



CENTER FOR TRANSPORTATION STUDIES

ITS INSTITUTE

**DEVELOPMENT OF
TRAFFIC SIMULATION LABORATORY
FOR DESIGN PLANNING
AND TRAFFIC OPERATIONS
(PHASE II)**

**Paul Telega and Panos Michalopoulos
Department of Civil Engineering
University of Minnesota**

REPRODUCED BY: **NTIS**
U.S. Department of Commerce
National Technical Information Service
Springfield, Virginia 22161

DISCLAIMER

The contents of this report reflect the views of the authors, who are responsible for the facts and the accuracy of the information presented herein. This document is disseminated under the sponsorship of the Department of Transportation, University Transportation Centers Program, in the interest of information exchange. The U.S. Government assumes no liability for the contents or use thereof.

*PROTECTED UNDER INTERNATIONAL COPYRIGHT
ALL RIGHTS RESERVED*
NATIONAL TECHNICAL INFORMATION SERVICE
U.S. DEPARTMENT OF COMMERCE

Reproduced from
best available copy.



1. Report No. CTS 00-02	2.	3. Recipient's Accession No.	
4. Title and Subtitle Development of Traffic Simulation Laboratory for Design Planning and Traffic Operations (Phase II)		5. Report Date June 2000	
7. Author(s) Paul Telega and Panos Michalopoulos		6.	
9. Performing Organization Name and Address University of Minnesota Civil Engineering Department 500 Pillsbury Drive SE Minneapolis, MN 55455		8. Performing Organization Report No.	
12. Sponsoring Organization Name and Address Intelligent Transportation Systems Institute of the Center for Transportation Studies University of Minnesota 200 Safety and Transportation Building 511 Washington Avenue SE Minneapolis, MN 55455-0375		10. Project/Task/Work Unit No.	
15. Supplementary Notes		11. Contract (C) or Grant (G) No. (C) (G)	
16. Abstract (Limit: 200 words) A key element in improving traffic operations and performing effective real-time traffic management is using simulation to assess the effectiveness of various alternatives prior to implementation. The research conducted here is Phase II of a three phased project with the ultimate goal of creating and running traffic simulation experiments in real-time. In the first phase, a set of well-known freeway simulators was evaluated. Major difficulties were a lack of real data, and the time consuming effort required to prepare data for each simulator. Phase I found that developing an integrated traffic analysis environment, where data processing, simulation and output analysis can be automated as efficiently as possible was of critical importance in improving traffic management and operations. In the second phase, the development of an Automated Simulations Tool (AST) was of critical importance. Phase II was partitioned into four major tasks: development of a Geometry Data Container (GDC), Creation of a partial Twin Cities Freeway Geometry, development of an AST, and specification of a real-time AST framework. Each part of this phase was essentially prescribed by the Phase I results. The GDC would be the design and implementation of a common geometry database that could be shared among different simulators. Initially, only a micro simulator would be implemented, but later other simulators could be added. Creation of a partial Twin Cities Freeway Geometry would be the base level common geometry that is entered. All the detail needed for a micro simulation is entered including freeway sections, detector locations and types, ramp meters, and other fine details. This work needs to be done only once with our design. Any subnetwork of the original entered network can be selected with a mouse and saved as a new network. Development of the AST will allow traffic engineers to select a freeway and a time period for simulation and then essentially run a simulation without any direct manipulation of data. Traffic engineers will not need to now anything about the data formats of either geometry or flow data in order to run a simulation.		13. Type of Report and Period Covered	
17. Document Analysis/Descriptors		14. Sponsoring Agency Code	
19. Security Class (this report) Unclassified	20. Security Class (this page) Unclassified	18. Availability Statement No restrictions. Document available from: National Technical Information Services, Springfield, Virginia 22161	21. No. of Pages 22. Price



**DEVELOPMENT OF TRAFFIC SIMULATION
LABORATORY FOR DESIGN PLANNING AND TRAFFIC
OPERATIONS (PHASE II)**

FINAL REPORT

Prepared by

Paul Telega and Panos Michalopoulos
Department of Civil Engineering
University of Minnesota
Minneapolis, MN 55455

June 2000

Published by

Intelligent Transportation Systems Institute of the
Center for Transportation Studies
University of Minnesota
200 Safety and Transportation Building
511 Washington Avenue SE
Minneapolis, MN 55455-0375

This report represents the results of research conducted by the authors and does not necessarily represent the views or policy of the Center for Transportation Studies. This report does not contain a standard or specified technique.



ACKNOWLEDGEMENTS

This research was supported by the Intelligent Transportation Systems (ITS) Institute in the Center for Transportation Studies (CTS) at the University of Minnesota and Guidestar. The authors would also like to thank the following individuals for their contributions to this document.

James Aswegan – *Minnesota Department of Transportation*

Lowell A. Benson – *Center for Transportation Studies*

Ron Dahl – *Mn/DOT's Traffic Management Center*

John Hourdakis – *University of Minnesota Department of Civil Engineering*

Dr. Eil Kwon – *Center for Transportation Studies*



Table of Contents

TABLE OF CONTENTS.....	III
TABLE OF FIGURES.....	V
EXECUTIVE SUMMARY	
1 INTRODUCTION.....	1
1.1 PROBLEM STATEMENT.....	1
1.2 RESEARCH OBJECTIVES.....	1
1.3 BACKGROUND/ HISTORY/ PAST WORK.....	3
1.4 WORK SUMMARY.....	4
1.5 REPORT ORGANIZATION.....	4
2 GEOMETRY DATA CONTAINER.....	5
2.1 OBJECTIVES.....	5
2.1.1 <i>Centralized site for storing a network and One-time creation of data for each network</i>	5
2.1.2 <i>Allow data extensibility in the GDC</i>	6
2.1.3 <i>Generic access protocol</i>	6
2.1.4 <i>Data manipulation ability</i>	7
2.2 G.I.S CONSIDERATIONS.....	7
2.2.1 <i>Capability (lack of)</i>	7
2.2.2 <i>Ease of use</i>	8
2.2.3 <i>Cost of software</i>	8
2.2.4 <i>Uniformity and availability of G.I.S.</i>	9
2.3 DESIGN OF THE GDC.....	9
2.3.1 <i>Relational database model chosen</i>	10
2.3.2 <i>ODBC/SQL chosen to access the database</i>	10
2.3.3 <i>Relationships between tables</i>	14
2.4 IMPLEMENTATION.....	16
2.4.1 <i>GDC Prototype Implementation</i>	16
2.4.2 <i>Blocks</i>	17
2.4.3 <i>Texts</i>	18
2.4.4 <i>Vehicle Classes</i>	18
2.4.5 <i>Global Messages</i>	18
2.4.6 <i>Sections</i>	19
2.4.7 <i>Nodes</i>	19
2.4.8 <i>Controllers</i>	19
2.4.9 <i>Centroids, Roguis, and Routes</i>	20
2.4.10 <i>Network</i>	20
2.4.11 <i>ODBC Functions</i>	20
2.5 I35WI94 TWIN CITY FREEWAY GEOMETRY.....	20
2.5.1 <i>Background Image of the Network</i>	22
2.5.2 <i>Trace over the Sections in the Network</i>	24
2.5.3 <i>Add additional detail</i>	25
2.5.4 <i>Add meters, detectors, and controllers</i>	26
2.5.5 <i>Save the resulting network</i>	26
2.5.6 <i>Problems and solutions</i>	27
3 SENSOR AUTOMATION DEVELOPMENT.....	29
3.1 OBJECTIVES.....	29
3.2 WHAT WE NEED.....	29

3.2.1	<i>Determine user input for geometry selection (point and click) and modify traffic editor to save subnetworks</i>	30
3.2.2	<i>Modify/augment geometry database to include flows</i>	31
3.2.3	<i>Define user input for traffic volume selection</i>	32
3.2.4	<i>Design an algorithm to add traffic volume to each entrance section for each simulation state</i>	33
3.3	PUTTING IT ALL TOGETHER	34
3.4	MAKING A SIMULATION RUN	35
4	SENSOR DATA AUTOMATION USE	37
4.1	OBJECTIVES	38
4.1.1	<i>Visually (with a mouse) select the network to be simulated</i>	38
4.1.2	<i>Eliminate the need for any manual entry of traffic network geometry (other than a one time entry)</i>	39
4.1.3	<i>Automatically retrieve traffic volume data for the selected network from the traffic volumes</i>	41
4.1.4	<i>Automatically generate the necessary simulation states using the selected network and the traffic volumes</i>	42
4.2	SPECIFICATION OF DATA REQUIREMENTS FOR SIMULATION	42
4.2.1	<i>Geometry selected and saved</i>	42
4.2.2	<i>Traffic volume database</i>	43
4.2.3	<i>User selection network, date, and time periods to simulate</i>	43
4.2.4	<i>Control plan for the selected geometry</i>	44
4.2.5	<i>Simulation states generated by the AST tool</i>	44
5	FRAMEWORK FOR PHASE III WORK	45
5.1	AUTOMATION OF CONTROL PLANS	45
5.2	AUTOMATION OF RESULTS PRESENTATION	46
5.3	INTEGRATING ANOTHER SIMULATOR WITH THE GDC	46
5.4	REAL-TIME ACCESS TO MNDOT DETECTOR DATA FOR SIMULATION	47
6	CONCLUSIONS AND RECOMMENDATIONS	49
6.1	PROBLEMS THAT WE ENCOUNTERED	49
6.1.1	<i>Problems with the GDC</i>	49
6.1.2	<i>Problems with the Automated Simulation Tool</i>	53
6.2	THINGS THAT WE LEARNED	54
6.3	WHAT WE WOULD DO DIFFERENT, IF WE HAD TO DO IT AGAIN	55
6.4	MAJOR ACCOMPLISHMENTS	56
6.5	CONCLUSIONS	57
Appendix A: Database Specifications		A-1
Appendix B: Relational Database Functions		B-1
Appendix C: Sample of Traffic Volumes provided by Mn/DOT		C-1

Table of Figures

FIGURE 2-1: TRAFFIC EDITOR AND SIMULATOR WITH AN API CONNECTING IT TO THE GEOMETRY.....	10
FIGURE 2-2 : SIMPLE BLOCK DRAWN IN THE TRAFFIC EDITOR.....	12
FIGURE 2-3: REPRESENTATION OF THE BLOCK IN FIGURE 2-2 IN ASCII NETWORK FORM	13
FIGURE 2-4: REPRESENTATION OF THE BLOCK IN FIGURE 2-2 IN DATABASE FORM.....	14
FIGURE 2-5: RELATIONAL DATABASE WITH TABLES AND RELATIONSHIPS	15
FIGURE 2-6: I35W FROM I694-I494 VIEW FROM WITHIN THE TRAFFIC EDITOR	21
FIGURE 2-7: BACKGROUND MAP OF TWIN CITIES ROADWAYS (GIS VIEW)	22
FIGURE 2-8: SIMPLIFIED MAP WITH MUCH LESS DETAIL (ESSENTIALLY I35W).....	23
FIGURE 2-9: CLOVERLEAF VIEW FROM WITHIN THE TRAFFIC EDITOR	25
FIGURE 3-1: VIEW OF THE AST ACCESSING THE NETWORK GEOMETRY	31
FIGURE 3-2: THE AST WITH INPUTS AND THE AST GENERATED SIMULATION STATES.....	34
FIGURE 4-1 OVERVIEW OF THE AST, THE SIMULATOR, THEIR INPUTS, AND THE RESULTS.....	37
FIGURE 4-2 I35W AT I494, SUBNETWORK SELECTED IN GRAY	39
FIGURE 4-3: SECTION 717 AT I35W-I494 CLOVERLEAF. SECTION 717 HAS NO DETECTORS IN IT.	40
FIGURE 4-4: FREEWAY SECTION WITH DETECTORS, METERS, AND CONTROLLERS	41
FIGURE 6-1: SOFTWARE ARCHITECTURE FROM THE AST TO THE GDC/ASCII NETWORK	50



EXECUTIVE SUMMARY

A key element in improving traffic operations and performing effective real-time traffic management is using simulation to assess the effectiveness of various alternatives prior to implementation. The research conducted here is Phase II of a three phased project with the ultimate goal of creating and running traffic simulation experiments in real-time. In the first phase, a set of well-known freeway simulators was evaluated. Major difficulties were a lack of real data, and the time consuming effort required to prepare data for each simulator. Phase I found that developing an integrated traffic analysis environment, where data processing, simulation and output analysis can be automated as efficiently as possible was of critical importance in improving traffic management and operations.

In the second phase, the development of an Automated Simulations Tool (AST) was of critical importance. Phase II was partitioned into four major tasks: development of a Geometry Data Container (GDC), Creation of a partial Twin Cities Freeway Geometry, development of an AST, and specification of a real-time AST framework. Each part of this phase was essentially prescribed by the Phase I results.

The GDC would be the design and implementation of a common geometry database that could be shared among different simulators. Initially, only a micro simulator would be implemented, but later other simulators could be added.

Creation of a partial Twin Cities Freeway Geometry would be the base level common geometry that is entered. All the detail needed for a micro simulation is entered including freeway sections, detector locations and types, ramp meters, and other fine

details. This work needs to be done only once with our design. Any subnetwork of the original entered network can be selected with a mouse and saved as a new network.

Development of the AST will allow traffic engineers to select a freeway and a time period for simulation and then essentially run a simulation without any direct manipulation of data. Traffic engineers will not need to know anything about the data formats of either geometry or flow data in order to run a simulation.

Finally, from what we have learned in Phase II we can specify what needs to be done for a real-time implementation of the AST: Phase III. By real-time we mean using the detector data as soon as it is available from Mn/DOT for simulation.

1 Introduction

1.1 Problem Statement

The key element in improving traffic operations and performing real-time management is the ability to assess the effectiveness of various alternatives prior to implementation. Simulation methods have long been recognized as the most effective tool for such analysis, and various simulators have been developed by different agencies for analyzing freeway and/or arterial networks. However, simulation has not yet become a suitable tool for practical application. One reason for this is the extensive manual labor required to input the different kinds of traffic data into most simulation programs, which points to the need for a simulation tool that provides automatic access to large amounts of traffic data. The purpose of this research in Phase II, therefore, is to develop an *Automated Simulation Tool* (AST) with automatic access to both traffic geometry and traffic measurement data. The AST will be part of a *Laboratory Environment for TRaffic ANalysis* (LETRAN), which will provide easy and efficient access to various kinds of traffic data for use in simulation, control, incident detection, and other types of traffic analysis applications to be deployed in a next-generation traffic management center.

1.2 Research Objectives

Our major objective is to develop viable, intelligent, and automated traffic flow simulation systems. These systems can then be deployed in the next-generation traffic

management center, and can be enabling advanced research as part of the ITS Laboratory infrastructure.

The ultimate objective of this research (in collaboration with other projects) is to develop a *Laboratory Environment for TRaffic ANalysis* (LETRAN) as part of the CTS ITS Lab. Such an environment would provide an integrated platform for traffic data management, model development and deployment, and the investigation of next-generation traffic management applications. The specific goal of this research is to develop an *Automated Simulation Tool* (AST) as an important part of this environment. This goal can be subdivided into two parts. The first is the creation of a *Geometry Data Container* (GDC) which provides a centralized and extensible location to store and manage traffic geometry data to be used by simulators and other lab analysis tools. The second is the development of the AST, which uses both the GDC and the *Traffic Data Management System* (TDMS) created in the “I-394 Lab” project for automated creation of simulation experiments.

In Phase II, we divided the project into four major objectives:

- Development of the Geometry Data Container (GDC)
- Creation of the I35W Freeway Geometry
- Development of an Automated Simulation Tool (AST)
- Specification of a Real-Time AST Framework

1.3 Background/ History/ Past Work

The full Simulation Lab Project is designed to be executed in three major phases:

- Phase I: Freeway simulator testing and evaluation
- Phase II: Development of the Automated Simulation Tool (AST)
- Phase III: Integration of Simulation Lab and I-394 Lab.

Phase I of this research evaluated a set of well-known freeway simulators, including AIMSUN, FREFLO, FREQ, FRESIM, INTEGRATION, and KRONOS. Standard test cases were developed and run on different simulators using real data collected by the Minnesota Department of Transportation (Mn/DOT). This evaluation process has already highlighted the difficulties of using current simulation software in a practical fashion, primarily due to the large amount of effort required to create traffic geometries and input traffic data. This work has therefore pointed to a need for the development of an automated traffic simulator where manual data input is minimized and automated input of data to the simulator is maximized.

An additional concept for improving simulation which was explored in Phase I was the idea of a *Common User-Interface* (CUI). A CUI would provide a single interface through which a user could create geometries, input traffic data, and run multiple simulation tools. A prototype CUI was developed with the assistance of Dr. Jaime Barcelo and researchers from the Universitat Politecnica De Catalunya from Barcelona, Spain which uses the GETRAM interface to create simulation input files not only for AIMSUN (for which it was created), but also for the traffic simulator developed at the University of Minnesota (KRONOS). This process has helped to determine that the

development of a CUI for even a small number of simulators is not a feasible task, primarily because of the vastly diverse modeling methodologies and approaches that different simulators possess. Therefore, a more feasible and equally useful goal was chosen: the development of an *Automated Simulation Tool* (AST) for making traffic simulation faster, easier, and more accessible to traffic engineers and managers.

In Phase II, which is discussed in the paper, we developed and implemented an Automated Simulation Tool that minimizes the tedious data entry, which is required for microscopic simulation. Traffic data from the Minnesota Department of Transportation can be automatically prepared for use in simulation.

In Phase III, the work that was done in Phase II will be extended so that real-time traffic data can be used in simulation and integrate this with the I-394 Lab and TRACLAB.

1.4 Work Summary

In completing this project, the following major work was completed:

- Creation of a traffic geometry database system
- Creation of a Twin Cities freeway geometry (partial)
- Development of the automated simulation tool

1.5 Report Organization

This report is organized essentially in order of major task.

2 Geometry Data Container

2.1 Objectives

The fundamental objectives of designing this Geometric Data Container (GDC) are to create:

- Centralized site for storing a network
- One-time creation of data for each network
- Allow data extensibility in the GDC
- Generic access protocol
- Data manipulation ability

2.1.1 Centralized site for storing a network and One-time creation of data for each network

A centralized site for storing all the major networks that are studied will exist at CTS in the ITS Laboratory. Only a one-time entry of each major network will be necessary. It will eliminate duplication of effort in the entering of and the updating of such networks. Researchers can then take advantage of already existing networks and use them in their work. Extra efforts will be made to keep the central database for each network up to date.

2.1.2 Allow data extensibility in the GDC

One major feature of database systems is that they allow augmentation of an existing database without disrupting current applications. That is, additional data fields and/or tables can be added to the current database to accommodate additional applications. In our case, additional fields or tables would contain information that allow other simulators (when suitably modified) to use the same database. This means that other simulators (once modified) can potentially use the same central geometry database that is created by our traffic editor for simulation and select only the information that they need in order to build the simulation model.

2.1.3 Generic access protocol

A generic access protocol to the database was selected called Open DataBase Connectivity (ODBC). ODBC allows any database system that supports the ODBC protocol to be used for information retrieval. This means that we do not need to program database commands for a specific database. All of our programs will be able to use any database that is ODBC compliant. Currently, Microsoft ACCESS, ORACLE, SYBASE, and most other major database systems have ODBC drivers. So by writing our programs using ODBC, we will have the capability to access many different database systems without reprogramming. For deployment in the field, this means that our system can more easily adapt to a commercial database that an engineering group already has without costly reprogramming.

2.1.4 Data manipulation ability

Choosing a database also shifts the burden of data management to the database system. That is, with a real database system all we need to do in an application is make store and retrieve requests of the database. The database system will then execute these commands for us. We don't need to worry about actually writing these to separate files or exactly where the data is. Not only will the database execute commands from the TSS, but any other program can also send commands to the database to retrieve or update information. The ASCII network file version will not allow data manipulation to be done without writing specific programs. For example, another program or the database tools could be used to query the I35W database for the number, type, and age of the detectors.

2.2 *Gis Considerations*

The Geographical Information Systems (GIS) that were reviewed for this project were not able to provide enough detail for microscopic simulation. This review of GISs was completed in Phase I. The following paragraphs describe some of our reasons for not selecting a GIS at this time.

2.2.1 Capability (lack of)

GISs organize information into layers. For example, a map could be constructed that has a pavement layer, a soil layer, and a hydrology layer. Each of these layers is separate and do not interact with the other layers. In the highway systems that we design and work with a roadway will pass over another roadway and then further down it will

then pass under the same roadway. No GIS reviewed can handle this type of situation. In this case, it cannot be determined in which layer the freeway passing under and over another freeway belongs. This is a fundamental problem for GIS systems that model highways.

GISs are based on geometric shapes such as lines, arcs, and polygons, but not intuitive objects such as a directed roadway with three lanes. It may be possible to modify a GIS to handle this situation, but it would be cumbersome and non-intuitive to use. Generally, trying to adapt a program to do something that it was not intended to do will usually result in frustration and an inadequate representation of the problem itself.

2.2.2 Ease of use

Most GIS's reviewed were not simple to use. They required special editors that were not what a traffic engineer would be familiar with, and they provided many more types of modeling features most of which were not needed for traffic modeling. These extra modeling features could lead to confusion and frustration.

2.2.3 Cost of software

Finally, the cost of GIS specific software was very high when deployed to the field. Commercial pricing could be as high as \$50,000 or higher. In addition, special training would be necessary to use it properly.

2.2.4 Uniformity and availability of G.I.S.

Most GIS's had no standard way of storing data. Each GIS had its own method of dealing with saved data making it difficult to interface with other programs. In addition, it was not entirely clear that any of the GISs provided an application programming interface (API) that could be used to get at the geometric features that it managed.

2.3 *Design of the GDC*

The Geometry Data Container designed here will be implemented in a relational database. The database design required analysis of the original structure of the information written to the ASCII file system, and the development of a set of tables and relationships between the tables. The relationships between the tables are defined by how information is used in the original ASCII file design. The GDC contains the geometry, detector, metering, and variable message sign information.

Changing the underlying structure of the geometry to create the GDC will have no visual or operational impact on the user of the traffic simulation system. The format of the geometry data is essentially hidden from the user of the system. It makes no difference if this information is stored in separate files, a relational database, or an object database because the application programming interface (API) actually interacts with the information directly—not the user. In figure 2-1, the GDC in the lower left part is the new feature being added with major additions and modifications to the API.

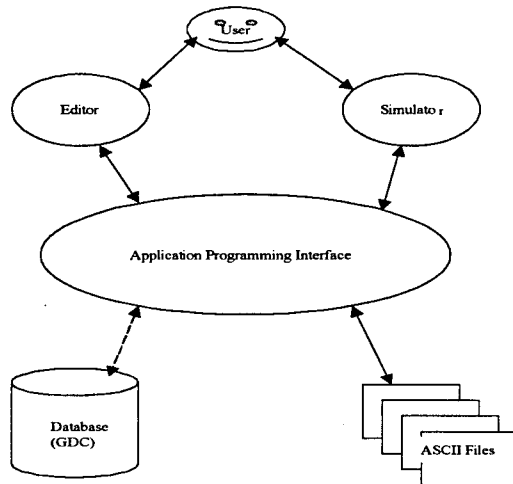


Figure 2-1: Traffic editor and simulator with an API connecting it to the geometry.

2.3.1 Relational database model chosen

Originally our design was to include an object database, but the complexity of implementing completely object oriented database was too much. The TSS data that was stored was not directly object oriented, but it behaved somewhat like an object oriented system. Unfortunately, the API in the TSS was not object oriented and would be very difficult to convert. Therefore, since a relational database would provide many of the features we wanted, was much less expensive, and the learning curve was much less, we decided to use a relational database system.

2.3.2 ODBC/SQL chosen to access the database

Once the database was chosen, we had to make another decision on how to communicate with it. We again had two choices: pick a particular database and use its

proprietary commands, or choose a generic protocol that most databases would be able to communicate with.

In the first case, we would have to design functions for a particular database. If we decided to switch to a different database later, it would cause many of the functions to be redesigned for a different database. This would be very costly.

In the second case, a generic protocol such as Open Database Connectivity (ODBC) with embedded Structured Query Language (SQL) would allow most functions designed to work with many different database systems with little reprogramming. There is some small cost because generic protocols many run somewhat slower than native database functions, but the flexibility of using other databases far outweighs this cost.

Finally since the SQL commands are embedded within the ODBC, additional functions can be easily written by a non-expert to search the database for specific patterns or other information of interest to the engineer. Without a database and ODBC/SQL these types of searches require custom programming which can be very expensive.

In order to build a database from the original format of the geometry data, we had to do an analysis of the current ASCII Network system of files and define a mapping that would carry the information from the files to a database. The ASCII Network was composed of eleven files: network, blocks, texts, vehicle classes, global messages, sections, nodes, controllers, centroids, roguis, and routes. Each of these files contains specific information on some part of the geometry.

The simplest file is Blocks. It contains a list of blocks including block points, which provide a visual background for the user. Each block could represent a building, while each block point is defines the actual outline of the block. All this information

exists in one file. In mapping this to a database, two tables are defined: blocks, and blockpoints. In figure 2-2, a simple block is drawn in the traffic editor and its representation in ASCII Network format is in figure 2-3. Finally, figure 2-4 represents the GDC representation in two tables.

