date Population Program

...write\DatePopulationRewrite\DatePopulationRewrite\Link.cs 1

```
1  //Link is sort of the appropriate name for this class
2
3  using System.Data;
4  using System.Collections.Generic;
5  using System.IO;
6
7  namespace DatePopulationRewrite
8  {
9      class Link
10     {
11
12         public string stnId { get; set; }
13         public string wislrId { get; set; }
14         public string usedStartDate { get; set; } //Holds latest start valid
15         public string usedEndDate { get; set; } //Holds latest end valid
16         public int errors { get; set; }
17
18         private void writeError(string errorText)
19         {
20
21             errors++;
22
23             if (!File.Exists("ErrorList.txt")) //Check if error list exists
24                 using (StreamWriter streamWriter = new StreamWriter
                       ("ErrorList.txt")) //Error list doesn't exist so create a new
                       one
25                 {
26
27                     streamWriter.WriteLine(errorText);
28
29                 }
30             else
31                 using (StreamWriter streamWriter = File.AppendText
                       ("ErrorList.txt")) //Error list exists so append to it
32                 {
33
34                     streamWriter.WriteLine(errorText);
35
36                 }
37
38         }
39
40         public Link(DataTable dataTable, Dictionary<string, DataRow> stnDate,
             Dictionary<string, DataRow> wislrDate, int row)
41         {
42
43             errors = 0;
44
45             //Get the stn and wislr id from the link_link (first) table
46             stnId = dataTable.Rows[row].ItemArray[0].ToString();
47             wislrId = dataTable.Rows[row].ItemArray[3].ToString();
48
```

```
49              //Find stn link in stn date table
50              DataRow stnDateRow = dataTable.NewRow();
51              if (stnDate.ContainsKey(stnId)) //Make sure the stnid exists in the
                  date table
52              {
53
54                  stnDateRow = stnDate[stnId]; //Get the stn date row based on stn
                      id
55
56              }
57              else
58              {
59
60                  writeError("No matching date for the stnid '" + stnId + "'"); //
                      Write errors to list
61
62              }
63
64              //Find wislr link in wislr date table
65              DataRow wislrDateRow = dataTable.NewRow(); //Make sure the wislr id
                  exists in the date table
66              if (wislrDate.ContainsKey(wislrId))
67              {
68
69                  wislrDateRow = wislrDate[wislrId];  //Get the wislr date row based
                      on wislr id
70
71              }
72              else
73              {
74
75                  writeError("No matching date for the wislrid '" + wislrId +
                      "'"); //Write errors to list
76
77              }
78
79
80              //Compare wislr and stn dates then get larger date via datetools
81              usedStartDate = DateTools.getLaterDate(stnDateRow.ItemArray
                  [1].ToString(), wislrDateRow.ItemArray[1].ToString());
82              usedEndDate = DateTools.getLaterDate(stnDateRow.ItemArray[2].ToString
                  (), wislrDateRow.ItemArray[2].ToString());
83
84          }
85
86      }
87
88  }
89
```

...\DatePopulationRewrite\DatePopulationRewrite\DateTools.cs 1

```
1  //Date tools takes 2 dates in (as strings) and returns the latest date of the 2.
2
3  using System;
4
5  namespace DatePopulationRewrite
6  {
7      sealed class DateTools
8      {
9
10         //Returns greater date of the 2 passed in
11         public static string getLaterDate(string date1, string date2) //Who needs
             singletons when we can just have static methods
12         {
13
14             if (!string.IsNullOrEmpty(date1) && !string.IsNullOrEmpty(date2)) //
                 Make sure the date strings aren't null
15             {
16
17                 //convert date strings to DateTime
18                 DateTime convertedDate1 = Convert.ToDateTime(date1);
19                 DateTime convertedDate2 = Convert.ToDateTime(date2);
20
21                 if (DateTime.Compare(convertedDate1, convertedDate2) < 0) //Check
                     if date1 is earlier
22                 {
23
24                     return date2;
25
26                 }
27
28                 return date1; //happens if dates are same or date1 is bigger
29
30             }
31
32             return string.IsNullOrEmpty(date2) ? date1: date2; //Return the date
                 that isn't empty
33
34         }
35
36     }
37 }
38
```

