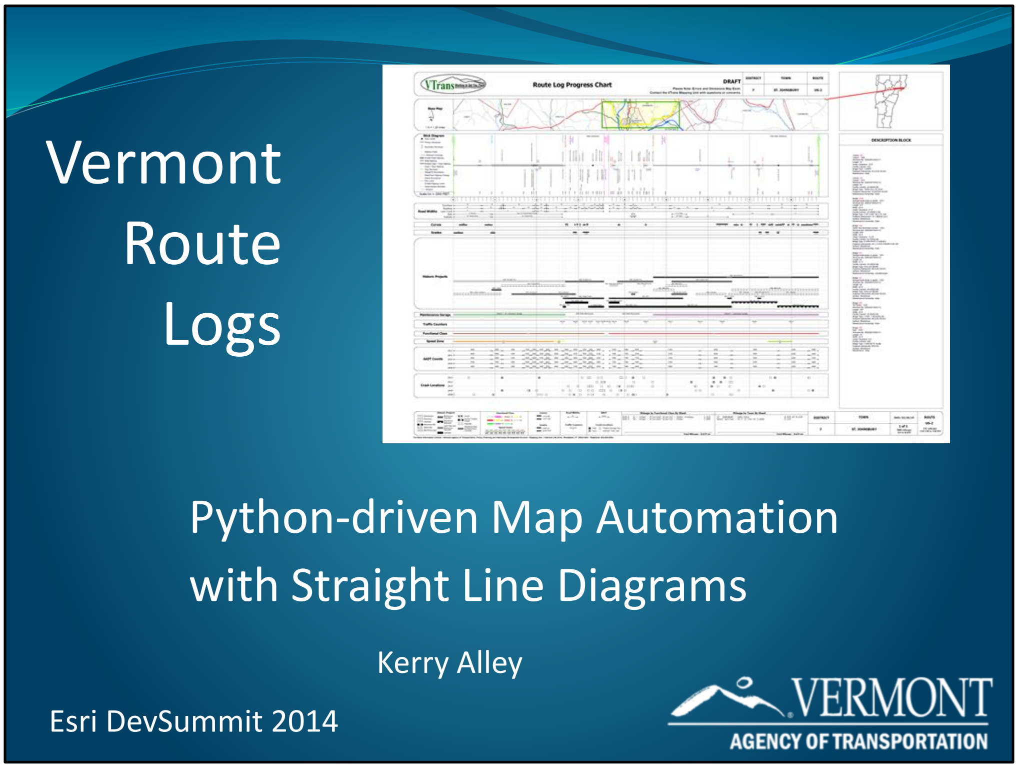
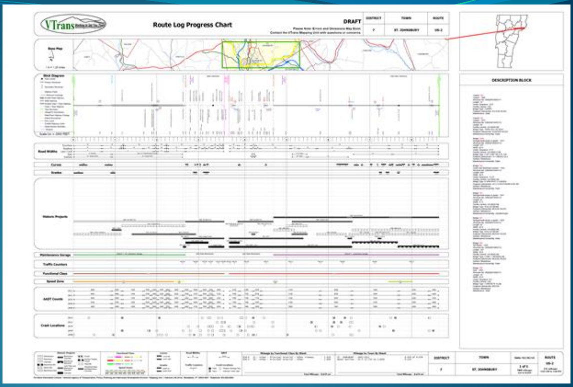


Vermont Route Logs



Python-driven Map Automation with Straight Line Diagrams

Kerry Alley

Esri DevSummit 2014

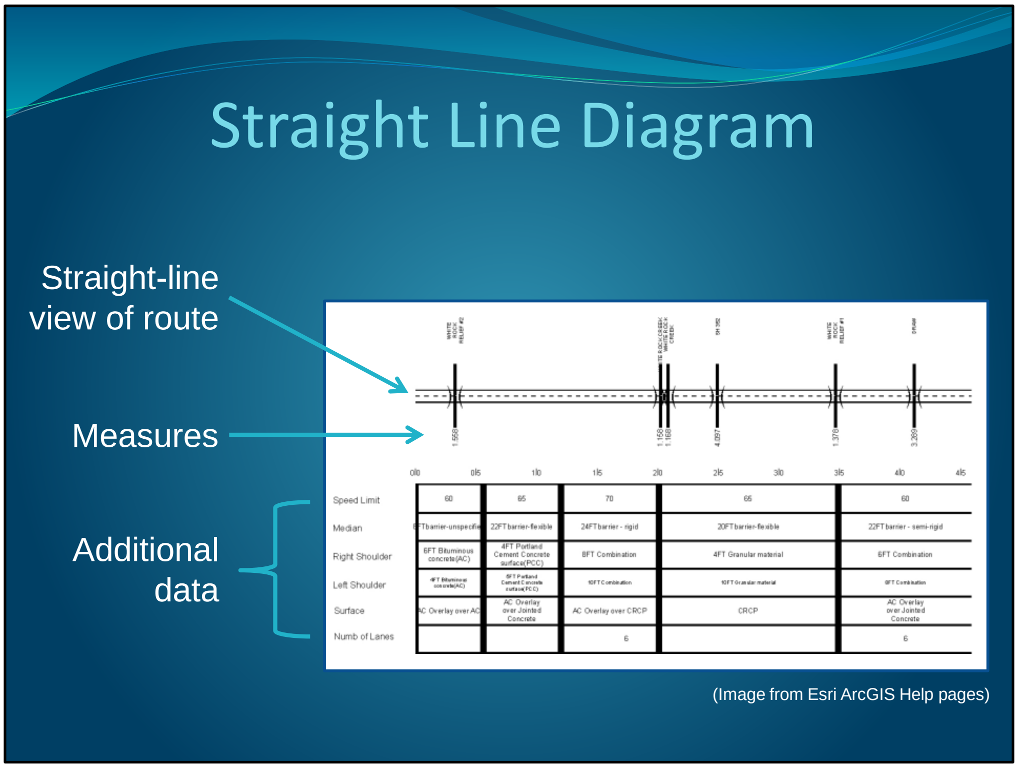
VERMONT AGENCY OF TRANSPORTATION

I'll be talking about our new Route Log System which is how we produce Route Logs.

The message that I hope you take home from this presentation is that Python can do so much more than just automate your map series, it can solve the problems that automation problematic.

Before I go into the project itself, I'd like describe SLDs in general, and introduce you to an actual Route Log

Straight Line Diagram



Speed Limit	60	65	70	65	60
Median	6FT barrier-unspecifie	22FT barrier-flexible	24FT barrier - rigid	20FT barrier-flexible	22FT barrier - semi-rigid
Right Shoulder	6FT Bituminous concrete(AC)	4FT Portland Cement Concrete surface(PCC)	8FT Combination	4FT Granular material	6FT Combination
Left Shoulder	4FT Bituminous concrete(AC)	6FT Portland Cement Concrete surface(PCC)	10FT Combination	10FT Granular material	8FT Combination
Surface	AC Overlay over AC	AC Overlay over Jointed Concrete	AC Overlay over CRCP	CRCP	AC Overlay over Jointed Concrete
Numb of Lanes			6		6

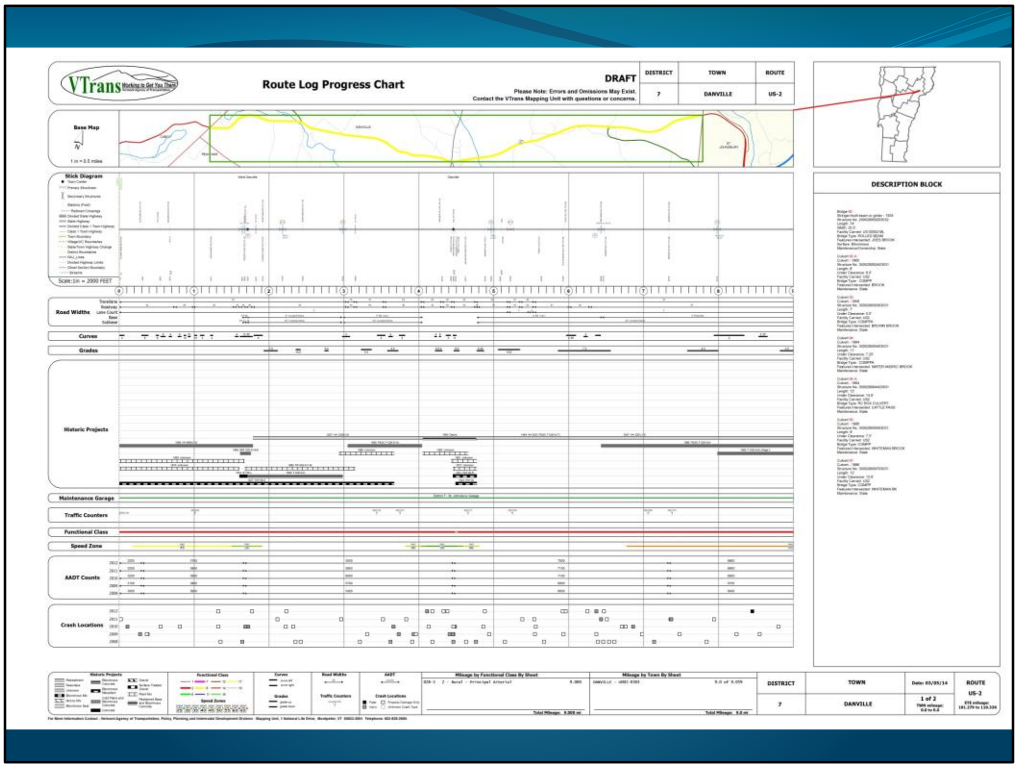
(Image from Esri ArcGIS Help pages)

A SLD provides a straight line view of a route and its associated data.