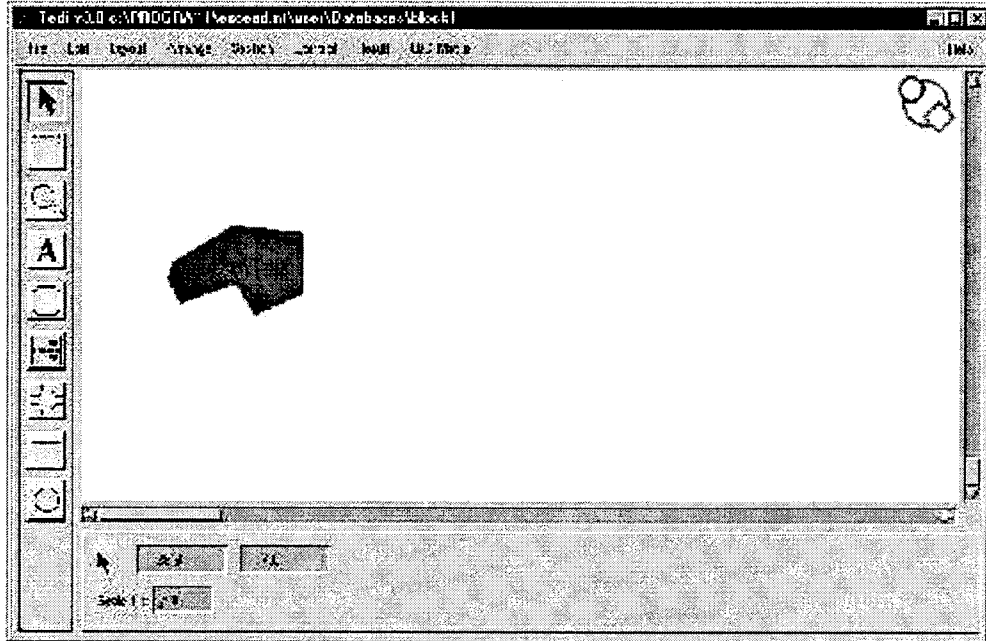
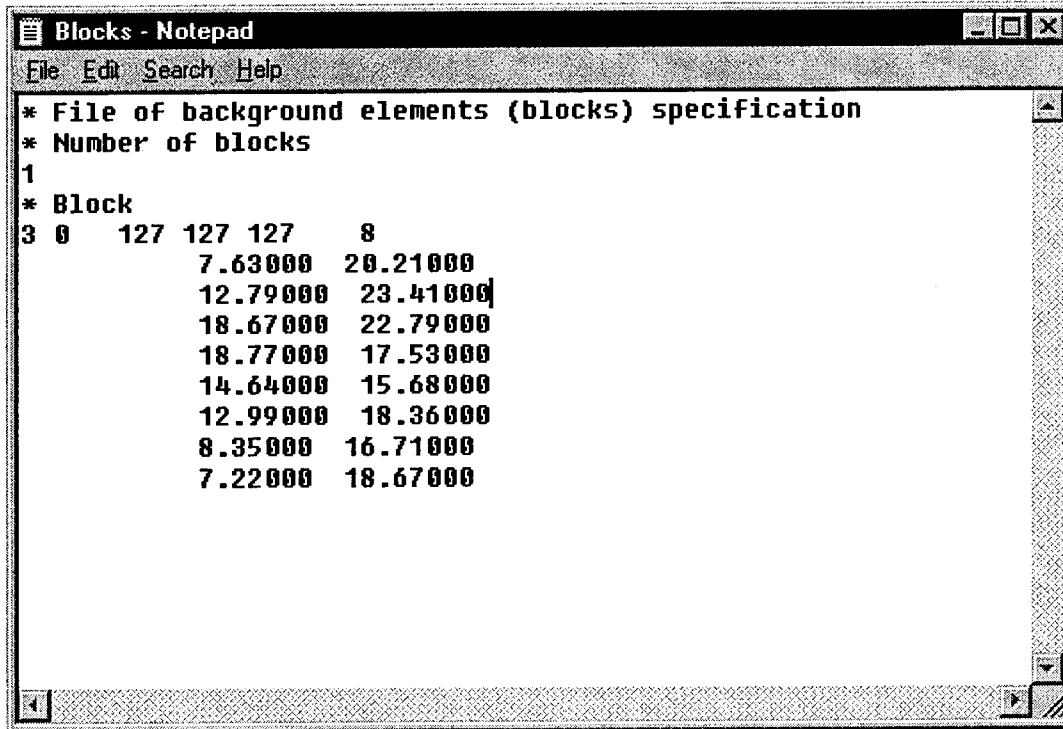


Figure 2-2 : Simple block drawn in the traffic editor



```
Blocks - Notepad
File Edit Search Help
* File of background elements (blocks) specification
* Number of blocks
1
* Block
3 0 127 127 127 8
    7.63000 20.21000
    12.79000 23.41000
    18.67000 22.79000
    18.77000 17.53000
    14.64000 15.68000
    12.99000 18.36000
    8.35000 16.71000
    7.22000 18.67000
```

Figure 2-3: Representation of the block in Figure 2-2 in ASCII Network form

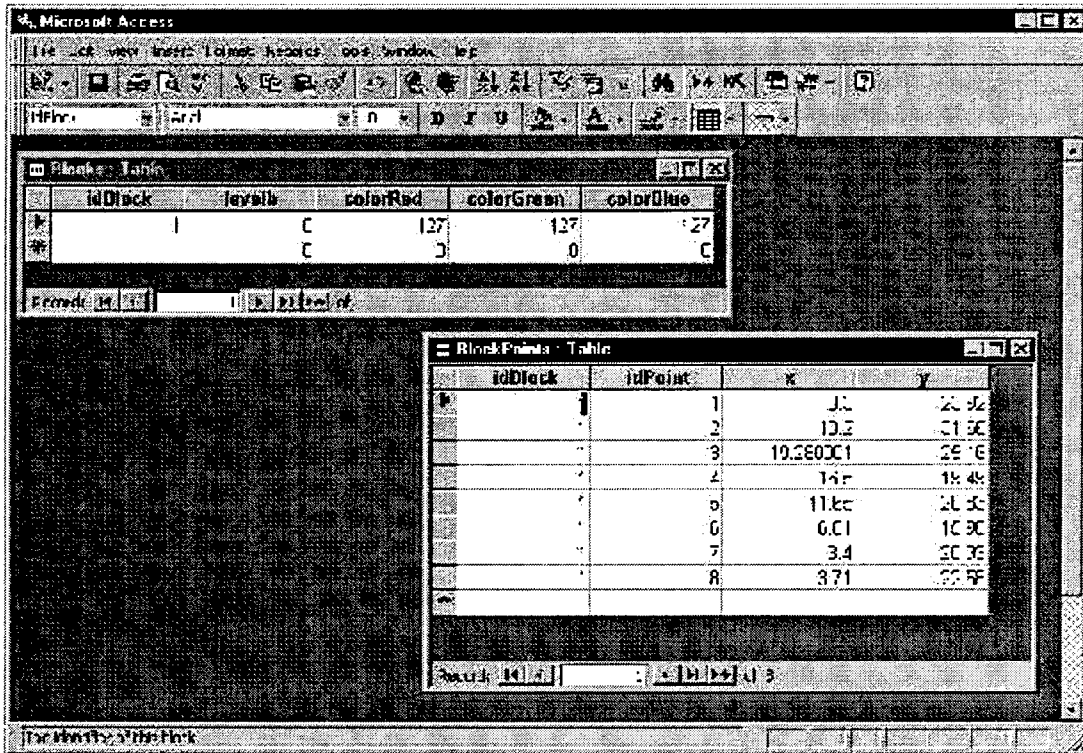


Figure 2-4: Representation of the block in Figure 2-2 in database form.

The most complex file is Sections. This file contains all the information about sections including meterings, detectors, turnings, rights, capacity, speed, elevation, and other geometry information. Detailed information on the table SECTIONS and other tables can be found in Appendix A.

2.3.3 Relationships between tables

It is not enough just to define tables. To use a relational database efficiently, relationships between the tables must be defined. Usually defining these relationships occurs naturally from the way that the tables were derived.

2.4 Implementation

This design of a database required the introduction of approximately 105 low level functions that will interact with the database during implementation. The functions are grouped into classes, which are similar to the ASCII network version of these functions. All the functions will be defined and implemented in the API of the system. They will be transparent to any user of the system. All the details of the function definitions are contained in Appendix B. In this section we give a basic overview of the functions and what they do.

2.4.1 GDC Prototype Implementation

The geometry data container (GDC) is implementation of the original geometry system done in ASCII Files, but redesigned to make it functional in a relational database format. This redesign of functions in the input/output portion of the traffic simulation system (TSS) is not a trivial rewriting of code. Our initial design of the database includes more than 105 completely new functions at the lowest level. Each function accesses the relational database system for either retrieval, updating, or saving of geometry or geometry-like information. These functions can be grouped into functional categories that operate on specific parts of the geometry. The functional classes correspond to their analogs in the original ASCII version of the geometry and are defined as follows:

- **Blocks:** blocks help in visualizing the area being simulated
- **Texts:** text used to identify objects being viewed

- Vehicle Classes: vehicle class definitions such as HOV->TRUCKS+BUSES
- Global Messages: global messages for drivers
- Sections: geometry information on sections of roadways
- Nodes: nodes link sections to form networks
- Controllers: controllers link meters and detectors to nodes/sections
- Centroids: used in origin/destination and route selection
- Roguis: used in origin/destination and route selection
- Routes: used in origin/destination and route selection
- Network: general network characteristics

In order to explain the implementation, it is necessary to understand not only the overall structure of the classes listed above, but the individual functions that need to be implemented for the relational database to function. In the following, we will explain how each category is used in the implementation and a definition and explanation of each function that needs to be implemented. More than 105 new functions had to be designed and implemented to make the GDC fully operational.

2.4.2 Blocks

Blocks are background objects (polygons) used to help visualize the traffic network. Blocks are polygons that can have shape and color. For example, blocks can be used visualize physical features such as rivers, buildings, and other like scenes. In figure 1, a block is drawn using high level functions that ultimately call the low level Block functions described below.

Implementing Blocks in the relational database requires defining eight new functions. In order to understand the implementation, it is necessary to explain each function, its parameters, and what action that it should perform on the database—Appendix B defines these functions in detail.

2.4.3 Texts

Texts are background texts, which are used to identify parts of the traffic network. Texts can be used to identify scenes such as rivers, buildings, other scenes, as well as parts of the traffic network. They are not used directly in simulation.

2.4.4 Vehicle Classes

Vehicle Classes are groupings of types of vehicles. For example, one might define an HOV1 (High Occupancy Vehicle) class with taxis and buses, and another, HOV2, with trucks and large trucks. Now lanes could be reserved in sections just for HOV1 and HOV2 type vehicles and no others.

2.4.5 Global Messages

Global Messages are messages that are visible on either variable message signs, or changeable message signs that are used on modern freeways. Each section can contain a variable message sign with the potential messages and potential actions by drivers once the message sign is activated. For example, if freeway traffic is heavy then the sign may

be set to read "Congestion Ahead, Use Alternate Routes." Once the sign is activated then a potential action would be for 10% of the drivers to take the next exit.

2.4.6 Sections

Sections contain the basic geometry of the network. A network itself is composed of a set of sections linked together by nodes. The sections themselves contain information about the physical characteristics of each section in the network such as number of lanes, lane length, lane width, capacity, maximum speed, type of roadway, slope, and contour.

2.4.7 Nodes

Nodes are the links that form an actual network. Nodes link the sections together to form networks. In general, there are two type of nodes: junctions, and junctures. Junctions are links between sections that have driver selectable turnings, and junctures are links between sections that have no driver selectable turnings—no intersections.

2.4.8 Controllers

Controllers connect detectors and meters for traffic counting and controlled entry. They provide a way to implement controls plans for a network. Any information that is received from detectors or sent to meterings and variable message signs is essentially routed through a controller.

2.4.9 Centroids, Roguis, and Routes

Centroids, roguis, and routes are used by the TSS system to help determine traffic flow given origin/destination matrices.

2.4.10 Network

Network contains the basic information about the network such as type, the default values for different types of structures. It also includes indices for background files that can be traced over to produce a network.

2.4.11 ODBC Functions

ODBC functions represent a new set of functions that perform all the storing/retrieval to and from the relational database. Any of the other functions defined above must call these functions in order to communicate with the database.

2.5 *I35W/I94 Twin City Freeway Geometry*

In order to test our tools, we chose a forty mile section of I35W from I694 to I494 and the intersection of I35W and I94 for about three miles. Figure 2-6 shows the zoomed out view of this segment of I35W in the traffic editor. This set of sections should be enough geometry to sufficiently test the simulation modeling, and the tools that we are building. In addition, building this section of the Twin Cities traffic network should give us a good way to measure the time and effort that it takes to enter other traffic networks of similar complexity.

The software that we have chosen has a traffic network editor that allows visual input of the network geometry. This greatly simplifies the entry, and the accuracy of the data entered. The general plan of attack for entering a new network is: get a background image of the network, trace over the sections in the network, add additional detail, add detectors and controllers, and save the resulting network. Finally, we discuss some of the problems that we encountered and how we solved them. Each of these tasks is quite a bit more complex than it seems. For example, entering a cloverleaf took much more time to complete than entering a straight ramp. The following sections describe the effort in entering a section of freeway.

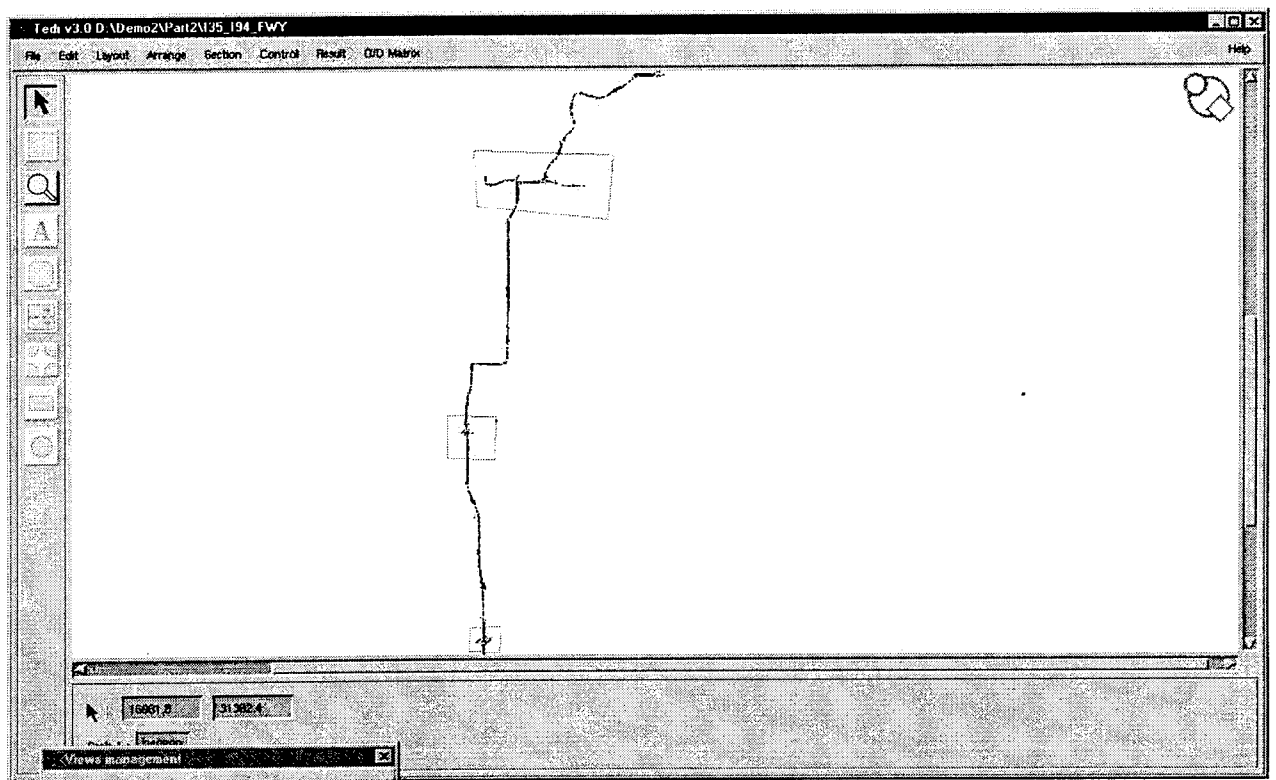


Figure 2-6: I35W from I694–I494 view from within the Traffic Editor

2.5.1 Background Image of the Network

A background image of the I35W/I94 freeway was provided to our research group by MN/DOT. The image was done in Microstation, a CAD program that Mn/DOT uses for keeping track of the freeway system. The image contained much more information than we needed. The image, figure 2-7, that we received contained the entire network of state aid roads for the Twin Cities. The size of the file was 54MB and took thirty minutes to load on a Silicon Graphics Workstation using the traffic network editor. Because the file contained more than what we needed and because the loading time was so significant we edited out essentially all the details except for the freeway sections of interest.

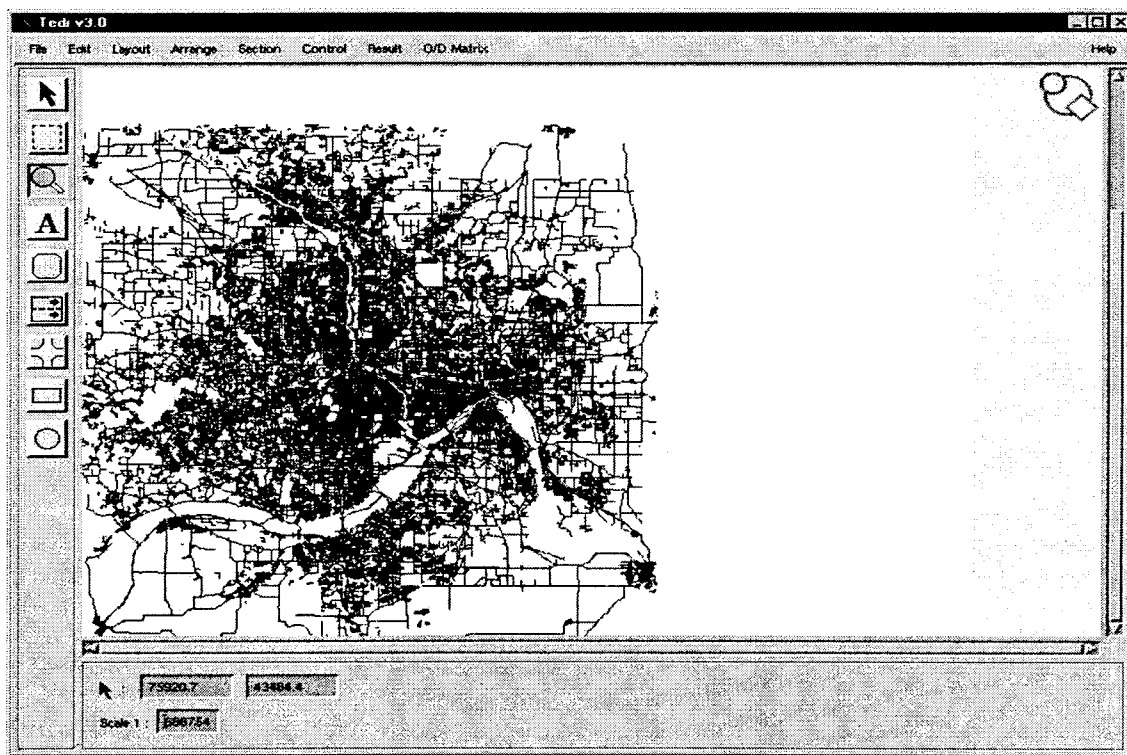


Figure 2-7: Background map of Twin Cities roadways (GIS view)

This new file was much smaller, about 3MB and took less than one minute to load. The background file is only needed for the initial entry of the geometry data explained in the next step. In figure 2-8, the difference in the number of details is visible. The background image only provides a centerline drawing of the freeway. Additional detail, such as number of lanes, turnings, detectors, and controllers, is provided by the ASB files, which were also provided by Mn/DOT. This background image and the ASB files are used to essentially trace out a geometry for the network.

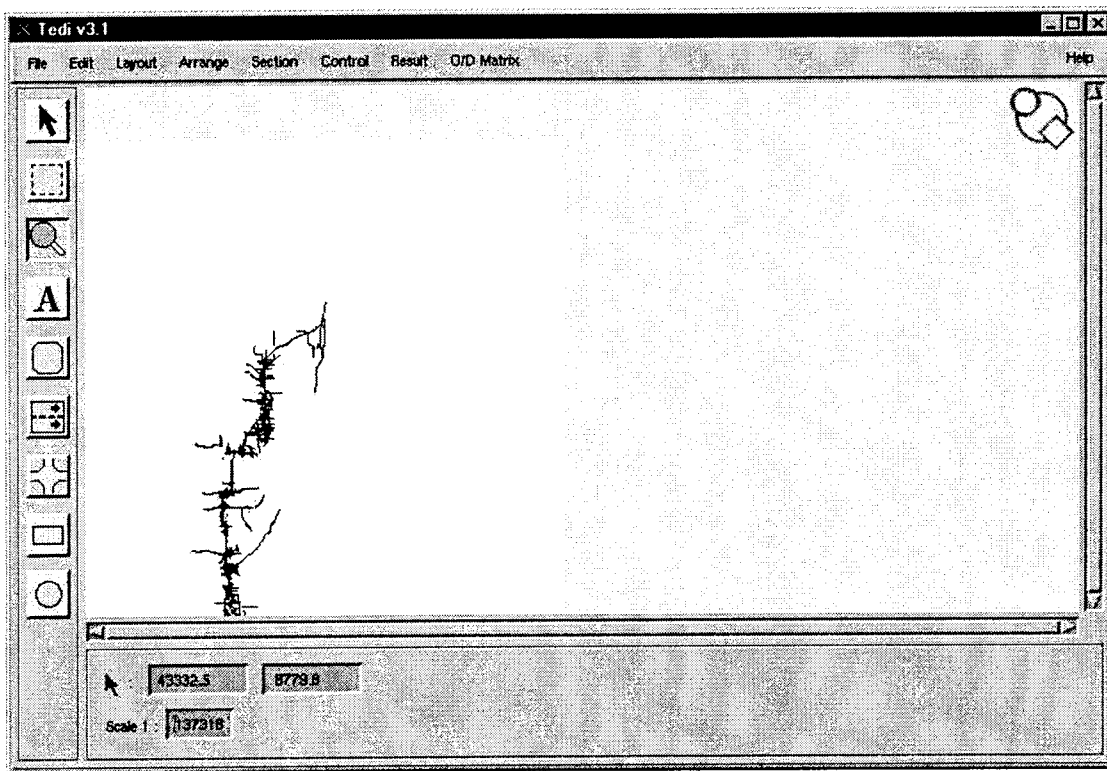


Figure 2-8: Simplified Map with much less detail (essentially I35W)

2.5.2 Trace over the Sections in the Network

Using the traffic network editor, the background image is opened and registered. Now we just start from the top and begin drawing freeway sections over the centerline image and adjust the detail to what the ASB files describe. For example, a section with 4 lanes , no turnings, no detectors, and no controllers. This drawing of sections is very tedious and time consuming, but it is visually accurate, and only needs to be done once. Drawing it with other systems would require the entering of coordinates for each section—this would even be more effort.

Actually, drawing the sections only becomes the most difficult with cloverleaves because these roadways have very sharp turning curves. The system that we are using has only straight sections for modeling networks, but sections can be angled where they meet other sections to produce curves. The result is that a single curve in a cloverleaf may be composed of 10-20 sections, and a full cloverleaf may contain 40-80 sections. Figure 2-9 shows a partial cloverleaf that was entered with the traffic editor, the numbers on the curves are the section identifiers.

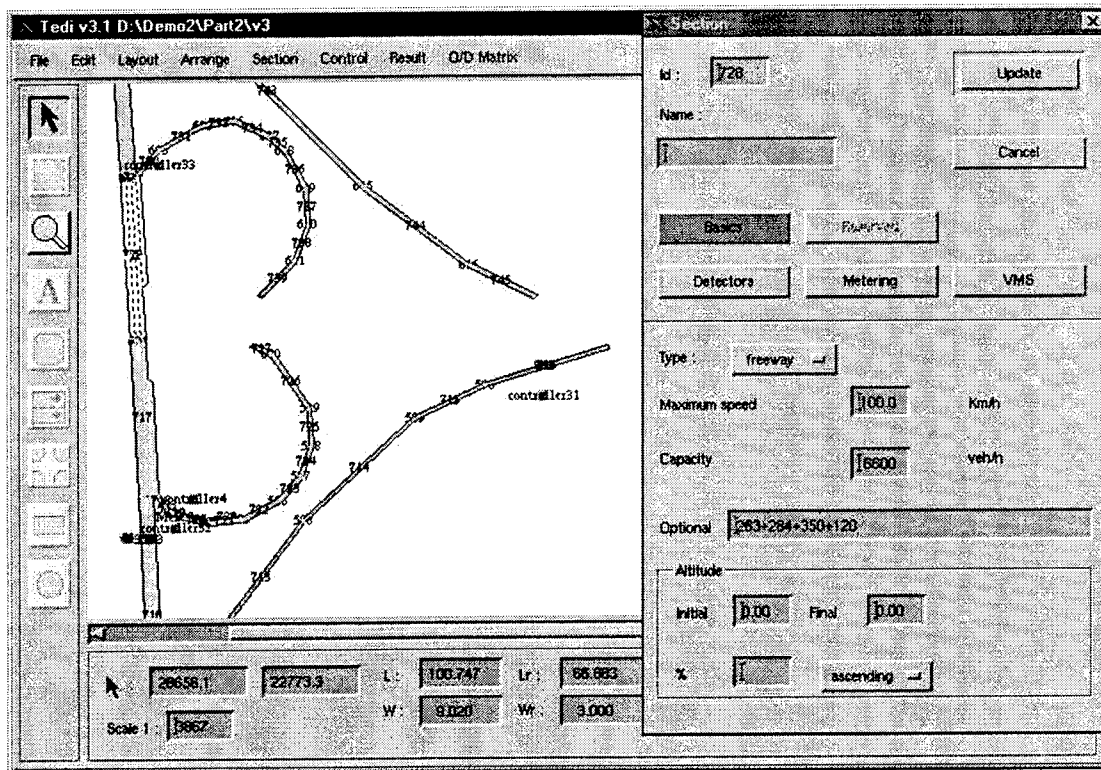


Figure 2-9: Cloverleaf view from within the traffic editor

Freeway sections need to be entered for each direction vehicles are to travel in because each section in the network can only provide for traffic flow in one direction.

2.5.3 Add additional detail

Once the basic geometry of the freeway is entered, the other details of the network geometry are added. This information is available from the ASB files such as section capacity, maximum speed, type of roadway, level, and slope. All these parameters help to determine the actual flow in a section based purely on the geometry. Although the network appears visually, all the detail parameters used here are entered numerically. They provide information to the simulator so that it can calculate

acceleration, deceleration, and speed for vehicles in a section. In addition, they also can be used to help calculate pollution caused by the vehicles used during simulation.

2.5.4 Add meters, detectors, and controllers

Now that all the geometry has been entered, meters, detectors, and controllers can be added. Again these devices are attached to each section in the geometry. Meters are classified by type (green-time, delay, and flow) and are usually connected to a controller. Detectors are also classified by type (inductance loop, pneumatic tube, coaxial cable, tape switch, and queue-length). In our example, we used green time for meters, and inductance loops for detectors. Detectors also had to have the correct identification so that when volume information is requested from one of the Mn/DOT traffic volume databases the correct detector volume information is transferred. This correspondence will be important when using the tool to automatically create the simulation states for a simulation. Mn/DOT provided the detector location file so that the correct detector identifiers were correct.

2.5.5 Save the resulting network

Once all the geometry information, along with metering and detector information is entered, the next step is to name and save the network. Actually, after entering several sections it's a good idea to save the work. There were times when entering section data that the system became inoperable and all the work was lost since the last save. So, as in

using all software it is prudent to save often and keep multiple versions so that major system crashes do not cause loss of much data and effort.

2.5.6 Problems and solutions

Generally all the problems that we had could be classified into two types: underestimating of the scope of work, and deciding how to model freeway characteristics that are not overtly in the model.

Our original intent was to model the entire Twin Cities freeway network because with a graphical editor this would be fairly fast—we thought. Entering the freeways with all the detail that we have took about three hours per mile of freeway. See figure 2-4 for how complex some freeways can be.

Some freeway sections merged from four lanes into three lanes. This particular characteristic was not directly available in the model, but our solution was to model two sections: one with four lanes and the other with three lanes with an intersection connecting them. The only additional detail that had to be done was to determine the correct turnings for vehicles moving from one section to the other.

The simulation system that we use has very many primitive structures that can be used to create complex systems, but this requires a good knowledge of the system in order to come up with a solution.

3 Sensor Automation Development

3.1 Objectives

The primary objective is to develop a program that takes as input network geometry, traffic volume data, and user specified simulation times to generate the simulation input files. That is, reduce the amount of time it takes to prepare data for a simulation. In the phase I report, it was found that input file preparation could take anywhere from one to twenty days, but in most cases it took only one day. The AST tool described here can reduce that day to a few minutes. This time saving should help make simulation a process that can be used more often and with much less effort.

3.2 What we need

In order to automate the simulation, the following need to be developed:

- Determine user input for geometry selection (point and click)
- Modify traffic editor to save subnetworks
- Modify/augment geometry database to include flows (traffic volumes)
- Define user input for traffic volume selection
- Design an algorithm to add traffic volume to each entrance section for each simulation state.

3.2.1 Determine user input for geometry selection (point and click) and modify traffic editor to save subnetworks.

It is assumed that the geometry network is either the complete network to be simulated or a superset of the network to be simulated. If the entire network is to be simulated then no modification of the geometry database is necessary.

On the other hand, if the network to be simulated is a subset of the current network then the user of the traffic editor must somehow select it. How is this to be accomplished? Well, the original version of the traffic editor allowed users of it to make certain views of the whole network. These views showed only the area selected, not the whole network. Unfortunately, the views still held all the information in memory and did not give up any space. Views are generally created by using a mouse to draw a polygon around the subnetwork of interest. We modified the traffic editor so that a view could be created and then saved as a complete network. In addition, the memory that was not in the saved view was released making the size of the geometry smaller.

Using the view for selection of a subnetwork and saving this view as a new network we now have the complete subnetwork (network) to work with. In our design, the AST tool works with the entire geometry presented to it. Therefore, it is necessary to save only that part of the network that is needed for simulation.

In the selection of a subnetwork, it does not matter whether the network is a relational database or an ASCII network. The AST tool uses the Application Programming Interface (API) to access the geometry. The API itself actually makes the connection to the network. Figure 3-1 shows essentially how the communication to the geometry is made. If you look closely, it does not matter what form the geometry is

stored in. The detail of whether the true underlying data are stored in a database or an ASCII network is hidden from the user. There is no need for the user to understand all the details just to do a simulation. A normal user of this system will work at the level of a traffic engineer and work with objects that he will understand, not fine details that make the program work.