...ePopulationRewrite\DatePopulationRewrite\DataBaseTools.cs 1

```
 1  using System;
 2  using System.Collections.Generic;
 3  using System.Data;
 4  using System.Data.OleDb;
 5  using System.Windows;
 6  using System.IO;
 7  using System.Windows.Shell;
 8
 9  namespace DatePopulationRewrite
10  {
11
12      public delegate void ProgressUpdated(object sender,
          DBToolsProgressUpdatedEventArgs e); //Lets us callback to the mainwindow
          when progress changes
13      public delegate void FatalError(object sender, DBToolsFatalErrorEventArgs
          e); //Lets us callback to the mainwindow when we get a bad error
14
15      class DataBaseTools
16      {
17
18          int currentProgress;
19          string currentProcess;
20
21          public bool shouldAbort = false; //This flag is checked in case we catch
              a bad error
22
23          public event ProgressUpdated progressUpdated;
24          public event FatalError didReceiveFatalError;
25
26          const int Link_Table = 0; //Link table index
27          const int STN_Date_Table = 1; //STN table index
28          const int WISLR_Date_Table = 2; //WISLR table index
29
30          private void throwError(string errorMessage)
31          {
32
33              shouldAbort = true; //Make sure we stop the operation
34              progressUpdated(this, new DBToolsProgressUpdatedEventArgs
                  (currentProgress, currentProcess,
                  TaskbarItemProgressState.Error)); //Show error in progress bar
35              didReceiveFatalError(this, new DBToolsFatalErrorEventArgs
                  (errorMessage)); //Display our masterfully created error message
36
37          }
38
39          //Reads in all fields from each of the tables the user has selected
40          public DataTable[] getDataTables(string[] filePaths, string[] tables)
41          {
42
43              List<DataTable> dataTables = new List<DataTable>();
44
45              currentProgress = 0;
```

```
46              currentProcess = "Reading Tables";
47              progressUpdated(this, new DBToolsProgressUpdatedEventArgs
                  (currentProgress, currentProcess,
                  TaskbarItemProgressState.Normal)); //Report initial progress so the
                   textblock text is set
48
49              for (int i = 0; i < filePaths.Length; i++) //iterate through the
                  filepaths list
50              {
51
52                  if (!shouldAbort)
53                  {
54
55                      dataTables.Add(getDataTable(filePaths[i], tables[i])); //Gets
                         the datatable from the table name at the filepath
56
57                      currentProgress = Convert.ToInt32(i / Convert.ToDouble
                        (filePaths.Length - 1) * 100);
58                      progressUpdated(this, new DBToolsProgressUpdatedEventArgs
                        (currentProgress, currentProcess,
                        TaskbarItemProgressState.Normal)); //Update the progressbar
                        thing
59
60                  }
61
62              }
63
64              return dataTables.ToArray(); //Return the datatables list as an array
65
66          }
67
68          //Returns a datatable from the filepath with the table name
69          public DataTable getDataTable(string filePath, string table)
70          {
71
72              List<string> dbConnectionStrings = new List<string>();
73              List<string> selectionQueries = new List<string>();
74              DataTable dataTable = new DataTable();
75
76              using (OleDbConnection connection = new OleDbConnection
                  ("Provider=Microsoft.ACE.OLEDB.12.0;Data Source=" + filePath))
77              using (OleDbCommand selectTablesCommand = new OleDbCommand("Select
                  *FROM [" + table + "]", connection)) //When we use the "using" like
                   this OleDBConnection and OleDBCommand will be released once we are
                   done with it and our connection is closed
78              {
79
80                  connection.Open(); //Open a connection to the database file
                      containing the selected table
81                  OleDbDataReader reader = selectTablesCommand.ExecuteReader(); //
                      Create a dbreader from the dbcommand
82                  dataTable.Load(reader); //Read in the opened datatable and store
```

```
                    it as a datatable object
83
84              }
85
86              return dataTable; //Return the datatable
87
88          }
89
90          //Returns a link array with start and end valid dates
91          private Link[] getLinksWithDates(string[] filePaths, string[] tables)
92          {
93
94              List<Link> links = new List<Link>();
95              DataTable[] dataTables = getDataTables(filePaths, tables); //get the
                  table the user has selected
96              List<Dictionary<string, DataRow>> dateList = new
                  List<Dictionary<string, DataRow>>(); //We need to look up the dates
                   based on wislr and stn id so dictionaries make the lookup almost
                  instant
97              double progress = 0.0;
98              int errorCount = 0;
99
100             File.Delete("ErrorList.txt"); //We don't want the old errors showing
                  up for this pass so lets remove the old one
101
102             for (int i = 1; i < dataTables.Length; i++) //Start i at 1 because we
                    need only the first and second tables in the dataTables array
103             {
104
105                 dateList.Add(new Dictionary<string, DataRow>()); //Add a new
                      dictionary to the dictionary list
106
107                 foreach (DataRow row in dataTables[i].Rows) //Iterate over each
                      row in the datatables
108                 {
109
110                     dateList[i - 1][row.ItemArray[0].ToString()] = row; //Add the
                          row to the dictionary the first should be stn and the second
                          wislr
111
112                 }
113
114             }
115
116             for (int i = 0; i < dataTables[0].Rows.Count; i++)
117             {
118                 switch (shouldAbort) //Check if we should abort before every
                      iteration
119                 {
120
121                     case false: //Shouldn't abort
122
```

```
123                         try //This may fail if the user didn't select the tables in the correct sequence
124                         {
125
126                             Link link = new Link(dataTables[0], dateList[0], dateList[1], i);//Create a link from the tables and current row
127                             links.Add(link); //Add the newly created link to the links list
128                             errorCount += link.errors;
129
130                             currentProgress = Convert.ToInt32(++progress / Convert.ToDouble(dataTables[0].Rows.Count) * 100);
131                             currentProcess = "Comparing Dates";
132                             progressUpdated(this, new DBToolsProgressUpdatedEventArgs(currentProgress, currentProcess, TaskbarItemProgressState.Normal)); //Fire the progress updated event
133
134                         }
135                         catch (Exception e) //Hopefully this never happens
136                         {
137
138                             Console.WriteLine(e);
139                             throwError("This exception is usually thrown when the link, WISLR, and STN tables are not opened in the correct order, or an STN or WISLR date cannot be found in the STN or WISLR date table. Crashed on row " + (i + 1) + ".");  //Fire the fatal error event and prepare for death
140
141                         }
142
143                         break;
144
145                     default: //Something is broken if this happens
146
147                         break;
148
149                 }
150
151             }
152
153             if (errorCount > 0)
154             {
155
156                 progressUpdated(this, new DBToolsProgressUpdatedEventArgs
                      (currentProgress, currentProcess,
                      TaskbarItemProgressState.Error)); //Show error in taskbar
157
158                 if (MessageBox.Show("There were " + errorCount + " errors found
                      in the date comparing process. Would you like like to write to
                      the access file anyway?", "", MessageBoxButton.YesNo,
```

```
                    MessageBoxImage.Exclamation) == MessageBoxResult.No) //Check if
                     user wants to open the file we wrote the dates to
159                 {
160
161                     shouldAbort = true;
162
163                 }
164
165                 System.Diagnostics.Process.Start("notepad.exe",
                      "ErrorList.txt"); //Open the error list file
166
167             }
168
169             return links.ToArray(); //Return the completed links array containing
                   links with start and end valid dates
170
171         }
172
173         //Writes dates to the first table in the tables array
174         private void writeDatesToTable(Link[] links, string filePath, string
              table)
175         {
176
177             string[] columns = { "Start_Valid", "End_Valid" }; //Define our start
                   and end valid column names
178
179             try
180             { //Something may break here but we hope not
181
182                 using (OleDbConnection connection = new OleDbConnection
                      ("Provider=Microsoft.ACE.OLEDB.12.0;Data Source=" +
                      filePath)) //Create a connection!
183                 {
184
185                     connection.Open();
186
187                     foreach (string column in columns) //Iterate over the column
                       array these next several is absolutely hideous but it must be
                        done or we crash =/
188                     {
189
190                         using (OleDbCommand updateCommand = new OleDbCommand
                        ("UPDATE [" + table + "] SET " + column + " = @date WHERE
                        STNid = @stnId AND WISLRid = @wislrId", connection)) //When
                        we use the "using" like this OleDBConnection should be
                        discarded once it's done
191                         {
192
193                             OleDbDataAdapter dataAdapter = new OleDbDataAdapter
                        (updateCommand);
194                             //Add parameters to write values in tables the date
                        is written while the stn and wislr ids are used to lookup
```

```
                        rows
195                     updateCommand.Parameters.Add("@date",
                        OleDbType.DBDate);
196                     updateCommand.Parameters.Add("@stnId",
                        OleDbType.Integer, int.MaxValue);
197                     updateCommand.Parameters.Add("@wislrId",
                        OleDbType.Integer, int.MaxValue);
198
199                     updateCommand.Prepare();
200                     OleDbTransaction transaction =
                        connection.BeginTransaction(); //Somehow this speeds things
                        up a bit
201                     updateCommand.Transaction = transaction;
202
203                     for (int i = 0; i < links.Length; i++) //Begin
                        iteration over links
204                     {
205
206                         if (!shouldAbort)
207                         {
208
209                             Link link = links[i];
210
211                             switch (column == columns[0])
212                             {
213
214                                 case true: //If is start valid
215
216                                     //Set values of sql parameters (this
                                    all probably should be in a method)
217                                     updateCommand.Parameters
                                    ["@date"].Value = Convert.ToDateTime(link.usedStartDate);
218                                     updateCommand.Parameters
                                    ["@stnId"].Value = link.stnId;
219                                     updateCommand.Parameters
                                    ["@wislrId"].Value = link.wislrId;
220                                     updateCommand.ExecuteNonQuery(); //
                                    Executes the query built from the insertCommand and these
                                    params
221
222                                     break;
223
224                                 default: //end valid
225
226                                     switch (string.IsNullOrEmpty
                                    (link.usedEndDate)) //Check if we have a date
227                                     {
228
229                                         case false: //Write the date if
                                        we have one
230
231                                             updateCommand.Parameters
```

```
                        ["@date"].Value = Convert.ToDateTime(link.usedEndDate);
232                                             updateCommand.Parameters
                        ["@stnId"].Value = link.stnId;
233                                             updateCommand.Parameters
                        ["@wislrId"].Value = link.wislrId;
234                                             updateCommand.ExecuteNonQuery
                        (); //Executes the query built from the insertCommand and
                        these params
235
236                                             break;
237
238                                         default: //If both dates were
                        null do nothing (if we try to write a blank date here ms
                        access complains)
239                                             break;
240
241                                     }
242
243                                     break;
244
245                                 }
246
247                             }
248
249                             currentProgress = Convert.ToInt32(i /
                        Convert.ToDouble(links.Length - 1) * 100);
250                             currentProcess = "Writing to " + column;
251                             progressUpdated(this, new
                        DBToolsProgressUpdatedEventArgs(currentProgress,
                        currentProcess, TaskbarItemProgressState.Normal)); //Update
                        progress
252
253                         }
254
255                         transaction.Commit();
256                         transaction.Dispose();
257
258                     }
259
260                 }
261
262             }
263
264         }
265         catch (Exception e) //This shouldn't happen much
266         {
267
268             Console.WriteLine(e);
269             throwError("An error occurred during the write process. Please
                make sure the Microsoft Access file being modified was not
                moved or deleted during the process."); //Heres our pretty
                general error message
```

```
270
271             }
272
273         }
274
275         //Returns an array containing table names for each of the tables in the
              hopefully existing access file (usually called to populate comboboxes)
276         public string[] getTableNames(string filePath)
277         {
278
279             List<string> tableNames = new List<string>();
280
281             try
282             {
283
284                 using (OleDbConnection connection = new OleDbConnection
                      ("Provider=Microsoft.ACE.OLEDB.12.0;Data Source=" +
                      filePath)) //When we use the "using" like this OleDBConnection
                      will not be disposed once its done we hope
285                 {
286
287                     string[] restrictions = new string[4];
288                     restrictions[3] = "Table";
289
290                     connection.Open();
291                     DataTable tables = connection.GetSchema("Tables",
                       restrictions); //Returns a datatable containing rows
                       containing table names
292                     connection.Close();
293
294                     foreach (DataRow row in tables.Rows) //Iterate over the rows
                       in the table we got from the access file
295                     {
296
297                         tableNames.Add(row.ItemArray[2].ToString()); //In this
                      case (and hopefully every case) the table names where in the
                      third column so this adds them to the tablename list
298
299                     }
300
301                 }
302
303             }
304
305             catch (Exception e) //This won't happen normally unless the users
                  attempts to open that annoying temporary file access puts next to
                  the original file when a file is open otherwise this shouldn't be
                  possible
306             {
307
308                 Console.WriteLine(e);
309                 MessageBox.Show("Make sure that the file you are attempting to
```

```
                    open is a Microsoft Access file. Also, if you do not have the
                    Microsoft Access Database Engine 2010 Redistributable please
                    download it here: http://www.microsoft.com/en-us/download/
                    details.aspx?id=13255", "The file could not be opened.",
                    MessageBoxButton.OK, MessageBoxImage.Error); //This isn't
                    fatal! display the messagebox
310
311             }
312
313             return tableNames.ToArray(); //Everything worked lets return our
                  tableNames
314
315         }
316
317         //Writes the start and end valid dates to the first table in the first
              file
318         public void populateDates(string[] filePaths, string[] tables)
319         {
320
321             Link[] links = getLinksWithDates(filePaths, tables); //Creates a link
                  array with links that have start and end valid dates
322
323             if (!shouldAbort) //Check if we should continue
324             {
325
326                 writeDatesToTable(links, filePaths[0], tables[0]); //Write the
                      dates to the first table in the first file specified by the
                      user
327
328             }
329
330         }
331
332     }
333
334     //These EventArgs subclasses allow us to pass things to the event
335     public class DBToolsProgressUpdatedEventArgs : EventArgs
336     {
337
338         public int progress { get; } //Define our progress readonly property
              (displayed via progress bar)
339         public string currentProcess { get; } //Define our process readonly
              property (displayed via label above progress bar)
340         public TaskbarItemProgressState progressState { get; }
341
342         public DBToolsProgressUpdatedEventArgs(int progress, string process,
              TaskbarItemProgressState progressState)
343         {
344
345             this.progress = progress;
346             currentProcess = process;
347             this.progressState = progressState;
```

```
348
349         }
350
351     }
352
353     public class DBToolsFatalErrorEventArgs : EventArgs
354     {
355
356         public string error { get; } //Deine our error readonly property
              (displayed in body of messagebox)
357
358         public DBToolsFatalErrorEventArgs(string errorText)
359         {
360
361             error = errorText;
362
363         }
364
365     }
366
367 }
368
```

```
 1  //Created by Greyson Wright 2015
 2
 3  using System;
 4  using System.Collections.Generic;
 5  using System.Windows;
 6  using System.Windows.Controls;
 7  using System.Windows.Media;
 8  using Microsoft.Win32;
 9  using System.Windows.Media.Animation;
10  using System.ComponentModel;
11  using System.IO;
12
13  namespace DatePopulationRewrite
14  {
15
16      /// <summary>
17      /// Interaction logic for MainWindow.xaml
18      /// </summary>
19      public partial class MainWindow : Window
20      {
21
22          bool shouldClose = true; //Flag that is checked when application is
              closing if date pop is running it presents a messagebox saying it can't
               close
23          bool errorPresented = false; //Flag checked when an error is thrown
24
25          List<string> filePaths = new List<string>();
26
27          enum Pages { Link, WISLR, STN };
28          Pages currentPage = Pages.Link;
29
30          BackgroundWorker backgroundWorker; //This runs dbtools and such
31          DataBaseTools dbTools;
32
33          ProgressDialog progressDialog; //This pops up when dbtools runs
34          string currentProcess = ""; //Name of the current process running for
              progress bar (this is badly placed i know)
35          System.Windows.Shell.TaskbarItemProgressState progressState;
36
37          //#--Layout and animation--#
38
39          //-Animation events-
40          //Will be called once our animation in moveGridViews completes
41          private void Animation_Completed(object sender, EventArgs e)
42          {
43
44              mainContainer.IsEnabled = true; //This is set to false during page
                  transition so that the user cannot click things and break
                  everything
45
46          }
47
```

```
48          //Just a little method that moves the pages simulating a page change
49          private void moveGridViews(double newX)
50          {
51
52              TranslateTransform trans = new TranslateTransform();
53              grid0.RenderTransform = trans; //We animate each grid sepparately
                  because it was breaking if we moved the background
54              grid1.RenderTransform = trans;
55              grid2.RenderTransform = trans;
56              DoubleAnimation animation = new DoubleAnimation
                  (grid0.TransformToAncestor(mainContainer).Transform(new Point(0,
                  0)).X, newX, TimeSpan.FromSeconds(0.1)); //Create a fast animation
57              animation.Completed += Animation_Completed;
58              trans.BeginAnimation(TranslateTransform.XProperty, animation);
59
60          }
61
62          //-Page transition- (These methods should be called when next and back
              are pressed)
63          private void transitionToNextPageWithAnimation()
64          {
65
66              var currentX = grid0.TransformToAncestor(mainContainer).Transform(new
                   Point(0, 0)).X; //Gets the x pistion we need to move
67              moveGridViews(currentX - 300); //Moves the pages to the left from the
                   x position
68
69          }
70
71          private void transitionToPreviousPageWithAnimation()
72          {
73
74              var currentX = grid0.TransformToAncestor(mainContainer).Transform(new
                   Point(0, 0)).X; //Gets the x pistion we need to move
75              moveGridViews(currentX + 300); //Moves the pages to the right from
                  the x position
76
77          }
78
79          //-Setup-
80          //Called at startup and when Date pop resets
81          private void setUpComboBoxes()
82          {
83
84              //Clear table combobox dropdown
85              linkComboBox.Items.Clear();
86              wislrComboBox.Items.Clear();
87              stnComboBox.Items.Clear();
88
89              //Set combobox text to default
90              linkComboBox.Items.Add("Select Table...");
91              linkComboBox.Text = "Select Table...";
```

```
 92
 93              wislrComboBox.Items.Add("Select Table...");
 94              wislrComboBox.Text = "Select Table...";
 95
 96              stnComboBox.Items.Add("Select Table...");
 97              stnComboBox.Text = "Select Table...";
 98
 99          }
100
101          //Gets the current comboBox based on currentPage
102          private ComboBox getCurrentComboBox()
103          {
104
105              switch (currentPage)
106              {
107
108                  case Pages.Link:
109
110                      return linkComboBox;
111
112                  case Pages.STN:
113
114                      return stnComboBox;
115
116                  default:
117
118                      return wislrComboBox;
119
120              }
121
122          }
123
124          //Gets filePath based on page
125          private int getCurrentFilePathIndex()
126          {
127
128              switch (currentPage)
129              {
130
131                  case Pages.Link:
132
133                      return 0;
134
135                  case Pages.STN:
136
137                      return 1;
138
139                  default:
140
141                      return 2;
142
143              }
```

```
144
145         }
146
147         //Gets current label based on page
148         private Label getCurrentLabel()
149         {
150
151             switch (currentPage)
152             {
153
154                 case Pages.Link:
155
156                     return linkLabel;
157
158                 case Pages.STN:
159
160                     return stnLabel;
161
162                 default:
163
164                     return wislrLabel;
165
166             }
167
168         }
169
170         //Called when we need to set everything back to the way it looked right
              after startup (it is usually called after date pop has finnished
              populating)
171         private void resetWindow()
172         {
173
174             setUpComboBoxes(); //Sets combo boxes back to default
175
176             //The following labels appear on the different pages we are just
                  setting their text back to normal
177             linkLabel.Content = "Open the file containing your link table.";
178             stnLabel.Content = "Open the file containing your STN Date Table.";
179             wislrLabel.Content = "Open the file containing your WISLR Date
                  table.";
180
181             currentPage = Pages.Link; //Pages.Link is the starting value of
                  currentPage
182             moveGridViews(0); //Set the content in the link page to appear (the
                  page change is animated so it kinda looks weird)
183             filePaths = new List<string>();
184
185             GC.Collect(); //This is an ok time to call collect
186
187             if (File.Exists("ErrorList.txt")) //Delete the error list once
                  completed
188                 File.Delete("ErrorList.txt");
```

```
189
190         }
191
192         //#--Window lifetime--#
193         public MainWindow()
194         {
195             InitializeComponent();
196
197         }
198
199         private void Window_Loaded(object sender, RoutedEventArgs e)
200         {
201
202             setUpComboBoxes(); //Clears our comboboxes out and sets the select
                  table text
203
204         }
205
206         private void Window_Closing(object sender,
              System.ComponentModel.CancelEventArgs e)
207         {
208
209             e.Cancel = !shouldClose;
210
211             if (!shouldClose)
212             {
213
214                 MessageBox.Show("Please wait for the Date Population tool to
                      finish.", "", MessageBoxButton.OK,
                      MessageBoxImage.Exclamation); //Let the user know we can't quit
                       yet
215
216             }
217
218         }
219
220         //#--Buttons--#
221         private void openButton_Click(object sender, RoutedEventArgs e)
222         {
223
224             ComboBox currentComboBox; //Holds current combobox based on page
225             int filePathIndex = 0; //Holds filePath index based on page
226
227             OpenFileDialog openDialog = new OpenFileDialog();
228             openDialog.Filter = "Access 2007 (*.accdb)|*accdb|Access 2000-2003
                  (*.mdb)|*.mdb"; //Set file filter for MS Access files only
229
230             filePathIndex = getCurrentFilePathIndex();
231             currentComboBox = getCurrentComboBox();
232
233             if (openDialog.ShowDialog() == true) //Check if a file gets opened
234             {
```

```
235
236                 if (filePaths.Count != filePathIndex) //Check if the filepath has
                        been added to the filePaths list
237                 {
238
239                     filePaths[filePathIndex] = openDialog.FileName; //A file path
                         has been added to the list for this page so we need to
                        overwrite it here
240
241                 }
242                 else
243                 {
244
245                     filePaths.Add(openDialog.FileName); //Looks like we haven't
                        added a filepath for this page yet so add a new one
246
247                 }
248
249                 currentComboBox.Items.Clear(); //Clear out the existing items so
                      we don't have table names from different files
250                 currentComboBox.Items.Add("Select Table...");
251                 currentComboBox.Text = "Select Table...";
252
253                 getCurrentLabel().Content = System.IO.Path.GetFileName
                      (openDialog.FileName); //Display the filename in the fileLabel
254                 dbTools = new DataBaseTools();
255                 foreach (string tableName in dbTools.getTableNames
                      (openDialog.FileName)) //Pull tables names out of the access db
                        file and iterate over them
256                 {
257
258                     currentComboBox.Items.Add(tableName); //Add the table names
                        to the comboxbox so the user can select the one he or she
                        wants
259
260                 }
261
262             }
263
264             dbTools = null;
265
266         }
267
268         private void nextButton_Click(object sender, RoutedEventArgs e)
269         {
270
271             if (getCurrentComboBox().Text != "Select Table...") //Check if user
                    has selected a table
272             {
273
274                 mainContainer.IsEnabled = false; //Disable the user interaction
                      while the page transitions
```

```
275
276                 if (grid0.TransformToAncestor(mainContainer).Transform(new Point
                      (0, 0)).X == 0) //If Link page is showing
277                 {
278
279                     currentPage = Pages.STN; //Set currentPage to middle page
280
281                 }
282                 else //If STN page is showing
283                 {
284
285                     currentPage = Pages.WISLR; //Set currentPage to last page
286
287                 }
288
289                 transitionToNextPageWithAnimation(); //Move the views to the left
290
291             }
292             else
293             {
294
295                 MessageBox.Show("You must select a table before advancing.", "",
                      MessageBoxButton.OK, MessageBoxImage.Exclamation); //Don't
                      allow the user to advance until they pick a table
296
297             }
298
299         }
300
301         private void backButton_Click(object sender, RoutedEventArgs e)
302         {
303
304             mainContainer.IsEnabled = false; //Disable user interaction during
                  transition
305
306             if (grid0.TransformToAncestor(mainContainer).Transform(new Point(0,
                  0)).X == -300) // if STN page is showing
307             {
308
309                 currentPage = Pages.Link; //Set currentPage to first page
310
311             }
312             else //If last page is showing
313             {
314
315                 currentPage = Pages.STN; //Set currentPage to middle page
316
317             }
318
319             transitionToPreviousPageWithAnimation(); //Move views to the right
320
321         }
```

```
322
323         private void goButton_Click(object sender, RoutedEventArgs e)
324         {
325
326             if (getCurrentComboBox().Text != "Select Table...") //Check if user
                  has selected a table
327             {
328
329                 shouldClose = false; //Don't let the user close date pop while
                      its populating dates
330                 string[] tables = { linkComboBox.Text, stnComboBox.Text,
                      wislrComboBox.Text }; //Get table names from the comboboxes
331
332                 //Setup our background worker
333                 backgroundWorker = new BackgroundWorker();
334                 backgroundWorker.WorkerReportsProgress = true;
335                 backgroundWorker.WorkerSupportsCancellation = true;
336                 backgroundWorker.DoWork += backgroundWorker_DoWork;
337                 backgroundWorker.RunWorkerCompleted +=
                      backgroundWorker_RunWorkerCompleted;
338                 backgroundWorker.ProgressChanged +=
                      backgroundWorker_ProgressChanged;
339                 backgroundWorker.RunWorkerAsync(tables); //Start background
                      worker
340
341                 progressDialog = new ProgressDialog();
342                 progressDialog.WindowStartupLocation =
                      WindowStartupLocation.CenterOwner;
343                 progressDialog.Owner = this;
344                 progressDialog.cancelWork += ProgressDialog_cancelWork;
345                 progressDialog.ShowDialog(); //Present progress bar
346
347             }
348
349             else
350             {
351
352                 MessageBox.Show("You must select a table before advancing.", "",
                      MessageBoxButton.OK, MessageBoxImage.Exclamation); //Don't let
                      the user continue until a table is selected
353
354             }
355
356         }
357
358         //#--Background Worker--#
359         //Fired when backgroundWorker.startasync is called (executes on a
              different thread)
360         private void backgroundWorker_DoWork(object sender, DoWorkEventArgs e)
361         {
362
363             dbTools = new DataBaseTools();
```

```
364             dbTools.progressUpdated += dbTools_ProgressUpdated;
365             dbTools.didReceiveFatalError += dbTools_didReceiveFatalError;
366             dbTools.populateDates(filePaths.ToArray(), (string[])e.Argument); //
                  Begin population
367
368         }
369
370         //Fired when backgroundWorker is done
371         private void backgroundWorker_RunWorkerCompleted(object sender,
              RunWorkerCompletedEventArgs e)
372         {
373
374             shouldClose = true; //The user now has permission to kill date pop so
                   don't get mad
375             progressDialog.closeDialog(); //Progress dialog should go away we
                  don't need it anymore
376             Dispatcher.Invoke(() => { //We have to invoke this other stuff
                  because its crossthreaded
377
378                 string firstTableName = linkComboBox.Text;//Holds the table name
                      that our work is written to
379                 string firstFilePath = filePaths[0];
380
381                 resetWindow(); //Resets window to default
382
383                 if (dbTools.shouldAbort == false)
384                 {
385
386                     if (MessageBox.Show("The Date Population tool is finished and
                          has updated the \"" + firstTableName + "\" table in" + "\""
                        + firstFilePath + "\". Would you like to open the file now?",
                          "", MessageBoxButton.YesNo, MessageBoxImage.Exclamation) ==
                        MessageBoxResult.Yes) //Check if user wants to open the file
                        we wrote the dates to
387                     {
388
389                         System.Diagnostics.Process.Start("msaccess.exe", "\"" +
                        firstFilePath + "\""); //This is a great way to open the file
                          so that access doesn't close when we do
390
391                     }
392
393                 }
394
395                 dbTools = null;
396
397             });
398
399         }
400
401         //Fired when backgroundworker.reportProgress is called
402         private void backgroundWorker_ProgressChanged(object sender,
```

```
            ProgressChangedEventArgs e)
403         {
404
405             progressDialog.setProgress(e.ProgressPercentage, currentProcess,
                  progressState); //Sets the progress of the progress bar
406
407         }
408
409         //#--DB Tools--#
410         //Fired when the progress changes (gives us access to the background
              worker from inside dbtools)
411         public void dbTools_ProgressUpdated(object sender,
              DBToolsProgressUpdatedEventArgs e)
412         {
413
414             currentProcess = e.currentProcess; //Sets the process name
415             progressState = e.progressState;
416             backgroundWorker.ReportProgress(e.progress); //Sets the progress
                  percentage of the progress bar
417
418         }
419
420         //Fired when we catch a fatal exception in dbtools
421         public void dbTools_didReceiveFatalError(object sender,
              DBToolsFatalErrorEventArgs e)
422         {
423
424             Dispatcher.Invoke(() => //Cross threaded operations mean we must
                  invoke this stuff
425             {
426
427                 if (!errorPresented) //Check if error is already presented
428                 {
429
430                     errorPresented = true; //This shouldn't happen more than once
                        or we get multiple messageboxes
431                     shouldClose = true; //Let the application close
432                     backgroundWorker.CancelAsync(); //Tell backgroundworker to
                        stop its work
433                     progressDialog.taskBarItemInfo.ProgressState =
                        System.Windows.Shell.TaskbarItemProgressState.Error;
434                     MessageBox.Show(e.error, "An unexpected exception has
                        occured!", MessageBoxButton.OK, MessageBoxImage.Error); //
                        Display error
435                     Application.Current.Shutdown(); //Stop the application
436
437                 }
438
439             });
440
441         }
442
```

```
443         private void ProgressDialog_cancelWork(object sender, EventArgs e)
444         {
445
446             dbTools.shouldAbort = true;
447
448         }
449
450     }
451
452 }
453
```