A SLD consists of the reference route with notable features (e.g. intersecting streets and bridges), measures displayed as labels or as hatches like on a ruler, and any relevant data parallel to the reference route corresponding to the portion of the route it aligns with.

BTW, with data in the right format, and a Production Mapping license, you too can make a SLD like this with the Straight Line Diagram Wizard!!

Very effective way to view dissimilar data for the same stretch of road simultaneously.

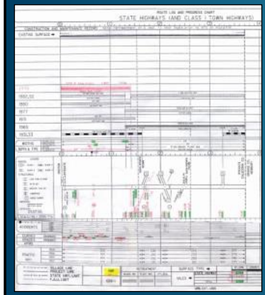


And we have a lot of data!

We create route logs for all Federal Aid Highways, not the basic town highways

This is the Route Log for US-2 in Danville, VT
Point out: Base map, Locator map, Stick diagram, Other data frames, Bridge descriptions, Header/Footer, Legends, Summary statistics.
Although the Mapping Unit maintains some of this data, much of it comes from other divisions within the Agency.

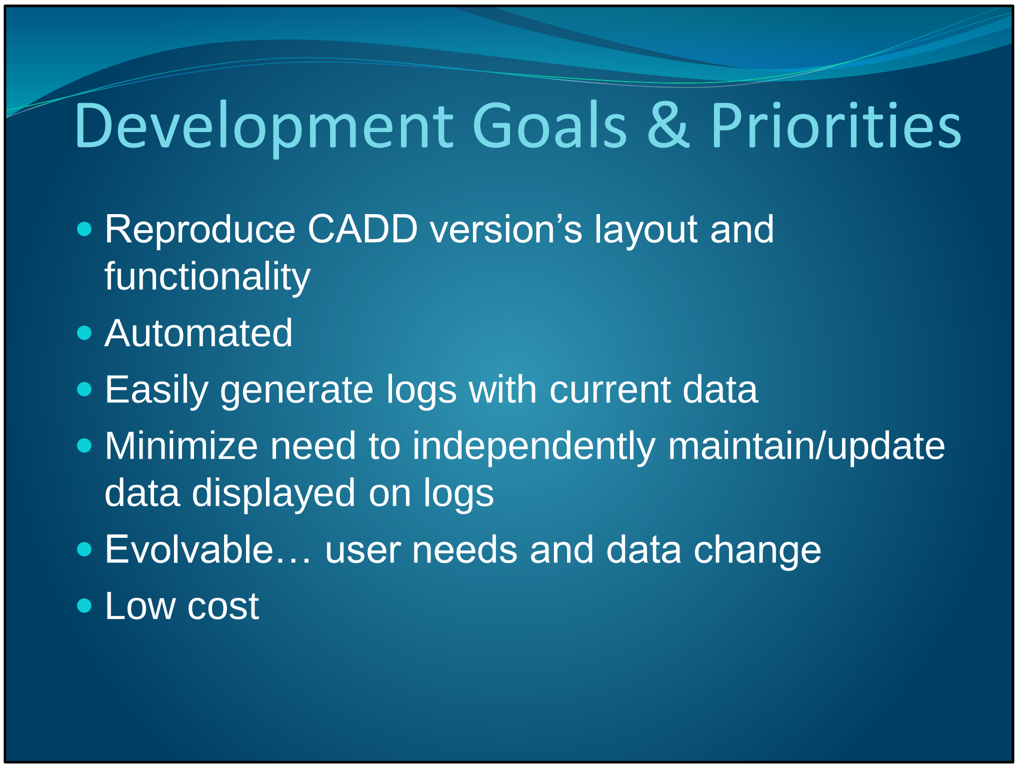
If you really want a good look, I have full sized copies with me and can show you after the talk.



Route Logs at VTrans

- 1950's
  - The first Logs were drawn by hand
  - Developed during the building of the Interstate System
- 1980's
  - Logs converted to CADD using Intergraph software
- 1990's
  - The Route Log System becomes defunct & Master series maintained with hand markups
- 2006-2010
  - Contractor developed ArcGIS/VBA system. Also an online version. Most users still preferred CADD w/ markups.

We have a long Route Log history at VTrans, but a downside of rapidly changing technology is the speed at which technology can become obsolete.
We were in desperate need of a new System without a clear solution



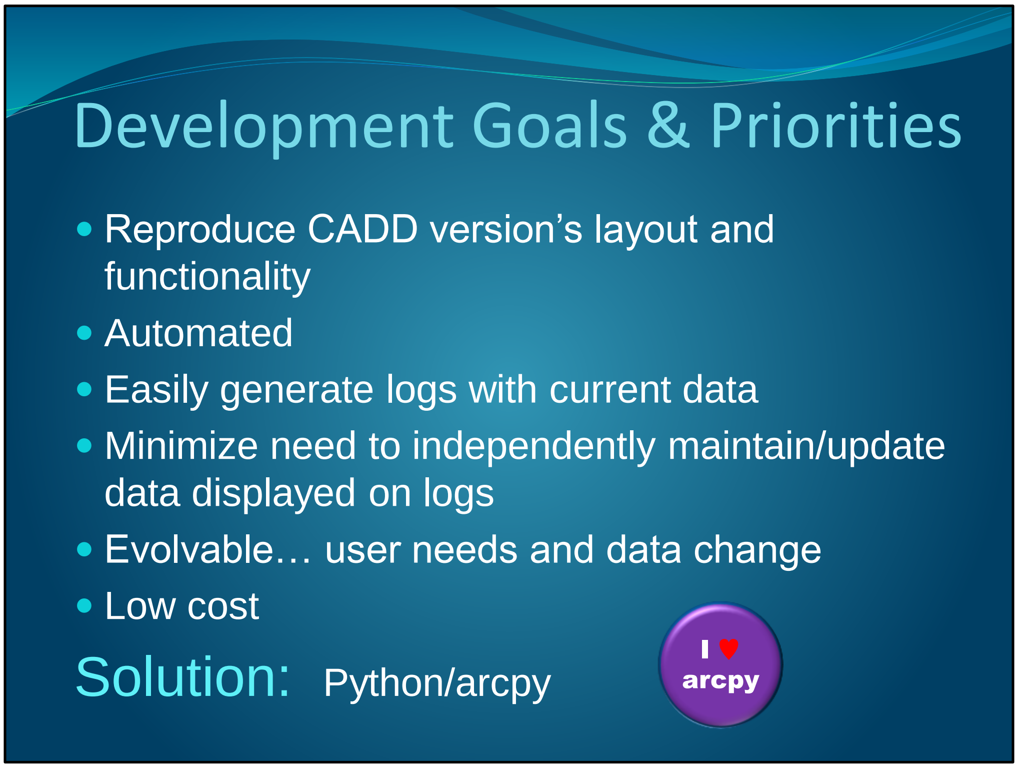
Development Goals & Priorities

- Reproduce CADD version's layout and functionality
- Automated
- Easily generate logs with current data
- Minimize need to independently maintain/update data displayed on logs
- Evolvable… user needs and data change
- Low cost

But we knew what we wanted!

Many of the challenges faced during development were related to users wanting the layout and functionality of the historic CADD Route Logs. We needed to create engineering-looking tool with GIS data.