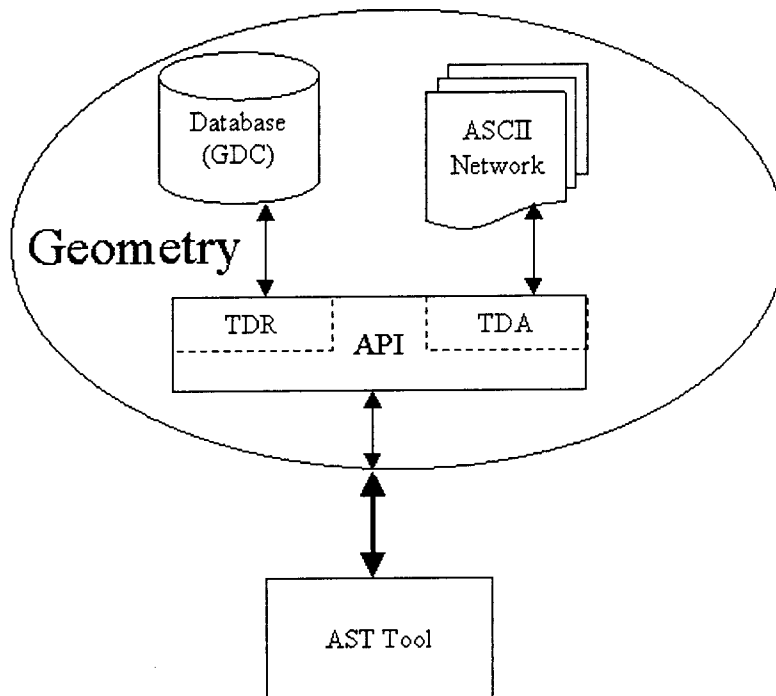


Figure 3-1: View of the AST accessing the network geometry

3.2.2 Modify/augment geometry database to include flows.

In order to simulate the network saved in the previous step, the geometry database had to be augmented because the saved network did not preserve all the detector flow information. That is, some sections may be entrance (boundary) sections, but they may

not be associated with a detector. Since detectors essentially give the flows for this model, no detector information would be provided to these sections and the simulation states could not be constructed.

A simple solution to this problem was to determine by hand which upstream detector(s) provided flow information for each section. This information was then added to the optional input field for each section header. In entering the flows, there are two cases that could exist: only entrance detectors, mixed entrance and exit detectors. Entrance detectors are handled by encoding the detectors with plus signs between them. Mixed detectors are encoded with plus or minus signs depending on whether the detector is at an entrance or exit. Entering this extra information is just a small additional task that can be completed at the same time the detectors are input into the geometry.

3.2.3 Define user input for traffic volume selection

Provide an input menu for the user to make selections. The selections should include the traffic database, the date and a range of times for traffic volume data. An Open DataBase Connectivity (ODBC) compliant database is used to store traffic volume data collected by Mn/DOT. The traffic volume data to be retrieved will be selected from this database by detector number, date, and time. The detectors do not discriminate among the different types of vehicles on the freeway: only a simple vehicle count is given. Although the simulator being used can handle several different vehicle modalities (for example, car, truck, bus), currently only a simple vehicle count is used.

3.2.4 Design an algorithm to add traffic volume to each entrance section for each simulation state

The selection of traffic volumes for each entrance section in the selected network is no trivial task. Both the traffic editor and the micro simulator have a very good application programming interface (API) connecting them to the geometry data. The API provides the programmer with access to any element that exists in the geometry.

There are API functions that determine which sections are entrance sections to the selected network. Each time an entrance section is found, then a special field encoded in the section is read. The information read lists the detectors that determine the flow for this section. If more than one detector is listed then there must be a '+' or a '-' between them indicating whether the value given by the detector is to be added or subtracted from the flow. A subtraction would occur if there were an exit ramp in this section.

Once the entrance sections and the flows are determined, the simulation states can be defined. Simulation states are the time slices that provide new input for each time period in the simulation. Producing these simulation states is the main purpose of the AST program.

In generating simulation states, for each time slice (for example, 5 minute period) the special encoded field is read from each entrance section (for example, 5+8-10), the flows are calculated from the detectors listed here, and the net flow is written to this simulation state section. These steps are repeated for each entrance section in the network. Now a simulation state file can be saved for this time slice. Next repeat the same process for the next time slice, and then repeat until all the simulation states have been computed. All these files created are stored in the same directory. Collectively,

they are called the simulation states. The file extension on all the files created is '.stt', indicating system states. These files are exactly what the micro simulator needs to perform a simulation. Without the AST, this complete process would need to be done manually via the traffic editor interface.

3.3 Putting it all together

Combining the geometry, the traffic volume data, and user specified simulation times a sequence of simulation state files is generated. These files will be input for the microsimulator for the traffic network. Figure 3-2, depicts the architecture of the AST tool.

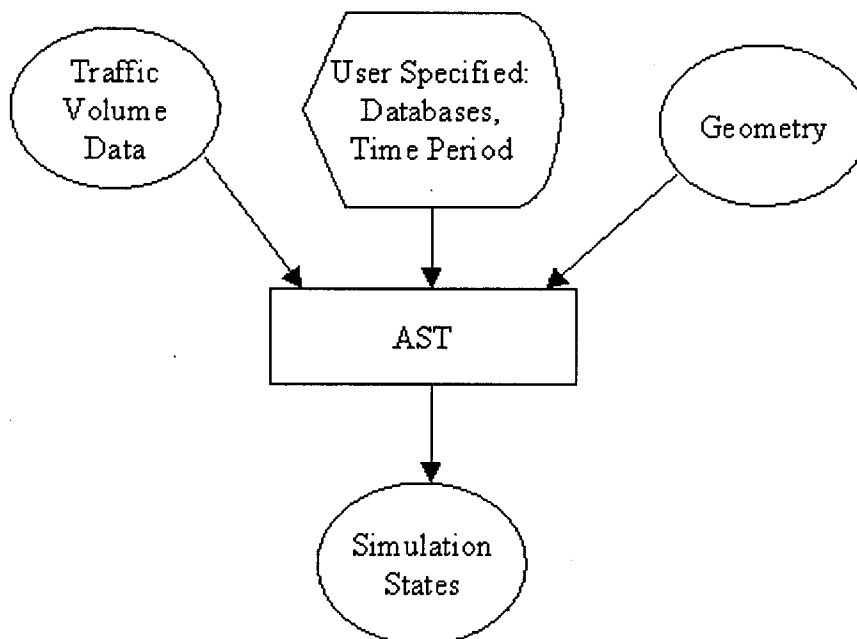


Figure 3-2: The AST with inputs and the AST generated simulation states

3.4 Making a Simulation Run

Finally all the information that is needed for simulation is ready. The simulation states have been generated (and traffic volumes), the geometry is set, a control plan has been defined. The next section will describe how to use the AST tool and then run an actual simulation.

4 Sensor Data Automation Use

In using the Automated Simulation Tool (AST), essentially all manual data entry is eliminated. The AST takes as input the traffic geometry, a user specified time period, and the traffic volumes to generate the simulation states (files) needed for simulation. Figure 4-1 gives a dataflow view of this process. All the real work collecting detector data, and entering a geometry has already been completed. The only real input required is the time period for the simulation states to be generated.

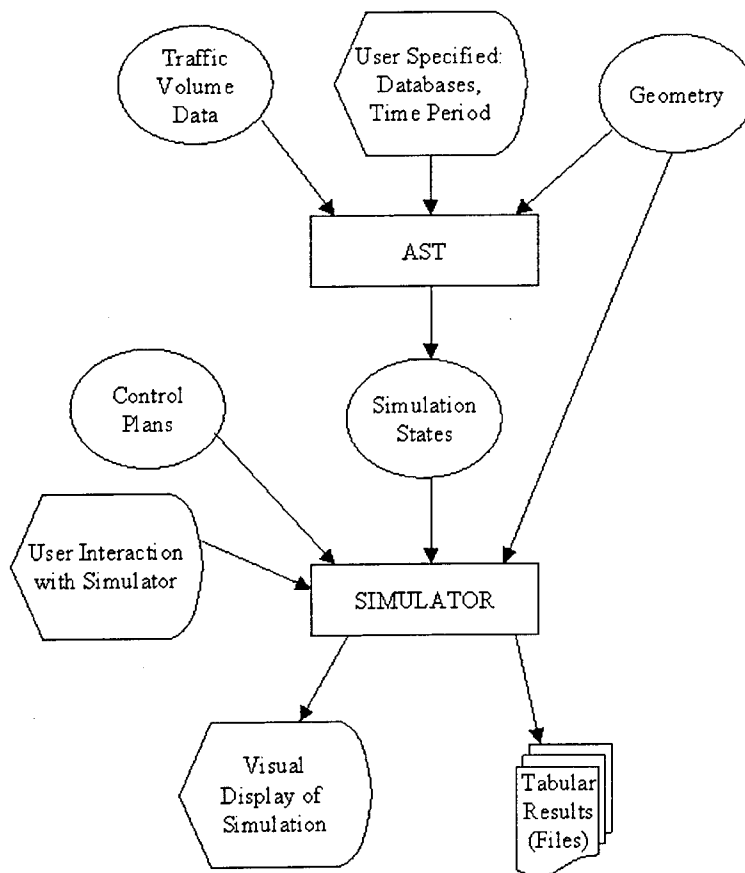


Figure 4-1 Overview of the AST, the Simulator, their inputs, and the results

4.1 Objectives

In general, our objectives were to automate the simulation process so that a simulation could be performed within minutes of the selection of an existing geometry.

- Visually (with a mouse) select the (sub)network to be simulated
- Eliminate the need for any manual entry of traffic network geometry (other than a one time entry)
- Automatically retrieve traffic volume data for the selected network from a traffic volume database
- Automatically generate the necessary simulation states using the selected network and the traffic volumes

4.1.1 Visually (with a mouse) select the network to be simulated.

Once the complete network is entered, we may be interested in simulating just a portion or a subnetwork of the original network. The traffic network editor was modified so that one could use a mouse to outline only that part of the network of interest and save it as a new network. This new network then has only the subnetwork selected, none of the other information is saved relating to parts of the original network. This is precisely why we encoded flows in the optional field in the previous section—so flow information would not be lost by selecting a subnetwork.

Figure 4-2 shows an original network with one area highlighted (rectangle around). This rectangle represents the subnetwork that we have defined. Figure 4-3, shows the view after the subnetwork is saved as a new network. Notice that the optional

field contains "120+283+284+350." This list indicates which detectors are needed to generate the correct flow for section 717.

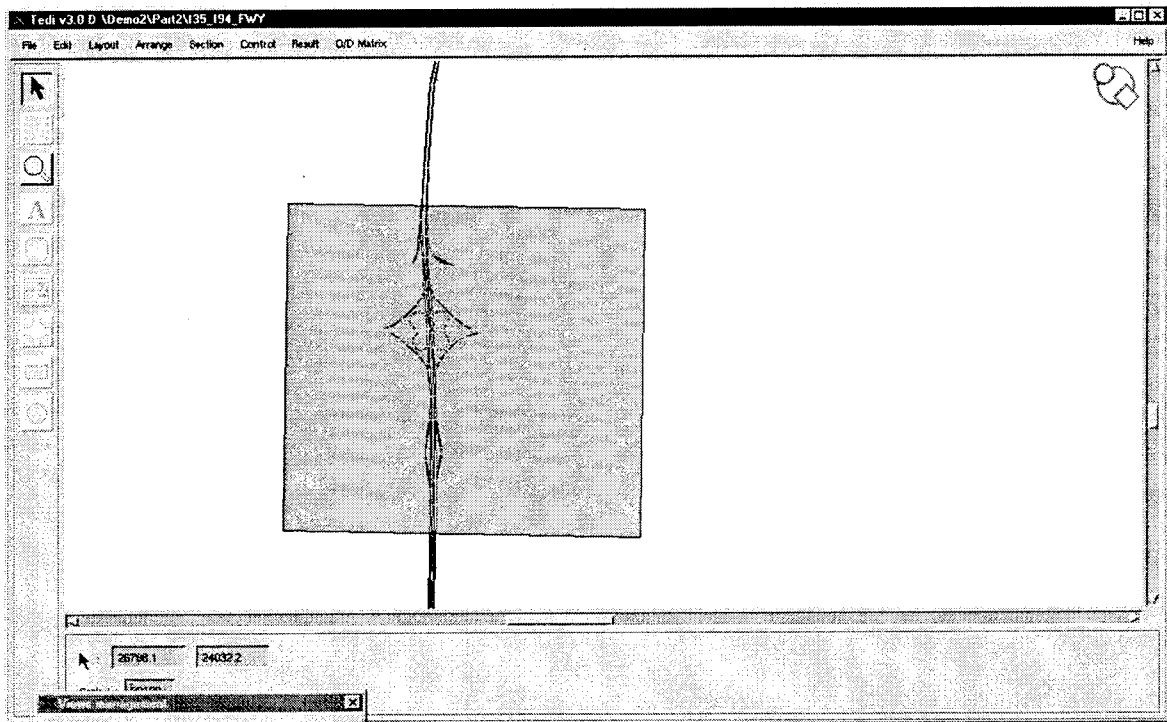


Figure 4-2 I35W at I494, subnetwork selected in gray

4.1.2 Eliminate the need for any manual entry of traffic network geometry (other than a one time entry).

One of the most time consuming activities is entering the traffic network itself. In the previous section, we have already created a traffic network that is part of the Twin Cities freeways so that the only additional pieces of information that are needed for the geometry are the list of entrance sections and their flows. We can encode this into the original geometry by determining which detectors give the flow for each section. These additions need to be done only once for each geometry.

In figure 4-4, the detectors in the first section are numbered 8, 9, and 10. Notice that the optional field to the right of the section contains “8+9+10.” This is the additional information that is encoded in section one indicating which detectors determine the flow for it. It implies that the total flow for this section is completely determined by summing the flows for detectors 8, 9, and 10.

If we extend this concept to the next section (section two), then the optional field here will be “8+9+10+11.” Detector eleven is included because there is an entrance ramp with detector eleven on it. So by extending this idea to all sections, we can select any subnetwork and be able to determine the flow for each section by viewing the optional field.

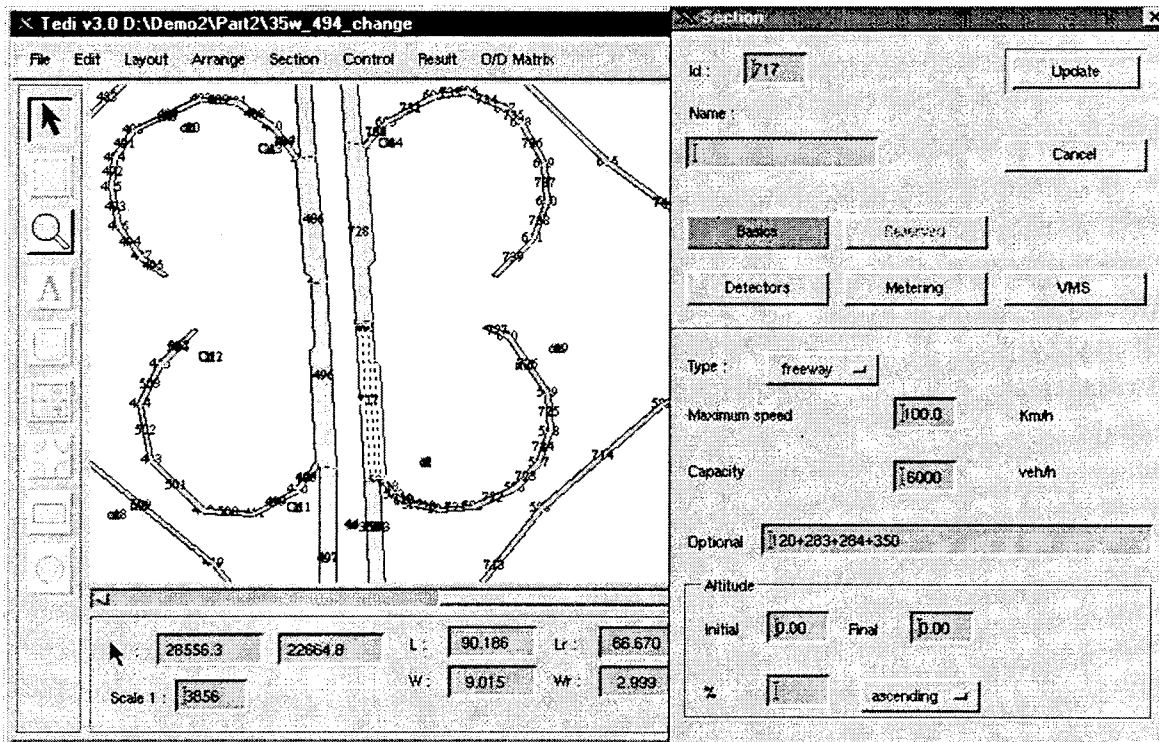


Figure 4-3: Section 717 at I35W-I494 cloverleaf. Section 717 has no detectors in it.

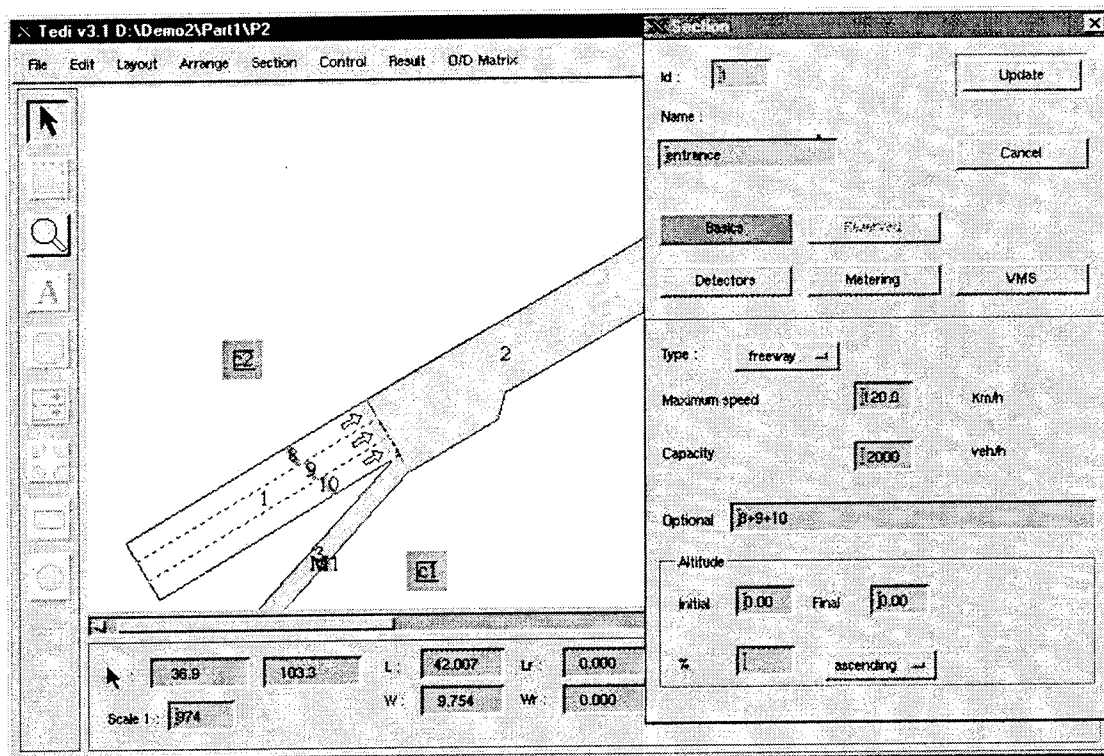


Figure 4-4: Freeway section with detectors, meters, and controllers

4.1.3 Automatically retrieve traffic volume data for the selected network from the traffic volumes.

Our automated tool is designed to gather all the traffic volumes for a specified time period for the set of detectors generating entrance flows for the network to be simulated. The entrance flows are easily determined by making a special call to an API function and reading a special optional field which was encoded with a list of detectors that determine the flow for that section (see figure 4-4).

4.1.4 Automatically generate the necessary simulation states using the selected network and the traffic volumes.

For each simulation state, detector information is provided from the traffic volume database. The data may need to be adjusted to reflect vehicles per hour flows rather than sampling period flows. The algorithm described in section 3 describes how the simulation states are generated. For the user, generation of the simulation states is completely automatic and should not have to worry about any details.

4.2 *Specification of data requirements for simulation*

In order to do an automated simulation the following components are needed:

- Geometry network selected and saved
- Traffic volume database
- User selection network, date, and time periods to simulate
- Control plan for the selected geometry
- Simulation states generated by the AST tool

4.2.1 Geometry selected and saved

Simulation of a particular geometry can be done quickly and easily by opening a previously stored network with the traffic network editor, selecting an area (subnetwork) for simulation, and then saving this selected area as a new network. If the entire network is to be used then no subnetwork selection is necessary—just use the entire network.

4.2.2 Traffic volume database

The traffic volume database for the time period that simulation is to be done for must be available. Usually Mn/DOT can provide this information. To use the data it must be imported into an ACCESS (or other ODBC compliant database) with the following columns as described in Appendix D. The next step is to register the database with the ODBC32 Administration program in Windows NT/95. The name of the database should be LOOPDATA. This name can be changed later, but the current program will look for a database named LOOPDATA.

Once the database is registered then it becomes available for any ODBC compliant program to retrieve and store data in. That is, any of the simulation tools that we designed can make use of it.

4.2.3 User selection network, date, and time periods to simulate

The user of the AST program can select any existing network (previously saved) for the geometry, and the date, start, and end times for detector data. Any detector data that cannot be found is defaulted to some preset value and a message is written to the user's screen stating that no detector data was found. In our experience, detector data is not always available for all the detectors in a particular network so substituting some data helps to complete the simulation. In the future, this addition of missing data can be made more intelligent, but a quick fix is just to substitute in a fixed value.

4.2.4 Control plan for the selected geometry

The microsimulator that is being used must have a control plan to control the rate at which vehicles enter the freeway. Otherwise, any vehicle on a ramp enters the freeway at the average speed without any waiting. This effect is not desirable and therefore a control plan must be entered.

4.2.5 Simulation states generated by the AST tool

The simulation states (time slices) for each time period for each section are generated by the AST tool with the information gathered from the user, the geometry and the detector values. Each state generated is written to a file and contains information on volumes for the entrance sections. For example, if the traffic engineer wants to simulate the traffic on the network from 0600 to 0900 with 5-minute data then there will be 12×3 or 36 state files generated for this simulation.

In our simulation, all the simulation state files are saved in a folder (directory) called a results container. The results container itself can have any name, but it makes sense to choose a meaningful one. Later when a simulation network is being loaded a particular results container will need to be specified in order to run the simulation.

5 Framework for Phase III work

In this phase of the project, we automated the geometry entry and generation of simulation states. In the next phase, we will automate the control plans and the results output by the simulator.

- Automate control plans
- One time geometry entry: including all ramps, detectors, etc.
- Automatic selection of detector data from a traffic database that is ODBC compliant
- Automatic generation of simulation states (more fully integrated)
- Generic/Integrated geometry for both a macro/micro simulator
- Real-time access to Mn/DOT detector data for simulation

5.1 Automation of control plans

Currently control plans are essentially fixed in this version of the simulator. That is, once a network is opened and a control plan is selected then no other control plan can be opened for this network. Our plans call for automating control plans so that multiple control plans can be used during any one simulation. In addition, we would implement Mn/DOTs control plan so that it could be used in the field.

Automated control plans will allow the simulator to dynamically modify the ramp control for each simulation state. This process will allow the traffic engineer to build algorithms based on current traffic flow to control ramp meters, not on metering based on

historical data or specific time periods during the day. The development of automated control plans will be an important step on the way to real-time traffic management for freeways.

5.2 Automation of results presentation

Once the simulation is completed, it would be good to automatically see the results of the simulation visually. Not just simple X-Y graphics, but some kind of visualization that makes traffic congestion easily stand out. Currently, the simulation data can be saved in spreadsheet style files that can later be used by other programs to generate graphs and perform statistical analyses. None of these graphic views or analyses are integrated with the simulator. Our goal will be to integrate these features within one graphical user interface (GUI) so that a traffic engineer could run a simulation and immediately view the results with just a few menu selections. Some research will need to be done to define exactly what type of visualization would be helpful to traffic engineers.

5.3 Integrating another simulator with the GDC

Integrating a macroscopic simulator so that it can make use of the GDC. Currently only a microscopic simulator works with the GDC. Maintaining only one GDC (geometry) will eliminate the cost of generating a special geometry for a macro simulator. In addition, more effort can be put into keeping the GDC up to date allowing more researchers to actually work on traffic simulation and analysis not geometry or data entry.

The ITS Laboratory at the University of Minnesota is continuing to develop the macro simulator: KRONOS. Since this development is being done in the lab, we have access to the developers and the source code, so that GDC geometry can be augmented

with additional information to generate macro simulator geometries from the macro geometry currently in the GDC.

5.4 *Real-time access to Mn/DOT detector data for simulation*

Finally, real-time access to Mn/DOT detector data from their data feed for real-time simulation. In another project (Activation of the I394 Lab), a prototype was built that received real-time data from Mn/DOT and stored it in a file system.

6 Conclusions and Recommendations

6.1 *Problems that we encountered*

6.1.1 Problems with the GDC

- Debugging the system
- Determining which functions to test first
- Finding internal bugs
- Defining a test suite to determine if things are done correctly
- Real-time and deleting an object on the screen will permanently delete it in the database
- Database cannot translate from relational to ascii.

6.1.1.1 Debugging the system

The system that we designed was difficult to debug because we did not foresee the system as large and as complex as it was. We had multiple layers of software to develop and understand. In figure 6-1, the API has at least four levels of software. All the names in figure with an asterisk (*) were new software that had to be implemented. At the TDR level there were more than 105 new functions to design and implement. The TDG functions all had to be modified to make the correct calls to the TDR functions. In addition, the TDI functions on the right side of the figure were not available for use by the TDR function on the left side. TDI functions provide the capability to create and modify internal API objects. They are utility functions.

Since these TDI utility functions did not exist for many TDR functions that we designed, retrieving objects from the GDC was much more difficult. We had to write some of our own translation functions, and debug them without knowing very much about the internal structure of objects in the API. In addition, at the lowest level ODBC was also a problem to work with because the special 'handles' required to use it were difficult to add to the API. In some cases, the macro definitions of the API conflicted with the macro definitions of Microsoft windows. This conflict resulted in errors during compilation.

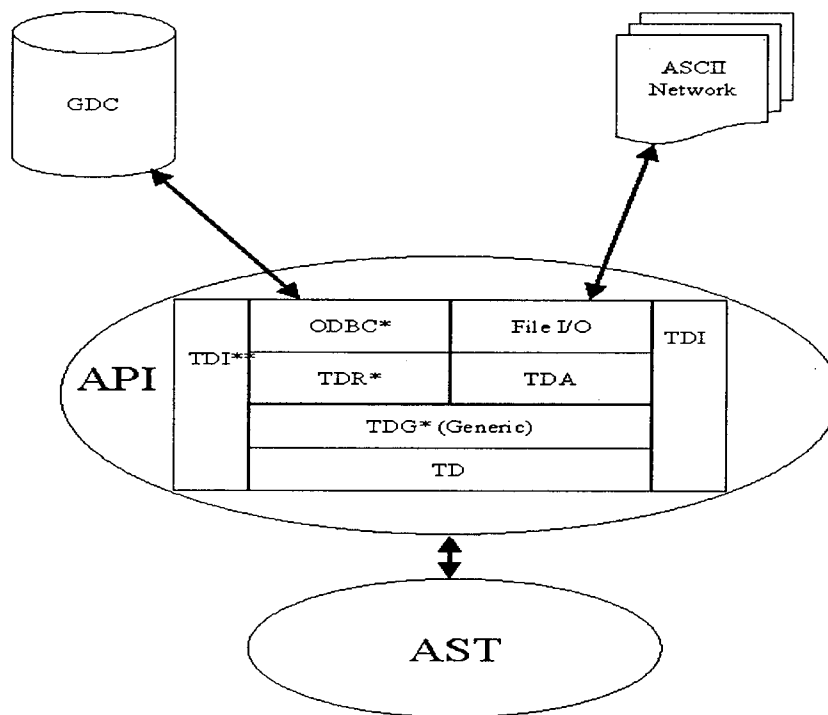


Figure 6-1: Software architecture from the AST to the GDC/ASCII Network

6.1.1.2 Determining which functions to test first

Since there were so many functions to design and test, the most logical step was to start with the network object and functions. Unfortunately, this network object while being one of the more important was not so simple and had many parts to implement before being able to see any results. Of course, nothing else would work without the network object first working. Most of this was done by trial and error because we had no knowledge of the internal working of this environment. In fact, a variable in the ODBC database was named 'level,' which conflicted with some function calls. This was a very difficult error to find, but easy to correct. The name was changed from 'level' to 'levelb.'