XV. Appendix: Table Merge SQL Statement

SELECT *

FROM Adams; UNION SELECT * FROM Ashland; UNION SELECT * FROM Barron; UNION SELECT * FROM Bayfield; UNION SELECT * FROM Brown; UNION SELECT * FROM Buffalo; UNION SELECT * FROM Burnett; UNION SELECT * FROM Calumet; UNION SELECT * FROM Chippewa; UNION SELECT * FROM Clark; UNION SELECT * FROM Columbia; UNION SELECT * FROM Crawford; UNION SELECT * FROM Dane; UNION SELECT * FROM Dodge; UNION SELECT * FROM Door; UNION SELECT * FROM Douglas; UNION SELECT * FROM Dunn; UNION SELECT * FROM Eau_Claire; UNION SELECT * FROM Florence; UNION SELECT * FROM Fond_Du_Lac; UNION SELECT * FROM Forest; UNION SELECT * FROM Grant; UNION SELECT * FROM Green; UNION SELECT * FROM Green_Lake; UNION SELECT * FROM Iowa; UNION SELECT * FROM Iron; UNION SELECT * FROM Jackson; UNION SELECT * FROM Jefferson; UNION SELECT * FROM Juneau; UNION SELECT * FROM Kenosha; UNION SELECT * FROM Kewaunee; UNION SELECT * FROM La_Crosse; UNION SELECT * FROM Lafayette; UNION SELECT * FROM Langlade; UNION SELECT * FROM Lincoln; UNION SELECT * FROM Manitowoc; UNION SELECT * FROM Marathon; UNION SELECT * FROM Marinette; UNION SELECT * FROM Marquette; UNION SELECT * FROM Menominee; UNION SELECT * FROM Milwaukee; UNION SELECT * FROM Monroe; UNION SELECT * FROM Oconto; UNION SELECT * FROM Oneida; UNION SELECT * FROM Outagamie; UNION SELECT * FROM Ozaukee; UNION SELECT * FROM Pepin; UNION SELECT * FROM Pierce; UNION SELECT * FROM Polk; UNION SELECT * FROM Portage; UNION SELECT * FROM Price; UNION SELECT * FROM Racine; UNION SELECT * FROM

Richland; UNION SELECT * FROM Rock; UNION SELECT * FROM Rusk; UNION SELECT * FROM Saint_Croix; UNION SELECT * FROM Sauk; UNION SELECT * FROM Sawyer; UNION SELECT * FROM Shawano; UNION SELECT * FROM Sheboygan; UNION SELECT * FROM Taylor; UNION SELECT * FROM Trempealeau; UNION SELECT * FROM Vernon; UNION SELECT * FROM Vilas; UNION SELECT * FROM Walworth; UNION SELECT * FROM Washburn; UNION SELECT * FROM Washington; UNION SELECT * FROM Waukesha; UNION SELECT * FROM Waupaca; UNION SELECT * FROM Waushara; UNION SELECT * FROM Winnebago; UNION SELECT * FROM Wood;