We did not have a solution until Esri's Jeff Barrett created a tailor-made, Python-based map automation system for generating Vermont Town Highway Maps. This system was proof of concept and a programmer's Rosetta Stone.

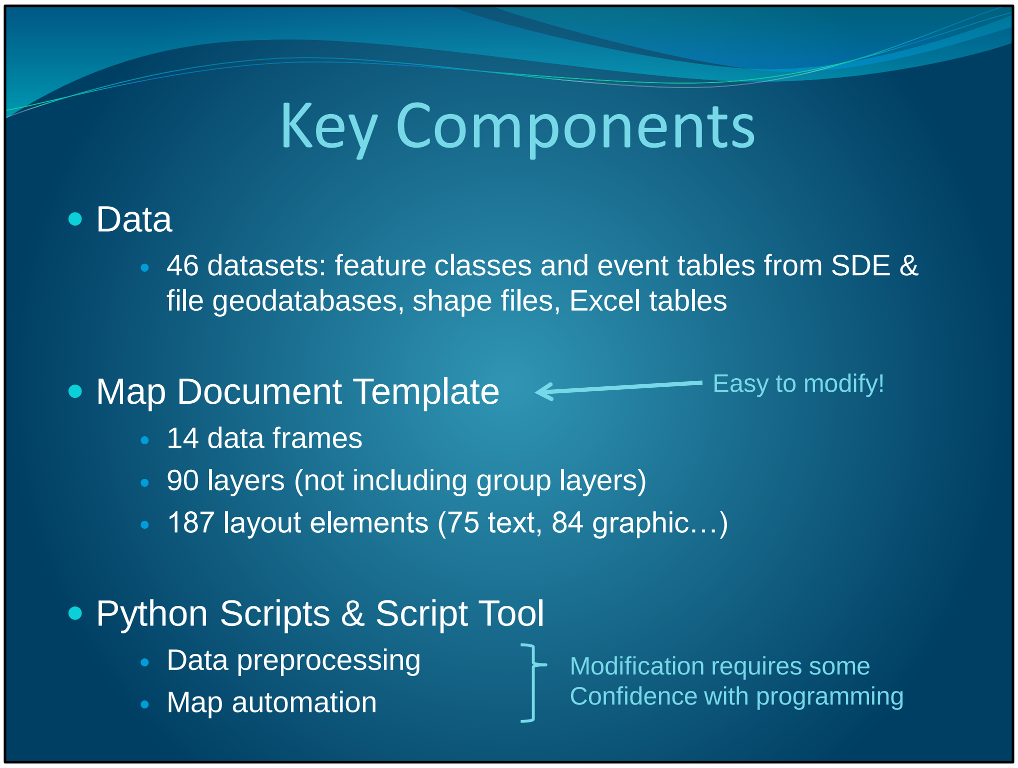


Development Goals & Priorities

- Reproduce CADD version's layout and functionality
- Automated
- Easily generate logs with current data
- Minimize need to independently maintain/update data displayed on logs
- Evolvable… user needs and data change
- Low cost

Solution: Python/arcpy

I ♥ arcpy



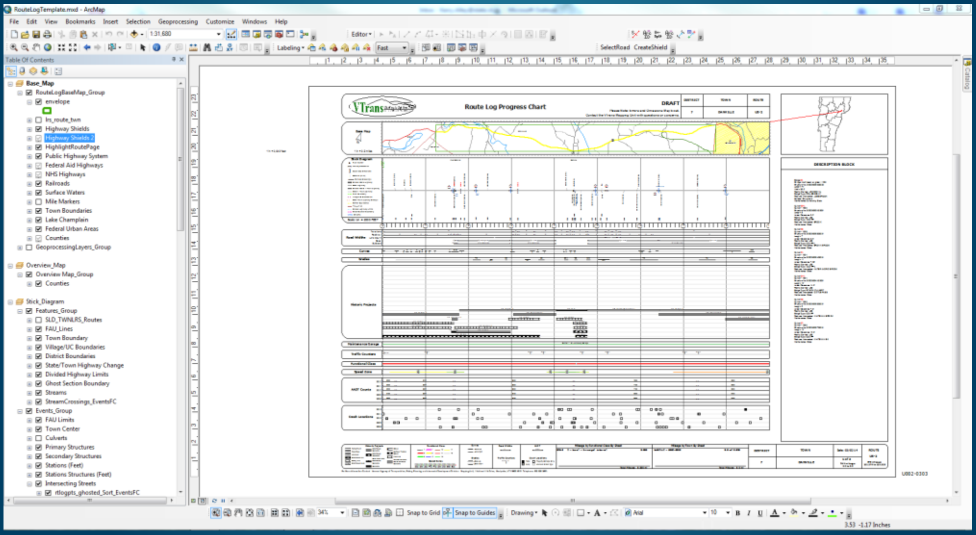
Key Components

- Data
  - 46 datasets: feature classes and event tables from SDE & file geodatabases, shape files, Excel tables
- Map Document Template ← Easy to modify!
  - 14 data frames
  - 90 layers (not including group layers)
  - 187 layout elements (75 text, 84 graphic…)
- Python Scripts & Script Tool
  - Data preprocessing
  - Map automation

} Modification requires some Confidence with programming

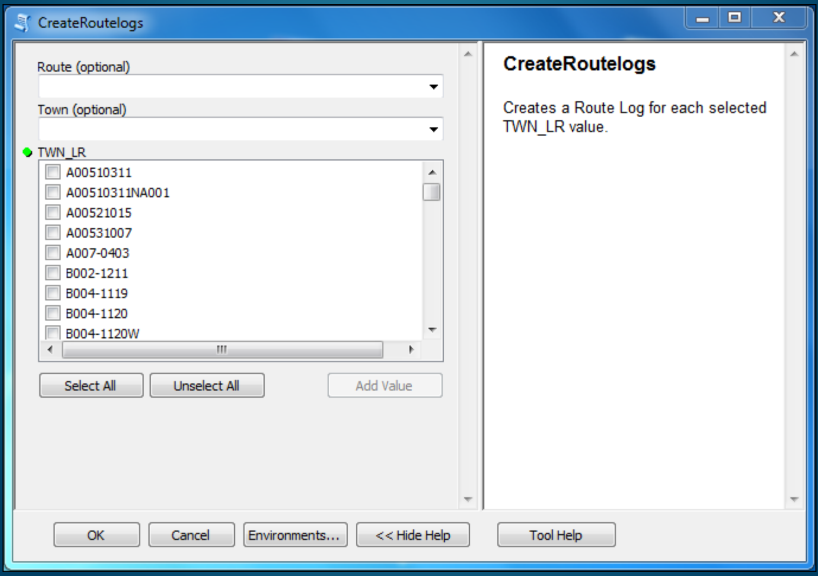
Now I'd like to describe the present system

Map Document



Here is a view of the template document. It’s never “empty.”

Script Tool


CreateRoutelogs

Route (optional)

Town (optional)

TWN_LR

- [ ] A00510311
- [ ] A00510311NA001
- [ ] A00521015
- [ ] A00531007
- [ ] A007-0403
- [ ] B002-1211
- [ ] B004-1119
- [ ] B004-1120
- [ ] B004-1120W

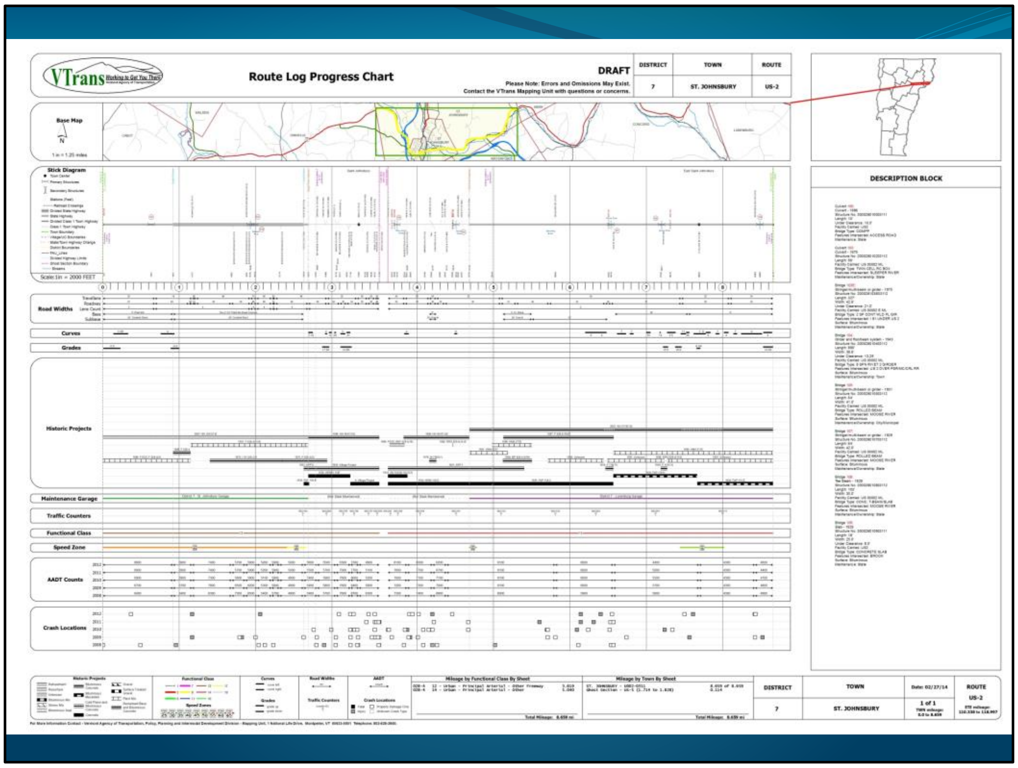
CreateRoutelogs

Creates a Route Log for each selected TWN_LR value.