Once the network object was finished, the block object was next. Unfortunately, the same problem was encountered here with the name 'level' as in the network object. Basically, there are conflicts in names between what ODBC will accept in calls to the database.

6.1.1.3 Finding internal bugs

In addition to general debugging problems, the software that we were using had additional internal bugs related to the modifications made to call the new TDR functions. These types of errors required much contact with the developers of the simulation suite in order solve these problems. In some cases, solutions were not found.

6.1.1.4 Defining a test suite to determine if things are done correctly

Since the system we were developing was so complex and so many changes were made to the software, it was difficult to come up with a simple plan to test the pieces of

the system we were building. Initially, we thought that we could do end to end testing. That is, use the traffic network editor to make a drawing, which is saved as a database, and also as an ASCII network.

6.1.1.5 Real-time and deleting an object on the screen will permanently delete it in the database

An unintended side effect of the implementation of the GDC was the dynamic nature of an object delete. For example, if an object was open in the GDC and being viewed with the network traffic editor, then any change to that object would immediately be a directed change to the GDC. Since the GDC was changed in this operation even closing the network traffic editor without saving it will not preserve or bring back the original object. This could be a significant problem for any user of the system. With the current design this problem cannot be easily fixed. This is also inconsistent with the ASCII network implementation of the system. A user of the system should be able to perform the same operations with the same results independent of the whether the system is to be saved in a GDC or an ASCII network. This feature is a design fault and will be fixed in the next phase of the project.

6.1.1.6 Implementation cannot translate from GCD to ASCII network.

The design of the GDC system would allow automatic translation from GDC to ASCII network and vice versa. Unfortunately, the implementation of the GDC could only translate from GDC to ASCII network, not ASCII network to GDC. Since the software is very complex, it was not possible to completely test the design before

implementation. Consequently, there were some internal errors and mismatches that should have been caught earlier, but were not. Part of this problem is related to the number of layers that had to be modified in the software. Refer to figure 6-1 to see the layers of software that were modified.

6.1.2 Problems with the Automated Simulation Tool

Although we were successful in building the AST, we did encounter problems along the way. The following is a list of the major problems that we encountered in designing the tool.

- Many detectors have missing data or do not work
- Sometimes the system fails to find a detector that exists
- Not all flows were computed accurately if turnings exist

6.1.2.1 Many detectors have missing data or do not work

In running the AST to generate the system flows, many times the detector information was missing or invalid. In order to correct for potential missing detectors, we automatically entered a default flow for any missing or invalid detector. Substituting a default value was a good idea, but a default value for a detector cannot be generalized across the whole network. The solution is to get complete and reliable data from the detectors or systems that translate detector information. This is not a job for the AST to complete.

6.1.2.2 Sometimes the system fails to find a detector that exists

In several of our test runs of the AST, some detectors could not be found in the database even though they existed. This was not a programming error on our part; it was an error in the software connection to the traffic volume database. The underlying ODBC drivers had an error that caused some calls to the traffic volume database to not find a detector. This problem was corrected with new drivers for the ACCESS database.

6.1.2.3 Not all flows were computed accurately if turnings exist

When we compute the flows in a network, our algorithm specifies which detectors to select from the traffic volume database and where to put the flows. Unfortunately if a vehicle has choice of a turning in a section then the simulator assumes that turns are of equal probability. In reality, this assumption may not be true. It may be that given a choice of going left or right that a driver more often will choose left. In this case, our flows would not be calculated correctly. A solution to this problem is to compute the flows from the detectors in the left and right turn areas and then adjust the turning probabilities to reflect the actual flows.

6.2 *Things that we learned*

- Adding a database capability to an application that is not currently using a database can be a very complex project.—even more so if the database must be integrated internally into the application. For example, just replacing the input

files with database calls is much simpler than modifying the application to use the database as a real-time updating tool.

- Problems in the HW/SW interface may not be apparent until implementation.
- Even though training costs \$\$, time lost trying to figure out software can be even more costly in \$\$ than a training class. Training on externally purchased software should be a must.
- Coordination on projects where details are not completely worked out can be a difficult problem to solve. As many details as possible must be nailed down before the project starts.
- Designing a new software system requires thorough planning and diagrams with all the interfaces nailed down before work is started.
- Test plans should be done at design time so that components can be tested
- Choosing ODBC protocol will pay off. Many other software products now have this protocol available.
- GDC implementation does not completely work. Some functions work, but a redesign of the connection to the database needs to be done

6.3 *What we would do different, if we had to do it again*

Do a more thorough design of the system. Define all the interface components, all the functions that need to be designed, all the levels of abstraction that must be considered, and ensure that the system design is consistent and workable. In our case, we did not do a complete detailed system design. We thought we did, but it was not detailed enough. We did not know what the interfaces would really look like; we did not have a

complete detailed paper design to follow. What we missed is that we did not have complete knowledge of the geometry software, we did not know ODBC, we did not know the process for building DLLs in the visual C++ environment, we did not know ACCESS, and the geometry translation from a set of ASCII files to a relational database was not a direct simple translation. All these factors combined together made this task overwhelming. In addition, the database design had a few errors that were difficult to find because of lack of knowledge of SQL and ODBC.

6.4 Major accomplishments

- GDC was designed. On the top the overall GDC looked like a good design, but implementation was not as easy. I think that too many things were changed to make a successful project without checkpoints to ensure that the software would eventually work.
- AST was designed and Implemented
- AST can generate simulation states with flow data from a selected network
- Some portions of the GDC are working, but they have no effect on whether the AST is working or not.
- I35W geometry with all ramps, meterings, and detectors was entered (it took approximately 3 hours/mile of freeway to enter the data).
- Currently an ITS lab user can take any contiguous portion of I35W and do a simulation with either current TMC data (relatively new data, not real-time), and in less than 30 minutes. This is a major step forward in being able to simulate so quickly.

6.5 Conclusions

The completion of the AST essentially completes most of the objectives of this project. The other objectives included partial entry of the Twin Cities Freeway network, and the design and implementation of the GDC. Although the GDC was designed and partly implemented, there were many problems both in the design and implementation of a complete GDC. Portions of the GDC worked, but the complete implementation with all 100+ functions was not completed. Fortunately, a redesign and a new implementation of the GDC have overcome many of these problems. The process of design and implementation of this project and all the problems that we had gave us great insight in how to improve our methodology for completion of the next project phase.

A major result of this project is that any traffic engineer can now select a network for simulation, determine a time period for simulation, and run the simulation without any manual entry of data or re-entry of geometry. This process alone reduces the amount of time that it takes to prepare data for a simulation run, and it also minimizes errors in data entry. Of course, the network must already be in the GDC (either in ASCII or database format), and the detector information must be available.



Appendix A

Database Specifications

Appendix A : Database Specifications

Introduction to Appendix A	3
Table: BackgroundLayers	3
Table: Backgrounds	4
Table: BlockPoints	5
Table: Blocks	6
Table: CentroidNodes	6
Table: Centroids	8
Table: CentroidSections	9
Table: ContourPoints	10
Table: ControllerDetectors	11
Table: ControllerMeterings	13
Table: ControllerNodes	14
Table: Controllers	16
Table: ControllerVMSs	17
Table: Detectors	18
Table: GlobMessages	19
Table: Laterals	20
Table: Meterings	21
Table: Network	22
Table: Nodes	24
Table: Rights	25
Table: RoguiRoutes	27
Table: RoguiRouteSections	28
Table: Roguis	30
Table: RoguiVMSs	31
Table: Sections	32
Table: SelPermissions	35
Table: Stages	36
Table: Texts	37
Table: Turnings	38
Table: VehClasses	40
Table: VMSs	41
Relationships: All	42

Introduction to Appendix A

This appendix contains the full description of the database tables that were defined in the ACCESS database. There is enough information here for each table to recreate the database. Each database table explanation is partitioned into three parts: columns, relationships, and table indexes.

Columns describe the type and size of data contained in a table. Each column that is defined is a data field with a name (e.g. fileName), a certain type (e.g. Text) and a particular size in bytes (e.g. text is 100 bytes long—characters).

Relationships specify how the tables in the database relate to each other. They provide an efficient means to retrieve and to store information in a database. For example, if a relationship exists between an element of blocks and records in blockpoints, and given that a block is composed of blockpoints, then deleting a block also deletes ALL the blockpoints associated with it automatically. So, just one delete command can delete many different records in RELATED tables. In addition, UPDATING a record will automatically update all records associated with it. Relationships also define the type of relationship between tables such as one-to-one, one-to-many, and many-to-many.

Table indexes provide efficient methods to search databases for information and provide keyword searching.

Table: BackgroundLayers

Columns

Name	Type	Size
fileName	Text	100
idLayer	Text	20

Relationships

BackgroundsBackgroundLayers

Backgrounds	BackgroundLayers
fileName	1 ∞ fileName

Attributes: Enforced, Cascade Updates, Cascade Deletes
 Attributes: One-To-Many

Table Indexes

Name	Number of Fields
BackgroundsBackgroundLayers	1
Clustered:	False
Distinct Count:	0

Foreign:	True
Ignore Nulls:	False
Name:	BackgroundsBackgroundLayers
Primary:	False
Required:	False
Unique:	False
Fields:	fileName, Ascending
fileName	1
Clustered:	False
Distinct Count:	0
Foreign:	False
Ignore Nulls:	False
Name:	fileName
Primary:	False
Required:	False
Unique:	False
Fields:	fileName, Ascending
idBackgroundLayer	2
Clustered:	False
Distinct Count:	0
Foreign:	False
Ignore Nulls:	False
Name:	idBackgroundLayer
Primary:	True
Required:	True
Unique:	True
Fields:	fileName, Ascending idLayer, Ascending

Table: Backgrounds

Columns

Name	Type	Size
fileName	Text	100
formatType	Number (Integer)	2
scale	Number (Single)	4
drawingDetail	Number (Integer)	2
setCoordinateOrigin	Yes/No	1
coordinateOriginX	Number (Double)	8
coordinateOriginY	Number (Double)	8
angle	Number (Single)	4
isRestrictLayers	Yes/No	1

Relationships

BackgroundsBackgroundLayers

Backgrounds	BackgroundLayers
fileName	1 ∞ fileName

Attributes:	Enforced, Cascade Updates, Cascade Deletes
Attributes:	One-To-Many

Table Indexes

Name	Number of Fields
------	------------------

idBackground	1
Clustering:	False
Distinct Count:	0
Foreign:	False
Ignore Nulls:	False
Name:	idBackground
Primary:	True
Required:	True
Unique:	True
Fields:	fileName, Ascending

Table: BlockPoints

Columns

Name	Type	Size
idBlock	Number (Long)	4
idPoint	Number (Integer)	2
x	Number (Double)	8
y	Number (Double)	8

Relationships

BlocksBlockPoints

Blocks	BlockPoints
idBlock	1 ∞ idBlock
Attributes:	Enforced, Cascade Updates, Cascade Deletes
Attributes:	One-To-Many

Table Indexes

Name	Number of Fields
BlocksBlockPoints	1
Clustering:	False
Distinct Count:	0
Foreign:	True
Ignore Nulls:	False
Name:	BlocksBlockPoints
Primary:	False
Required:	False
Unique:	False
Fields:	idBlock, Ascending
idBlock	1
Clustering:	False
Distinct Count:	0
Foreign:	False
Ignore Nulls:	False
Name:	idBlock
Primary:	False
Required:	False
Unique:	False
Fields:	idBlock, Ascending
idBlockPoint	2
Clustering:	False

Distinct Count:	0
Foreign:	False
Ignore Nulls:	False
Name:	idBlockPoint
Primary:	True
Required:	True
Unique:	True
Fields:	idBlock, Ascending idPoint, Ascending

Table: Blocks

Columns

Name	Type	Size
idBlock	Number (Long)	4
level	Number (Integer)	2
colorRed	Number (Long)	4
colorGreen	Number (Long)	4
colorBlue	Number (Long)	4

Relationships

BlocksBlockPoints

Blocks	BlockPoints
idBlock	1 ∞ idBlock
Attributes:	Enforced, Cascade Updates, Cascade Deletes
Attributes:	One-To-Many

Table Indexes

Name	Number of Fields
idBlock	1
Clustered:	False
Distinct Count:	0
Foreign:	False
Ignore Nulls:	False
Name:	idBlock
Primary:	True
Required:	True
Unique:	True
Fields:	idBlock, Ascending

Table: CentroidNodes

Columns

Name	Type	Size
idCentroid	Number (Long)	4
type	Number (Integer)	2
idNode	Number (Long)	4
percentage	Number (Single)	4

Relationships

CentroidsCentroidNodes

Centroids	CentroidNodes
idCentroid	1 ∞ idCentroid
Attributes:	Enforced, Cascade Updates, Cascade Deletes
Attributes:	One-To-Many

NodesCentroidNodes

Nodes	CentroidNodes
idNode	1 ∞ idNode
Attributes:	Enforced, Cascade Updates, Cascade Deletes
Attributes:	One-To-Many

Table Indexes

Name	Number of Fields
CentroidsCentroidNodes	1
Clustered:	False
Distinct Count:	0
Foreign:	True
Ignore Nulls:	False
Name:	CentroidsCentroidNodes
Primary:	False
Required:	False
Unique:	False
Fields:	idCentroid, Ascending
idCentroid	1
Clustered:	False
Distinct Count:	0
Foreign:	False
Ignore Nulls:	False
Name:	idCentroid
Primary:	False
Required:	False
Unique:	False
Fields:	idCentroid, Ascending
idCentroidNode	3
Clustered:	False
Distinct Count:	0
Foreign:	False
Ignore Nulls:	False
Name:	idCentroidNode
Primary:	True
Required:	True
Unique:	True
Fields:	idCentroid, Ascending type, Ascending idNode, Ascending
idNode	1

Clustering:	False
Distinct Count:	0
Foreign:	False
Ignore Nulls:	False
Name:	idNode
Primary:	False
Required:	False
Unique:	False
Fields:	idNode, Ascending

NodesCentroidNodes 1

Clustering:	False
Distinct Count:	0
Foreign:	True
Ignore Nulls:	False
Name:	NodesCentroidNodes
Primary:	False
Required:	False
Unique:	False
Fields:	idNode, Ascending

Table: Centroids

Columns

Name	Type	Size
idCentroid	Number (Long)	4
name	Text	20
level	Number (Integer)	2
x	Number (Double)	8
y	Number (Double)	8
autoPercentages	Yes/No	1

Relationships

CentroidsCentroidNodes

Centroids	CentroidNodes
idCentroid	1 ∞ idCentroid
Attributes:	Enforced, Cascade Updates, Cascade Deletes
Attributes:	One-To-Many

CentroidsCentroidSections

Centroids	CentroidSections
idCentroid	1 ∞ idCentroid
Attributes:	Enforced, Cascade Updates, Cascade Deletes
Attributes:	One-To-Many

Table Indexes

Name	Number of Fields
idCentroid	1

Clustered: False
 Distinct Count: 1
 Foreign: False
 Ignore Nulls: False
 Name: idCentroid
 Primary: True
 Required: True
 Unique: True
 Fields: idCentroid, Ascending

Table: CentroidSections

Columns

Name	Type	Size
idCentroid	Number (Long)	4
type	Number (Integer)	2
idSection	Number (Long)	4
position	Number (Single)	4
percentage	Number (Single)	4

Relationships

CentroidsCentroidSections

Centroids	CentroidSections
idCentroid	1 ∞ idCentroid

Attributes: Enforced, Cascade Updates, Cascade Deletes
 Attributes: One-To-Many

SectionsCentroidSections

Sections	CentroidSections
idSection	1 ∞ idSection

Attributes: Enforced, Cascade Updates, Cascade Deletes
 Attributes: One-To-Many

Table Indexes

Name	Number of Fields
CentroidsCentroidSections	1
Clustered:	False
Distinct Count:	0
Foreign:	True
Ignore Nulls:	False
Name:	CentroidsCentroidSections
Primary:	False
Required:	False
Unique:	False
Fields:	idCentroid, Ascending
idCentroid	1
Clustered:	False

Distinct Count:	0
Foreign:	False
Ignore Nulls:	False
Name:	idCentroid
Primary:	False
Required:	False
Unique:	False
Fields:	idCentroid, Ascending
idCentroidSection	4
Clustering:	False
Distinct Count:	0
Foreign:	False
Ignore Nulls:	False
Name:	idCentroidSection
Primary:	True
Required:	True
Unique:	True
Fields:	idCentroid, Ascending type, Ascending idSection, Ascending position, Ascending
idSection	1
Clustering:	False
Distinct Count:	0
Foreign:	False
Ignore Nulls:	False
Name:	idSection
Primary:	False
Required:	False
Unique:	False
Fields:	idSection, Ascending
SectionsCentroidSections	1
Clustering:	False
Distinct Count:	0
Foreign:	True
Ignore Nulls:	False
Name:	SectionsCentroidSections
Primary:	False
Required:	False
Unique:	False
Fields:	idSection, Ascending

Table: ContourPoints

Columns

Name	Type	Size
idSection	Number (Long)	4
side	Number (Integer)	2
idPoint	Number (Integer)	2
x	Number (Double)	8
y	Number (Double)	8

Relationships

SectionsContourPoints

Sections	ContourPoints
idSection	1 ∞ idSection

Attributes: Enforced, Cascade Updates, Cascade Deletes
 Attributes: One-To-Many

Table Indexes

Name	Number of Fields
idContourPoint	3
Clustered:	False
Distinct Count:	0
Foreign:	False
Ignore Nulls:	False
Name:	idContourPoint
Primary:	True
Required:	True
Unique:	True
Fields:	idSection, Ascending side, Ascending idPoint, Ascending
idSection	1
Clustered:	False
Distinct Count:	0
Foreign:	False
Ignore Nulls:	False
Name:	idSection
Primary:	False
Required:	False
Unique:	False
Fields:	idSection, Ascending
SectionsContourPoints	1
Clustered:	False
Distinct Count:	0
Foreign:	True
Ignore Nulls:	False
Name:	SectionsContourPoints
Primary:	False
Required:	False
Unique:	False
Fields:	idSection, Ascending

Table: ControllerDetectors

Columns

Name	Type	Size
idController	Text	20
idDetector	Text	20
idConnection	Number (Integer)	2
isActive	Yes/No	1

Relationships

ControllersControllerDetectors	Controllers	ControllerDetectors
	1	1
	idController	idController

Attributes: Unique, Enforced, Cascade Updates, Cascade Deletes
 Attributes: One-To-One

DetectorsControllerDetectors

Detectors	ControllerDetectors
idDetector	1 ∞ idDetector

Attributes: Enforced, Cascade Updates, Cascade Deletes
 Attributes: One-To-Many

Table Indexes

Name	Number of Fields
ControllersControllerDetectors	1
Clustered:	False
Distinct Count:	0
Foreign:	True
Ignore Nulls:	False
Name:	ControllersControllerDetectors
Primary:	False
Required:	False
Unique:	True
Fields:	idController, Ascending
DetectorsControllerDetectors	1
Clustered:	False
Distinct Count:	0
Foreign:	True
Ignore Nulls:	False
Name:	DetectorsControllerDetectors
Primary:	False
Required:	False
Unique:	False
Fields:	idDetector, Ascending
idController	1
Clustered:	False
Distinct Count:	0
Foreign:	False
Ignore Nulls:	False
Name:	idController
Primary:	False
Required:	False
Unique:	True
Fields:	idController, Ascending
idControllerDetector	2
Clustered:	False
Distinct Count:	0
Foreign:	False
Ignore Nulls:	False
Name:	idControllerDetector
Primary:	True
Required:	True
Unique:	True
Fields:	idController, Ascending idDetector, Ascending
idDetector	1
Clustered:	False
Distinct Count:	0
Foreign:	False
Ignore Nulls:	False
Name:	idDetector
Primary:	False
Required:	False
Unique:	False

Fields: idDetector, Ascending

Table: ControllerMeterings

Columns

Name	Type	Size
idController	Text	20
idMetering	Text	20
idConnection	Number (Integer)	2
isActive	Yes/No	1

Relationships

ControllersControllerMeterings

Controllers	ControllerMeterings
idController	idController
Attributes:	Unique, Enforced, Cascade Updates, Cascade Deletes
Attributes:	One-To-One

MeteringsControllerMeterings

Meterings	ControllerMeterings
idMetering	idMetering
Attributes:	Enforced, Cascade Updates, Cascade Deletes
Attributes:	One-To-Many

Table Indexes

Name	Number of Fields
ControllersControllerMeterings	1
Clustered:	False
Distinct Count:	0
Foreign:	True
Ignore Nulls:	False
Name:	ControllersControllerMeterings
Primary:	False
Required:	False
Unique:	True
Fields:	idController, Ascending
idController	1
Clustered:	False
Distinct Count:	0
Foreign:	False
Ignore Nulls:	False
Name:	idController
Primary:	False
Required:	False
Unique:	True
Fields:	idController, Ascending

idControllerMetering	2
Clustered:	False
Distinct Count:	0
Foreign:	False
Ignore Nulls:	False
Name:	idControllerMetering
Primary:	True
Required:	True
Unique:	True
Fields:	idController, Ascending idMetering, Ascending
idMetering	1
Clustered:	False
Distinct Count:	0
Foreign:	False
Ignore Nulls:	False
Name:	idMetering
Primary:	False
Required:	False
Unique:	False
Fields:	idMetering, Ascending
MeteringsControllerMeterings	1
Clustered:	False
Distinct Count:	0
Foreign:	True
Ignore Nulls:	False
Name:	MeteringsControllerMeterings
Primary:	False
Required:	False
Unique:	False
Fields:	idMetering, Ascending

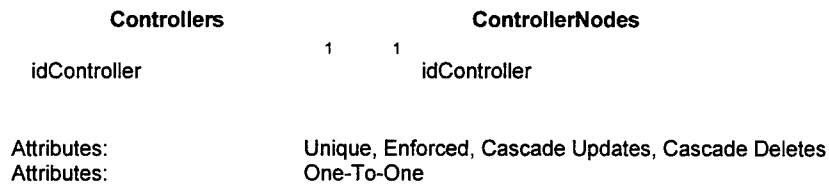
Table: ControllerNodes

Columns

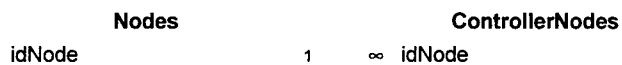
Name	Type	Size
idController	Text	20
idNode	Number (Long)	4
idConnection	Number (Integer)	2
isActive	Yes/No	1

Relationships

ControllersControllerNodes



NodesControllerNodes



Attributes: Enforced, Cascade Updates, Cascade Deletes
 Attributes: One-To-Many

Table Indexes

Name	Number of Fields
ControllersControllerNodes	1
Clustered:	False
Distinct Count:	0
Foreign:	True
Ignore Nulls:	False
Name:	ControllersControllerNodes
Primary:	False
Required:	False
Unique:	True
Fields:	idController, Ascending
idController	1
Clustered:	False
Distinct Count:	0
Foreign:	False
Ignore Nulls:	False
Name:	idController
Primary:	False
Required:	False
Unique:	True
Fields:	idController, Ascending
idControllerNode	2
Clustered:	False
Distinct Count:	0
Foreign:	False
Ignore Nulls:	False
Name:	idControllerNode
Primary:	True
Required:	True
Unique:	True
Fields:	idController, Ascending idNode, Ascending
idNode	1
Clustered:	False
Distinct Count:	0
Foreign:	False
Ignore Nulls:	False
Name:	idNode
Primary:	False
Required:	False
Unique:	False
Fields:	idNode, Ascending
NodesControllerNodes	1
Clustered:	False
Distinct Count:	0
Foreign:	True
Ignore Nulls:	False
Name:	NodesControllerNodes
Primary:	False
Required:	False
Unique:	False
Fields:	idNode, Ascending

Table: Controllers

Columns

Name	Type	Size
idController	Text	20
name	Text	20
level	Number (Integer)	2
x	Number (Double)	8
y	Number (Double)	8

Relationships

ControllersControllerDetectors

Controllers	ControllerDetectors
idController	idController
1	1
Attributes:	Unique, Enforced, Cascade Updates, Cascade Deletes
Attributes:	One-To-One

ControllersControllerMeterings

Controllers	ControllerMeterings
idController	idController
1	1
Attributes:	Unique, Enforced, Cascade Updates, Cascade Deletes
Attributes:	One-To-One

ControllersControllerNodes

Controllers	ControllerNodes
idController	idController
1	1
Attributes:	Unique, Enforced, Cascade Updates, Cascade Deletes
Attributes:	One-To-One

ControllersControllerVMSs

Controllers	ControllerVMSs
idController	idController
1	1
Attributes:	Unique, Enforced, Cascade Updates, Cascade Deletes
Attributes:	One-To-One

Table Indexes

Name	Number of Fields
idController	1
Clustered:	False

Distinct Count: 0
 Foreign: False
 Ignore Nulls: False
 Name: idController
 Primary: True
 Required: True
 Unique: True
 Fields: idController, Ascending

Table: ControllerVMSs

Columns

Name	Type	Size
idController	Text	20
idVMS	Text	20
idConnection	Number (Integer)	2
isActive	Yes/No	1

Relationships

ControllersControllerVMSs

Controllers	ControllerVMSs
idController	idController
Attributes:	Unique, Enforced, Cascade Updates, Cascade Deletes
Attributes:	One-To-One

VMSsControllerVMSs

VMSs	ControllerVMSs
idVMS	idVMS
Attributes:	Enforced, Cascade Updates, Cascade Deletes
Attributes:	One-To-Many

Table Indexes

Name	Number of Fields
ControllersControllerVMSs	1
Clustering:	False
Distinct Count:	0
Foreign:	True
Ignore Nulls:	False
Name:	ControllersControllerVMSs
Primary:	False
Required:	False
Unique:	True
Fields:	idController, Ascending
idController	1
Clustering:	False
Distinct Count:	0

	Foreign:	False
	Ignore Nulls:	False
	Name:	idController
	Primary:	False
	Required:	False
	Unique:	True
	Fields:	idController, Ascending
idVMS		1
	Clustered:	False
	Distinct Count:	0
	Foreign:	False
	Ignore Nulls:	False
	Name:	idVMS
	Primary:	False
	Required:	False
	Unique:	False
	Fields:	idVMS, Ascending
PrimaryKey		2
	Clustered:	False
	Distinct Count:	0
	Foreign:	False
	Ignore Nulls:	False
	Name:	PrimaryKey
	Primary:	True
	Required:	True
	Unique:	True
	Fields:	idController, Ascending idVMS, Ascending
VMSsControllerVMSs		1
	Clustered:	False
	Distinct Count:	0
	Foreign:	True
	Ignore Nulls:	False
	Name:	VMSsControllerVMSs
	Primary:	False
	Required:	False
	Unique:	False
	Fields:	idVMS, Ascending

Table: Detectors

Columns

Name	Type	Size
idDetector	Text	20
idSection	Number (Long)	4
type	Number (Integer)	2
measuringCapability	Number (Long)	4
idLaneFirst	Number (Integer)	2
idLaneLast	Number (Integer)	2
distance	Number (Single)	4
length	Number (Single)	4

Relationships

DetectorsControllerDetectors

Detectors		ControllerDetectors
idDetector	1	∞ idDetector

Attributes: Enforced, Cascade Updates, Cascade Deletes
 Attributes: One-To-Many

SectionsDetectors

Sections	Detectors
idSection	1 ∞ idSection

Attributes: Enforced, Cascade Updates, Cascade Deletes
 Attributes: One-To-Many

Table Indexes

Name	Number of Fields
idDetector	1
Clustered:	False
Distinct Count:	0
Foreign:	False
Ignore Nulls:	False
Name:	idDetector
Primary:	True
Required:	True
Unique:	True
Fields:	idDetector, Ascending
idSection	1
Clustered:	False
Distinct Count:	0
Foreign:	False
Ignore Nulls:	False
Name:	idSection
Primary:	False
Required:	False
Unique:	False
Fields:	idSection, Ascending
SectionsDetectors	1
Clustered:	False
Distinct Count:	0
Foreign:	True
Ignore Nulls:	False
Name:	SectionsDetectors
Primary:	False
Required:	False
Unique:	False
Fields:	idSection, Ascending