XVI. Appendix: Date Check Program

...e\date-check\Date Check Tool\Date Check Tool\DateTools.cs 1

```
1  //Date tools takes 2 dates in (as strings) and returns the latest date of the 2.
2  //Optimized for Date Check
3
4  using System;
5
6  namespace Date_Check_Tool
7  {
8      sealed class DateTools
9      {
10
11         //Returns greater date of the 2 passed in
12         public static string getLaterDate(string date1, string date2) //Who needs
             singletons when we can just have static methods
13         {
14
15             if (!string.IsNullOrEmpty(date1) && !string.IsNullOrEmpty(date2)) //
                 Make sure the date strings aren't null
16             {
17
18                 //convert date strings to DateTime
19                 DateTime convertedDate1 = Convert.ToDateTime(date1);
20                 DateTime convertedDate2 = Convert.ToDateTime(date2);
21
22                 if (DateTime.Compare(convertedDate1, convertedDate2) < 0) //Check
                     if date1 is earlier
23                 {
24
25                     return date2;
26
27                 }
28
29                 return date1; //happens if dates are same or date1 is bigger
30
31             }
32
33             return "NULL";
34         }
35
36     }
37
38 }
39
```

```
1  //Optimized for Date Check
2
3  using System;
4  using System.Collections.Generic;
5  using System.Linq;
6  using System.Threading.Tasks;
7  using System.Data;
8  using System.Data.OleDb;
9  using System.Windows;
10
11 namespace Date_Check_Tool
12 {
13
14     public delegate void ProgressUpdated(object sender,
         DBToolsProgressUpdatedEventArgs e); //Lets us callback to the mainwindow
         when progress changes
15     public delegate void FatalError(object sender, DBToolsFatalErrorEventArgs
         e); //Lets us callback to the mainwindow when we get a bad error
16
17     class DataBaseTools
18     {
19
20         bool shouldAbort = false; //This flag is checked in case we catch a bad
             error
21
22         public event ProgressUpdated progressUpdated;
23         public event FatalError didReceiveFatalError;
24
25         const int Link_Table = 0; //Link table index
26         const int STN_Date_Table = 1; //STN table index
27         const int WISLR_Date_Table = 2; //WISLR table index
28
29         //Reads in all fields from each of the tables the user has selected
30         public DataTable[] getDataTables(string[] filePaths, string[] tables)
31         {
32
33             List<DataTable> dataTables = new List<DataTable>();
34
35             progressUpdated(this, new DBToolsProgressUpdatedEventArgs(0, "Reading
                Tables")); //Report initial progress so the textblock text is set
36
37             for (int i = 0; i < filePaths.Length; i++) //iterate the filepaths
                 list
38             {
39
40                 dataTables.Add(getDataTable(filePaths[i], tables[i])); //Gets the
                    datatable from the table name at the filepath
41                 progressUpdated(this, new DBToolsProgressUpdatedEventArgs
                     (Convert.ToInt32(i / Convert.ToDouble(filePaths.Length - 1) *
                     100), "Reading Tables")); //Update progress
42
43             }
```

```
44
45              return dataTables.ToArray(); //Return the datatables list as an array
46
47          }
48
49          //Returns a datatable from the filepath with the table name
50          public DataTable getDataTable(string filePath, string table)
51          {
52
53              List<string> dbConnectionStrings = new List<string>();
54              List<string> selectionQueries = new List<string>();
55              DataTable dataTable = new DataTable();
56
57              using (OleDbConnection connection = new OleDbConnection
                  ("Provider=Microsoft.ACE.OLEDB.12.0;Data Source=" + filePath))
58              using (OleDbCommand selectTablesCommand = new OleDbCommand("Select
                  *FROM [" + table + "]", connection)) //When we use the "using" like
                   this OleDBConnection and OleDBCommand will be released once we are
                   done with it and our connection is closed
59              {
60
61                  connection.Open(); //Open a connection to the database file
                      containing the selected table
62                  OleDbDataReader reader = selectTablesCommand.ExecuteReader(); //
                      Create a dbreader from the dbcommand
63                  dataTable.Load(reader); //Read in the opened datatable and store
                      it as a datatable object
64
65              }
66
67              return dataTable; //Return the datatable
68
69          }
70
71          //Returns a link array with start and end valid dates
72          public Link[] getLinksWithDates(string[] filePaths, string[] tables)
73          {
74
75              List<Link> links = new List<Link>();
76              DataTable[] dataTables = getDataTables(filePaths, tables); //get the
                  table the user has selected
77              List<DataRow> stnRows = dataTables[STN_Date_Table].Rows.Cast<DataRow>
                  ().ToList();
78              List<DataRow> wislrRows = dataTables
                  [WISLR_Date_Table].Rows.Cast<DataRow>().ToList();
79              double progress = 0.0;
80
81              Parallel.For(0, dataTables[0].Rows.Count, (i, loopState) => //
                  Parallel for loops are awesome because they do stuff asynchronously
82              {
83
84                  switch (shouldAbort) //Check if we should abort before every
```

```
                    iteration
85                  {
86
87                      case false: //Shouldn't abort
88
89                          try //This may fail if the user didn't select the tables in the correct sequence
90                          {
91
92                              //Get the stn and wislr id from the link_link (first) table
93                              string stnId = dataTables[Link_Table].Rows[i].ItemArray[0].ToString();
94                              string wislrId = dataTables[Link_Table].Rows[i].ItemArray[3].ToString();
95
96                              Link link = new Link(dataTables, i);//Create a link from the tables and current row
97                              links.Add(link); //Add the newly created link to the links list
98
99                              progressUpdated(this, new DBToolsProgressUpdatedEventArgs(Convert.ToInt32(++progress / Convert.ToDouble(dataTables[0].Rows.Count) * 100), "Comparing Dates")); //Fire the progress updated event
100
101                         }
102                         catch (Exception e)
103                         {
104
105                             Console.WriteLine(e);
106                             shouldAbort = true; //Stop Checking
107                             didReceiveFatalError(this, new DBToolsFatalErrorEventArgs("This exception is usually thrown when the link, WISLR, and STN tables are not opened in the correct order, or an STN or WISLR date cannot be found in the STN or WISLR date table. Crashed on row " + (i + 1) + "."); //Fire the fatal error event and prepare for death
108
109                         }
110
111                         break;
112
113                     default:
114
115                         loopState.Break(); // basically this is the same as saying break; in a normal forloop
116
117                         break;
118
119                 }
120
```

```
121             });
122
123             return links.ToArray(); //Return the completed links array containing
                   links with start and end valid dates
124
125         }
126
127         //Writes dates to the first table in the tables array (remember thats the
               array in mainwindow.cs that gets passed into every method above this)
128         public void writeDatesToTable(Link[] links, string filePath, string
              table)
129         {
130
131             string[] columns = { "Start_Valid", "End_Valid" }; //Define our start
                   and end valid column names
132
133             try { //Something may break here but we hope not
134
135                 using (OleDbConnection connection = new OleDbConnection
                      ("Provider=Microsoft.ACE.OLEDB.12.0;Data Source=" +
                      filePath)) //Create a connection!
136                 {
137
138                     connection.Open();
139
140                     foreach (string column in columns) //Iterate over the column
                       array these next several is absolutely hideous but it must be
                        done or we crash =/
141                     {
142
143                         using (OleDbCommand insertCommand = new OleDbCommand
                        ("UPDATE " + table + " SET " + column + " = @date WHERE STNid
                         = @stnId AND WISLRid = @wislrId", connection)) //When we use
                         the "using" like this OleDBConnection should be discarded
                        once it's done
144                         {
145
146                             OleDbDataAdapter dataAdapter = new OleDbDataAdapter
                        (insertCommand);
147                             //Add parameters to write values in tables the date
                        is written while the stn and wislr ids are used to lookup
                        rows
148                             insertCommand.Parameters.AddWithValue("@date", "");
149                             insertCommand.Parameters.AddWithValue("@stnId", "");
150                             insertCommand.Parameters.AddWithValue("@wislrId",
                        "");
151
152                             for (int i = 0; i < links.Length; i++) //Begin
                        iteration over links
153                             {
154
155                                 Link link = links[i];
```

```
156
157                         switch (column == columns[0]) //This kinda looks bad probably could've been done better
158                         {
159
160                             case true: //If is start valid
161
162                                 //Set values of sql parameters
163                                 insertCommand.Parameters["@date"].Value = link.usedStartDate;
164                                 insertCommand.Parameters["@stnId"].Value = link.stnId;
165                                 insertCommand.Parameters["@wislrId"].Value = link.wislrId;
166                                 insertCommand.ExecuteNonQuery(); // Executes the query built from the insertCommand and these params
167
168                                 break;
169
170                             default: //end valid
171
172                                 switch (string.IsNullOrEmpty(link.usedEndDate)) //Check if we have a date
173                                 {
174
175                                     case false: //Write the date if we have one
176
177                                         //Set values of sql parameters
178                                         insertCommand.Parameters["@date"].Value = link.usedEndDate;
179                                         insertCommand.Parameters["@stnId"].Value = link.stnId;
180                                         insertCommand.Parameters["@wislrId"].Value = link.wislrId;
181                                         insertCommand.ExecuteNonQuery(); //Executes the query built from the insertCommand and these params
182
183                                         break;
184
185                                     default: //If both dates were null do nothing (if we try to write a blank date here ms access complains)
186                                         break;
187
188                                 }
189
190                                 break;
191
192                         }
```

```
193
194                                 progressUpdated(this, new
                            DBToolsProgressUpdatedEventArgs(Convert.ToInt32(i /
                            Convert.ToDouble(links.Length - 1) * 100), "Writing to " +
                            column)); //Fire the progressUpdated event
195
196                                 }
197
198                             }
199
200                         }
201
202                     }
203                 }
204                 catch(Exception ex)
205                 {
206
207                     Console.WriteLine(ex);
208                     shouldAbort = true; //Stop Date Check
209                     didReceiveFatalError(this, new DBToolsFatalErrorEventArgs("An
                        error occurred during the write process. Please make sure the
                        Microsoft Access file being modified was not moved or deleted
                        during the process.")); //Heres our pretty general error
                        message
210
211                 }
212
213             }
214
215             public void writeToTable(DataTable table, string filePath)
216             {
217
218                 using (OleDbConnection connection = new OleDbConnection
                      ("Provider=Microsoft.ACE.OLEDB.12.0;Data Source=" + filePath))
219                 using (OleDbCommand createCommand = new OleDbCommand("CREATE TABLE
                      [Date_Errors]([STNID] number, [WISLRID] number, [START_VALID] text,
                       [END_VALID] text)", connection))
220                 using (OleDbCommand deleteCommand = new OleDbCommand("DROP TABLE
                      [Date_Errors]", connection))
221                 using (OleDbDataAdapter dataAdapter = new OleDbDataAdapter("Select
                      *FROM [Date_Errors]", "Provider=Microsoft.ACE.OLEDB.12.0;Data
                      Source=" + filePath))
222                 using (OleDbCommandBuilder commandBuilder = new OleDbCommandBuilder
                      (dataAdapter))
223                 {
224                     Console.WriteLine("rows" + table.Rows.Count);
225                     connection.Open();
226
227                     try
228                     {
229
230                         createCommand.ExecuteNonQuery();
```

```
231
232                 }
233                 catch(Exception ex)
234                 {
235
236                     deleteCommand.ExecuteNonQuery();
237                     createCommand.ExecuteNonQuery();
238
239                     Console.WriteLine(ex);
240
241                 }
242
243                 dataAdapter.AcceptChangesDuringFill = true;
244                 dataAdapter.Fill(table);
245                 dataAdapter.Update(table);
246             }
247
248
249         }
250
251         //Returns an array containing table names for each of the tables in the
              hopefully existing access file (usually called to populate comboboxes)
252         public string[] getTableNames(string filePath)
253         {
254
255             List<string> tableNames = new List<string>();
256
257             try {
258
259                 using (OleDbConnection connection = new OleDbConnection
                      ("Provider=Microsoft.ACE.OLEDB.12.0;Data Source=" +
                      filePath)) //When we use the "using" like this OleDBConnection
                      will not be disposed once its done we hope
260                 {
261
262                     string[] restrictions = new string[4];
263                     restrictions[3] = "Table";
264
265                     connection.Open();
266                     DataTable tables = connection.GetSchema("Tables",
                         restrictions);
267                     connection.Close();
268
269                     foreach (DataRow row in tables.Rows) //Iterate over the rows
                         in the table we got from the access file
270                     {
271
272                         tableNames.Add(row.ItemArray[2].ToString()); //In this
                        case (and hopefully every case) the table names where in the
                        third column so this adds them to the tablename list
273
274                     }
```

```
275
276                 }
277
278             }
279
280             catch(Exception ex) //This won't happen normally unless the users attempts to open that annoying temporary file access creates when a file is open otherwise this shouldn't be possible
281             {
282
283                 Console.WriteLine(ex);
284
285                 MessageBox.Show("Make sure that the file you are attempting to open is a Microsoft Access file.", "The file could not be opened.", MessageBoxButton.OK, MessageBoxImage.Error); //This isn't fatal! display the messagebox
286
287             }
288
289             return tableNames.ToArray(); //Everything worked lets return our tableNames
290
291         }
292
293         //Writes the start and end valid dates to the first table in the first file
294         public void populateDates(string[] filePaths, string[] tables)
295         {
296
297             Link[] links = getLinksWithDates(filePaths, tables); //Creates a link array with links that have start and end valid dates
298
299             if (!shouldAbort) //Check if we should continue
300             {
301
302                 writeDatesToTable(links, filePaths[0], tables[0]); //Write the dates to the first table in the first file specified by the user
303
304             }
305
306         }
307
308     }
309
310     //These EventArgs subclasses allow us to pass things to the event
311     public class DBToolsProgressUpdatedEventArgs: EventArgs
312     {
313
314         public int progress { get; } //Define our progress readonly property (displayed via progress bar)
315         public string currentProcess { get; } //Define our process readonly
```

```
            property (displayed via label above progress bar)
316
317         public DBToolsProgressUpdatedEventArgs(int progress, string process)
318         {
319
320             this.progress = progress;
321             currentProcess = process;
322
323         }
324
325     }
326
327     public class DBToolsFatalErrorEventArgs : EventArgs
328     {
329
330         public string error { get; } //Deine our error readonly property
              (displayed in body of messagebox)
331
332         public DBToolsFatalErrorEventArgs(string errorText)
333         {
334
335             error = errorText;
336
337         }
338
339     }
340
341 }
342
```

```
1  //Created by Greyson Wright 2015
2
3  using System.Collections.Generic;
4  using System.Windows;
5  using Microsoft.Win32;
6  using System.ComponentModel;
7  using System.Data;
8  using System.IO;
9  using System;
10 using System.Linq;
11
12 namespace Date_Check_Tool
13 {
14     /// <summary>
15     /// Interaction logic for MainWindow.xaml
16     /// </summary>
17     public partial class MainWindow : Window
18     {
19
20         //Global variables bother me
21         BackgroundWorker backgroundWorker;
22         DataTable outTable;
23         string currentProcess;
24         string filePath;
25         string tableName;
26         bool errorPresented = false;
27         bool shouldClose = true;
28         int errorCount;
29
30         public MainWindow()
31         {
32             InitializeComponent();
33
34             //Displays select table in combo box
35             tableComboBox.Items.Add("Select Table...");
36             tableComboBox.Text = "Select Table...";
37
38         }
39
40         //A method that returns an array that will later be written to a text
             file each element is a line of a text file
41         private string[] getBadDates()
42         {
43
44             DataTable table;
45             outTable = new DataTable();
46             List<string> errorTypeCount = new List<string>();
47             List<string> writeContents = new List<string>();
48             string[] columNames = {"STNID", "WISLRID", "END_VALID",
                 "START_VALID"};
49
50             DataBaseTools dbTools = new DataBaseTools();
```

```
51              dbTools.progressUpdated += dbTools_ProgressUpdated;
52              dbTools.didReceiveFatalError += dbTools_didReceiveFatalError;
53              table = dbTools.getDataTable(filePath, tableName);
54
55              string stnId;
56              string wislrId;
57              string recordCreated;
58              string recordHistoric;
59              string startValid;
60              string endValid;
61              int localErrorCount; //Local errorCount keeps up with errors per row
                  while errorCount is total erros over the whole file the error count
                    is displayed once date check is done
62              int[] errors = { 0, 0, 0, 0 };
63
64              foreach (string name in columNames)
65              {
66
67                  outTable.Columns.Add(name);
68
69              }
70
71              for (int i = 0; i < table.Rows.Count; i++)
72              {
73
74                  stnId = table.Rows[i].ItemArray[0].ToString();
75                  wislrId = table.Rows[i].ItemArray[3].ToString();
76                  recordCreated = table.Rows[i].ItemArray[12].ToString();
77                  recordHistoric = table.Rows[i].ItemArray[13].ToString();
78                  startValid = table.Rows[i].ItemArray[14].ToString();
79                  endValid = table.Rows[i].ItemArray[15].ToString();
80                  localErrorCount = 0;
81
82                  writeContents.Add("STNID: " + stnId +" WISLRID: " + wislrId +
                      " {"); //The rows aren't always the same in our datatables and
                      the datatable in access so we give them the stnid and wislrid
83
84                  if (string.IsNullOrEmpty(recordHistoric) && !string.IsNullOrEmpty
                      (endValid)) //No end valid date for record historic
85                  {
86
87                      writeContents.Add(Environment.NewLine + "    There is no
                       Record Historic for End Valid (" + endValid + ")" +
                       Environment.NewLine);
88                      errors[0]++;
89                      localErrorCount++;
90
91                  }
92
93                  if (DateTools.getLaterDate(endValid, startValid) == startValid)
94                  {
95
```

```
 96                      writeContents.Add(Environment.NewLine + "    Start Valid is more recent than End Valid (" + endValid + ")" + Environment.NewLine); //End valid is earlier than the start valid
 97                      errors[1]++;
 98                      localErrorCount++;
 99
100                  }
101
102                  if (DateTools.getLaterDate(recordHistoric, startValid) == startValid) //Start valid is more recent than record historic
103                  {
104
105                      writeContents.Add(Environment.NewLine + "    Start Valid is more recent than Record Historic" + Environment.NewLine);
106                      errors[2]++;
107                      localErrorCount++;
108
109                  }
110
111                  if (DateTools.getLaterDate(recordCreated, startValid) == startValid) //Start valid is more recent than record created
112                  {
113
114                      writeContents.Add(Environment.NewLine + "    Start Valid is more recent than Record Created" + Environment.NewLine);
115                      errors[3]++;
116                      localErrorCount++;
117
118                  }
119
120                  if (localErrorCount == 0) //If we don't have errors here remove the opening line and brace
121                  {
122
123                      writeContents.RemoveAt(writeContents.Count - 1);
124
125                  }
126                  else //If we have local errors close it off with a brace and a new line
127                  {
128
129                      errorCount += localErrorCount; //Keep track of all errors
130                      writeContents.Add("}\n");
131
132                      DataRow currentRow = outTable.NewRow();
133                      currentRow["STNID"] = stnId;
134                      currentRow["WISLRID"] = wislrId;
135                      currentRow["END_VALID"] = endValid;
136                      currentRow["START_VALID"] = startValid;
137                      outTable.Rows.Add(currentRow);
138
```

```
139                 }
140
141             }
142
143             errorTypeCount.Add("Error types:\nThere is no Record Historic for End
                    Valid: " + errors[0] + "\n");
144             errorTypeCount.Add("Start Valid is more recent than End Valid: " +
                  errors[1] + "\n");
145             errorTypeCount.Add("Start Valid is more recent than Record Historic:
                  " + errors[2] + "\n");
146             errorTypeCount.Add("Start Valid is more recent than Record Created: "
                    + errors[3] + "\n" + Environment.NewLine);
147
148             return errorTypeCount.ToArray().Concat(writeContents.ToArray
                  ()).ToArray(); //Return array of lines
149
150         }
151
152         private void browseButton_Click(object sender, RoutedEventArgs e)
153         {
154
155             OpenFileDialog openDialog = new OpenFileDialog();
156             openDialog.Filter = "Access 2007 (*.accdb)|*accdb|Access 2000-2003
                  (*.mdb)|*.mdb"; //Set file filter for MS Access files only
157
158             if (openDialog.ShowDialog() == true) //Check if a file gets opened
159             {
160
161                 filePath = openDialog.FileName; //Looks like we haven't added a
                      filepath for this page yet so add a new one
162
163                 tableComboBox.Items.Clear(); //Clear out the existing items so we
                       don't have table names from different files
164                 tableComboBox.Items.Add("Select Table...");
165                 tableComboBox.Text = "Select Table...";
166
167                 nameLabel.Content = System.IO.Path.GetFileName
                      (openDialog.FileName); //Display the filename in the fileLabel
168                 DataBaseTools dbTools = new DataBaseTools();
169                 foreach (string tableName in dbTools.getTableNames
                      (openDialog.FileName)) //Pull tables names out of the access db
                       file and iterate over them
170                 {
171
172                     tableComboBox.Items.Add(tableName); //Add the table names to
                       the comboxbox so the user can select the one he or she wants
173
174                 }
175
176             }
177         }
178
```

```
179         private void goButton_Click(object sender, RoutedEventArgs e)
180         {
181
182             if (filePath != null && tableComboBox.Text != "Select table...") //
                  Check if user has selected a table
183             {
184
185                 shouldClose = false; //Don't let user close us until we finish
186                 tableName = tableComboBox.Text; //Get the tablename from the
                      combo box
187                 backgroundWorker = new BackgroundWorker(); //Initialize a
                      background worker
188                 backgroundWorker.WorkerReportsProgress = true;
189                 backgroundWorker.WorkerSupportsCancellation = true;
190                 backgroundWorker.DoWork += backgroundWorker_DoWork;
191                 backgroundWorker.RunWorkerCompleted +=
                      backgroundWorker_RunWorkerCompleted;
192                 backgroundWorker.RunWorkerAsync(tableComboBox.Text); //Start
                      background worker
193
194             }
195             else
196             {
197
198                 MessageBox.Show("Select a link table before proceeding.");
199
200             }
201
202         }
203
204         //#--Background Worker--#
205         //Fired when backgroundWorker.startasync is called (executes on a
              different thread)
206         private void backgroundWorker_DoWork(object sender, DoWorkEventArgs e)
207         {
208
209             string[] writeContents = getBadDates();
210             if (errorCount > 0) //Don't worry about writing if there are no
                  errors
211             {
212                 using (StreamWriter streamWriter = new StreamWriter
                      ("output.txt"))
213                 {
214
215                     for (int i = 0; i < writeContents.Length; i++) //Iterate over
                          the array
216                     {
217
218                         streamWriter.WriteLine(writeContents[i]); //Write each
                        element in the array to the file
219
220                     }
```

```
221
222                 }
223
224             }
225         }
226
227         //Fired when backgroundWorker is done
228         private void backgroundWorker_RunWorkerCompleted(object sender,
              RunWorkerCompletedEventArgs e)
229         {
230
231             shouldClose = true; //The user now has permission to kill date pop so
                  don't get mad
232             Dispatcher.Invoke(() => { //We have to invoke this other stuff
                  because its crossthreaded
233
234                 if (errorCount > 0) //Don't worrry about opening the file unless
                      we have errors
235                 {
236
237                     MessageBox.Show("The Date Check Tool finished and found
                        errors.");
238
239                     DataBaseTools dbTools = new DataBaseTools();
240                     dbTools.writeToTable(outTable, filePath);
241
242                     System.Diagnostics.Process.Start("msaccess.exe",
                        filePath); //Opens the text file in notepad
243                     System.Diagnostics.Process.Start("notepad.exe",
                        "output.txt"); //Opens the text file in notepad
244
245                 }
246                 else
247                 {
248
249                     MessageBox.Show("The Date Check Tool finished and found no
                        errors.");
250
251                 }
252
253                 errorCount = 0;
254
255             });
256
257         }
258
259         //#--DB Tools--#
260         //Fired when the progress changes (gives us access to the background
              worker from inside dbtools)
261         public void dbTools_ProgressUpdated(object sender,
              DBToolsProgressUpdatedEventArgs e)
262         {
```

```
263
264             currentProcess = e.currentProcess; //Sets the process name
265             backgroundWorker.ReportProgress(e.progress); //Sets the progress
                  percentage of the progress bar
266
267         }
268
269         //Fired when we catch a fatal exception in dbtools
270         public void dbTools_didReceiveFatalError(object sender,
              DBToolsFatalErrorEventArgs e)
271         {
272
273             Dispatcher.Invoke(() => //Cross threaded operations mean we must
                  invoke this stuff
274             {
275
276                 if (!errorPresented) //Check if error is already presented
277                 {
278
279                     errorPresented = true; //This shouldn't happen more than once
                      or we get multiple messageboxes
280                     shouldClose = true; //Let the application close
281                     backgroundWorker.CancelAsync(); //Tell backgroundworker to
                      stop its work
282                     MessageBox.Show(e.error, "An unexpected exception has
                      occured!", MessageBoxButton.OK, MessageBoxImage.Error); //
                      Display error
283                     Application.Current.Shutdown(); //Quit date check
284
285                 }
286
287             });
288
289         }
290
291         private void Window_Closing(object sender, CancelEventArgs e)
292         {
293
294             e.Cancel = !shouldClose;
295
296         }
297
298     }
299
300 }
301
```