Here is the script tool used to drive the Route Log System.

The user has to select at least one town route code. Since there are about 2300 routes, there's the option of filtering the choices by Route # or by town.

When you hit OK, the PDFs appear with informative naming in an output folder.

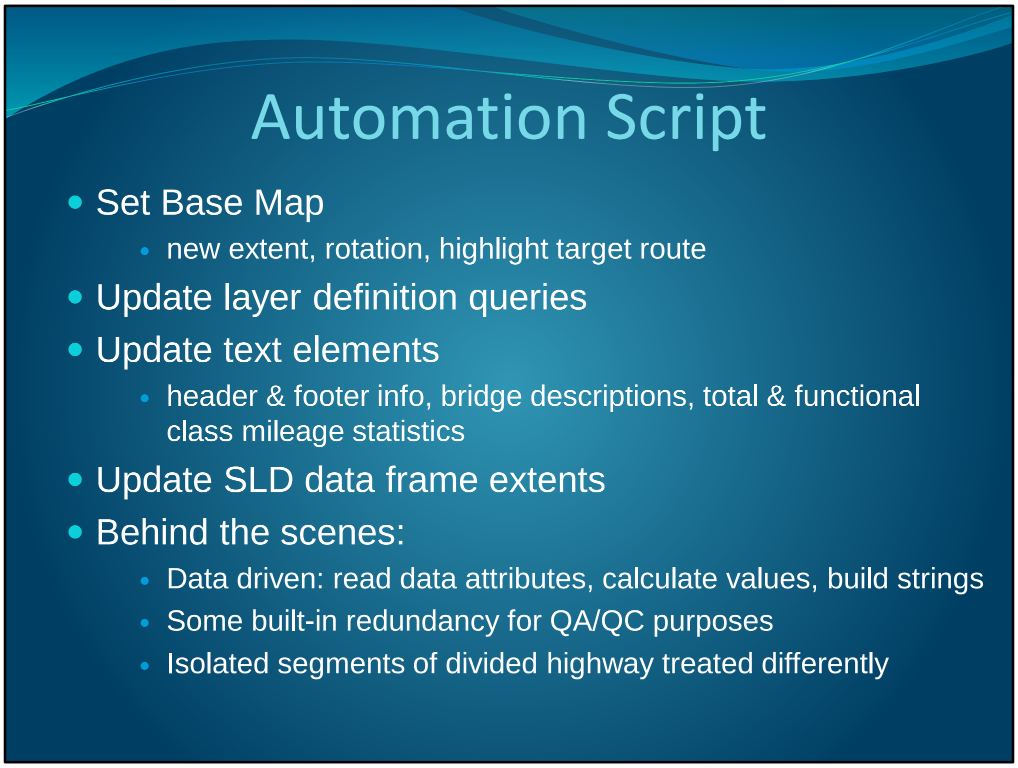


Outline map automation scripts while showing this slide:

The map automation scripts change the mxd from one Route Log to the next.

Set base map (extent, rotation, highlight target route)
Update layer definition queries
Update text elements (header & footer info, bridge descriptions, total & functional class mileage statistics)
Update SLD data frame extents
Behind the scenes:

> Data driven: read data attributes, calculate values, build strings/tables
> Some built-in redundancy for QA/QC purposes
> Isolated segments of divided highway treated differently

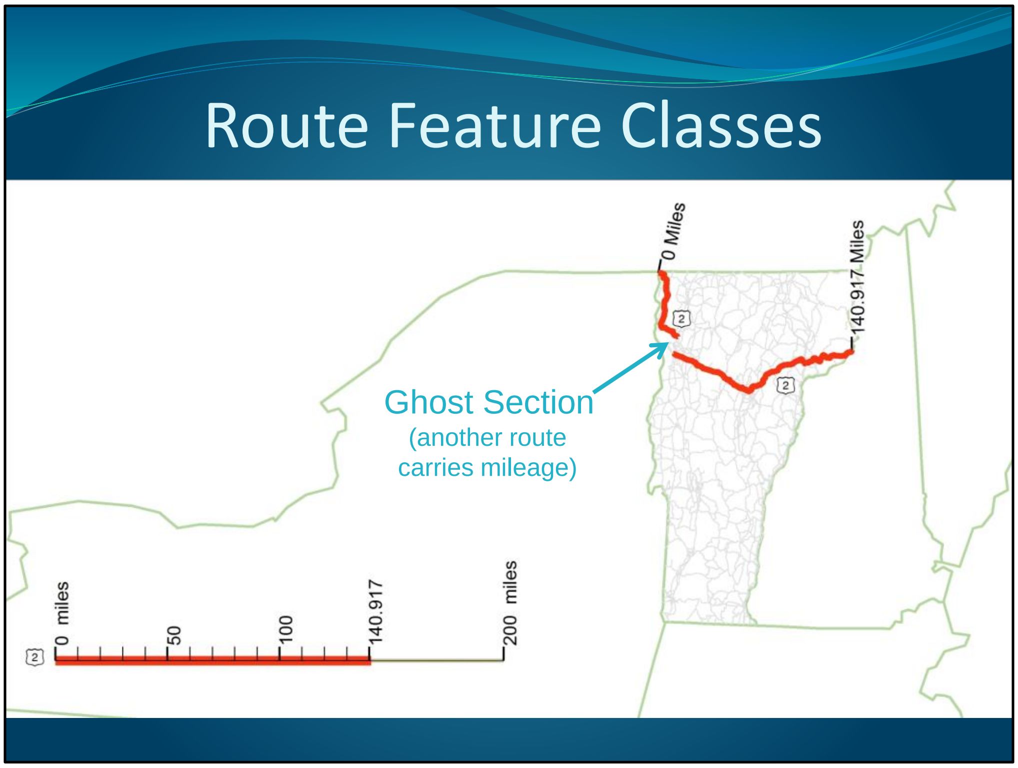


Automation Script

- Set Base Map
  - new extent, rotation, highlight target route
- Update layer definition queries
- Update text elements
  - header & footer info, bridge descriptions, total & functional class mileage statistics
- Update SLD data frame extents
- Behind the scenes:
  - Data driven: read data attributes, calculate values, build strings
  - Some built-in redundancy for QA/QC purposes
  - Isolated segments of divided highway treated differently

Before I summarize what is happening in the data preprocessing scripts, I want to give you a peek "under the hood" at the routes and measures I work with. It will make the data preprocessing summary a bit more clear.

Now we're getting to the fun stuff!