Table: GlobMessages

Columns

Name	Type	Size
idGlobMessage	Number (Long)	4
type	Number (Integer)	2
priority	Number (Integer)	2
description	Text	20

Table Indexes

Name	Number of Fields
idGlobMessage	1
Clustered:	False
Distinct Count:	0
Foreign:	False
Ignore Nulls:	False
Name:	idGlobMessage
Primary:	True
Required:	True
Unique:	True
Fields:	idGlobMessage, Ascending

Table: Laterals

Columns

Name	Type	Size
idSection	Number (Long)	4
type	Number (Integer)	2
width	Number (Single)	4
length	Number (Single)	4

Relationships

SectionsLaterals

Sections	Laterals
idSection	1 ∞ idSection
Attributes:	Enforced, Cascade Updates, Cascade Deletes
Attributes:	One-To-Many

Table Indexes

Name	Number of Fields
idLateral	2
Clustered:	False
Distinct Count:	0
Foreign:	False
Ignore Nulls:	False
Name:	idLateral
Primary:	True
Required:	True
Unique:	True
Fields:	idSection, Ascending type, Ascending
idSection	1
Clustered:	False
Distinct Count:	0
Foreign:	False
Ignore Nulls:	False
Name:	idSection
Primary:	False
Required:	False
Unique:	False

Fields:	idSection, Ascending
SectionsLaterals	1
Clustered:	False
Distinct Count:	0
Foreign:	True
Ignore Nulls:	False
Name:	SectionsLaterals
Primary:	False
Required:	False
Unique:	False
Fields:	idSection, Ascending

Table: Meterings

Columns

Name	Type	Size
idMetering	Text	20
idSection	Number (Long)	4
type	Number (Integer)	2
platoonLength	Number (Integer)	2
distance	Number (Single)	4

Relationships

MeteringsControllerMeterings

Meterings	ControllerMeterings
idMetering	1 ∞ idMetering

Attributes:	Enforced, Cascade Updates, Cascade Deletes
Attributes:	One-To-Many

SectionsMeterings

Sections	Meterings
idSection	1 ∞ idSection

Attributes:	Enforced, Cascade Updates, Cascade Deletes
Attributes:	One-To-Many

Table Indexes

Name	Number of Fields
idMetering	1
Clustered:	False
Distinct Count:	0
Foreign:	False
Ignore Nulls:	False
Name:	idMetering
Primary:	True
Required:	True
Unique:	True

	Fields:	idMetering, Ascending
idSection		1
	Clustered:	False
	Distinct Count:	0
	Foreign:	False
	Ignore Nulls:	False
	Name:	idSection
	Primary:	False
	Required:	False
	Unique:	False
	Fields:	idSection, Ascending
SectionsMeterings		1
	Clustered:	False
	Distinct Count:	0
	Foreign:	True
	Ignore Nulls:	False
	Name:	SectionsMeterings
	Primary:	False
	Required:	False
	Unique:	False
	Fields:	idSection, Ascending

Table: Network

Columns

Name	Type	Size
dummyId	Number (Integer)	2
version	Number (Single)	4
isPointerStandar	Yes/No	1
showDistance	Yes/No	1
isGridSet	Yes/No	1
showGrid	Yes/No	1
gridDistance	Number (Single)	4
xMin	Number (Long)	4
yMin	Number (Long)	4
xMax	Number (Long)	4
yMax	Number (Long)	4
xMinDw	Number (Long)	4
yMinDw	Number (Long)	4
xMaxDw	Number (Long)	4
yMaxDw	Number (Long)	4
backgroundColorRed	Number (Long)	4
backgroundColorGreen	Number (Long)	4
backgroundColorBlue	Number (Long)	4
textColorRed	Number (Long)	4
textColorGreen	Number (Long)	4
textColorBlue	Number (Long)	4
sectionBaseColorRed	Number (Long)	4
sectionBaseColorGreen	Number (Long)	4
sectionBaseColorBlue	Number (Long)	4
sectionLinesColorRed	Number (Long)	4
sectionLinesColorGreen	Number (Long)	4
sectionLinesColorBlue	Number (Long)	4
sectionSelPermissionColorRed	Number (Long)	4
sectionSelPermissionColorGreen	Number (Long)	4
sectionSelPermissionColorBlue	Number (Long)	4
sectionSelectedLaneColorRed	Number (Long)	4
sectionSelectedLaneColorGreen	Number (Long)	4
sectionSelectedLaneColorBlue	Number (Long)	4
nodeColorRed	Number (Long)	4
nodeColorGreen	Number (Long)	4

nodeColorBlue	Number (Long)	4
controllerColorRed	Number (Long)	4
controllerColorGreen	Number (Long)	4
controllerColorBlue	Number (Long)	4
centroidColorRed	Number (Long)	4
centroidColorGreen	Number (Long)	4
centroidColorBlue	Number (Long)	4
backgroundLevel	Number (Long)	4
textLevel	Number (Long)	4
blockLevel	Number (Long)	4
sectionLevel	Number (Long)	4
controllerLevel	Number (Long)	4
centroidLevel	Number (Long)	4
defaultLevel	Number (Long)	4
defaultNbLanes	Number (Integer)	2
defaultLaneWidth	Number (Single)	4
defaultRoadType	Number (Integer)	2
defaultArterialMaxspeed	Number (Single)	4
defaultRoadMaxspeed	Number (Single)	4
defaultFreewayMaxspeed	Number (Single)	4
showObjects	Number (Long)	4
isDetectorDisregate	Yes/No	1
isTrafficLeftSide	Yes/No	1
nbbackgrounds	Text	50
file	Text	50
format	Text	50
backscale	Text	50
detail	Text	50
set_coordinate_origin	Text	50
coordinate_origin	Text	50
angle	Text	50
restrict_layers	Text	50
nblayers	Text	50
idlayer	Text	50

Table Indexes

Name	Number of Fields
idlayer	1
Clustered:	False
Distinct Count:	0
Foreign:	False
Ignore Nulls:	False
Name:	idlayer
Primary:	False
Required:	False
Unique:	False
Fields:	idlayer, Ascending
idNetwork	1
Clustered:	False
Distinct Count:	0
Foreign:	False
Ignore Nulls:	False
Name:	idNetwork
Primary:	False
Required:	False
Unique:	False
Fields:	dummyId, Ascending
PrimaryKey	1
Clustered:	False
Distinct Count:	0
Foreign:	False
Ignore Nulls:	False
Name:	PrimaryKey
Primary:	True
Required:	True
Unique:	True
Fields:	dummyId, Ascending

Table: Nodes

Columns

Name	Type	Size
idNode	Number (Long)	4
type	Number (Integer)	2
isYellowBox	Yes/No	1

Relationships

NodesCentroidNodes

Nodes	CentroidNodes
idNode	1 ∞ idNode
Attributes:	Enforced, Cascade Updates, Cascade Deletes
Attributes:	One-To-Many

NodesControllerNodes

Nodes	ControllerNodes
idNode	1 ∞ idNode
Attributes:	Enforced, Cascade Updates, Cascade Deletes
Attributes:	One-To-Many

NodesStages

Nodes	Rights
idNode	1 ∞ idNode
Attributes:	Enforced, Cascade Updates, Cascade Deletes
Attributes:	One-To-Many

NodesStages1

Nodes	Stages
idNode	1 ∞ idNode
Attributes:	Enforced, Cascade Updates, Cascade Deletes
Attributes:	One-To-Many

NodesTurnings

Nodes	Turnings
idNode	1 ∞ idNode

Attributes: Enforced, Cascade Updates, Cascade Deletes
 Attributes: One-To-Many

Table Indexes

Name	Number of Fields
idNode	1
Clustered:	False
Distinct Count:	0
Foreign:	False
Ignore Nulls:	False
Name:	idNode
Primary:	True
Required:	True
Unique:	True
Fields:	idNode, Ascending

Table: Rights

Columns

Name	Type	Size
idNode	Number (Long)	4
idStage	Number (Integer)	2
idSectionOrigin	Number (Long)	4
idSectionDest	Number (Long)	4

Relationships

NodesStages

	Nodes		Rights
idNode	1	∞	idNode
Attributes:	Enforced, Cascade Updates, Cascade Deletes		
Attributes:	One-To-Many		

SectionsStages

	Sections		Rights
idSection	1	∞	idSectionOrigin
Attributes:	Enforced, Cascade Updates, Cascade Deletes		
Attributes:	One-To-Many		

SectionsStages1

	Sections		Rights
idSection	1	∞	idSectionDest
Attributes:	Enforced, Cascade Updates, Cascade Deletes		

Attributes: One-To-Many

StagesRights

	Stages		Rights
idNode		1	∞ idNode
idStage		1	∞ idStage
Attributes:			Enforced, Cascade Updates, Cascade Deletes
Attributes:			One-To-Many

Table Indexes

Name	Number of Fields
idNode	1
Clustered:	False
Distinct Count:	0
Foreign:	False
Ignore Nulls:	False
Name:	idNode
Primary:	False
Required:	False
Unique:	False
Fields:	idNode, Ascending
idSectionDest	1
Clustered:	False
Distinct Count:	0
Foreign:	False
Ignore Nulls:	False
Name:	idSectionDest
Primary:	False
Required:	False
Unique:	False
Fields:	idSectionDest, Ascending
idSectionOrigin	1
Clustered:	False
Distinct Count:	0
Foreign:	False
Ignore Nulls:	False
Name:	idSectionOrigin
Primary:	False
Required:	False
Unique:	False
Fields:	idSectionOrigin, Ascending
idStage	4
Clustered:	False
Distinct Count:	0
Foreign:	False
Ignore Nulls:	False
Name:	idStage
Primary:	True
Required:	True
Unique:	True
Fields:	idNode, Ascending idSectionOrigin, Ascending idSectionDest, Ascending idStage, Ascending
NodesStages	1
Clustered:	False
Distinct Count:	0
Foreign:	True
Ignore Nulls:	False
Name:	NodesStages
Primary:	False

```

Required: False
Unique: False
Fields: idNode, Ascending
SectionsStages 1
  Clustered: False
  Distinct Count: 0
  Foreign: True
  Ignore Nulls: False
  Name: SectionsStages
  Primary: False
  Required: False
  Unique: False
  Fields: idSectionOrigin, Ascending
SectionsStages1 1
  Clustered: False
  Distinct Count: 0
  Foreign: True
  Ignore Nulls: False
  Name: SectionsStages1
  Primary: False
  Required: False
  Unique: False
  Fields: idSectionDest, Ascending
StagesRights 2
  Clustered: False
  Distinct Count: 0
  Foreign: True
  Ignore Nulls: False
  Name: StagesRights
  Primary: False
  Required: False
  Unique: False
  Fields: idNode, Ascending
           idStage, Ascending

```

Table: RoguiRoutes

Columns

Name	Type	Size
idRogui	Text	20
idRoute	Text	20
name	Text	20
type	Number (Integer)	2

Relationships

RoguiRoutesRoguiRouteSections

RoguiRoutes		RoguiRouteSections	
idRogui	1	∞	idRogui
idRoute	1	∞	idRoute

Attributes: Enforced
 Attributes: One-To-Many

RoguisRoguiRoutes

Roguis	RoguiRoutes
idRogui	1 ∞ idRogui
Attributes:	Enforced, Cascade Updates, Cascade Deletes
Attributes:	One-To-Many

Table Indexes

Name	Number of Fields
idRogui	1
Clustered:	False
Distinct Count:	0
Foreign:	False
Ignore Nulls:	False
Name:	idRogui
Primary:	False
Required:	False
Unique:	False
Fields:	idRogui, Ascending
idRoguiRoute	2
Clustered:	False
Distinct Count:	0
Foreign:	False
Ignore Nulls:	False
Name:	idRoguiRoute
Primary:	True
Required:	True
Unique:	True
Fields:	idRogui, Ascending idRoute, Ascending
idRoute	1
Clustered:	False
Distinct Count:	0
Foreign:	False
Ignore Nulls:	False
Name:	idRoute
Primary:	False
Required:	False
Unique:	False
Fields:	idRoute, Ascending
RoguisRoguiRoutes	1
Clustered:	False
Distinct Count:	0
Foreign:	True
Ignore Nulls:	False
Name:	RoguisRoguiRoutes
Primary:	False
Required:	False
Unique:	False
Fields:	idRogui, Ascending

Table: RoguiRouteSections

Columns

Name	Type	Size
------	------	------

idRogui	Text	20
idRoute	Text	20
idSection	Number (Long)	4

Relationships

RoguiRoutesRoguiRouteSections

	RoguiRoutes		RoguiRouteSections
idRogui	1	∞	idRogui
idRoute	1	∞	idRoute
Attributes:			Enforced
Attributes:			One-To-Many

SectionsRoguiRouteSections

	Sections		RoguiRouteSections
idSection	1	∞	idSection
Attributes:			Enforced, Cascade Updates, Cascade Deletes
Attributes:			One-To-Many

Table Indexes

Name	Number of Fields
idRogui	1
Clustered:	False
Distinct Count:	0
Foreign:	False
Ignore Nulls:	False
Name:	idRogui
Primary:	False
Required:	False
Unique:	False
Fields:	idRogui, Ascending
idRoguiRouteSection	3
Clustered:	False
Distinct Count:	0
Foreign:	False
Ignore Nulls:	False
Name:	idRoguiRouteSection
Primary:	True
Required:	True
Unique:	True
Fields:	idRogui, Ascending idRoute, Ascending idSection, Ascending
idRoute	1
Clustered:	False
Distinct Count:	0
Foreign:	False
Ignore Nulls:	False
Name:	idRoute
Primary:	False
Required:	False
Unique:	False
Fields:	idRoute, Ascending
idSection	1
Clustered:	False
Distinct Count:	0

Foreign:	False
Ignore Nulls:	False
Name:	idSection
Primary:	False
Required:	False
Unique:	False
Fields:	idSection, Ascending
RoguiRoutesRoguiRouteSections	2
Clustered:	False
Distinct Count:	0
Foreign:	True
Ignore Nulls:	False
Name:	RoguiRoutesRoguiRouteSections
Primary:	False
Required:	False
Unique:	False
Fields:	idRogui, Ascending idRoute, Ascending
SectionsRoguiRouteSections	1
Clustered:	False
Distinct Count:	0
Foreign:	True
Ignore Nulls:	False
Name:	SectionsRoguiRouteSections
Primary:	False
Required:	False
Unique:	False
Fields:	idSection, Ascending

Table: Roguis

Columns

Name	Type	Size
idRogui	Text	20
name	Text	20
idNodeOrigin	Number (Long)	4
idNodeDest	Number (Long)	4

Relationships

RoguisRoguiRoutes

Roguis	RoguiRoutes
idRogui	idRogui
1	∞
Attributes:	Enforced, Cascade Updates, Cascade Deletes
Attributes:	One-To-Many

RoguisRoguiVMSs

Roguis	RoguiVMSs
idRogui	idRogui
1	∞
Attributes:	Enforced, Cascade Updates, Cascade Deletes

Attributes: One-To-Many

Table Indexes

Name	Number of Fields
idNodeDest	1
Clustered:	False
Distinct Count:	0
Foreign:	False
Ignore Nulls:	False
Name:	idNodeDest
Primary:	False
Required:	False
Unique:	False
Fields:	idNodeDest, Ascending
idNodeOrigin	1
Clustered:	False
Distinct Count:	0
Foreign:	False
Ignore Nulls:	False
Name:	idNodeOrigin
Primary:	False
Required:	False
Unique:	False
Fields:	idNodeOrigin, Ascending
idRogui	1
Clustered:	False
Distinct Count:	0
Foreign:	False
Ignore Nulls:	False
Name:	idRogui
Primary:	True
Required:	True
Unique:	True
Fields:	idRogui, Ascending

Table: RoguiVMSs

Columns

Name	Type	Size
idRogui	Text	20
idVMS	Text	20

Relationships

RoguisRoguiVMSs

Roguis	RoguiVMSs
idRogui	idRogui
1	∞
Attributes:	Enforced, Cascade Updates, Cascade Deletes
Attributes:	One-To-Many

Table Indexes

Name	Number of Fields
idRogui	1
Clustered:	False
Distinct Count:	0
Foreign:	False
Ignore Nulls:	False
Name:	idRogui
Primary:	False
Required:	False
Unique:	False
Fields:	idRogui, Ascending
idRoguiVMS	2
Clustered:	False
Distinct Count:	0
Foreign:	False
Ignore Nulls:	False
Name:	idRoguiVMS
Primary:	True
Required:	True
Unique:	True
Fields:	idRogui, Ascending idVMS, Ascending
idVMS	1
Clustered:	False
Distinct Count:	0
Foreign:	False
Ignore Nulls:	False
Name:	idVMS
Primary:	False
Required:	False
Unique:	False
Fields:	idVMS, Ascending
RoguisRoguiVMSs	1
Clustered:	False
Distinct Count:	0
Foreign:	True
Ignore Nulls:	False
Name:	RoguisRoguiVMSs
Primary:	False
Required:	False
Unique:	False
Fields:	idRogui, Ascending

Table: Sections

Columns

Name	Type	Size
idSection	Number (Long)	4
name	Text	20
level	Number (Integer)	2
nbLanes	Number (Integer)	2
slopeType	Number (Integer)	2
slopePercentage	Number (Single)	4
heightInitial	Number (Long)	4
heightEnd	Number (Long)	4
contourType	Number (Integer)	2
roadType	Number (Integer)	2
maxSpeed	Number (Single)	4
capacity	Number (Single)	4

optional

Text

20

Relationships

SectionsCentroidSections

Sections		CentroidSections
idSection	1 ∞	idSection
Attributes:		Enforced, Cascade Updates, Cascade Deletes
Attributes:		One-To-Many

SectionsContourPoints

Sections		ContourPoints
idSection	1 ∞	idSection
Attributes:		One-To-Many
Attributes:		Enforced, Cascade Updates, Cascade Deletes

SectionsDetectors

Sections		Detectors
idSection	1 ∞	idSection
Attributes:		One-To-Many
Attributes:		Enforced, Cascade Updates, Cascade Deletes

SectionsLaterals

Sections		Laterals
idSection	1 ∞	idSection
Attributes:		One-To-Many
Attributes:		Enforced, Cascade Updates, Cascade Deletes

SectionsMeterings

Sections		Meterings
idSection	1 ∞	idSection
Attributes:		One-To-Many
Attributes:		Enforced, Cascade Updates, Cascade Deletes

SectionsRoguiRouteSections

Sections		RoguiRouteSections
idSection	1 ∞	idSection
Attributes:		Enforced, Cascade Updates, Cascade Deletes
Attributes:		One-To-Many

SectionsSelPermissions

Sections		SelPermissions
idSection	1 1	idSection
Attributes:		Unique, Enforced, Cascade Updates, Cascade Deletes
Attributes:		One-To-One

SectionsStages

Sections		Rights
idSection	1 ∞	idSectionOrigin
Attributes:		One-To-Many
Attributes:		Enforced, Cascade Updates, Cascade Deletes

SectionsStages1

Sections		Rights
idSection	1 ∞	idSectionDest
Attributes:		One-To-Many
Attributes:		Enforced, Cascade Updates, Cascade Deletes

SectionsTurnings

Sections		Turnings
idSection	1 ∞	idSectionOrigin
Attributes:		One-To-Many
Attributes:		Enforced, Cascade Updates, Cascade Deletes

SectionsTurnings1

Sections		Turnings
idSection	1 ∞	idSectionDest
Attributes:		One-To-Many
Attributes:		Enforced, Cascade Updates, Cascade Deletes

SectionsVMSs

Sections		VMSs
idSection	1 ∞	idSection
Attributes:		Enforced, Cascade Updates, Cascade Deletes
Attributes:		One-To-Many

Table Indexes

Name	Number of Fields
------	------------------

idSection	1
Clustering:	False
Distinct Count:	2
Foreign:	False
Ignore Nulls:	False
Name:	idSection
Primary:	True
Required:	True
Unique:	True
Fields:	idSection, Ascending

Table: SelPermissions

Columns

	Type	Size
Name	Number (Long)	4
idSection	Number (Integer)	2
idLaneFirst	Number (Integer)	2
idLaneLast	Text	20

Relationships

SectionsSelPermissions

Sections	SelPermissions
idSection	idSection
Attributes:	Unique, Enforced, Cascade Updates, Cascade Deletes
Attributes:	One-To-One

VehClassesSelPermissions

VehClasses	SelPermissions
idVehClass	idVehClass
Attributes:	Enforced, Cascade Updates, Cascade Deletes
Attributes:	One-To-Many

Table Indexes

Name	Number of Fields
idClassVehicle	1
Clustering:	False
Distinct Count:	0
Foreign:	False
Ignore Nulls:	False
Name:	idClassVehicle
Primary:	False
Required:	False
Unique:	False
Fields:	idVehClass, Ascending
idSection	1

Clustered:	False
Distinct Count:	0
Foreign:	False
Ignore Nulls:	False
Name:	idSection
Primary:	True
Required:	True
Unique:	True
Fields:	idSection, Ascending
SectionsSelPermissions	1
Clustered:	False
Distinct Count:	0
Foreign:	True
Ignore Nulls:	False
Name:	SectionsSelPermissions
Primary:	False
Required:	False
Unique:	True
Fields:	idSection, Ascending
VehClassesSelPermissions	1
Clustered:	False
Distinct Count:	0
Foreign:	True
Ignore Nulls:	False
Name:	VehClassesSelPermissions
Primary:	False
Required:	False
Unique:	False
Fields:	idVehClass, Ascending

Table: Stages

Columns

Name	Type	Size
idNode	Number (Long)	4
idStage	Number (Integer)	2
type	Number (Integer)	2

Relationships

NodesStages1

Nodes	Stages
idNode	1 ∞ idNode
Attributes:	Enforced, Cascade Updates, Cascade Deletes
Attributes:	One-To-Many

StagesRights

Stages	Rights
idNode	1 ∞ idNode
idStage	1 ∞ idStage

Attributes: Enforced, Cascade Updates, Cascade Deletes
 Attributes: One-To-Many

Table Indexes

Name	Number of Fields
idNode	1
Clustered:	False
Distinct Count:	0
Foreign:	False
Ignore Nulls:	False
Name:	idNode
Primary:	False
Required:	False
Unique:	False
Fields:	idNode, Ascending
NodesStages1	1
Clustered:	False
Distinct Count:	0
Foreign:	True
Ignore Nulls:	False
Name:	NodesStages1
Primary:	False
Required:	False
Unique:	False
Fields:	idNode, Ascending
PrimaryKey	2
Clustered:	False
Distinct Count:	0
Foreign:	False
Ignore Nulls:	False
Name:	PrimaryKey
Primary:	True
Required:	True
Unique:	True
Fields:	idNode, Ascending idStage, Ascending

Table: Texts

Columns

Name	Type	Size
idText	Number (Long)	4
level	Number (Integer)	2
string	Text	200
x	Number (Double)	8
y	Number (Double)	8
fontName	Text	27
fontType	Number (Long)	4
fontHeight	Number (Single)	4
colorRed	Number (Long)	4
colorGreen	Number (Long)	4
colorBlue	Number (Long)	4

Table Indexes

Name	Number of Fields
idText	1

Clustered:	False
Distinct Count:	0
Foreign:	False
Ignore Nulls:	False
Name:	idText
Primary:	False
Required:	False
Unique:	False
Fields:	idText, Ascending
PrimaryKey	1
Clustered:	False
Distinct Count:	0
Foreign:	False
Ignore Nulls:	False
Name:	PrimaryKey
Primary:	True
Required:	True
Unique:	True
Fields:	idText, Ascending

Table: Turnings

Columns

Name	Type	Size
idNode	Number (Long)	4
idSectionOrigin	Number (Long)	4
idSectionDest	Number (Long)	4
originIdLaneFirst	Number (Integer)	2
originIdLaneLast	Number (Integer)	2
destIdLaneFirst	Number (Integer)	2
destIdLaneLast	Number (Integer)	2
giveawayType	Number (Long)	4
maxSpeedAutomatic	Yes/No	1
maxSpeed	Number (Single)	4
optional	Text	20

Relationships

NodesTurnings

Nodes	Turnings
idNode	1 ∞ idNode
Attributes:	Enforced, Cascade Updates, Cascade Deletes
Attributes:	One-To-Many

SectionsTurnings

Sections	Turnings
idSection	1 ∞ idSectionOrigin
Attributes:	Enforced, Cascade Updates, Cascade Deletes
Attributes:	One-To-Many

SectionsTurnings1

Sections	Turnings
idSection	1 ∞ idSectionDest
Attributes:	Enforced, Cascade Updates, Cascade Deletes
Attributes:	One-To-Many

Table Indexes

Name	Number of Fields
idNode	1
Clustered:	False
Distinct Count:	0
Foreign:	False
Ignore Nulls:	False
Name:	idNode
Primary:	False
Required:	False
Unique:	False
Fields:	idNode, Ascending
idSectionDest	1
Clustered:	False
Distinct Count:	0
Foreign:	False
Ignore Nulls:	False
Name:	idSectionDest
Primary:	False
Required:	False
Unique:	False
Fields:	idSectionDest, Ascending
IdSectionOrigin	1
Clustered:	False
Distinct Count:	0
Foreign:	False
Ignore Nulls:	False
Name:	IdSectionOrigin
Primary:	False
Required:	False
Unique:	False
Fields:	idSectionOrigin, Ascending
idTurning	3
Clustered:	False
Distinct Count:	0
Foreign:	False
Ignore Nulls:	False
Name:	idTurning
Primary:	True
Required:	True
Unique:	True
Fields:	idNode, Ascending idSectionOrigin, Ascending idSectionDest, Ascending
NodesTurnings	1
Clustered:	False
Distinct Count:	0
Foreign:	True
Ignore Nulls:	False
Name:	NodesTurnings
Primary:	False
Required:	False
Unique:	False
Fields:	idNode, Ascending
SectionsTurnings	1
Clustered:	False
Distinct Count:	0
Foreign:	True
Ignore Nulls:	False

Name:	SectionsTurnings
Primary:	False
Required:	False
Unique:	False
Fields:	idSectionOrigin, Ascending
SectionsTurnings1	1
Clustered:	False
Distinct Count:	0
Foreign:	True
Ignore Nulls:	False
Name:	SectionsTurnings1
Primary:	False
Required:	False
Unique:	False
Fields:	idSectionDest, Ascending

Table: VehClasses

Columns

Name	Type	Size
idVehClass	Text	20

Relationships

VehClassesSelPermissions

VehClasses	SelPermissions
idVehClass	1 ∞ idVehClass
Attributes:	Enforced, Cascade Updates, Cascade Deletes
Attributes:	One-To-Many

Table Indexes

Name	Number of Fields
idVehClass	1
Clustered:	False
Distinct Count:	0
Foreign:	False
Ignore Nulls:	False
Name:	idVehClass
Primary:	True
Required:	True
Unique:	True
Fields:	idVehClass, Ascending

Table: VMSs

Columns

Name	Type	Size
idVMS	Text	20
idSection	Number (Long)	4
distance	Number (Single)	4

Relationships

SectionsVMSs

Sections	VMSs
idSection	1 ∞ idSection
Attributes:	Enforced, Cascade Updates, Cascade Deletes
Attributes:	One-To-Many

VMSsControllerVMSs

VMSs	ControllerVMSs
idVMS	1 ∞ idVMS
Attributes:	Enforced, Cascade Updates, Cascade Deletes
Attributes:	One-To-Many

Table Indexes

Name	Number of Fields
idSection	1
Clustering:	False
Distinct Count:	0
Foreign:	False
Ignore Nulls:	False
Name:	idSection
Primary:	False
Required:	False
Unique:	False
Fields:	idSection, Ascending
idVMS	1
Clustering:	False
Distinct Count:	0
Foreign:	False
Ignore Nulls:	False
Name:	idVMS
Primary:	True
Required:	True
Unique:	True
Fields:	idVMS, Ascending
SectionsVMSs	1
Clustering:	False
Distinct Count:	0
Foreign:	True
Ignore Nulls:	False
Name:	SectionsVMSs
Primary:	False
Required:	False