APPENDIX B-NAMING CONVENTION AND RP DIAGRAMS

The document included in this appendix was developed in close collaboration with Wisconsin Department of Transportation staff during the initial phases of this research. The naming convention document establishes strict naming guidelines to be used during WISLR editing activities to ensure data integrity. The naming document also serves as a guide for WISLR editors to interpret Reference Point diagrams when naming and categorizing state highway features and has been incorporated into the data management plan at WisDOT.

Contents

Naming Convention for UA / ARNOLD / ILT Project

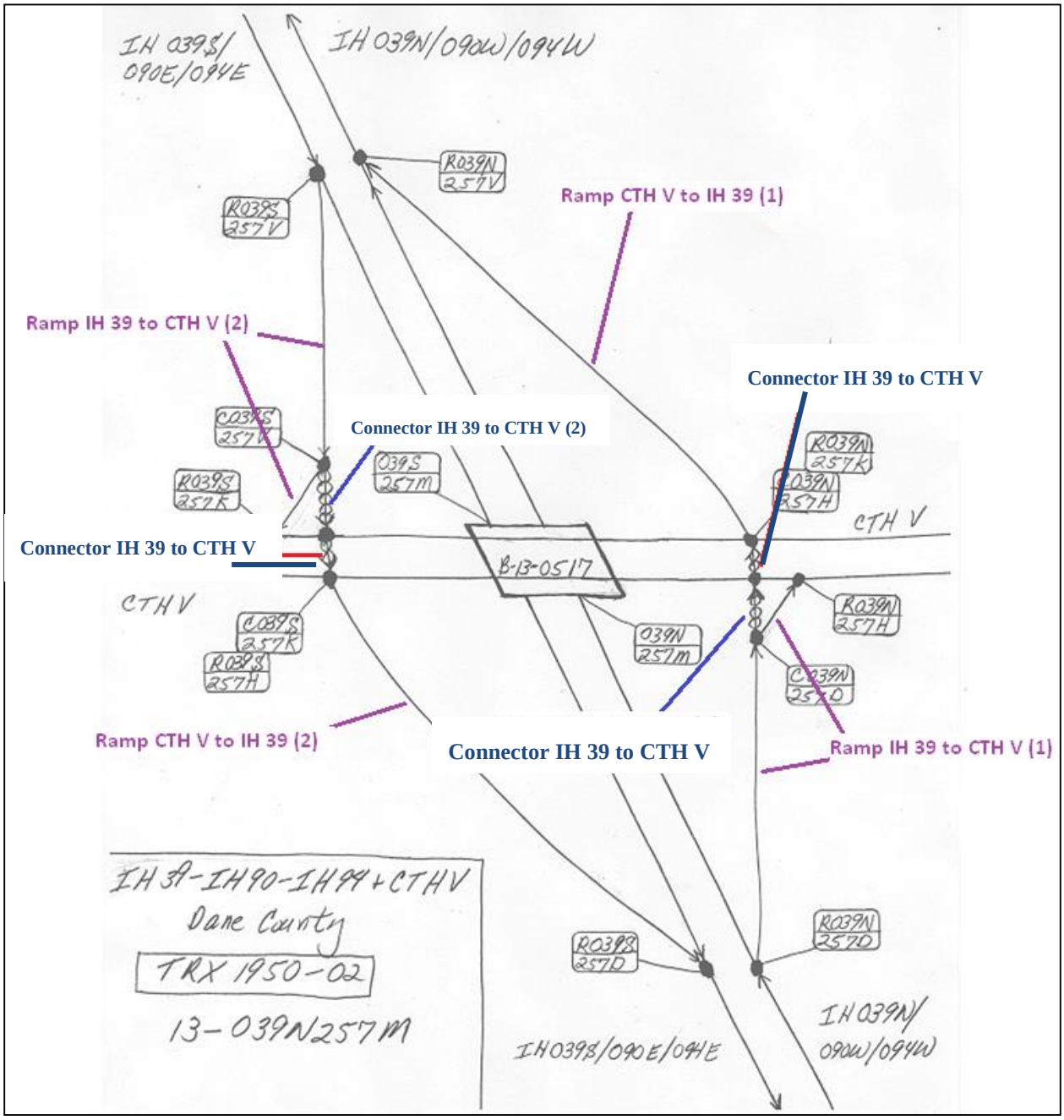
Highways and Facilities	WISLR LCM Naming Method
Mainline	USH 14, STH 14, IH 39
Ramps	Ramp USH 14 to CTH MM (2)*****
Connectors	Connector USH 141 to CTH MM (1)*
Connectors that connect two portions of the same highway	Connector USH 141
Connectors with Crossovers	Connector USH 141 to CTH MM (3)*
Roundabouts – Marked as Connectors	RAB USH 141 (1)*
Roundabouts – Marked as Mainline	USH 141
Frontage Road	Frontage Rd IH 43 (1) * ***€***
Weigh Stations	SWEF # 19 (*1*)ŧ
Brake Check Area	Brake Check Area USH 8 (1)ŧ
Park and Ride Lots	P&R College Ave (1)ŧ
Rest Areas	Rest Area # 22 (1)ŧ
Waysides	Wayside (1)ŧ
Turn Lanes	Turn Lane USH 141 to *"insert local name"*
J-Turns	J-Turn Connector USH 141 (1)
Business Route	USH B51

**** Extensions should start with cardinal directions (North or East as extension 1 (1)). Specifically, extension numbering shall follow the road labeling. Only necessary if more than one route within a given municipality share the same name.***

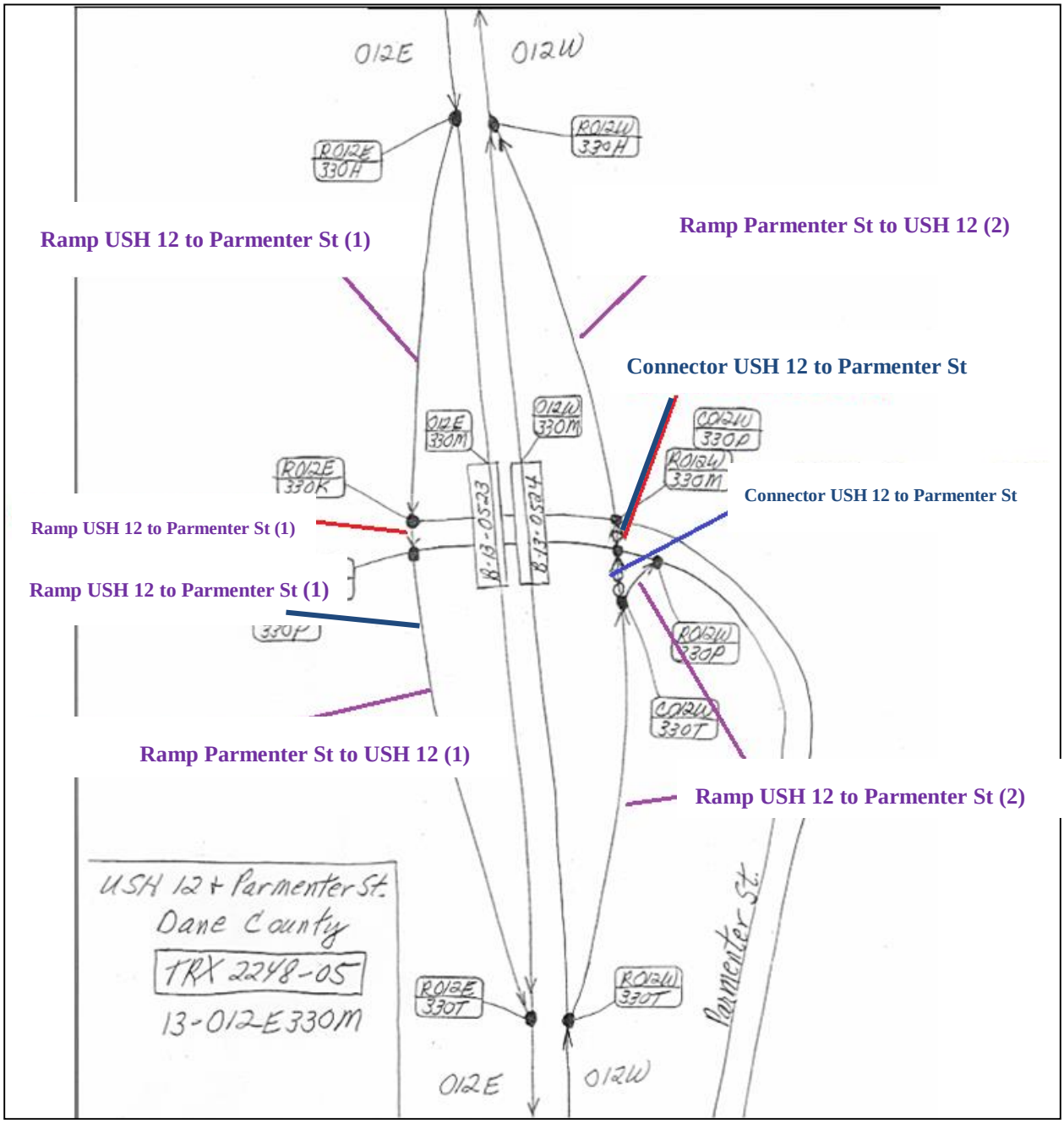
ŧ ***TRX diagrams must be referenced when assigning these extensions. Typically, the main route through the area will receive the extension (1). The remaining extensions should be assigned in a logical order.***

€ The Frontage road will receive the category of the road it follows, then select "5-Frontage road" under Subcategory.