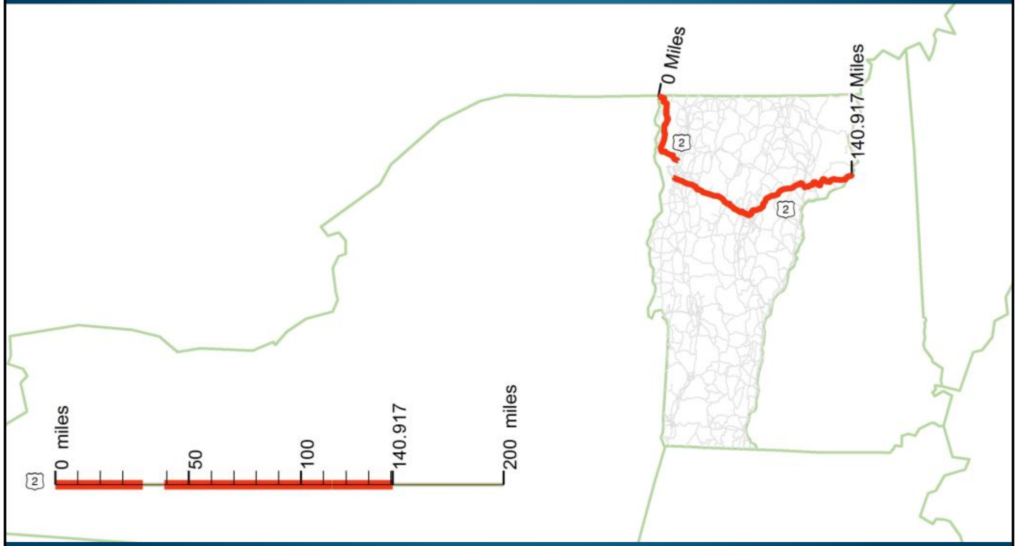
Route Feature Classes

A route feature class (the keystone of LRS) has an "built in" measuring system. Routes can be created from road centerline features, or in the case of a SLD, from straight line features, and measures can be defined however you want.

Even though its really only the linear measures that matters spatially with a SLD route, the lines have to be put *somewhere* and have a length… so I put them along a line that starts at coordinate (0,0) Vermont State Plane coordinates, and set their lengths equal to route lengths in our records. So, the straight lines in the Vermont Route Logs are actually located in NY!

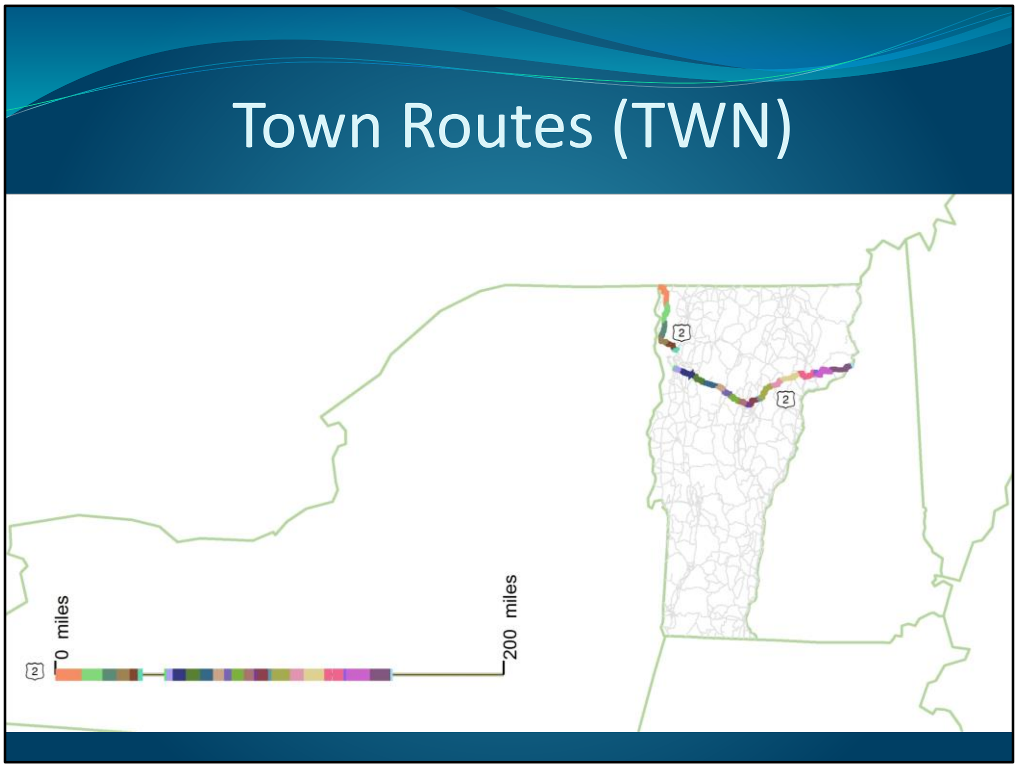
If I simply used the measures in our existing LRS, the SLD would look like this, even with a ghost section…. But I wanted it to look like this…

End-to-End Routes (ETE)



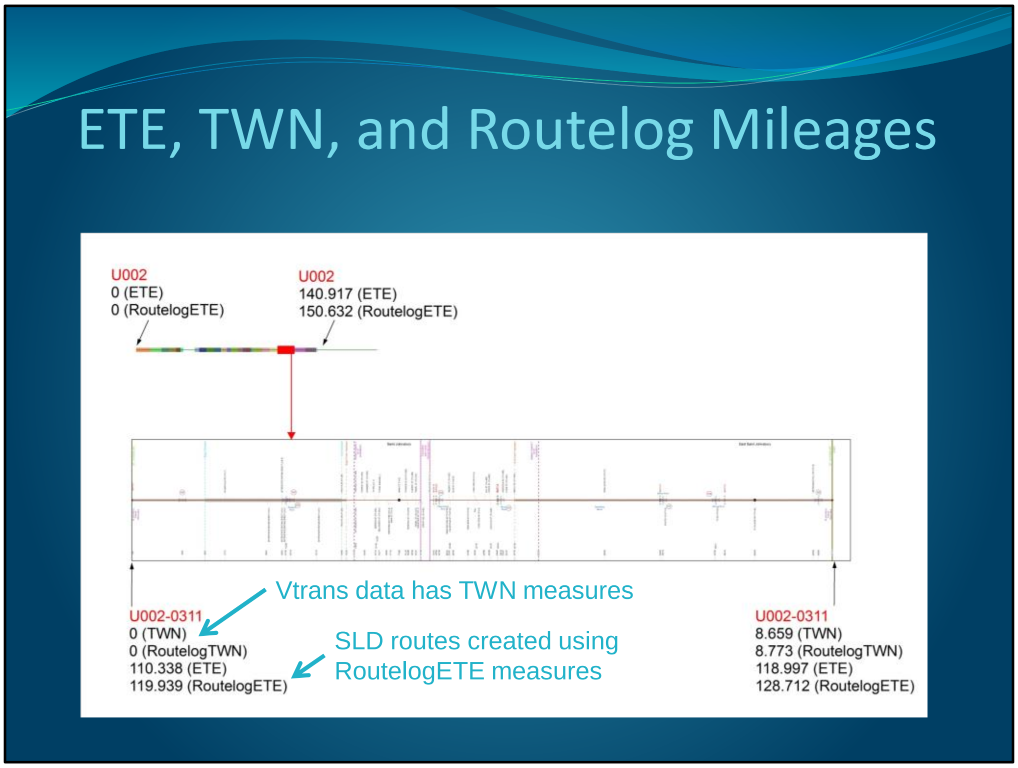
Here there is a gap in both the feature and the measurement hatches.

In the past two slides, the route reflects mileage accumulated across the state, so it is the US-2 ETE route

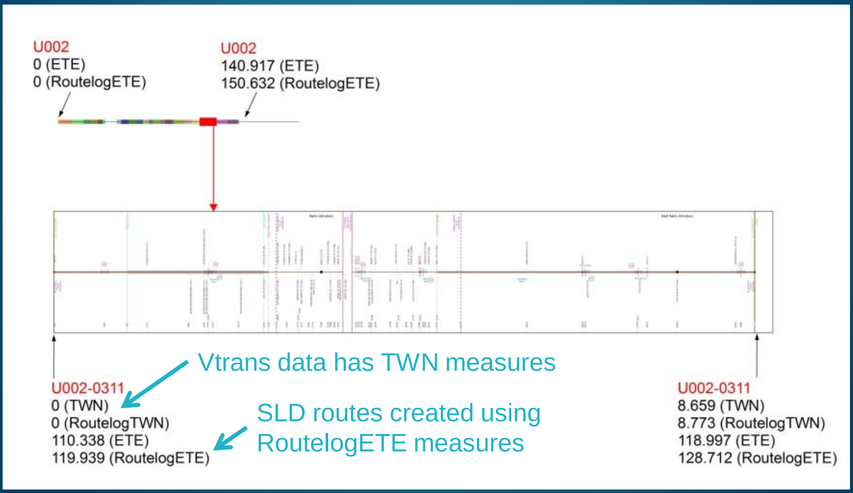


Town Routes (TWN)

However, for practical purposes, the highway-related data associated with VT routes is stored using town-based measures
Each differently colored segment is a different route. Each segment starts with measure 0 and connects nicely with upstream and downstream routes.
We already maintained both ETE and TWN map-view routes, although the TWN routes are most widely used in the agency.