Unique: False
Fields: idSection, Ascending

Properties

AccessVersion:	07.53	Build:	4122
Collating Order:	General	Def. Updatable:	True
Query Timeout:	60	Records Affected:	0
Transactions:	True	Version :	3.0

User Permissions

admin

Group Permissions

Admins
Users

Relationships: All

Relationships

BackgroundsBackgroundLayers

Backgrounds		BackgroundLayers
fileName	1 ∞	fileName
Attributes:		One-To-Many Enforced, Cascade Updates, Cascade Deletes

BlocksBlockPoints

Blocks		BlockPoints
idBlock	1 ∞	idBlock
Attributes:		One-To-Many Enforced, Cascade Updates, Cascade Deletes

CentroidsCentroidNodes

Centroids		CentroidNodes
idCentroid	1 ∞	idCentroid
Attributes:		One-To-Many Enforced, Cascade Updates, Cascade Deletes

CentroidsCentroidSections

Centroids		CentroidSections
idCentroid	1 ∞	idCentroid
Attributes:		Enforced, Cascade Updates, Cascade Deletes
Attributes:		One-To-Many

ControllersControllerDetectors

Controllers		ControllerDetectors
idController	1 1	idController
Attributes:		One-To-One
Attributes:		Unique, Enforced, Cascade Updates, Cascade Deletes

ControllersControllerMeterings

Controllers		ControllerMeterings
idController	1 1	idController
Attributes:		One-To-One
Attributes:		Unique, Enforced, Cascade Updates, Cascade Deletes

ControllersControllerNodes

Controllers		ControllerNodes
idController	1 1	idController
Attributes:		One-To-One
Attributes:		Unique, Enforced, Cascade Updates, Cascade Deletes

ControllersControllerVMSs

Controllers		ControllerVMSs
idController	1 1	idController
Attributes:		Unique, Enforced, Cascade Updates, Cascade Deletes
Attributes:		One-To-One

DetectorsControllerDetectors

Detectors		ControllerDetectors
idDetector	1 ∞	idDetector
Attributes:		Enforced, Cascade Updates, Cascade Deletes
Attributes:		One-To-Many

MeteringsControllerMeterings

Meterings		ControllerMeterings	
idMetering	1	∞ idMetering	
Attributes:		One-To-Many	
Attributes:		Enforced, Cascade Updates, Cascade Deletes	
NodesCentroidNodes			
Nodes		CentroidNodes	
idNode	1	∞ idNode	
Attributes:		One-To-Many	
Attributes:		Enforced, Cascade Updates, Cascade Deletes	
NodesControllerNodes			
Nodes		ControllerNodes	
idNode	1	∞ idNode	
Attributes:		Enforced, Cascade Updates, Cascade Deletes	
Attributes:		One-To-Many	
NodesStages			
Nodes		Rights	
idNode	1	∞ idNode	
Attributes:		One-To-Many	
Attributes:		Enforced, Cascade Updates, Cascade Deletes	
NodesStages1			
Nodes		Stages	
idNode	1	∞ idNode	
Attributes:		One-To-Many	
Attributes:		Enforced, Cascade Updates, Cascade Deletes	
NodesTurnings			
Nodes		Turnings	
idNode	1	∞ idNode	
Attributes:		One-To-Many	
Attributes:		Enforced, Cascade Updates, Cascade Deletes	
RoguiRoutesRoguiRouteSections			
RoguiRoutes		RoguiRouteSections	
idRogui	1	∞ idRogui	
idRoute	1	∞ idRoute	

Attributes: One-To-Many
Attributes: Enforced

RoguisRoguiRoutes

Roguis **RoguiRoutes**
idRogui 1 ∞ idRogui

Attributes: Enforced, Cascade Updates, Cascade Deletes
Attributes: One-To-Many

RoguisRoguiVMSs

Roguis **RoguiVMSs**
idRogui 1 ∞ idRogui

Attributes: One-To-Many
Attributes: Enforced, Cascade Updates, Cascade Deletes

SectionsCentroidSections

Sections **CentroidSections**
idSection 1 ∞ idSection

Attributes: One-To-Many
Attributes: Enforced, Cascade Updates, Cascade Deletes

SectionsContourPoints

Sections **ContourPoints**
idSection 1 ∞ idSection

Attributes: One-To-Many
Attributes: Enforced, Cascade Updates, Cascade Deletes

SectionsDetectors

Sections **Detectors**
idSection 1 ∞ idSection

Attributes: One-To-Many
Attributes: Enforced, Cascade Updates, Cascade Deletes

SectionsLaterals

Sections **Laterals**
idSection 1 ∞ idSection

Attributes: Enforced, Cascade Updates, Cascade Deletes
Attributes: One-To-Many

SectionsMeterings

Sections		Meterings
idSection	1 ∞	idSection
Attributes:		One-To-Many
Attributes:		Enforced, Cascade Updates, Cascade Deletes

SectionsRoguiRouteSections

Sections		RoguiRouteSections
idSection	1 ∞	idSection
Attributes:		One-To-Many
Attributes:		Enforced, Cascade Updates, Cascade Deletes

SectionsSelPermissions

Sections		SelPermissions
idSection	1 1	idSection
Attributes:		One-To-One
Attributes:		Unique, Enforced, Cascade Updates, Cascade Deletes

SectionsStages

Sections		Rights
idSection	1 ∞	idSectionOrigin
Attributes:		One-To-Many
Attributes:		Enforced, Cascade Updates, Cascade Deletes

SectionsStages1

Sections		Rights
idSection	1 ∞	idSectionDest
Attributes:		One-To-Many
Attributes:		Enforced, Cascade Updates, Cascade Deletes

SectionsTurnings

Sections		Turnings
idSection	1 ∞	idSectionOrigin
Attributes:		One-To-Many
Attributes:		Enforced, Cascade Updates, Cascade Deletes

SectionsTurnings1

Sections		Turnings
-----------------	--	-----------------

idSection 1 ∞ idSectionDest

Attributes: One-To-Many
Attributes: Enforced, Cascade Updates, Cascade Deletes

SectionsVMSs

Sections **VMSs**
idSection 1 ∞ idSection

Attributes: Enforced, Cascade Updates, Cascade Deletes
Attributes: One-To-Many

StagesRights

Stages **Rights**
idNode 1 ∞ idNode
idStage 1 ∞ idStage

Attributes: One-To-Many
Attributes: Enforced, Cascade Updates, Cascade Deletes

VehClassesSelPermissions

VehClasses **SelPermissions**
idVehClass 1 ∞ idVehClass

Attributes: One-To-Many
Attributes: Enforced, Cascade Updates, Cascade Deletes

VMSsControllerVMSs

VMSs **ControllerVMSs**
idVMS 1 ∞ idVMS

Attributes: Enforced, Cascade Updates, Cascade Deletes
Attributes: One-To-Many



Appendix B

Relational Database Functions

Appendix B: Relational Database Functions

Appendix B: Relational Database Functions.....	1
Implementation.....	5
GDC Prototype Implementation.....	5
Blocks.....	5
TDRRetrieveBlock.....	6
TDRRetrieveBlockPoints.....	7
TDRRetrieveBlockPoints.....	7
TDRStoreBlock.....	7
TDRExistBlock.....	8
TDRReadBlockIds.....	8
TDRDeleteBlock.....	8
TDRListBlocks.....	9
TDRNbBlocks.....	9
Texts.....	9
TDRRetrieveText.....	10
TDRStoreText.....	10
TDRExistText.....	10
TDRSearchText.....	11
TDRReadTextIds.....	11
TDRDeleteText.....	12
TDRListTexts.....	12
TDRNbTexts.....	12
Vehicle Classes.....	12
TDRRetrieveVehClass.....	13
TDRStoreVehClass.....	13
TDRExistVehClass.....	14
TDRDeleteVehClass.....	14
TDRListVehClasses.....	14
TDRNbVehClasses.....	15
Global Messages.....	15
TDRRetrieveGlobMessage.....	15
TDRStoreGlobMessage.....	16
TDRExistGlobMessage.....	16
TDRDeleteGlobMessage.....	16
TDRListGlobMessages.....	17
TDRNbGlobMessages.....	17
Sections.....	17
TDRRetrieveSection.....	18
TDRRetrieveSectionLaterals.....	18
TDRRetrieveSectionSelPermissions.....	19
TDRRetrieveSectionDetectors.....	19
TDRRetrieveSectionMeterings.....	19
TDRRetrieveSectionMessaSigns.....	20
TDRStoreSection.....	20

TDRExistSection.....	21
TDRSearchSection	21
TDRSearchSectionByOptional	22
TDRReadSectionIds	22
TDRDeleteSection.....	22
TDRListSections	23
TDRNbSections	23
TDRExistDetector	23
TDRSearchDetector.....	24
TDRExistMetering	24
TDRSearchMetering.....	24
TDRExistMessaSign	25
TDRSearchMessaSign.....	25
Nodes	26
TDRRetrieveNode	26
TDRRetrieveNodeTurnings	26
TDRRetrieveNodeStages.....	27
TDRStoreNode.....	27
TDRExistNode	28
TDRReadNodeIds	28
TDRSearchJuncture.....	29
TDRReadJunctureIds.....	29
TDRSearchJunction.....	30
TDRReadJunctionIds.....	30
TDRDeleteNode.....	30
TDRListNodes	31
TDRNbNodes.....	31
Controllers.....	31
TDRRetrieveController	32
TDRRetrieveControllerConDevices.....	32
TDRStoreController	32
TDRExistController.....	34
TDRReadControllerIds.....	34
TDRDeleteController	35
TDRListControllers.....	35
TDRNbControllers	35
Centroids	35
TDRRetrieveCentroid.....	36
TDRRetrieveCentroidCenConnections	36
TDRStoreCentroid.....	36
TDRExistCentroid.....	37
TDRReadCentroidIds	37
TDRReadObjectToCentroidConnections	38
TDRDeleteCentroid.....	38
TDRListCentroids	38
TDRNbCentroids.....	39

Roguis	39
TDRRetrieveRogui	39
TDRRetrieveRoguiRoutes	40
TDRRetrieveRoguiUpVMSs	40
TDRStoreRogui	40
TDRExistRogui	41
TDRReadRoguiIds	41
TDRDeleteRogui	41
TDRListRoguis	42
TDRNbRoguis	42
Routes	42
TDRRetrieveRoute	42
TDRStoreRoute	43
TDRExistRoute	43
TDRReadRouteIds	43
TDRDeleteRoute	44
TDRListRoutes	44
TDRNbRoutes	44
Network	45
TDROpenDatabase	45
TDRCloseDatabase	45
TDRPrepareNewNetwork	46
TDROpenNetwork	46
TDRGuessNetworkFormat	46
TDRReadNetworkDim	47
TDRCloseNetwork	47
TDRWriteNetworkGlobals	47
Relational Database	48
ODBCRegisterDataSource	48
ODBCUnRegisterDataSource	49
ODBCCopyDOSFile	49
ODBCGetTextData	50
ODBCGetIntData	50
ODBCGetLongIntData	51
ODBCGetFloatData	51
ODBCGetDoubleData	52
ODBCAllocStmt	52
ODBCFreeStmt	52
ODBCSelect	53
ODBCFetch	53
ODBCSelectFetch	53
ODBCTableSize	54

Implementation

GDC Prototype Implementation

The geometry data container (GDC) is implementation of the original geometry system done in ASCII Files, but redesigned to make it functional in a relational database format. All the new functions and their implementation is transparent to the user of this system. This redesign of functions in the input/output portion of the traffic simulation system (TSS) is not a trivial rewriting of code. Our initial design of the database includes more than 105 completely new functions at the lowest level. Each function accesses the relational database system for either retrieval, updating, or saving of geometry or geometry-like information. These functions can be grouped into functional categories that operate on specific parts of the geometry. The functional classes correspond to their analogs in the original ASCII version of the geometry and are defined as follows:

- Blocks: drawing blocks which help in visualizing the area being simulated
- Texts: text used to identify objects being viewed
- Vehicle Classes: vehicle class definitions such as HOV->TRUCKS+BUSES
- Global Messages: global messages for drivers
- Sections: geometry information on sections of roadways
- Nodes: nodes link sections to form networks
- Controllers: controllers link meters and detectors to nodes/sections
- Centroids: used for origin/destination routing
- Roguis: used for origin/destination routing
- Routes: used for origin/destination routing
- Network: general network characteristics

In order to explain the implementation, it is necessary to understand not only the overall structure of the classes listed above, but the individual functions that need to be implemented for the relational database to function. In the following, we will explain how each category is used in the implementation and a definition and explanation of each function that needs to be implemented. More than 100 new functions had to be designed and implemented to make the GDC fully operational.

Blocks

Blocks are background objects (polygons) used to help visualize the traffic network. Blocks are polygons that can have shape and color. For example, blocks can be used visualize physical features such as rivers, buildings, and other like scenes. In figure 1, a block is drawn using high level functions that ultimately call the low level Block functions described below.

Implementing Blocks in the relational database requires defining the following functions:

- TDRRetrieveBlock
- TDRRetrieveBlockPoints
- TDRStoreBlock
- TDRExistBlock
- TDRReadBlockIds
- TDRDeleteBlock

- TDRListBlocks
- TDRNbBlocks

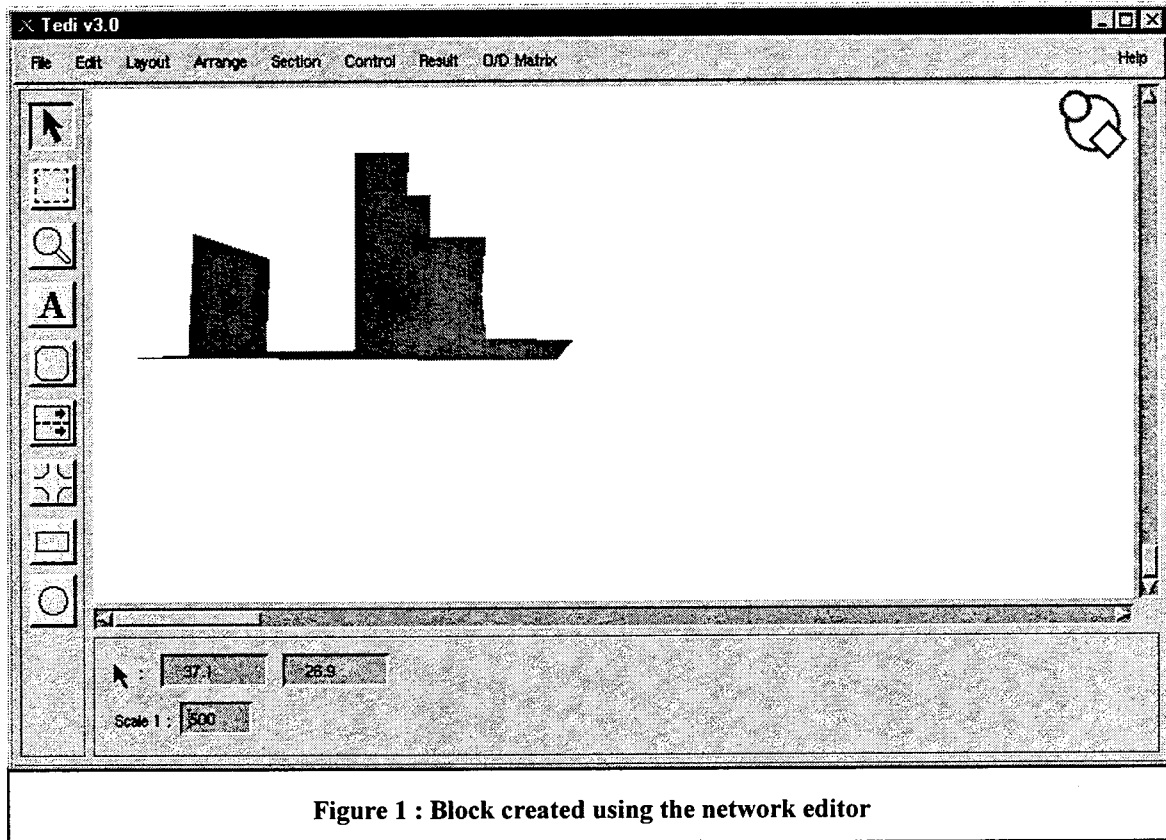


Figure 1 : Block created using the network editor

In order to understand the implementation, it is necessary to explain each function, its parameters, and what action that it should perform on the database. What follows is a detailed explanation of each of the Block functions.

Finally, another class of functions was defined to provide relational database access from the function classes defined in the geometry. This class of functions provides Open Database Connectivity (ODBC) calls to any ODBC compliant database allowing many different databases to be used with the same code without major, if any changes.

TDRRetrieveBlock

Summary	This function TDRRetrieveBlock retrieves the block from the database, builds a block object, and links it to the cache provided. An incremental retrieval is performed first, the remaining information will be retrieved when needed.	
Syntax	<pre>int TDRRetrieveBlock(int idblock, TDICacheObject *cache)</pre>	
Arguments	idblock [INPUT]	Id of block to read
	cache[INPUT/OUTPUT]	List to link new object to
Returns	0	No errors
	MemAlloc	Memory allocation error

BlockUnk

Unknown block

Comments This function is one of the TDR block functions that works exclusively with blocks and does not interact with other types of objects. The errors defined by the mnemonic are defined in TDErrors.h.

TDRRetrieveBlockPoints

Summary Retrieves from the relational database the points of the block currently in the block storage manipulation.

Syntax int TDRRetrieveBlockPoints(
TDIBlock *block)

Arguments *block [INPUT/OUTPUT] Link to internal storage format

Returns 0 No errors
MemAlloc Memory allocation error

Comments This function is one of the TDR block functions that works exclusively with blocks and does not interact with other types of objects. The errors defined by the mnemonic are defined in TDErrors.h.

TDRRetrieveBlockPoints

Summary Retrieves from the relational database the points of the block currently in the block storage manipulation.

Syntax int TDRRetrieveBlockPoints(
TDIBlock *block)

Arguments *block Link to block storage manipulation

Returns 0 no errors
MemAlloc memory allocation

Comments This function is one of the TDR block functions that works exclusively with blocks and does not interact with other types of objects. The errors defined by the mnemonic are defined in TDErrors.h.

TDRStoreBlock

Summary Stores the contents of the block storage manipulation to the relational database. If it is a new block, supposes that it does not exist in the relational database. CAUTION: The view is not updated here!!

Syntax int TDRStoreBlock(
TDIBlock *block_source)

Arguments *block_source Link to the block storage manipulation

Returns 0 no errors
ObjectNotOpen - the required object is not open

	ObjectInteg	the object does not comply integrity requirements
	MemAlloc	memory allocation
Comments	This function is one of the TDR block functions that works exclusively with blocks and does not interact with other types of objects. The errors defined by the mnemonic are defined in TDErrors.h.	

TDRExistBlock

Summary	Looks in the list of blocks in the relational database, looking for the block identifier idblock. Returns TRUE if found, FALSE otherwise.	
Syntax	Bool TDRExistBlock(int idblock)	
Arguments	idblock	Block identifier
Returns	TRUE FALSE	Block found Block not found
Comments	This function is one of the TDR block functions that works exclusively with blocks and does not interact with other types of objects. The errors defined by the mnemonic are defined in TDErrors.h.	

TDRReadBlockIds

Summary	Read the identifiers of the nodes in the relational database, and returns them in an array.	
Syntax	Int TDRReadBlockIds(int *maxids, int *nbids, int **ids, int *idcurrent)	
Arguments	*maxids *nbids **ids *idcurrent	Maximum number of Ids to read Number of Ids read from the database An array returning the Ids read Current Id
Returns	0 MemAlloc	no errors allocating memory
Comments	This function is one of the TDR block functions that works exclusively with blocks and does not interact with other types of objects. The errors defined by the mnemonic are defined in TDErrors.h.	

TDRDeleteBlock

Summary	Deletes this block from the relational database.	
Syntax	int TDRDeleteBlock(int idblock)	

Arguments	idblock	Identifier of block to delete
Returns	0 BlockUnk	no errors Block not found
Comments	This function is one of the TDR block functions that works exclusively with blocks and does not interact with other types of objects. The errors defined by the mnemonic are defined in TDErrors.h.	

TDRListBlocks

Summary	Lists the blocks in the relational database, with output to the screen	
Syntax	void TDRListBlocks()	
Arguments	void	None
Returns	void	no errors
Comments	This function is one of the TDR block functions that works exclusively with blocks and does not interact with other types of objects. The errors defined by the mnemonic are defined in TDErrors.h.	

TDRNbBlocks

Summary	Counts the blocks in the relational database.	
Syntax	Int TDRNbBlocks()	
Arguments	void	None
Returns	Number of blocks	Number of blocks in the network
Comments	This function is one of the TDR block functions that works exclusively with blocks and does not interact with other types of objects. The errors defined by the mnemonic are defined in TDErrors.h.	

Texts

Texts are background texts, which are used to identify parts of the traffic network. Texts can be used to identify scenes such as rivers, buildings, other scenes, as well as parts of the traffic network. Implementing Texts in the relational database requires defining the following functions:

- TDRRetrieveText
- TDRStoreText
- TDRExistText
- TDRSearchText
- TDRReadTextIds
- TDRDeleteText
- TDRListTexts
- TDRNbTexts

TDRRetrieveText

Summary	Retrieves from the relational database the text and puts it in the cache provided. Supposes an unallocated storage.	
Syntax	int TDRRetrieveText(int idtext, TDICacheObject *cache)	
Arguments	idtext *cache	Identifier of the requested text Link to internal cache
Returns	0 MemAlloc TextUnk	no errors memory allocation unknown text
Comments	This function is one of the TDR text functions that works exclusively with texts and does not interact with other types of objects. The errors defined by the mnemonic are defined in TDErrors.h.	

TDRStoreText

Summary	Stores the contents of the text storage manipulation to the relational database presentation. If it is a new text, supposes that it does not exist in the relational database. CAUTION: The view is not updated here!!	
Syntax	int TDRStoreText(TDIText *text_source)	
Arguments	text_source	Link to object to be stored to the relational database
Returns	0 ObjectNotOpen ObjectInteg MemAlloc	no errors the required object is not open the object does not comply integrity requirements memory allocation
Comments	This function is one of the TDR text functions that works exclusively with texts and does not interact with other types of objects. The errors defined by the mnemonic are defined in TDErrors.h.	

TDRExistText

Summary	Looks in the list of texts in the relational database, looking for the text with identifier idtext. Returns TRUE if found, FALSE otherwise.	
Syntax	Bool TDRExistText(int idtext)	

Arguments	idtext	Id of text to search for
Returns	TRUE FALSE	If text is found If text is not found
Comments	This function is one of the TDR text functions that works exclusively with texts and does not interact with other types of objects. The errors defined by the mnemonic are defined in TDErrors.h.	

TDRSearchText

Summary	Looks in the list of texts in the relational database, looking for the text with identifier idtext. Returns a pointer to the text or NULL if not found. The previous text is also returned.	
Syntax	TDIText TDRSearchText(int idtext, TDIText **text_prev)	
Arguments	idtext **text_prev	Id of the text being searched for Pointer to the previous text
Returns	NULL Pointer to text	Not found If found
Comments	This function is one of the TDR text functions that works exclusively with texts and does not interact with other types of objects. The errors defined by the mnemonic are defined in TDErrors.h.	

TDRReadTextIds

Summary	Read the identifiers of the nodes in the relational database, and returns them in an array.	
Syntax	Int TDRReadTextIds(int *maxids, int *nbids, int **ids, int *idcurrent)	
Arguments	*maxids *nbids **ids *idcurrent	Maximum number of Ids to return Number of Ids returned Array of Ids returned Current Id
Returns	0 MemAlloc	no errors allocating memory
Comments	This function is one of the TDR text functions that works exclusively with texts and does not interact with other types of objects. The errors defined by the mnemonic are defined in TDErrors.h.	

TDRDeleteText

Summary	Deletes this text from the relational database	
Syntax	int TDRDeleteText(int idtext)	
Arguments	idtext	Id of text to delete
Returns	0	no errors
Comments	This function is one of the TDR text functions that works exclusively with texts and does not interact with other types of objects. The errors defined by the mnemonic are defined in TDErrors.h.	

TDRListTexts

Summary	Lists the texts in the relational database, with output to the screen.	
Syntax	Void TDRListTexts()	
Arguments	Void	No arguments
Returns	Void	Nothing returned
Comments	This function is one of the TDR text functions that works exclusively with texts and does not interact with other types of objects. The errors defined by the mnemonic are defined in TDErrors.h.	

TDRNbTexts

Summary	Counts the texts in the relational database	
Syntax	Int TDRNbTexts()	
Arguments	Void	No arguments
Returns	0 Number of texts	No texts found Number of texts found
Comments	This function is one of the TDR text functions that works exclusively with texts and does not interact with other types of objects. The errors defined by the mnemonic are defined in TDErrors.h.	

Vehicle Classes

Vehicle Classes are groupings of types of vehicles. For example, one might define an HOV1 (High Occupancy Vehicle) class with taxis and buses, and another, HOV2, with trucks and large trucks. Now lanes could be reserved in sections just for HOV1 and HOV2 type vehicles and no others.

Implementing Vehicle Classes in the relational database requires defining the following functions:

- TDRRetrieveVehClass

- TDRStoreVehClass
- TDRExistVehClass
- TDRDeleteVehClass
- TDRListVehClasses
- TDRNbVehClasses

TDRRetrieveVehClass

Summary	Retrieves from the relational database the vehclass and puts it in the network storage manipulation. Supposes an unallocated storage	
Syntax	int TDRRetrieveVehClass(char *idvehclass, TDICacheObject *cache)	
Arguments	*idvehclass *cache	Link to internal cache
Returns	0 MemAlloc VehClassUnk	no errors memory allocation unknown class of vehicle modalities
Comments	This function is one of the TDR vehicle class functions that essentially defines classes of vehicles and their highway permissions. For example, a High Occupancy Vehicle (HOV) can have different lanes to use than a non HOV vehicle. The errors defined by the mnemonic are defined in TDErrors.h.	

TDRStoreVehClass

Summary	Stores the contents of the network storage manipulation to the relational database. If it is a new vehclass, supposes that it does not exist in the RDB. CAUTION: The view is not updated here!! If the vehclass was already existing and has changed its identifier, then it notifies that selpermissions have to be revised.	
Syntax	int TDRStoreVehClass(TDIVehClass *vehclass_source, Bool *check_selpermissions)	
Arguments	*vehclass_source *check_selpermissions	
Returns	0 ObjectInteg MemAlloc	no errors the object does not comply integrity requirements memory allocation
Comments	This function is one of the TDR vehicle class functions that essentially defines classes of vehicles and their highway permissions. For example, a High Occupancy Vehicle (HOV) can have different lanes to use than a non HOV vehicle. The errors defined by the mnemonic are defined in TDErrors.h.	

TDRExistVehClass

Summary	Looks in the list of vehicle classes in the network, looking for the one with this identifier. Returns TRUE if found, FALSE otherwise.	
Syntax	Bool TDRExistVehClass(char *idvehclass)	
Arguments	*idvehclass	Vehicle class identifier
Returns	0 Not equal 0	no errors Errors
Comments	This function is one of the TDR vehicle class functions that essentially defines classes of vehicles and their highway permissions. For example, a High Occupancy Vehicle (HOV) can have different lanes to use than a non HOV vehicle. The errors defined by the mnemonic are defined in TDErrors.h.	

TDRDeleteVehClass

Summary	Deletes a vehicle modality class from the RDB, deleting also all selective permissions with this class.	
Syntax	int TDRDeleteVehClass(char *idvehclass)	
Arguments	*idvehclass	Vehicle class identification
Returns	0 VehClassUnk	no errors unknown vehicle modality class
Comments	This function is one of the TDR vehicle class functions that essentially defines classes of vehicles and their highway permissions. For example, a High Occupancy Vehicle (HOV) can have different lanes to use than a non HOV vehicle. The errors defined by the mnemonic are defined in TDErrors.h.	