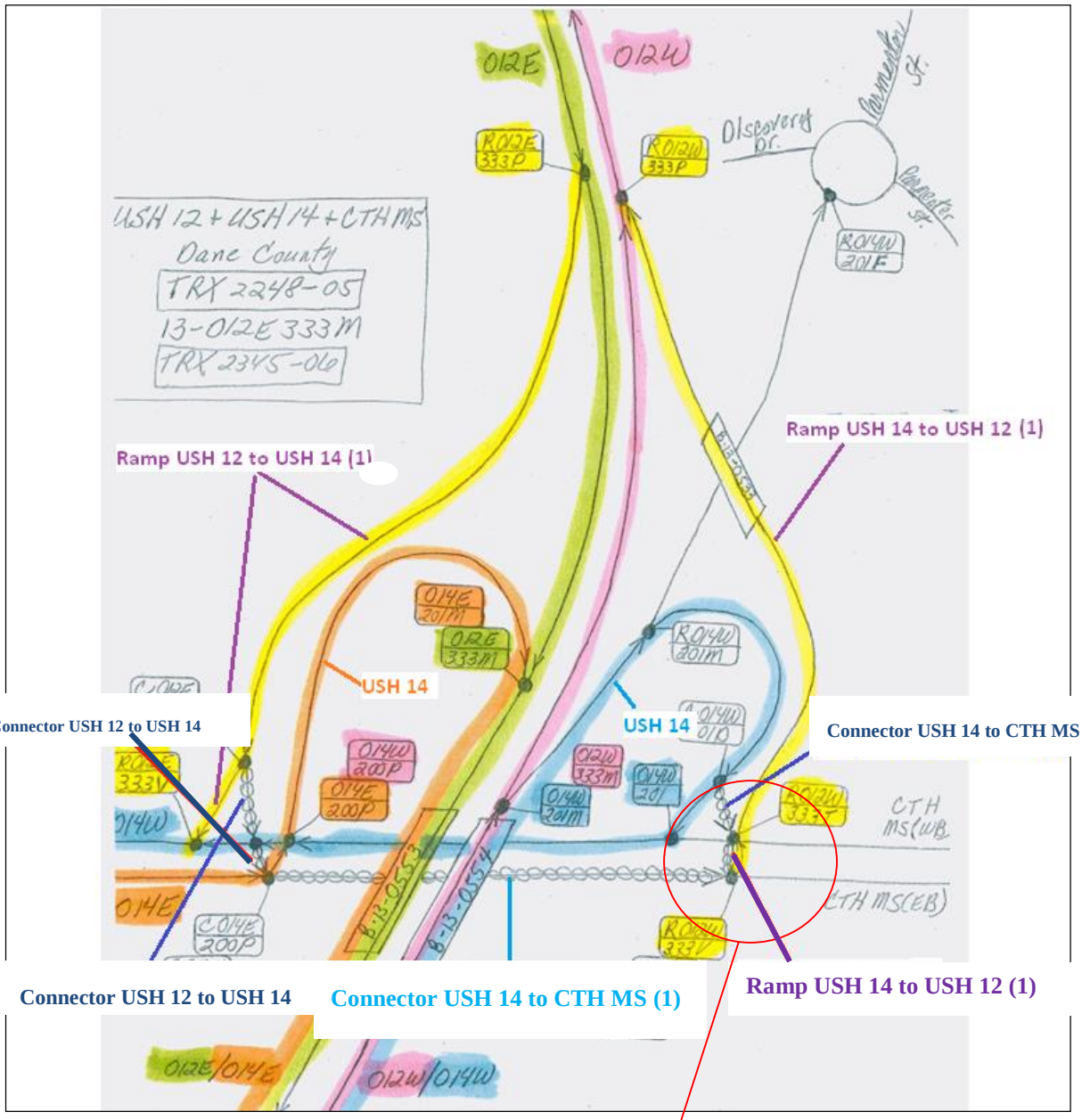
Example 1: Ramp with Turn Lanes



Example 2: Ramp to Local road. 1 Turn Lane, Local road doubled out. How to name

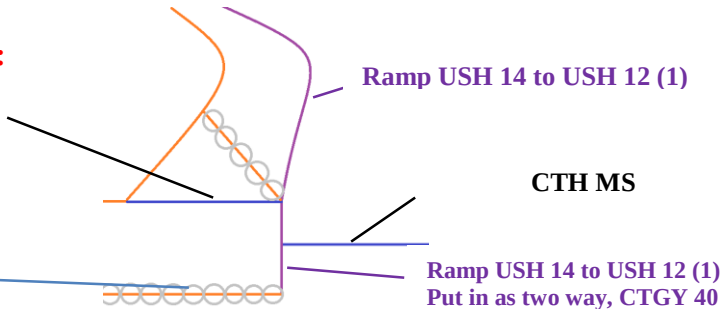


Example 3: Ramps with Main Line and connectors:

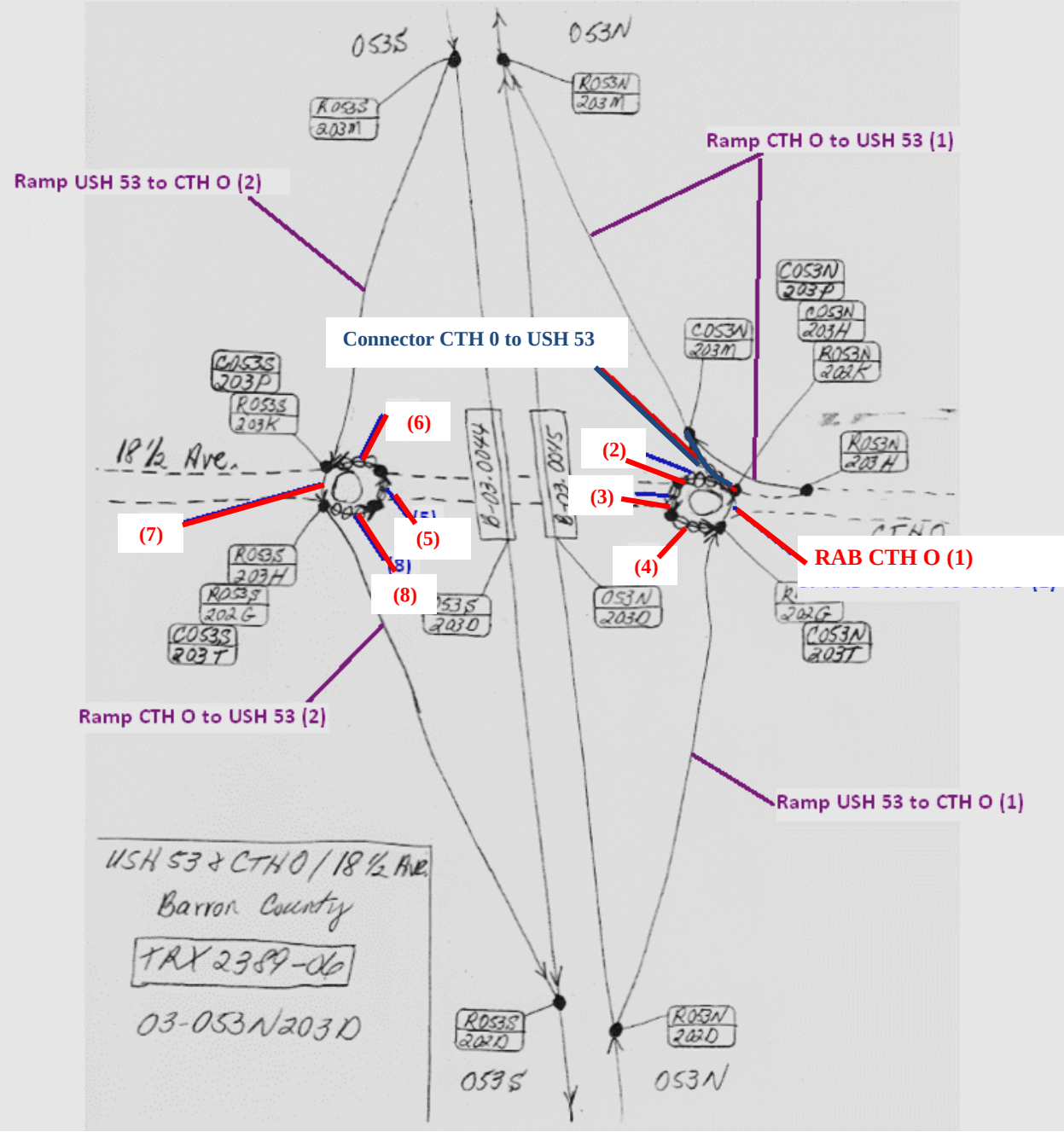


What to do in WISLR LCM:

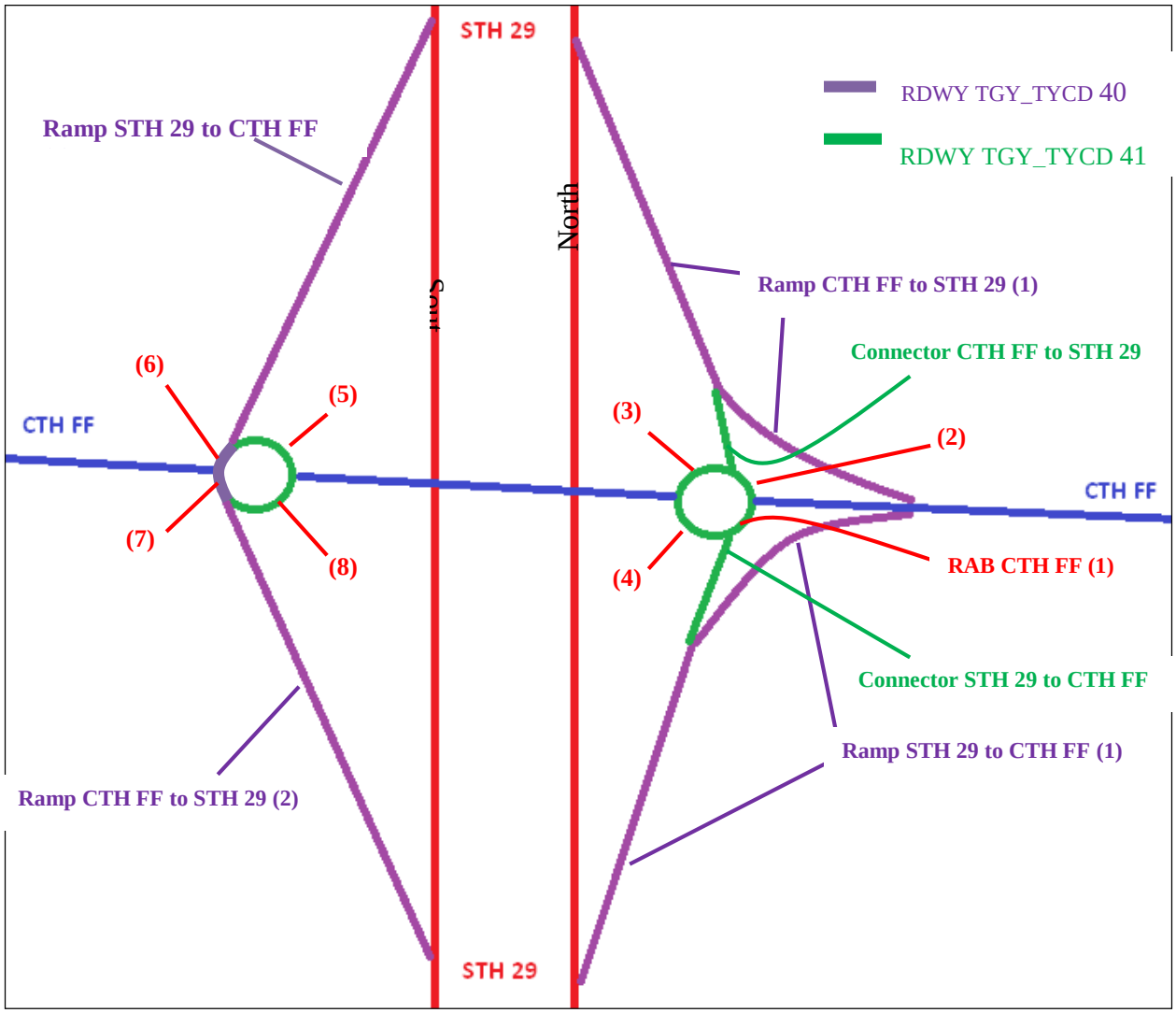
CTH MS.
Note: Email DOT to add the local/CTH Road Link