ETE, TWN, and Routelog Mileages



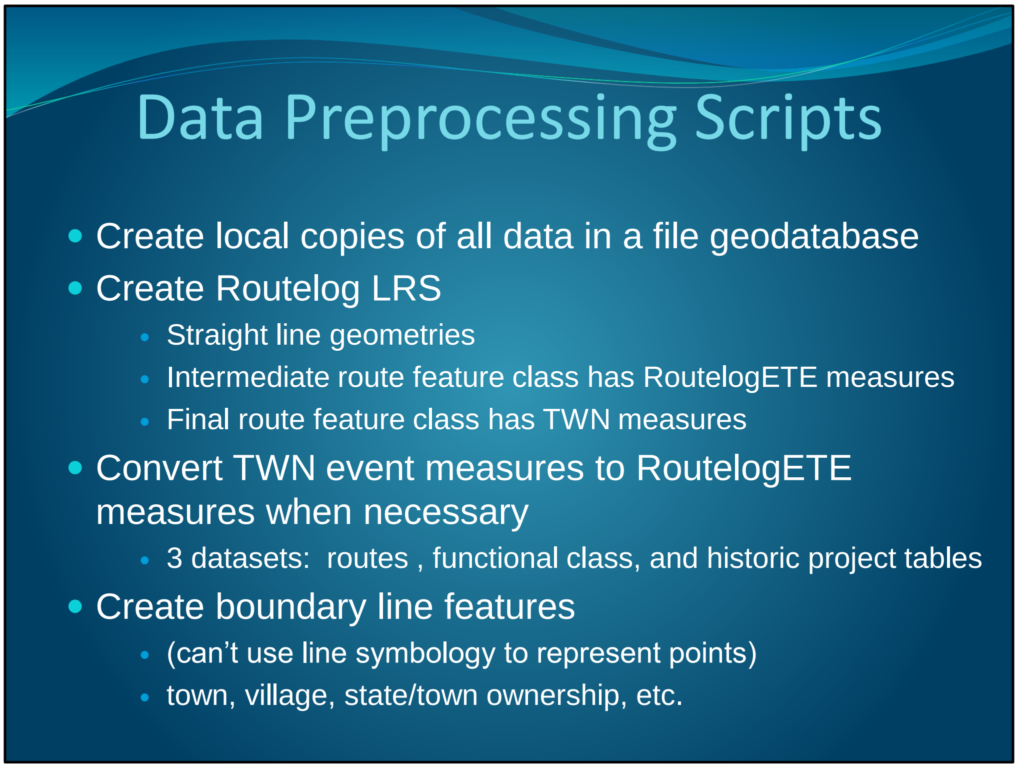
This slide shows the Stick Diagram for a particular town-based route found along US-2 and shows where it fits along US-2

Here you can compare the start and end measures according to the 4 different measuring systems.

ETE measures >= TWN measures (they're equal if the ETE route fits within a single town)

TWN and ETE Routelog measures are always greater than their non-Routelog counterparts because Routelog measures increase across ghost sections as well as along the route features.

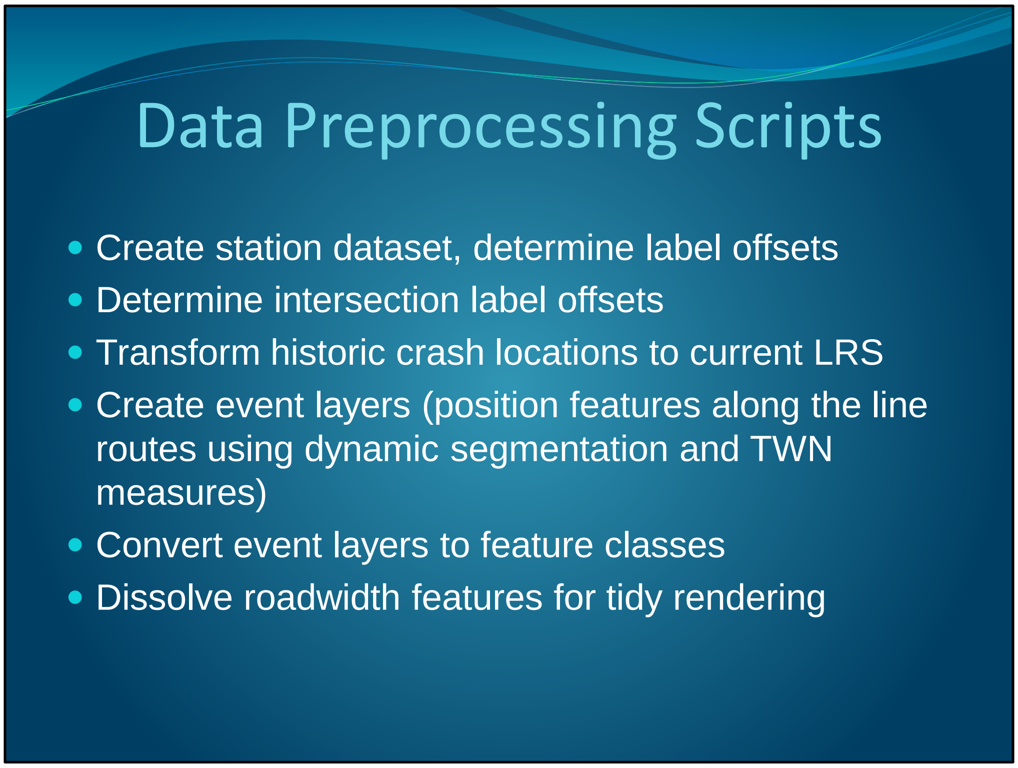
And so you don't think I'm doing all of this just for mathematical kicks, remember that data is stored with TWN measures, Route Log layout depends on Routelog measures



Data Preprocessing Scripts

- Create local copies of all data in a file geodatabase
- Create Routelog LRS
  - Straight line geometries
  - Intermediate route feature class has RoutelogETE measures
  - Final route feature class has TWN measures
- Convert TWN event measures to RoutelogETE measures when necessary
  - 3 datasets: routes , functional class, and historic project tables
- Create boundary line features
  - (can't use line symbology to represent points)
  - town, village, state/town ownership, etc.

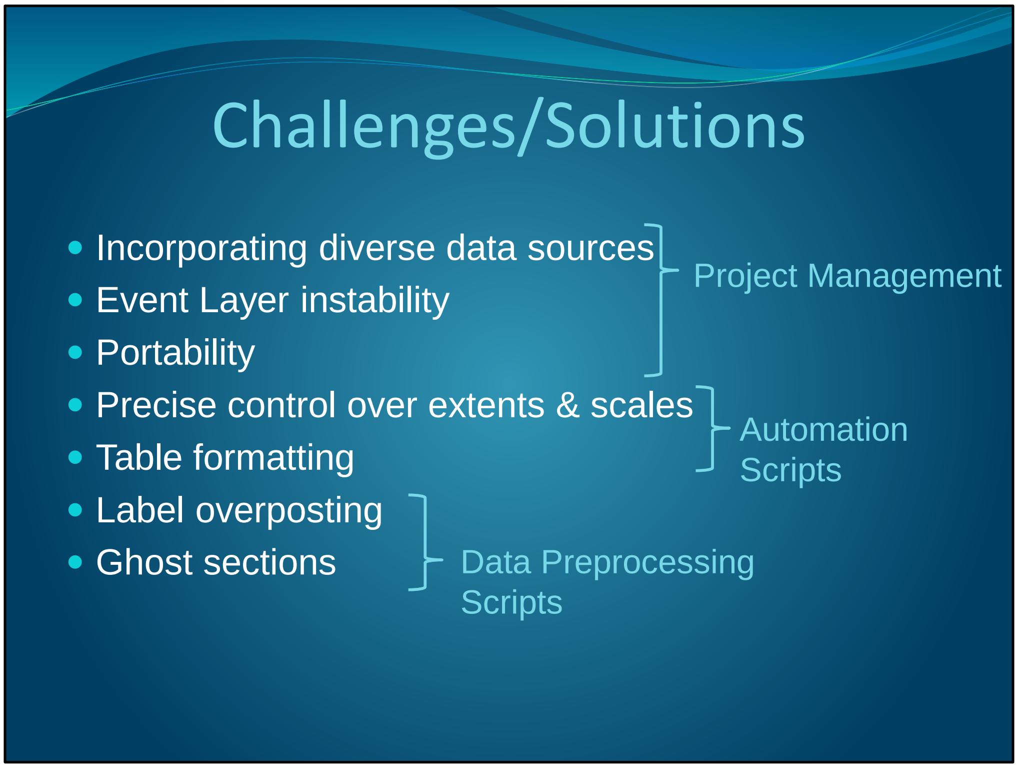
Back to Data PreProcessing Scripts….
The final route feature class has town measures so that most data with twn measures does not need to be converted.
Only 3 datasets need to be converted