TDRListVehClasses

Summary	Lists the vehicle modalities classes in the network, with output to the screen.	
Syntax	Void TDRListVehClasses()	
Arguments	Void	No arguments
Returns	Void	No return value
Comments	This function is one of the TDR vehicle class functions that essentially defines classes of vehicles and their highway permissions. For example, a High Occupancy Vehicle (HOV) can have different lanes to use than a non HOV vehicle. The errors defined by the mnemonic are defined in TDErrors.h.	

TDRNbVehClasses

Summary	Counts the classes of vehicles in the network.	
Syntax	Int TDRNbVehClasses().	
Arguments	Void	No arguments
Returns	0 Not equal 0	no errors Errors
Comments	This function is one of the TDR vehicle class functions that essentially defines classes of vehicles and their highway permissions. For example, a High Occupancy Vehicle (HOV) can have different lanes to use than a non HOV vehicle. The errors defined by the mnemonic are defined in TDErrors.h.	

Global Messages

Global Messages are messages that are visible on either variable message signs, or changeable message signs that are used on modern freeways. Each section can contain a variable message sign with the potential messages and potential actions by drivers once the message sign is activated. For example, if freeway traffic is heavy then the sign may be set to read "Congestion Ahead, Use Alternate Routes." Once the sign is activated then a potential action would be for 10% of the drivers to take the next exit.

Implementing Texts in the relational database requires defining the following functions:

- TDRRetrieveGlobMessage
- TDRStoreGlobMessage
- TDRExistGlobMessage
- TDRDeleteGlobMessage
- TDRListGlobMessages
- TDRNbGlobMessages

TDRRetrieveGlobMessage

Summary	Retrieves from the relational database the globmessage and puts it the network storage manipulation. Supposes an unallocated storage.	
Syntax	int TDRRetrieveGlobMessage(int idglobmessage, TDICacheObject *cache)	
Arguments	idglobmessage *cache	Identifier for global message Link to internal cache
Returns	0 MemAlloc GlobMessageUnk	no errors memory allocation unknown class of vehicle modalities
Comments	This function is one of the TDR global message class functions that provide information to drivers such as accident ahead, congestion ahead, speed reduced. In addition, driver reactions to these messages is modeled. The errors defined by the mnemonic are defined in TDErrors.h.	

TDRStoreGlobMessage

Summary	Stores the contents of the network storage manipulation to the relational database representation. If it is a new globmessage, supposes that it does not exist in the RDB. CAUTION: The view is not updated here!! If the globmessage was already existing and has changed its identifier, then it notifies that selpermissions have to be revised.	
Syntax	int TDRStoreGlobMessage(TDIGlobMessage *globmessage_source, Bool *check_messasigns)	
Arguments	globmessage_source *check_messasigns	Link to global message Status
Returns	0 ObjectInteg MemAlloc	no errors the object does not comply integrity requirements memory allocation
Comments	This function is one of the TDR global message class functions that provide information to drivers such as accident ahead, congestion ahead, speed reduced. In addition, driver reactions to these messages is modeled. The errors defined by the mnemonic are defined in TDErrors.h.	

TDRExistGlobMessage

Summary	Looks in the list of global messages in the network, looking for the one with this identifier. Returns TRUE if found, FALSE otherwise. *	
Syntax	Bool TDRExistGlobMessage(int idglobmessage)	
Arguments	idglobmessage	Identifier for global message
Returns	TRUE FALSE	Message found Message not found
Comments	This function is one of the TDR global message class functions that provide information to drivers such as accident ahead, congestion ahead, speed reduced. In addition, driver reactions to these messages is modeled. The errors defined by the mnemonic are defined in TDErrors.h.	

TDRDeleteGlobMessage

Summary	Deletes this global message from the RDB.	
Syntax	int TDRDeleteGlobMessage(int idglobmessage)	
Arguments	idglobmessage	Id of the global message
Returns	0	no errors

Not equal to 0

Errors

Comments This function is one of the TDR global message class functions that provide information to drivers such as accident ahead, congestion ahead, speed reduced. In addition, driver reactions to these messages is modeled. The errors defined by the mnemonic are defined in TDErrors.h.

TDRListGlobMessages

Summary Lists the global messages in the network, with output to the screen.

Syntax Void TDRListGlobMessages()

Arguments Void No arguments

Returns 0 no errors
Not equal to 0 Errors

Comments This function is one of the TDR global message class functions that provide information to drivers such as accident ahead, congestion ahead, speed reduced. In addition, driver reactions to these messages is modeled. The errors defined by the mnemonic are defined in TDErrors.h.

TDRNbGlobMessages

Summary Counts the global messages in the network.

Syntax Int TDRNbGlobMessages()

Arguments Void No arguments

Returns 0 no errors
Not equal to 0 errors

Comments This function is one of the TDR global message class functions that provide information to drivers such as accident ahead, congestion ahead, speed reduced. In addition, driver reactions to these messages is modeled. The errors defined by the mnemonic are defined in TDErrors.h.

Sections

Sections contain the basic geometry of the network. A network itself is composed of a set of sections linked together by nodes. The sections themselves contain information about the physical characteristics of each section in the network such as number of lanes, lane length, lane width, capacity, maximum speed, type of roadway, slope, and contour.

Implementing Sections in the relational database requires defining the following functions:

- TDRRetrieveSection
- TDRRetrieveSectionLaterals
- TDRRetrieveSectionSelPermissions

- TDRRetrieveSectionDetectors
- TDRRetrieveSectionMeterings
- TDRRetrieveSectionMessaSigns
- TDRStoreSection
- TDRExistSection;
- TDRSearchSection
- TDRSearchSectionByOptional
- TDRReadSectionIds
- TDRDeleteSection
- TDRListSections
- TDRNbSections
- TDRExistDetector
- *TDRSearchDetector
- TDRExistMetering
- TDRSearchMetering
- TDRExistMessaSign
- TDRSearchMessaSign

TDRRetrieveSection

Summary	Retrieves from the relational database the section and puts it in the cache provided. Supposes an unallocated storage. An incremental retrieval is performed: firstly, only the basic info is retrieved, the remaining will be retrieved when needed.	
Syntax	int TDRRetrieveSection(int idsection, TDICacheObject *cache)	
Arguments	idsection *cache	Id of the section to be retrieved Link to internal cache
Returns	0 MemAlloc SectionUnk	no errors memory allocation unknown section
Comments	This function is one of the TDR section class functions. The functions in this class are fundamental to drawing and retrieving the section geometry. The errors defined by the mnemonic are defined in TDErrors.h.	

TDRRetrieveSectionLaterals

Summary	Retrieves from the relational database the laterals of the section currently in the section storage manipulation.	
Syntax	int TDRRetrieveSectionLaterals(TDISection *section)	
Arguments	*section	Link to internal section object
Returns	0 MemAlloc	no errors memory allocation

Comments This function is one of the TDR section class functions. The functions in this class are fundamental to drawing and retrieving the section geometry. The errors defined by the mnemonic are defined in TDErrors.h.

TDRRetrieveSectionSelPermissions

Summary Retrieves from the relational database the selpermissions of the section currently in the section storage manipulation.

Syntax int TDRRetrieveSectionSelPermissions(
TDISection *section)

Arguments *section Link to section storage manipulation

Returns 0 no errors
MemAlloc memory allocation

Comments This function is one of the TDR section class functions. The functions in this class are fundamental to drawing and retrieving the section geometry. The errors defined by the mnemonic are defined in TDErrors.h.

TDRRetrieveSectionDetectors

Summary Retrieves from the relational database the detectors of the section currently in the section storage manipulation.

Syntax int TDRRetrieveSectionDetectors(
TDISection *section)

Arguments *section Link to section storage manipulation

Returns 0 no errors
MemAlloc memory allocation

Comments This function is one of the TDR section class functions. The functions in this class are fundamental to drawing and retrieving the section geometry. The errors defined by the mnemonic are defined in TDErrors.h.

TDRRetrieveSectionMeterings

Summary Retrieves from the relational database the meterings of the section currently in the section storage manipulation.

Syntax int TDRRetrieveSectionMeterings(
TDISection *section)

Arguments *section Link to section storage manipulation

Returns 0 no errors
MemAlloc memory allocation

Comments This function is one of the TDR section class functions. The functions in this class are fundamental to drawing and retrieving the section geometry. The errors defined by the mnemonic are defined in TDErrors.h.

TDRRetrieveSectionMessaSigns

Summary Retrieves from the relational database the messasigns of the section currently in the section storage manipulation.

Syntax int TDRRetrieveSectionMessaSigns(
TDISection *section)

Arguments *section Link to section storage manipulation

Returns 0 no errors
MemAlloc memory allocation

Comments This function is one of the TDR section class functions. The functions in this class are fundamental to drawing and retrieving the section geometry. The errors defined by the mnemonic are defined in TDErrors.h.

TDRStoreSection

Summary Stores the contents of the section storage manipulation to the RDB. If it is a new section, supposes that it does not exist in the RDB. CAUTION: The view is not updated here!! Indicates also if the turnings have to be checked.

Syntax int TDRStoreSection(
TDISection *section_source,
Bool *check_turnings,
Bool *check_speedturnings,
int *nbdetectors_deleted,
TDIConDev **detectors_deleted,
int *nbmeterings_deleted,
TDIConDev **meterings_deleted,
int *nbmessasigns_deleted,
TDIConDev **messasigns_deleted,
int *nbdetectors_updated,
TDIConDev **detectors_updated,
int *nbmeterings_updated,
TDIConDev **meterings_updated,
int *nbmessasigns_updated,
TDIConDev **messasigns_updated)

Arguments *section_source Link to internal section object
*check_turnings Boolean
*check_speedturnings Boolean
*nbdetectors_deleted Number of detectors deleted
**detectors_deleted Array of deleted detectors
*nbmeterings_deleted Number of meterings deleted
**meterings_deleted Array of deleted meterings
*nbmessasigns_deleted Number of messages deleted
**messasigns_deleted Array of deleted messages

	*nbdetectors_updated	Number of detectors updated
	**detectors_updated	Array of updated detectors
	*nbmeterings_updated	Number of meterings updated
	**meterings_updated	Array of updated meterings
	*nbmessasigns_updated	Number of messages updated
	**messasigns_updated	Array of updated messages
Returns	0	no errors
	ObjectNotOpen	the required object is not open
	ObjectInteg	the object does not comply integrity requirements
	MemAlloc	memory allocation
Comments	This function is one of the TDR section class functions. The functions in this class are fundamental to drawing and retrieving the section geometry. The errors defined by the mnemonic are defined in TDErrors.h.	

TDRExistSection

Summary	Looks in the list of sections in the network, looking for the section with identifier idsection. Returns TRUE if found, FALSE otherwise.	
Syntax	Bool TDRExistSection(int idsection)	
Arguments	idsection	Identifier for section
Returns	TRUE FALSE	Section found Section not found
Comments	This function is one of the TDR section class functions. The functions in this class are fundamental to drawing and retrieving the section geometry. The errors defined by the mnemonic are defined in TDErrors.h.	

TDRSearchSection

Summary	Looks in the list of sections in the relational database, looking for the section with identifier idsection. Returns a pointer to the section or NULL if not found. The previous section is also returned.	
Syntax	TDISection TDRSearchSection(int idsection, TDISection **section_prev)	
Arguments	idsection	Identifier of section
	**section_prev	Link to previous section
Returns	NULL	Section not found
	Not NULL	Pointer to found section
Comments	This function is one of the TDR section class functions. The functions in this class are fundamental to drawing and retrieving the section geometry. The errors defined by the mnemonic are defined in TDErrors.h.	

TDRSearchSectionByOptional

Summary	Searches in the list of sections in the network, returning the one with the optional field, or NULL otherwise. Depending on type, it searches for this exact string or for this string as part of another string.	
Syntax	TDISection *TDRSearchSectionByOptional(TDName name, int search_type)	
Arguments	name search_type	String to search for in optional fields Type of search
Returns	NULL Not NULL	Not found Pointer to found object
Comments	This function is one of the TDR section class functions. The functions in this class are fundamental to drawing and retrieving the section geometry. The errors defined by the mnemonic are defined in TDErrors.h.	

TDRReadSectionIds

Summary	Read the identifiers of the sections in the whole network, and returns them in an array.	
Syntax	int TDRReadSectionIds(int *maxids, int *nbids, int **ids, int *idcurrent)	
Arguments	*maxids *nbids **ids *idcurrent	Maximum number of ids to return Number of ids actually returned Array of ids returned Current id
Returns	0 MemAlloc	no errors allocating memory
Comments	This function is one of the TDR section class functions. The functions in this class are fundamental to drawing and retrieving the section geometry. The errors defined by the mnemonic are defined in TDErrors.h.	

TDRDeleteSection

Summary	Deletes this section from the RDB. All the subobjects as selective permissions, detectors, meterings,.. and the references of all this in the nodes, controllers, ... will be automatically deleted by cascade. The condition of that a node without turnings cannot exist is also dealt elsewhere.	
---------	---	--

Syntax	void TDRDeleteSection(int idsection)	
Arguments	idsection	Identifier of section to be deleted
Returns	0 Not equal to 0	no errors Errors
Comments	This function is one of the TDR section class functions. The functions in this class are fundamental to drawing and retrieving the section geometry. The errors defined by the mnemonic are defined in TDErrors.h.	

TDRListSections

Summary	Lists the vehicle modalities in the network, with output to the screen. This function is usually used for debugging.	
Syntax	void TDRListSections()	
Arguments	Void	No arguments
Returns	0 Not equal to 0	no errors Errors
Comments	This function is one of the TDR section class functions. The functions in this class are fundamental to drawing and retrieving the section geometry. The errors defined by the mnemonic are defined in TDErrors.h.	

TDRNbSections

Summary	Counts the sections in the relational database.	
Syntax	Int TDRNbSections()	
Arguments	void	No arguments
Returns	0 > 0 <0	no errors Number of sections in database Errors
Comments	This function is one of the TDR section class functions. The functions in this class are fundamental to drawing and retrieving the section geometry. The errors defined by the mnemonic are defined in TDErrors.h.	

TDRExistDetector

Summary	Looks in the list of detectors in the network, looking for the one with the required identifier. Returns TRUE if found, FALSE otherwise. Looks among all detectors in the network. Within a section, detectors are sorted alphabetically.	
Syntax	Bool TDRExistDetector(char iddetector[TDMAXNAMLEN + 1])	

Arguments	iddetector[TDMAXNAMLEN + 1]	Identifier for detector
Returns	TRUE FALSE	Detector found Detector not found
Comments	This function is one of the TDR section class functions. The functions in this class are fundamental to drawing and retrieving the section geometry. The errors defined by the mnemonic are defined in TDErrors.h.	

TDRSearchDetector

Summary	Searches in the list of detectors in the RDB, returning the one with the required identifier, or NULL otherwise. Looks among all detectors in the RDB. Within a section, detectors are sorted alphabetically.	
Syntax	TDIDetector *TDRSearchDetector(char iddetector[TDMAXNAMLEN + 1], int *idsection)	
Arguments	iddetector[TDMAXNAMLEN + 1] *idsection	Identifier of detector Link to section where detector exists
Returns	0 Pointer	no errors Pointer to detector found, if not NULL
Comments	This function is one of the TDR section class functions. The functions in this class are fundamental to drawing and retrieving the section geometry. The errors defined by the mnemonic are defined in TDErrors.h.	

TDRExistMetering

Summary	Looks in the list of meterings in the RDB, looking for the one with the required identifier. Returns TRUE if found, FALSE otherwise. Looks among all meterings in the network. Within a section, meterings are sorted alphabetically.	
Syntax	Bool TDRExistMetering(char idmetering[TDMAXNAMLEN + 1])	
Arguments	idmetering[TDMAXNAMLEN + 1]	Identifier for metering
Returns	0 1	Metering not found Metering found
Comments	This function is one of the TDR section class functions. The functions in this class are fundamental to drawing and retrieving the section geometry. The errors defined by the mnemonic are defined in TDErrors.h.	

TDRSearchMetering

Summary	Searches in the list of meterings in the RDB, returning the one with the required identifier, or NULL otherwise. Looks among all meterings in the network. Within a section, meterings are sorted alphabetically.	
---------	---	--

Syntax	TDIMetering *TDRSearchMetering(char idmetering[TDMAXNAMLEN + 1], int *idsection)	
Arguments	idmetering[TDMAXNAMLEN + 1] *idsection	Identifier for metering Link to section
Returns	NULL Not NULL	Metering not found Pointer to metering found
Comments	This function is one of the TDR section class functions. The functions in this class are fundamental to drawing and retrieving the section geometry. The errors defined by the mnemonic are defined in TDErrors.h.	

TDRExistMessaSign

Summary	Looks in the list of VMSs in the network, looking for the one with the required identifier. Returns TRUE if found, FALSE otherwise. Looks among all messasigns in the network. Within a section, messasigns are sorted alphabetically.	
Syntax	Bool TDRExistMessaSign(char idmessasign[TDMAXNAMLEN + 1])	
Arguments	idmessasign[TDMAXNAMLEN + 1]	Identifier for message
Returns	TRUE FALSE	Message found Message not found
Comments	This function is one of the TDR section class functions. The functions in this class are fundamental to drawing and retrieving the section geometry. The errors defined by the mnemonic are defined in TDErrors.h.	

TDRSearchMessaSign

Summary	Searches in the list of messages in the network, returning the one with the required identifier, or NULL otherwise. Looks among all messasigns in the network. Within a section, messasigns are sorted alphabetically.	
Syntax	TDIMessaSign *TDRSearchMessaSign(char idmessasign[TDMAXNAMLEN + 1], int *idsection)	
Arguments	idmessasign[TDMAXNAMLEN + 1] *idsection	Identifier for message Link to section where message is
Returns	NULL Not NULL	Message not found Link to message
Comments	This function is one of the TDR section class functions. The functions in this class are fundamental to drawing and retrieving the section geometry. The errors defined by the mnemonic are defined in TDErrors.h.	

Nodes

Nodes are the links that form an actual network. Nodes link the sections together to form networks. In general, there are two type of nodes: junctions, and junctures. Junctions are links between sections that have driver selectable turnings, and junctures are links between sections that have no driver selectable turnings—no intersections.

Implementing Nodes in the relational database requires defining the following functions:

- TDRRetrieveNode
- TDRRetrieveNodeTurnings
- TDRRetrieveNodeStages
- TDRStoreNode
- TDRExistNode
- TDRSearchNode
- TDRReadNodeIds
- TDRReadJunctureIds
- TDRSearchJunction
- TDRReadJunctionIds
- TDRDeleteNode
- TDRListNodes
- TDRNbNodes

TDRRetrieveNode

Summary	Retrieves from the RDB the node and puts it in the cache provided. Supposes an unallocated storage. A incremental retrieval is performed: firstly, only the basic info is retrieved, the remaining will be retrieved when needed.	
Syntax	int TDRRetrieveNode(int idnode, TDICacheObject *cache)	
Arguments	Idnode *cache	Identifier for node Link to internal cache
Returns	0 MemAlloc NodeUnk	no errors memory allocation unknown node
Comments	This function is one of the TDR node class functions. The functions in this class are fundamental to linking sections in the geometry. The errors defined by the mnemonic are defined in TDErrors.h.	

TDRRetrieveNodeTurnings

Summary	Retrieves from the RDB the turnings within the node.
Syntax	int TDRRetrieveNodeTurnings(TDINode *node)

Arguments	*node	Link to node
Returns	0 MemAlloc	no errors memory allocation
Comments	This function is one of the TDR node class functions. The functions in this class are fundamental to linking sections in the geometry. The errors defined by the mnemonic are defined in TDErrors.h.	

TDRRetrieveNodeStages

Summary	Retrieves from the RDB the stages within the node.	
Syntax	int TDRRetrieveNodeStages(TDINode *node)	
Arguments	*node	Link to node
Returns	0 MemAlloc	no errors memory allocation
Comments	This function is one of the TDR node class functions. The functions in this class are fundamental to linking sections in the geometry. The errors defined by the mnemonic are defined in TDErrors.h.	

TDRStoreNode

Summary	Stores the contents of the node storage manipulation to the RDB. If it is a new node, supposes that it does not exist in the RDB. CAUTION: The view is not updated here!! Returns the entrance and exiting sections which have been modified.	
Syntax	int TDRStoreNode(TDINode *node_source, Bool id_changed, int *nbentrances_existed, int **entrances_existed, int *nbexits_existed, int **exits_existed, int *nbentrances_deleted, int **entrances_deleted, int *nbexits_deleted, int **exits_deleted, int *nbentrances_added, int **entrances_added, int *nbexits_added, int **exits_added)	
Arguments	*node_source id_changed *nbentrances_existed **entrances_existed *nbexits_existed **exits_existed	Link to internal node Was the ID of this node changed? Number of entrances Array of existed entrances Number of exits Array of exits

	<code>*nbenrances_deleted</code>	Number of entrances deleted
	<code>**entrances_deleted</code>	Array of entrances deleted
	<code>*nbexits_deleted</code>	Number of exits deleted
	<code>**exits_deleted</code>	Array of deleted exits
	<code>*nbenrances_added</code>	Number of entrances added
	<code>**entrances_added</code>	Array of entrances added
	<code>*nbexits_added</code>	Number of exits added
	<code>**exits_added</code>	Array of exits added
Returns	0	no errors
	ObjectNotOpen *	the required object is not open
	ObjectInteg - *	the object does not comply integrity requirements
	MemAlloc	memory allocation
Comments	This function is one of the TDR node class functions. The functions in this class are fundamental to linking sections in the geometry. The errors defined by the mnemonic are defined in TDErrors.h.	

TDRExistNode

Summary	Looks in the list of nodes in the RDB, looking for the node with identifier idnode. Returns TRUE if found, FALSE otherwise.	
Syntax	Bool TDRExistNode(int idnode)	
Arguments	idnode	Identifier of node
Returns	TRUE	Node found
	FALSE	Node not found
Comments	This function is one of the TDR node class functions. The functions in this class are fundamental to linking sections in the geometry. The errors defined by the mnemonic are defined in TDErrors.h.	

TDRReadNodeIds

Summary	Read the identifiers of the nodes in the whole network, and returns them in an array.	
Syntax	Int TDRReadNodeIds(int *maxids, int *nbids, int **ids, int *idcurrent)	
Arguments	<code>*maxids</code>	Maximum number of Ids to return
	<code>*nbids</code>	Number of IDS returned
	<code>**ids</code>	Array of IDS
	<code>*idcurrent</code>	Current ID
Returns	0	no errors
	MemAlloc	allocating memory

Comments This function is one of the TDR node class functions. The functions in this class are fundamental to linking sections in the geometry. The errors defined by the mnemonic are defined in TDErrors.h.

TDRSearchJuncture

Summary Looks in the list of nodes in the RDB, looking for the juncture with identifier idnode. A juncture is a node with node->type = 0. Returns a pointer to the juncture or NULL if not found. The *exists boolean indicates if the node exists without being a juncture. The previous node is also returned.

Syntax TDINode *TDRSearchJuncture(
int idnode,
Bool *exists,
TDINode **node_prev)

Arguments	idnode	Identifier of node
	*exists	True if not a juncture, False otherwise
	**node_prev	Link to previous node

Returns	NULL	Not found
	Not NULL	Pointer to the juncture

Comments This function is one of the TDR node class functions. The functions in this class are fundamental to linking sections in the geometry. The errors defined by the mnemonic are defined in TDErrors.h.

TDRReadJunctureIds

Summary Read the identifiers of the junctures (nodes) in the whole RDB, and returns them in an array.

Syntax Int TDRReadJunctureIds(
int *maxids,
int *nbids,
int **ids,
int *idcurrent)

Arguments	*maxids	Maximum number of JunctureIDS
	*nbids	Number of IDS returned
	**ids	Array of IDS
	*idcurrent	Current ID

Returns	0	no errors
	MemAlloc	allocating memory

Comments This function is one of the TDR node class functions. The functions in this class are fundamental to linking sections in the geometry. The errors defined by the mnemonic are defined in TDErrors.h.

TDRSearchJunction

Summary	Looks in the list of nodes in the RDB, looking for the junction with identifier idnode. A junction is a node with node->type = 1. Returns a pointer to the junction or NULL if not found. The *exists boolean indicates if the node exists without being a junction. The previous node is also returned.	
Syntax	TDINode *TDRSearchJunction(int idnode, Bool *exists, TDINode **node_prev)	
Arguments	idnode *exists **node_prev	Id of node TRUE if junction, FALSE otherwise Link to previous node
Returns	NULL Not NULL	Not found Pointer to the junction
Comments	This function is one of the TDR node class functions. The functions in this class are fundamental to linking sections in the geometry. The errors defined by the mnemonic are defined in TDErrors.h.	

TDRReadJunctionIds

Summary	Read the identifiers of the junctions (nodes) in the whole network, and returns them in an array.	
Syntax	Int TDRReadJunctionIds(int *maxids, int *nbids, int **ids, int *idcurrent)	
Arguments	*maxids *nbids **ids *idcurrent	Maximum number of IDS to return Number of junctions returned Array of returned junctions Current ID
Returns	0 MemAlloc	No junctions found allocating memory
Comments	This function is one of the TDR node class functions. The functions in this class are fundamental to linking sections in the geometry. The errors defined by the mnemonic are defined in TDErrors.h.	

TDRDeleteNode

Summary	Deletes the node from the RDB
Syntax	void TDRDeleteNode(int idnode)

Arguments	Idnode	Identifier for node
Returns	0 Not 0	no errors Errors
Comments	This function is one of the TDR node class functions. The functions in this class are fundamental to linking sections in the geometry. The errors defined by the mnemonic are defined in TDErrors.h.	

TDRListNodes

Summary	Lists the nodes in the RDB, with output to the screen.	
Syntax	Void TDRListNodes()	
Arguments	Void	No arguments
Returns	Void	No return value
Comments	This function is one of the TDR node class functions. The functions in this class are fundamental to linking sections in the geometry. The errors defined by the mnemonic are defined in TDErrors.h.	

TDRNbNodes

Summary	Counts the nodes in the RDB.	
Syntax	Int TDRNbNodes()	
Arguments	Void	No arguments
Returns	0 >0	No nodes found Number of nodes in RDB
Comments	This function is one of the TDR node class functions. The functions in this class are fundamental to linking sections in the geometry. The errors defined by the mnemonic are defined in TDErrors.h.	

Controllers

Controllers connect detectors and meters for traffic counting and controlled entry to freeways.

Implementing Controllers in the relational database requires defining the following functions:

- TDRRetrieveController
- TDRRetrieveControllerConDevices
- TDRStoreController
- TDRExistController
- TDRReadControllerIds
- TDRDeleteController
- TDRListControllers
- TDRNbControllers

TDRRetrieveController

Summary	Retrieves from the RDB the controller and puts it in the cache provided. Supposes an unallocated storage. A incremental retrieval is performed: firstly, only the basic info is retrieved, the remaining will be retrieved when needed.	
Syntax	int TDRRetrieveController(char idcontroller[TDMAXNAMLEN + 1], TDICacheObject *cache)	
Arguments	idcontroller *cache	Identifier for controller Link to internal cache
Returns	0 MemAlloc ControllerUnk	no errors memory allocation unknown controller
Comments	This function is one of the TDR controller class functions. The functions in this class control meters and detector. The errors defined by the mnemonic are defined in TDErrors.h.	

TDRRetrieveControllerConDevices

Summary	Retrieves from the RDB the connections of the controller currently in the controller storage manipulation.	
Syntax	int TDRRetrieveControllerConDevices(TDIController *controller)	
Arguments	*controller	Link to controller
Returns	0 MemAlloc	no errors memory allocation
Comments	This function is one of the TDR controller class functions. The functions in this class control meters and detector. The errors defined by the mnemonic are defined in TDErrors.h.	