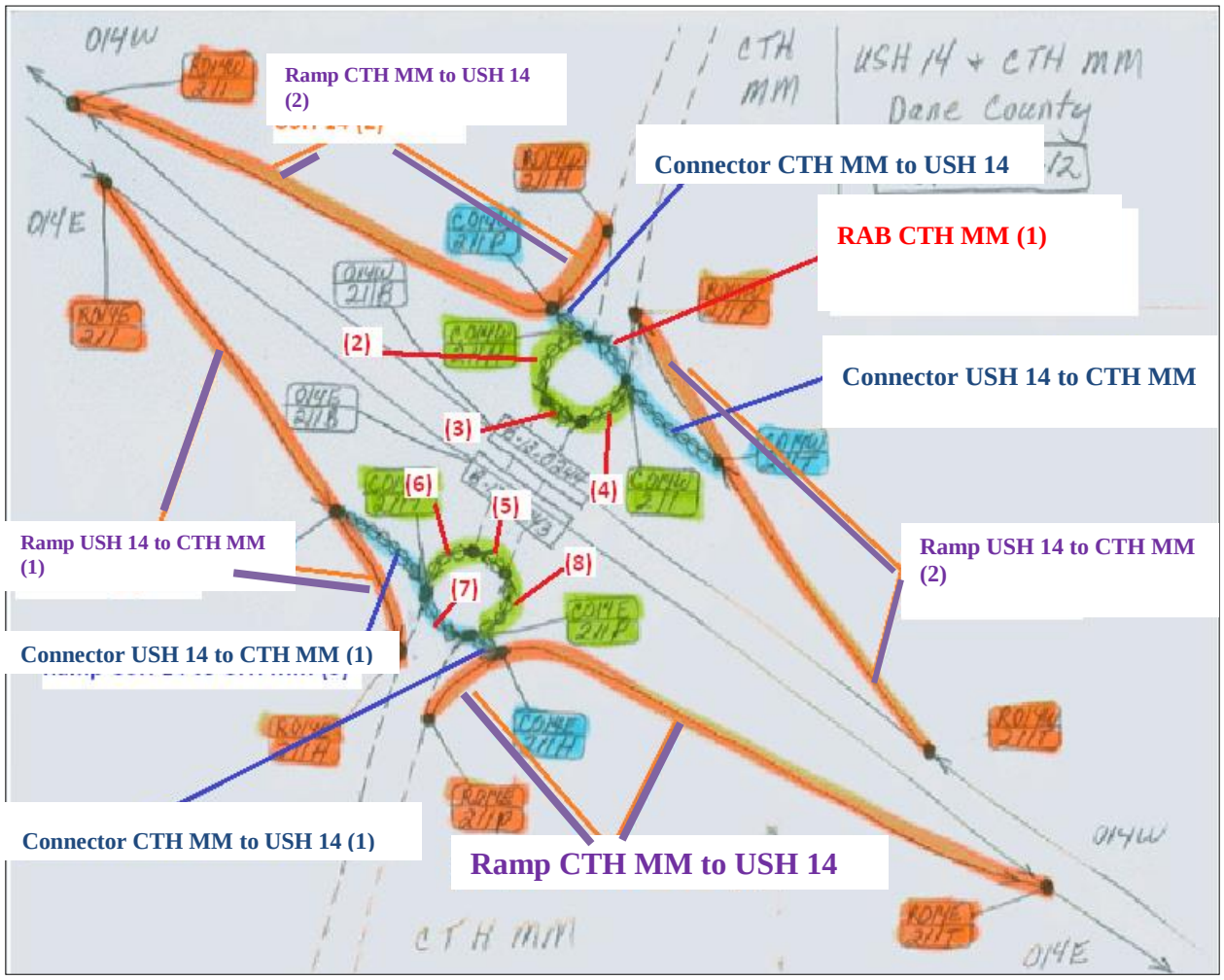
Example 4: Ramp with RAB to CTH/Local road and 1 Turn Lane:



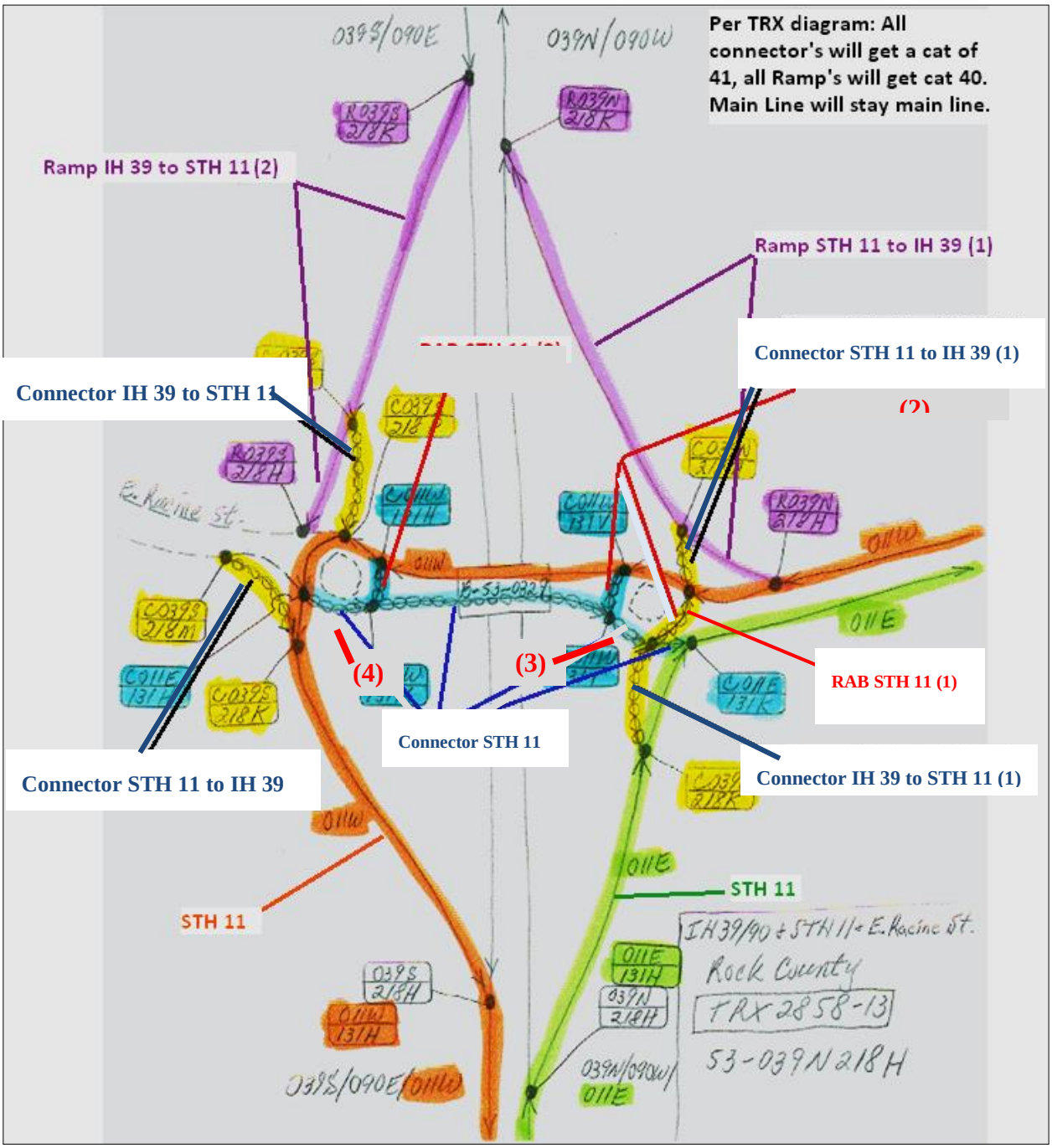
Example 5: Ramps with RAB and Connectors. What to do when Local/County roads are two-way routes



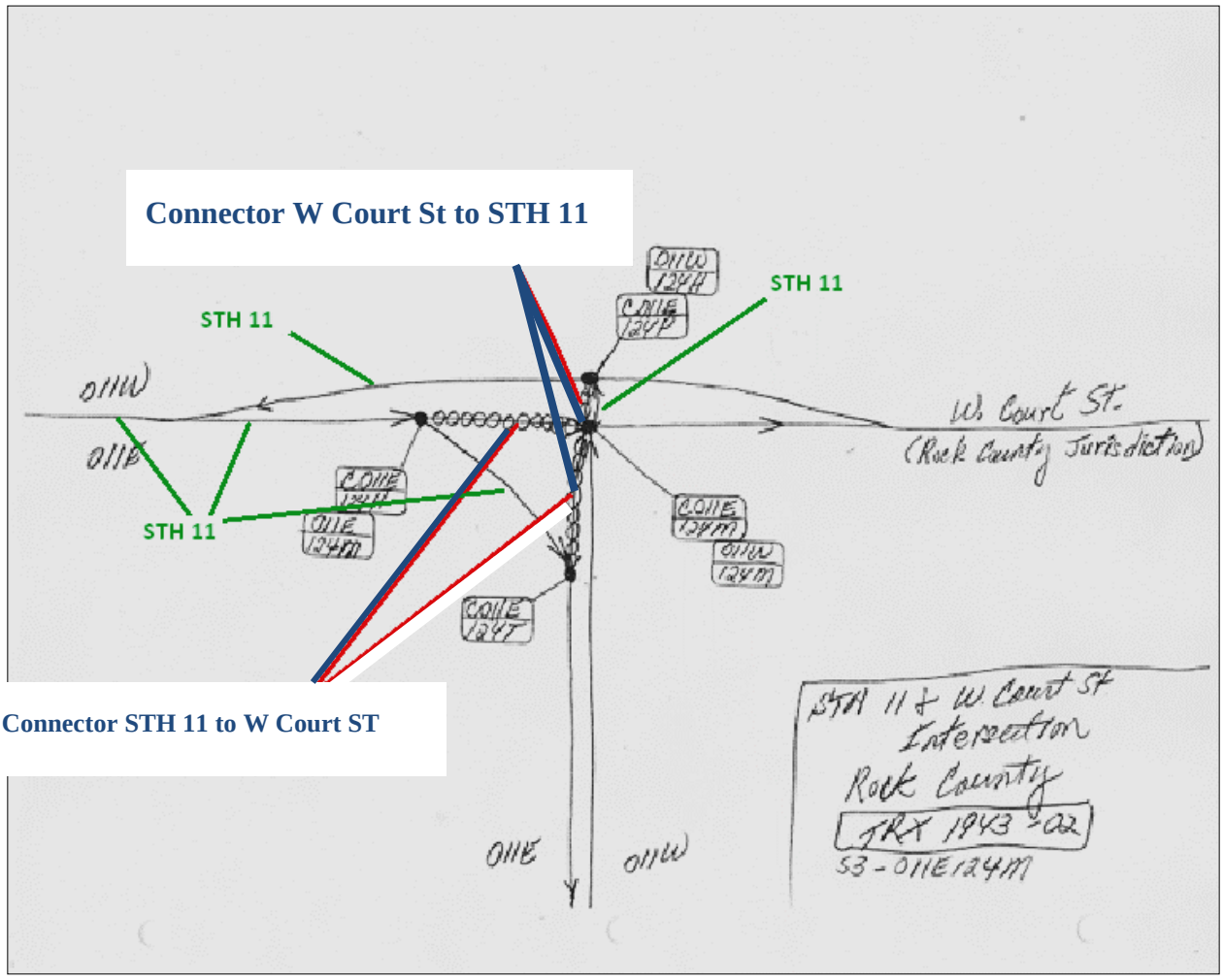
Example 6: Ramp with Turn Lane, Connector's and CTH RAB's:



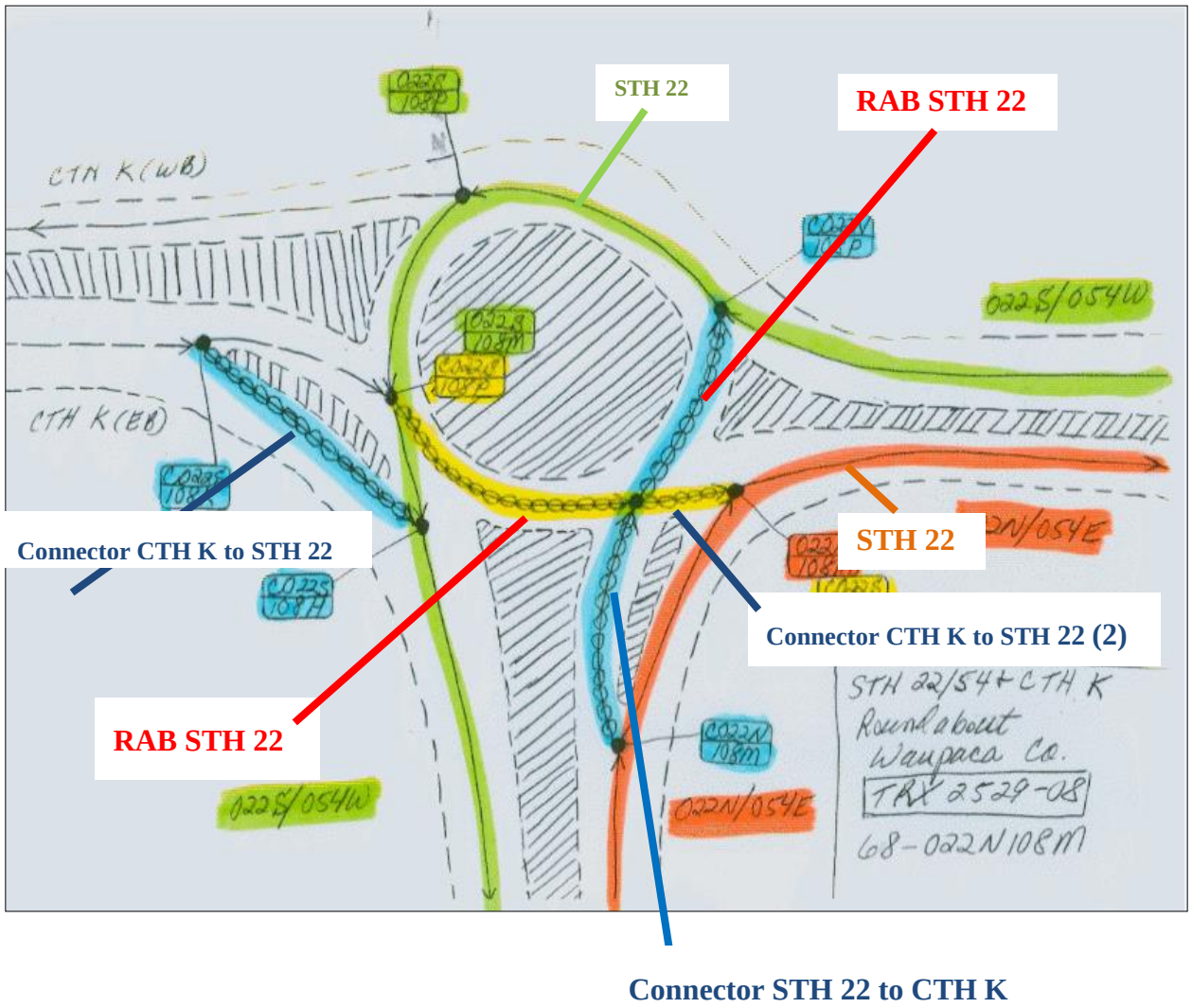
Example 7: IH Ramp with RAB, Turn lanes, and STH Main Line.