Data Preprocessing Scripts

- Create station dataset, determine label offsets
- Determine intersection label offsets
- Transform historic crash locations to current LRS
- Create event layers (position features along the line routes using dynamic segmentation and TWN measures)
- Convert event layers to feature classes
- Dissolve roadwidth features for tidy rendering

I’ll tell you more about stations when I describe how the labels are offset



Challenges/Solutions

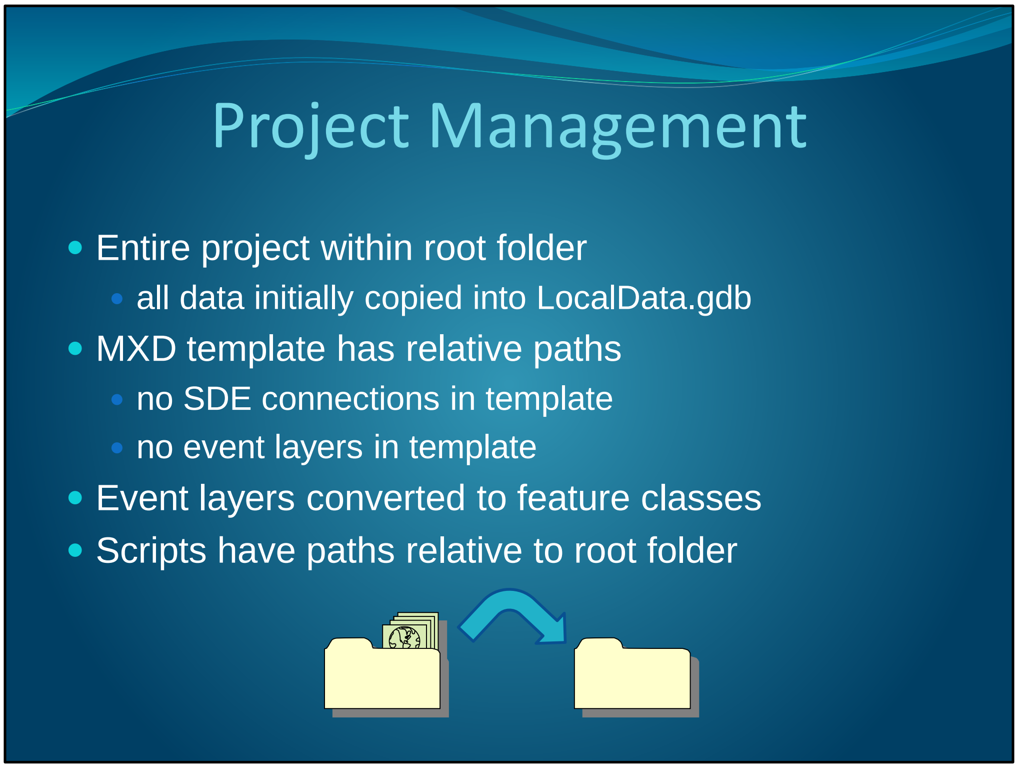
- Incorporating diverse data sources
- Event Layer instability
- Portability
- Precise control over extents & scales
- Table formatting
- Label overposting
- Ghost sections

$$\left.\begin{array}{l}\text{Incorporating diverse data sources}\\ \text{Event Layer instability}\\ \text{Portability}\end{array}\right\}\ \text{Project Management}$$

$$\left.\begin{array}{l}\text{Precise control over extents \& scales}\\ \text{Table formatting}\end{array}\right\}\ \text{Automation Scripts}$$

$$\left.\begin{array}{l}\text{Label overposting}\\ \text{Ghost sections}\end{array}\right\}\ \text{Data Preprocessing Scripts}$$

Now back to highlighting some challenges and solutions that I haven't already covered.



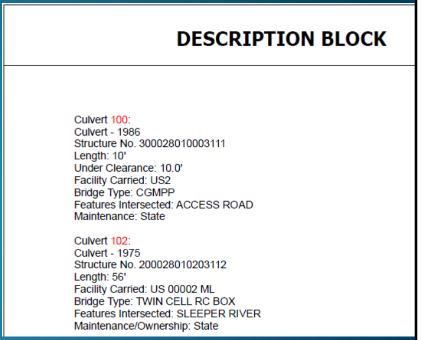
Project Management

- Entire project within root folder
  - all data initially copied into LocalData.gdb
- MXD template has relative paths
  - no SDE connections in template
  - no event layers in template
- Event layers converted to feature classes
- Scripts have paths relative to root folder

Easy one first!
If there is network reorganization that affects the paths of data sources, only one script is affected.

Bridge Description Table

- Table is a single string assembled during automation
- String includes Python and ArcGIS formatting tags



DESCRIPTION BLOCK

Culvert 100:
Culvert - 1986
Structure No. 300028010003111
Length: 10'
Under Clearance: 10.0'
Facility Carried: US2
Bridge Type: CGMPP
Features Intersected: ACCESS ROAD
Maintenance: State

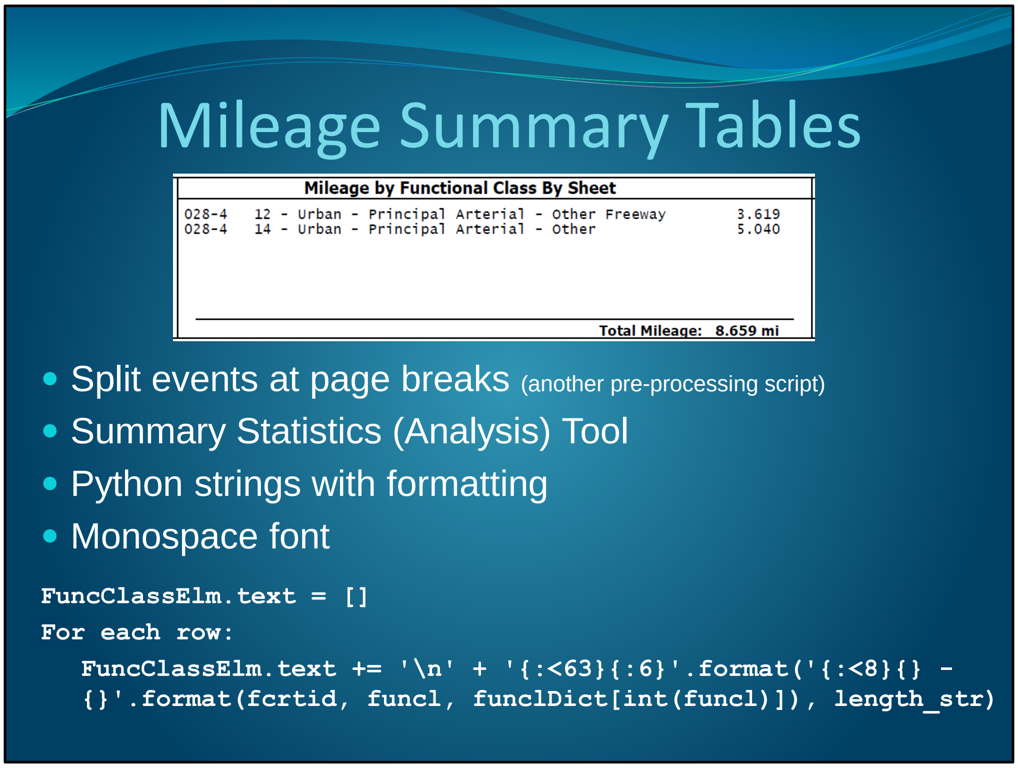
Culvert 102:
Culvert - 1975
Structure No. 200028010203112
Length: 56'
Facility Carried: US 00002 ML
Bridge Type: TWIN CELL RC BOX
Features Intersected: SLEEPER RIVER
Maintenance/Ownership: State

In Python:

```
BridgeDescriptionElm.text = '%s <CLR red = "255">'%struc_categ
   + label + '</CLR>: ' + '\r\n'
   + struc_type + ' - ' + str(yr_built) + '\r\n'
```

In ArcMap text element:

```
Culvert <CLR red = "255">100</CLR>:
Culvert - 1986
```



Mileage Summary Tables

Mileage by Functional Class By Sheet

```
028-4    12 - Urban - Principal Arterial - Other Freeway         3.619
028-4    14 - Urban - Principal Arterial - Other                 5.040
```

Total Mileage: 8.659 mi

- Split events at page breaks (another pre-processing script)
- Summary Statistics (Analysis) Tool
- Python strings with formatting
- Monospace font

```
FuncClassElm.text = []
For each row:
   FuncClassElm.text += '\n' + '{:<63}{:6}'.format('{:<8}{} - {}'.format(fcrtid, funcl, funclDict[int(funcl)]), length_str)
```

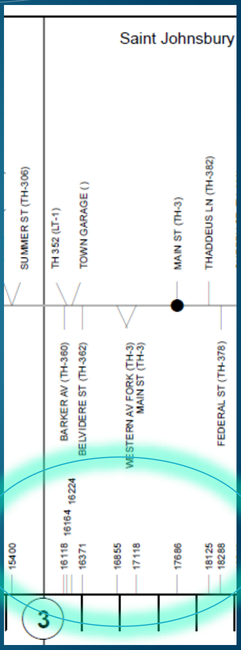
Here is an example of a Functional Class Mileage Summary Table

In order to create this table, I had to split Functional Class events at page breaks and run the event table through summary statistics
The table is a single string that grew line by line as it looped through the summary statistics table.

The values are extracted from datasets, and the formatting is native to Python. This table is using nested formatting.

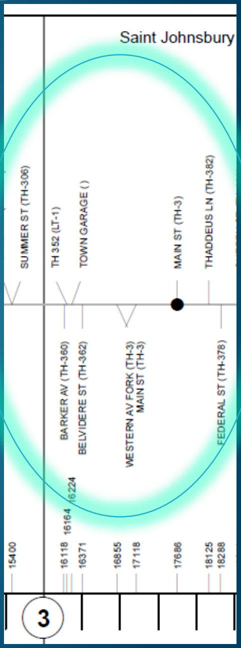
Station Labels

- Copy all stations data to one table
- Add “offset” field
- Define proximity cutoff
- Consider records L to R
- Is current label too close to nearest non-offset label to its left?
- If so, its offset = previous label’s offset + 1
- L to R order of labels strictly maintained

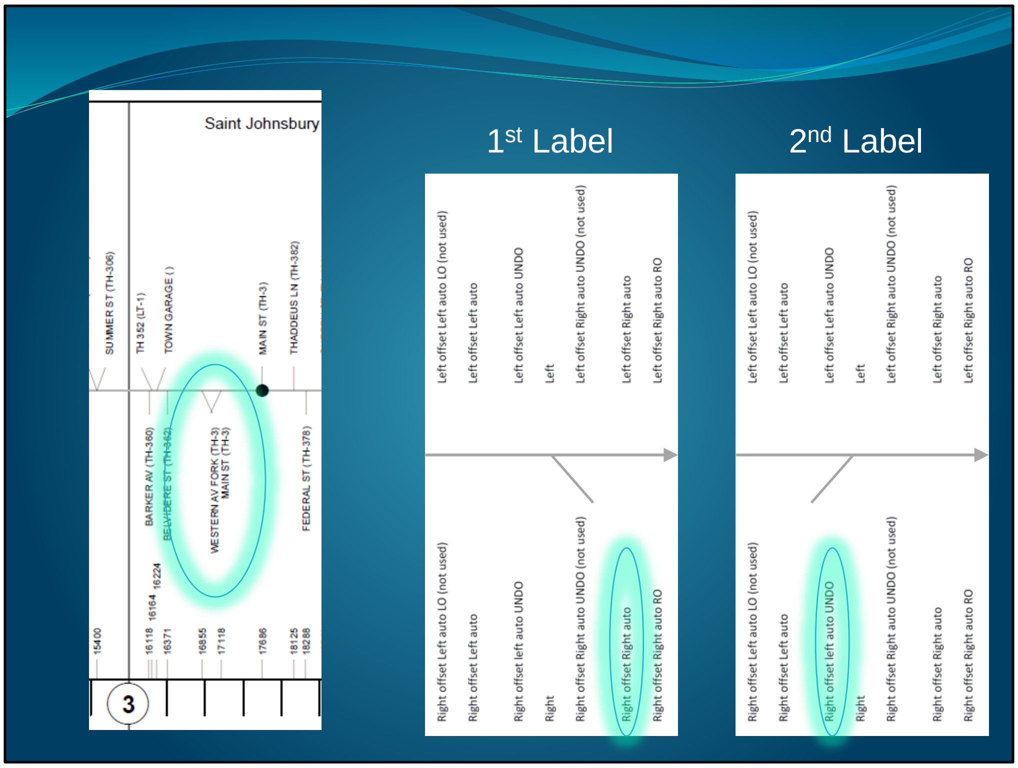


Intersection Labels

- Non-perpendicular intersection labels have offsets (if possible)
- Label have offsets to avoid labels to their left
- L to R order strictly maintained
- 36 label scenarios depending on:
  - Side of road
  - Previous intersection's angle
  - Previous intersection's offset type
  - Current intersection's angle
- 14 label classes



Its not perfect, but does make a big difference.



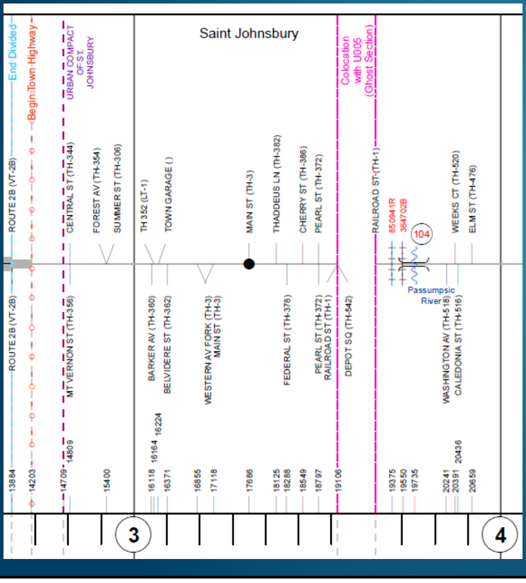
I'll talk you through how the label class was determined for these two labels:
1st label – Standard non-perpendicular offset
2nd label – Standard offset would cause it to overlap with the previous label that has an offset, but at least there is still room for it to shift slightly in that direction without causing overlap.

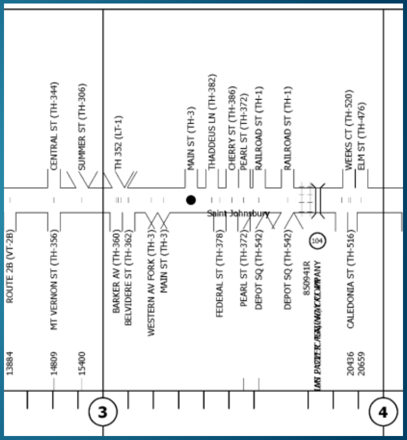
Why bother with that shift? The script doesn't know whether there will be another label closely following the one it's considering, so the shift could potentially provide needed space as well as align the label with the intersection better.

I just have to make sure that the script assigning label class considers offsets that are the same size as those set in the Label Classes.

Ghost Intersections

2014 Route Log


Saint Johnsbury
End Divided
Begin Town Highway
URBAN COMPACT OF ST. JOHNSBURY
Colocation with U005 (Ghost Section)
Passumpsic River
104
3
4



2008 Route Log

Ghost intersections are intersections that coincide with the beginning or end of a ghost section.

The mile markers for these intersections correspond to two locations along the SLD because the mile marker at the start of a ghost section is the same as the mile marker at the end of the ghost section.

So where is the intersection rendered???

In this case, two intersections belong on the upstream end of the ghost section, and one belongs on the downstream side. The linear referencing tools currently place them all at the upstream location. ArcGIS 9.X placed them all at both locations.

I didn’t want to introduce minute errors into the data just to get some points to plot downstream, so I flagged those points, and positioned those using the intermediate route feature class based on routelog measures that are different at the ends of ghost sections.

Thanks!

VTrans Mapping Unit:

- Johnathan Croft
- Michael Trunzo
- Sara Moulton
- Gary Smith
- David Narkewicz

Esri:

- Jeff Barrett
- Roads & Highways Team

VTrans Working to Get You There Vermont Agency of Transportation

Contact:

Kerry.Alley@state.vt.us