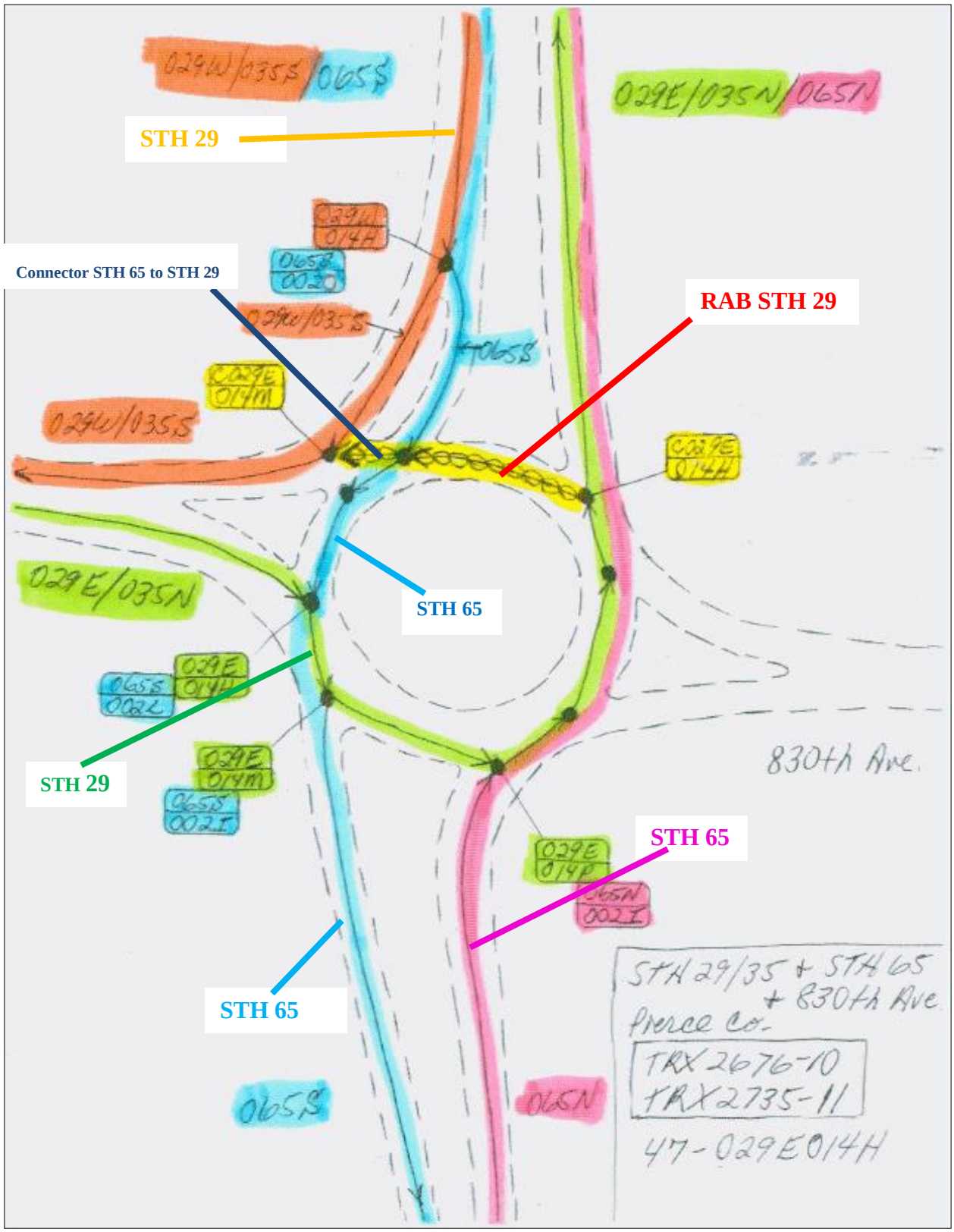
Example 8: Connector, turn lane, local road:



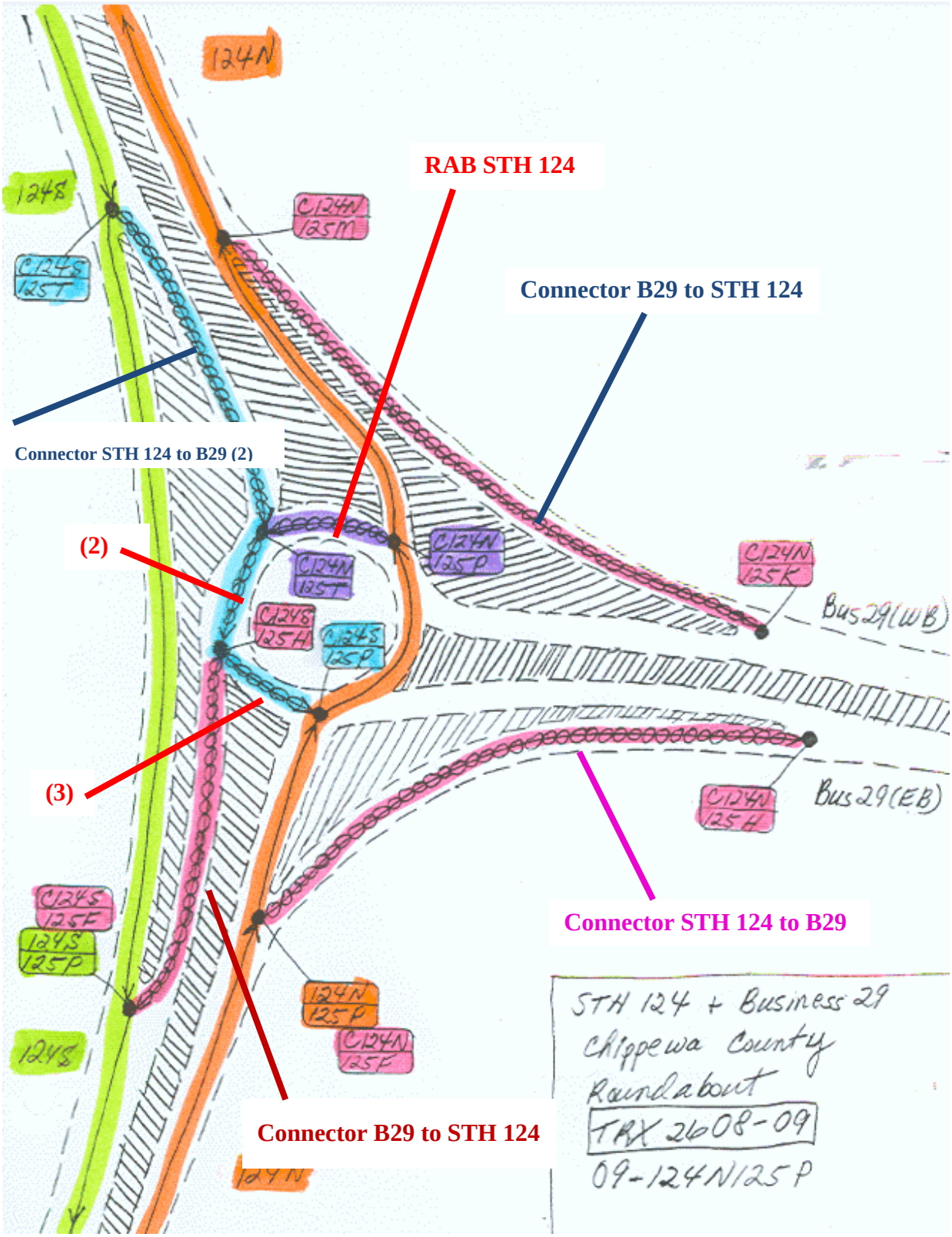
Example 9: RAB intersection with CTH and STN, with turn lanes



Example 10: RAB intersection with Local road and State Roads:

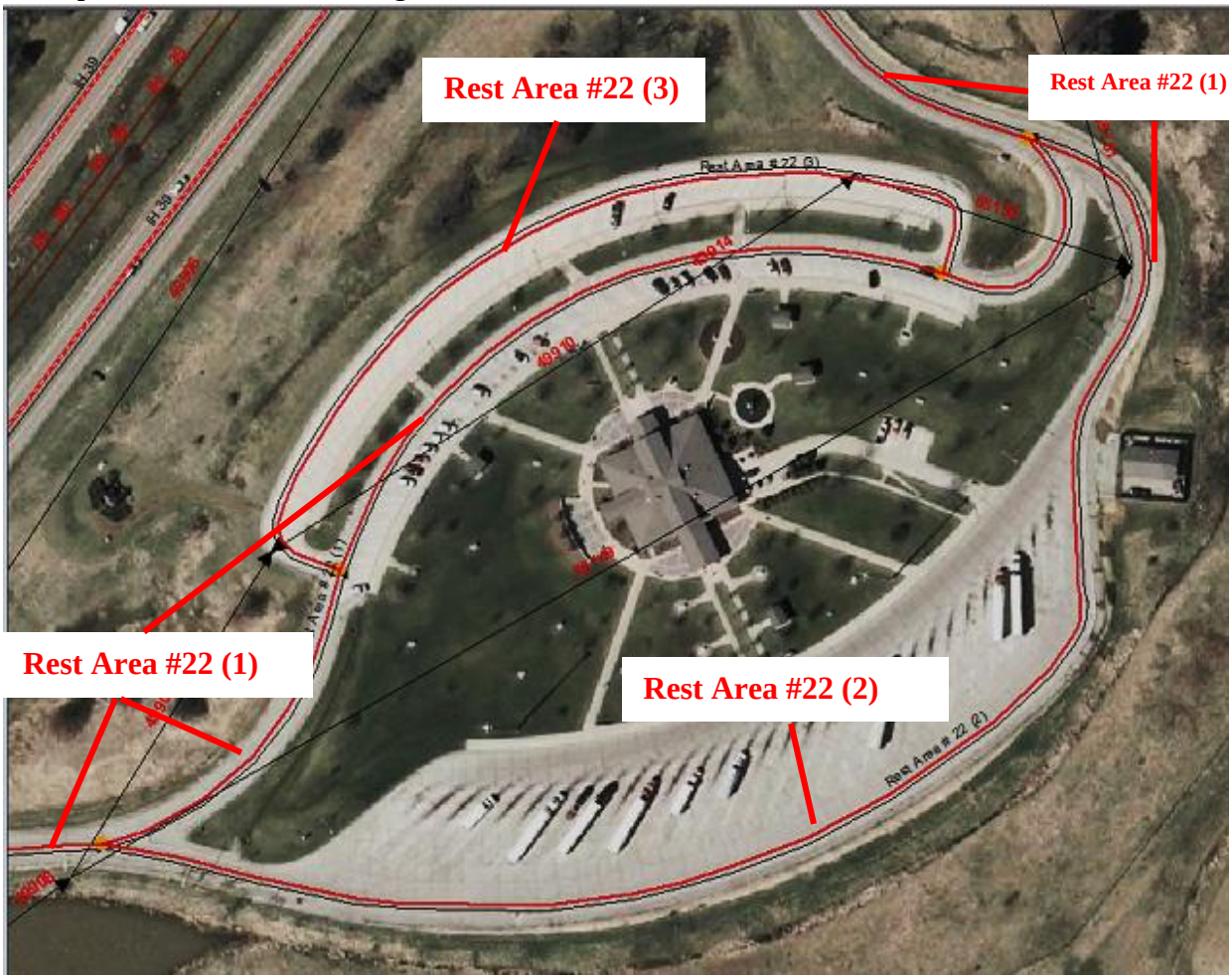


Example 11: All State RAB Intersection



Example 12: SWEF Naming

Example 13: Rest Area Naming



APPENDIX C-GORE POINT TABLE GENERATION

The steps outlined in this appendix describe the process to generate the tables containing the set of gore points used to calibrate the Link-Link table. The procedure set forth is based on tools available in ESRI's ArcMap 10.x; therefore, this guide assumes that the user has moderate knowledge and understanding of the tools in ArcGIS.

The following steps are performed once the gore point affected links have been identified and overlaid over an aerial image of the area. It has been found that the optimum scale to needed to most clearly view gore locations is 1:250.

Required Data

- Current WISLR link shape file
- Current WISLR Reference Sites shape file
- STN link shape file (for reference only)

Steps

1. Generate new polyline feature- This step will produced a polyline containing three vertices: a start, middle and end point.
    a. The start point is placed on the WISLR mainline line link, lined up as closely as possible to the gore point location as defined by STN.
    b. The middle point is placed on the WISLR reference site, closest to the gore location, and corresponding the WISLR gore point.
    c. The end point is placed on the ramp, or off mainline link, lined up as closely as possible to the same gore point location found in step 1(a).

d. When creating the polyline feature for gore location at an on-ramp, be sure to assign the unique identifier "1" to this feature. This is most easily accomplished through Organized Templates in the Create Features tool.

2. Spatially join the new polyline to the WISLR reference site shape file. This will attach the reference site id to the polyline feature attribute data.
3. The next step will create a point feature data set from the created polyline feature.
    a. Use Feature Vertices to Points tool to create a separate point shape file for the start, middle and end point.
    b. Must run the tool two times:
        i. First run the tool on "Start Point: to create a point shape file for the mainline gore point.
        ii. Second, run the tool on the "End Point" to create a point shape file for the ramp, or off main line gore point.
4. Use the "Create Route" tool for the WISLR link shape file to locate the previously created point files on the WISLR link. Run the tool twice.
    a. First run the tool using the "Start Point" shape file created in step 3 (b)(i). This locates the created point on the WISLR mainline link and captures the WISLR link ID. Export to .dbf named "Mainline_Gore_Point".
    b. Second run the tool using the "End Point" shape file created in step 3(b)(ii). This locates the created point on the WISLR ramp link and the captures the WISRL link ID. Export to .dbf named "Offmainline_Gore_Point".

The two exported files will contain a unique gore point ID, the WISLR link ID, the offset measure along the link from the "from" reference site, and the unique identifier.