TDRStoreController

Summary	Stores the contents of the controller storage manipulation to the RDB. If it is a new controller, supposes that it does not exist in the RDB. CAUTION: The view is not updated here!! Returns the connections to junctions, meterings, detectors, and VMSs that have been modified.
---------	---

Syntax

```

int TDRStoreController(
TDIController *controller_source,
Bool id_changed,
int *nbjunctions_existed,
TDIConDev **junctions_existed,
int *nbmeterings_existed,
TDIConDev **meterings_existed,
int *nbdetectors_existed,
TDIConDev **detectors_existed,
int *nbmessasigns_existed,
TDIConDev **messasigns_existed,
int *nbjunctions_deleted,
TDIConDev **junctions_deleted,
int *nbmeterings_deleted,
TDIConDev **meterings_deleted,
int *nbdetectors_deleted,
TDIConDev **detectors_deleted,
int *nbmessasigns_deleted,
TDIConDev **messasigns_deleted,
int *nbjunctions_added,
TDIConDev **junctions_added,
int *nbmeterings_added,
TDIConDev **meterings_added,
int *nbdetectors_added,
TDIConDev **detectors_added,
int *nbmessasigns_added,
TDIConDev **messasigns_added)

```

Arguments

```

*controller_source,
id_changed,
*nbjunctions_existed,
**junctions_existed,
*nbmeterings_existed,
**meterings_existed
*nbdetectors_existed
**detectors_existed,
*nbmessasigns_existed
**messasigns_existed
*nbjunctions_deleted
**junctions_deleted
*nbmeterings_deleted,
**meterings_deleted,
*nbdetectors_deleted,
**detectors_deleted
*nbmessasigns_deleted
**messasigns_deleted
*nbjunctions_added
**junctions_added
*nbmeterings_added,
**meterings_added
*nbdetectors_added
**detectors_added
*nbmessasigns_added
**messasigns_added

```

```

Link to controller object
Was the ID changed? TRUE/FALSE
Number of junctions existing
Array of junctions existing
Number of meterings existing
Array of meterings existing
Number of detectors existing
Array of detectors existing
Number of messages existing
Array of messages existing
Number of junctions deleted
Array of junctions deleted
Number of meterings deleted
Array of meterings deleted
Number of detectors deleted
Array of detectors deleted
Number of messaged deleted
Array of messages deleted
Number of junctions added
Array of junctions added
Number of meterings added
Array of meterings added
Number of detectors added
Array of detectors added
Number of messages added
Array of messages added

```

Returns

0

no errors

ObjectNotOpen	the required object is not open
ObjectInteg	the object does not comply integrity requirements
MemAlloc	memory allocation

Comments This function is one of the TDR controller class functions. The functions in this class control meters and detector. The errors defined by the mnemonic are defined in TDErrors.h.

TDRExistController

Summary Looks in the list of controllers in the RDB, looking for the controller with identifier idcontroller. Returns TRUE if found, FALSE otherwise.

Syntax Bool TDRExistController(
char idcontroller[TDMAXNAMLEN + 1])

Arguments idcontroller Identifier for controller

Returns TRUE Controller found
FALSE Not found

Comments This function is one of the TDR controller class functions. The functions in this class control meters and detector. The errors defined by the mnemonic are defined in TDErrors.h.

TDRReadControllerIds

Summary Read the identifiers of the controllers in the whole network, and returns them in an array.

Syntax Int TDRReadControllerIds(
int *maxids,
int *nbids,
char ***ids,
int *idcurrent)

Arguments *maxids Maximum number of controllers to return
*nbids Number of controllers in array
***ids Character array of controller identifiers
*idcurrent Current ID

Returns 0 no errors
MemAlloc allocating memory

Comments This function is one of the TDR controller class functions. The functions in this class control meters and detector. The errors defined by the mnemonic are defined in TDErrors.h.

TDRDeleteController

Summary	Lists the controllers in the RDB, with output to the screen.	
Syntax	Void TDRDeleteController(char *idcontroller)	
Arguments	*idcontroller	Identifier of controller
Returns	Void	No return values
Comments	This function is one of the TDR controller class functions. The functions in this class control meters and detector. The errors defined by the mnemonic are defined in TDErrors.h.	

TDRListControllers

Summary	Lists the controllers in the RDB, with output to the screen.	
Syntax	void TDRListControllers()	
Arguments	Void	No arguments
Returns	Void	No return values
Comments	This function is one of the TDR controller class functions. The functions in this class control meters and detector. The errors defined by the mnemonic are defined in TDErrors.h.	

TDRNbControllers

Summary	Counts the controllers in the RDB	
Syntax	Int TDRNbControllers()	
Arguments	Void	No arguments
Returns	>=0	Number of controllers in RDB
Comments	This function is one of the TDR controller class functions. The functions in this class control meters and detector. The errors defined by the mnemonic are defined in TDErrors.h.	

Centroids

Implementing Centroids in the relational database requires defining the following functions:

- TDRRetrieveCentroid
- TDRRetrieveCentroidCenConnections
- TDRStoreCentroid
- TDRExistCentroid

- TDRReadCentroidIds
- TDRReadObjectToCentroidConnections
- TDRDeleteCentroid
- TDRListCentroids
- TDRNbCentroids

TDRRetrieveCentroid

Summary	Retrieves from the RDB the centroid and puts it in the cache provided. Supposes an unallocated storage. A incremental retrieval is performed: firstly, only the basic info is retrieved, the remaining will be retrieved when needed.	
Syntax	int TDRRetrieveCentroid(int idcentroid, TDICacheObject *cache)	
Arguments	idcentroid *cache	Identifier for centroid. Link to internal cache
Returns	0 MemAlloc CentroidUnk	no errors memory allocation unknown centroid
Comments	This function is one of the TDR centroid class functions. The functions in this class help select routes. The errors defined by the mnemonic are defined in TDErrors.h.	

TDRRetrieveCentroidCenConnections

Summary	Retrieves from the RDB the connections of the centroid currently in the centroid storage manipulation.	
Syntax	int TDRRetrieveCentroidCenConnections(TDICentroid *centroid)	
Arguments	*centroid	Link to centroid storage manipulation
Returns	0 MemAlloc	no errors memory allocation
Comments	This function is one of the TDR centroid class functions. The functions in this class help select routes. The errors defined by the mnemonic are defined in TDErrors.h.	

TDRStoreCentroid

Summary	Stores the contents of the centroid storage manipulation to the RDB. If it is a new centroid, supposes that it does not exist in the RDB. CAUTION: The view is not updated here!!	
Syntax	int TDRStoreCentroid(TDICentroid *centroid_source, Bool id_changed)	

Arguments	*centroid_source id_changed	Link to Object to be stored TRUE if ID was change, otherwise FALSE
Returns	0 ObjectNotOpen ObjectInteg MemAlloc	no errors the required object is not the object does not comply integrity requirements memory allocation
Comments	This function is one of the TDR centroid class functions. The functions in this class help select routes. The errors defined by the mnemonic are defined in TDErrors.h.	

TDRExistCentroid

Summary	Looks in the list of centroids in the RDB, looking for the centroid with identifier idcentroid. Returns TRUE if found, FALSE otherwise.	
Syntax	Bool TDRExistCentroid(int idcentroid)	
Arguments	idcentroid	Identifier of centroid
Returns	TRUE FALSE	Centroid found Not found
Comments	This function is one of the TDR centroid class functions. The functions in this class help select routes. The errors defined by the mnemonic are defined in TDErrors.h.	

TDRReadCentroidIds

Summary	Read the identifiers of the centroids in the whole RDB, and returns them in an array.	
Syntax	Int TDRReadCentroidIds(int *maxids, int *nbids, int **ids, int *idcurrent)	
Arguments	*maxids *nbids **ids *idcurrent	Maximum number of Centroids to return Number of centroid IDS returned Array of centroid IDS Current ID
Returns	0 MemAlloc	no errors allocating memory
Comments	This function is one of the TDR centroid class functions. The functions in this class help select routes. The errors defined by the mnemonic are defined in TDErrors.h.	

TDRReadObjectToCentroidConnections

Summary	Returns an array of connection descriptors between this object and the centroids. The object has to be a section (type_object == 0) or a node (type_object == 1).	
Syntax	int TDRReadObjectToCentroidConnections(int type_object, int idobject, int *nbobconnections_ret, TDOBJECTTOCENTROID **TDOBconnections_ret)	
Arguments	type_object idobjec *nbobconnections_ret **TDOBconnections_ret	Section (=0), Node (=1) Identifier of object Number of connections Array of connections returned
Returns	0 ParamError NetNotOpen MemAlloc	no errors incorrect parameter there is no open network allocating memory
Comments	This function is one of the TDR centroid class functions. The functions in this class help select routes. The errors defined by the mnemonic are defined in TDErrors.h.	

TDRDeleteCentroid

Summary	Deletes this centroid from the RDB.	
Syntax	Void TDRDeleteCentroid(int idcentroid)	
Arguments	Idcentroid	Identifier for the centroid
Returns	0 Not equal 0	no errors Errors
Comments	This function is one of the TDR centroid class functions. The functions in this class help select routes. The errors defined by the mnemonic are defined in TDErrors.h.	

TDRListCentroids

Summary	Lists the centroids in the RDB, with output to the screen.	
Syntax	void TDRListCentroids()	
Arguments	Void	No arguments
Returns	Void	No return values
Comments	This function is one of the TDR centroid class functions. The functions in this class help select routes. The errors defined by the mnemonic are defined in TDErrors.h.	

TDRNbCentroids

Summary	Counts the centroids in the network.	
Syntax	Int TDRNbCentroids()	
Arguments	Void	No arguments
Returns	Number of centroids	Number of centroids in RDB
Comments	This function is one of the TDR centroid class functions. The functions in this class help select routes. The errors defined by the mnemonic are defined in TDErrors.h.	

Roguis

Implementing Roguis in the relational database requires defining the following functions:

- TDRRetrieveRogui
- TDRRetrieveRoguiRoute
- TDRRetrieveRoguiUpVMSs
- TDRStoreRogui
- TDRExistRogui
- TDRReadRoguiIds
- TDRDeleteRogui
- TDRListRoguis
- TDRNbRoguis

TDRRetrieveRogui

Summary	Retrieves from the RDB the rogui and puts it in the cache provided. Supposes an unallocated storage. A incremental retrieval is performed: firstly, only the basic info is retrieved, the remaining will be retrieved when needed.	
Syntax	int TDRRetrieveRogui(char idrogui[TDMAXNAMLEN + 1], TDICacheObject*cache)	
Arguments	idrogui *cache	Identifier for rogui Link to internal cache
Returns	0 MemAlloc RoguiUnk	no errors memory allocation unknown rogui
Comments	This function is one of the TDR rogue class functions. The errors defined by the mnemonic are defined in TDErrors.h.	

TDRRetrieveRoguiRoutes

Summary	Retrieves from the RDB the routes in the rogui that is currently in the storage manipulation.	
Syntax	int TDRRetrieveRoguiRoutes(TDIRogui *rogui)	
Arguments	*rogui	Link to internal rogui object
Returns	0 MemAlloc	no errors memory allocation
Comments	This function is one of the TDR rogue class functions. The errors defined by the mnemonic are defined in TDErrors.h.	

TDRRetrieveRoguiUpVMSs

Summary	Retrieves from the RDB the upstream VMSs in the rogui that is currently in the storage manipulation.	
Syntax	int TDRRetrieveRoguiUpVMSs(TDIRogui *rogui)	
Arguments	*rogui	Link to internal rogui object
Returns	0 MemAlloc	no errors memory allocation
Comments	This function is one of the TDR rogue class functions. The errors defined by the mnemonic are defined in TDErrors.h.	

TDRStoreRogui

Summary	Stores the contents of the rogui storage manipulation to the RDB. If it is a new rogui, supposes that it does not exist in the RDB. CAUTION: The view is not updated here!!	
Syntax	int TDRStoreRogui(TDIRogui *rogui_source, Bool id_changed)	
Arguments	*rogui_source id_changed	Link to internal rogui storage TRUE, if ID was changed, FALSE otherwise
Returns	0 ObjectNotOpen ObjectInteg MemAlloc	no errors the required object is not open the object does not comply integrity requirements memory allocation
Comments	This function is one of the TDR rogue class functions. The errors defined by the mnemonic are defined in TDErrors.h.	

TDRExistRogui

Summary	Looks in the list of roguis in the RDB, looking for the rogui with * identifier idrogui. Returns TRUE if found, FALSE otherwise.	
Syntax	Bool TDRExistRogui(char idrogui[TDMAXNAMLEN + 1])	
Arguments	idrogui	Identifier for the rogui
Returns	TRUE FALSE	Rogui found Rogui not found
Comments	This function is one of the TDR rogue class functions. The errors defined by the mnemonic are defined in TDErrors.h.	

TDRReadRoguids

Summary	Read the identifiers of the roguis in the whole RDB, and returns them in an array..	
Syntax	Int TDRReadRoguids(int *maxids, int *nbids, char ***ids, int *idcurrent)	
Arguments	*maxids *nbids ***ids *idcurrent	Maximum number of roguis to return Number of roguis returned Char array of roguis returned Current rogui
Returns	0 MemAlloc	no errors error allocating memory
Comments	This function is one of the TDR rogue class functions. The errors defined by the mnemonic are defined in TDErrors.h.	

TDRDeleteRogui

Summary	Deletes the rogui from the RDB.	
Syntax	Void TDRDeleteRogui(char *idrogui)	
Arguments	*idrogui	Identifier of rogui
Returns	Void	No return values
Comments	This function is one of the TDR rogue class functions. The errors defined by the mnemonic are defined in TDErrors.h.	

TDRListRoguis

Summary	Lists the roguis in the RDB, with output to the screen.	
Syntax	void TDRListRoguis();	
Arguments	Void	No arguments
Returns	Void	No return values
Comments	This function is one of the TDR rogue class functions. The errors defined by the mnemonic are defined in TDErrors.h.	

TDRNbRoguis

Summary	Counts the roguis in the RDB	
Syntax	int TDRNbRoguis();	
Arguments	Void	No arguments
Returns	Number of roguis	Number of roguis in the RDB
Comments	This function is one of the TDR rogue class functions. The errors defined by the mnemonic are defined in TDErrors.h.	

Routes

Implementing Routes in the relational database requires defining the following functions:

- TDRRetrieveRoute
- TDRStoreRoute
- TDRExistRoute
- TDRReadRouteIds
- TDRDeleteRoute
- TDRListRoute
- TDRNbRoutes

TDRRetrieveRoute

Summary	Retrieves from the RDB the route and puts it in the cache provided. Supposes an unallocated storage. A incremental retrieval is performed: firstly, only the basic info is retrieved, the remaining will be retrieved when needed.
Syntax	int TDRRetrieveRoute(char idroute[TDMAXNAMLEN + 1], TDICacheObject *cache)

Arguments	Idroute *cache	Identifier of route Link to internal storage
Returns	0 MemAlloc RouteUnk	no errors memory allocation unknown route
Comments	This function is one of the TDR route class functions. The errors defined by the mnemonic are defined in TDErrors.h.	

TDRStoreRoute

Summary	Stores the contents of the route storage manipulation to the RDB. If it is a new route, supposes that it does not exist in the RDB. CAUTION: The view is not updated here	
Syntax	int TDRStoreRoute(TDIRoute*route_source, Bool id_changed)	
Arguments	*route_source id_changed	Link to route TRUE if id changed, FALSE otherwise
Returns	0 ObjectNotOpen ObjectInteg MemAlloc	no errors the required object is not open the object does not comply integrity requirements memory allocation
Comments	This function is one of the TDR route class functions. The errors defined by the mnemonic are defined in TDErrors.h.	

TDRExistRoute

Summary	Looks in the list of routes in the RDB, looking for the route with identifier idroute. Returns TRUE if found, FALSE otherwise	
Syntax	Bool TDRExistRoute(char idroute[TDMAXNAMLEN + 1])	
Arguments	Idroute	Identifier of route
Returns	TRUE FALSE	Route exists Route not found
Comments	This function is one of the TDR route class functions. The errors defined by the mnemonic are defined in TDErrors.h.	

TDRReadRouteIds

Summary	Read the identifiers of the routes in the RDB, and returns them in an array.
---------	--

Syntax	Int TDRReadRouteIds(int *maxids, int *nbids, char ***ids, int *idcurrent)	
Arguments	*maxids *nbids ***ids *idcurrent	Maximum number of routes to be returned Number of routes returned Char array of returned routes Current ID
Returns	0 MemAlloc	no errors Error allocating memory
Comments	This function is one of the TDR route class functions. The errors defined by the mnemonic are defined in TDErrors.h.	

TDRDeleteRoute

Summary	Deletes the route from the RDB.	
Syntax	Void TDRDeleteRoute(char *idroute)	
Arguments	*idroute	Identifier for route
Returns	0 Not 0	no errors Errors
Comments	This function is one of the TDR route class functions. The errors defined by the mnemonic are defined in TDErrors.h.	

TDRListRoutes

Summary	Lists the routes in the RDB, with output to the screen.	
Syntax	void TDRListRoutes(void);	
Arguments	Void	Nor arguments
Returns	Void	No return values
Comments	This function is one of the TDR route class functions. The errors defined by the mnemonic are defined in TDErrors.h.	

TDRNbRoutes

Summary	Counts the routes in the RDB.	
Syntax	int TDRNbRoutes(void);	

Arguments	Void	No arguments
Returns	Number of routes	Number of routes found in RDB
Comments	This function is one of the TDR route class functions. The errors defined by the mnemonic are defined in TDErrors.h.	

Network

Implementing Network in the relational database requires defining the following functions:

- int TDROpenDatabase
- int TDRCloseDatabase
- int TDRPrepareNewNetwork
- int TDROpenNetwork
- Bool TDRGuessNetworkFormat
- int TDRReadNetworkDim
- int TDRCloseNetwork
- Int TDRWriteNetworkGlobals

TDROpenDatabase

Summary	Indicates, to the RDB part of the functions that this is the database should be used for this specific mode (geREAD or geWRITE). This function opens a ODBC connection to this database, so that next reads (Retrieve object execute faster).	
Syntax	int TDROpenDatabase(char name_full[TDMAXFILEPATH + 1], int mode);	
Arguments	name_full mode	Full path and name of database geREAD or geWRITE mode.
Returns	0	no errors
Comments	This function is one of the TDR network class functions. This class of functions forms the basis for all other calls to the traffic network. The errors defined by the mnemonic are defined in TDErrors.h.	

TDRCloseDatabase

Summary	Closes the database that was previously opened for this mode.	
Syntax	int TDRCloseDatabase(int mode);	
Arguments	mode	geREAD or geWRITE mode.
Returns	0	no errors

Comments This function is one of the TDR network class functions. This class of functions forms the basis for all other calls to the traffic network. The errors defined by the mnemonic are defined in TDErrors.h.

TDRPrepareNewNetwork

Summary Prepares a new network in RDB format. This means creating the database file and, inside it, creating the tables and relationships. name_full is the 'database name' + '.' + extension.

Syntax int TDRPrepareNewNetwork(
char newNameFull[TDMAXFILEPATH + 1]);

Arguments newNameFull Full path and name of new database

Returns 0 no errors
Not 0 Errors

Comments This function is one of the TDR network class functions. This class of functions forms the basis for all other calls to the traffic network. The errors defined by the mnemonic are defined in TDErrors.h.

TDROpenNetwork

Summary Opens the network with the given name, supposing RDB format.

Syntax int TDROpenNetwork (
char name_full[TDMAXFILEPATH + 1])

Arguments name_full Full path and name of new database

Returns 0 no errors
NetAIOpen there is already an open network
MemAlloc error allocating memory

Comments This function is one of the TDR network class functions. This class of functions forms the basis for all other calls to the traffic network. The errors defined by the mnemonic are defined in TDErrors.h.

TDRGuessNetworkFormat

Summary Looks if the network with this name is in RDB format.
NOTE: fileName is really the complete path (C:\...)

Syntax Bool TDRGuessNetworkFormat (
char fileName[TDMAXFILEPATH + 1])

Arguments fileName Full path and name of database

Returns TRUE if RDB format
FALSE Otherwise

Comments This function is one of the TDR network class functions. This class of functions forms the basis for all other calls to the traffic network. The errors defined by the mnemonic are defined in TDErrors.h.

TDRReadNetworkDim

Summary Reads the number of main objects in an existing RDB network. The network doesn't need to be open.

Syntax int TDRReadNetworkDim(
char name[TDMAXFILEPATH + 1],
TDNetDim*TDnetdim)

Arguments name Full path and name of database
*TDnetdim Link to where counts of objects are stored

Returns ParamError incorrect parameter
NetTypeUnsup unsupported type of network format
0 No errors

Comments This function is one of the TDR network class functions. This class of functions forms the basis for all other calls to the traffic network. The errors defined by the mnemonic are defined in TDErrors.h.

TDRCloseNetwork

Summary Frees the required data structures.

Syntax int TDRCloseNetwork(void);

Arguments Void No arguments

Returns 0 no errors
Not 0 Errors

Comments This function is one of the TDR network class functions. This class of functions forms the basis for all other calls to the traffic network. The errors defined by the mnemonic are defined in TDErrors.h.

TDRWriteNetworkGlobals

Summary Writes the file which contains the global variables, from the Network structure, or the view to the relational database.

Syntax Int TDRWriteNetworkGlobals(
Bool all_network)

Arguments all_network If TRUE do entire network, else do a view

Returns 0 no errors
FileOpen opening the file

FileWrite	writing the file
FileOpenAlready	opening an already opened file
FileClose	closing the file

Comments This function is one of the TDR network class functions. This class of functions forms the basis for all other calls to the traffic network. The errors defined by the mnemonic are defined in TDErrors.h.

Relational Database

The relational database functions use the ODBC 3.0 software development kit to build functions that will move and search for data in an ODBC compliant database. All the database calls made to the relational database (RDB) must use these functions to communicate with it. Although other functions may work, debugging is made much easier if all the database troubleshooting can be done through these functions. In addition, any function that has an error can be easily corrected here, in one place, saving much time.

- ODBCRegisterDataSource
- ODBCUnRegisterDataSource
- ODBC CopyDOSFile
- ODBCGetTextData
- ODBCGetIntData
- ODBCGetLongIntData
- ODBCGetFloatData
- ODBCGetDoubleData
- ODBCAllocStmt
- ODBCFreeStmt
- ODBCSelect
- ODBCFetch
- ODBCSelectFetch
- ODBCTableSize

ODBCRegisterDataSource

Summary This function registers the parameters being passed as an ODBC database in the ODBC32 administrator's table.

Syntax `int ODBCRegisterDataSource (
char szDSN[],
char szDefaultDir[],
char szDriverId[],
char szDriverName[],
char szDBQ[])`

Arguments	szDSN[] SzDefaultDir[], szDriverId[], SzDriverName[], szDBQ[]	Name for ODBC registry Directory of original DB ODBC driverid Access=25 "Microsoft Access Driver (*.mdb)" Original DB filename
-----------	---	--

Returns	0	no errors
---------	---	-----------

Not 0

Errors

Comments This function is one of the TDR database class functions. This class of functions forms the basis for all communication with the relational database. The errors defined by the mnemonic are defined in TDErrors.h.

ODBCUnRegisterDataSource

Summary This function removes the named ODBC registered database from the ODBC32 database registry.

Syntax int ODBCUnRegisterDataSource (
char szDSN[],
char szDefaultDir[],
char szDriverId[],
char szDriverName[],
char szDBQ[])

Arguments	szDSN[], SzDefaultDir[], szDriverId[], SzDriverName[], szDBQ[]	Name for ODBC registry Directory of original DB ODBC driverid Access=25 "Microsoft Access Driver (*.mdb)" Original DB filename
------------------	--	--

Returns	0 Not 0	no errors Errors
----------------	------------	---------------------

Comments This function is one of the TDR database class functions. This class of functions forms the basis for all communication with the relational database. The errors defined by the mnemonic are defined in TDErrors.h.

ODBCCopyDOSFile

Summary Copy the file in origin[] to dest[] using the DOS COPY command. This function allows certain files to be copied which will be needed by some of the other programs later.

Syntax Int ODBCCopyDOSFile (
char dest[],
const char origin[])

Arguments	dest[], origin[]	Path and name of where to copy file Path and name of file to make a copy of
------------------	---------------------	--

Returns	0 Not 0	no errors Errors (as defined by DOS)
----------------	------------	---

Comments This function is one of the ODBC database class functions. This class of functions forms the basis for all communication with the relational database. The errors defined by the mnemonic are defined in TDErrors.h.

ODBCGetTextData

Summary	Retrieve TEXT data from an ODBC database for a particular column. Assumes that an appropriate SELECT and FETCH have already been performed.	
Syntax	Int ODBCGetTextData(SQLHSTMT hstmt, Int col, Char* data, Int maxlength)	
Arguments	hstmt col *data maxlength	SQL statement handle col is an index to selected data data received maximum characters to accept
Returns	0 Not 0	no errors Errors
Comments	This function is one of the TDR database class functions. This class of functions forms the basis for all communication with the relational database. The errors defined by the mnemonic are defined in TDErrors.h.	

ODBCGetIntData

Summary	Retrieve INT data from an ODBC database for a particular column. Assumes that an appropriate SELECT and FETCH have already been performed.	
Syntax	Int ODBCGetIntData(SQLHSTMT hstmt, Int col, Int *data, Int minvalue, Int maxvalue)	
Arguments	hstmt col *data minvalue maxvalue	SQL statement handle col is an index to selected data data received minimum acceptable maximum acceptable
Returns	0 Not 0	no errors Errors
Comments	This function is one of the TDR database class functions. This class of functions forms the basis for all communication with the relational database. The errors defined by the mnemonic are defined in TDErrors.h.	

ODBCGetLongIntData

Summary	Retrieve LONG INT data from an ODBC database for a particular column. Assumes that an appropriate SELECT and FETCH have already been performed.	
Syntax	Int ODBCGetLongIntData (SQLHSTMT hstmt, Int col, long int *data, long int minvalue, long int maxvalue)	
Arguments	hstmt col *data minvalue maxvalue	SQL statement handle col is an index to selected data data received minimum acceptable maximum acceptable
Returns	0 Not 0	no errors Errors
Comments	This function is one of the TDR database class functions. This class of functions forms the basis for all communication with the relational database. The errors defined by the mnemonic are defined in TDErrors.h.	

ODBCGetFloatData

Summary	Retrieve FLOAT data from an ODBC database for a particular column. Assumes that an appropriate SELECT and FETCH have already been performed.	
Syntax	Int ODBCGetFloatData(SQLHSTMT hstmt, Int col, Float *data, Float minvalue, Float maxvalue)	
Arguments	hstmt col data minvalue maxvalue	SQL statement handle which column is the data in for this select data received minimum acceptable maximum acceptable
Returns	0 Not 0	no errors Errors
Comments	This function is one of the TDR database class functions. This class of functions forms the basis for all communication with the relational database. The errors defined by the mnemonic are defined in TDErrors.h.	

ODBCGetDoubleData

Summary	Retrieve DOUBLE data from an ODBC database for a particular column. Assumes that an appropriate SELECT and FETCH have already been performed.	
Syntax	Int ODBCGetDoubleData(SQLHSTMT hstmt, int col, double* data, double minvalue, double maxvalue)	
Arguments	hstmt col Data minvalue maxvalue	SQL statement handle which column is the data in for this select data received minimum acceptable maximum acceptable
Returns	0 Not 0	no errors Errors
Comments	This function is one of the TDR database class functions. This class of functions forms the basis for all communication with the relational database. The errors defined by the mnemonic are defined in TDErrors.h.	

ODBCAllocStmt

Summary	This function uses ODBC software development kit functions to allocate a statement handle given a connection handle.	
Syntax	Int ODBCAllocStmt (SQLHDBC FAR *hstmt, SQLHDBC hdbc)	
Arguments	hstmt hdbc	pointer to SQL stmt handle to allocate existing connection handle to link to
Returns	0 Not 0	no errors Errors
Comments	This function is one of the TDR database class functions. This class of functions forms the basis for all communication with the relational database. The errors defined by the mnemonic are defined in TDErrors.h.	

ODBCFreeStmt

Summary	Free up an ODBC statement handle
Syntax	Int ODBCFreeStmt (SQLHSTMT hstmt)

Arguments	hstmt	pointer to stmt to free
Returns	0 Not 0	no errors Errors
Comments	This function is one of the TDR database class functions. This class of functions forms the basis for all communication with the relational database. The errors defined by the mnemonic are defined in TDErrors.h.	

ODBCSelect

Summary	This function sends the SELECT statement that is contained in the variable "selectxx" for the ODBC database that is connected to the handle "hstmt".	
Syntax	Int ODBCSelect (SQLHSTMT hstmt, SQLCHAR *selectxx)	
Arguments	hstmt selectxx	pointer to SQL statement handle SQL SELECT data command
Returns	0 Not 0	no errors Errors
Comments	This function is one of the TDR database class functions. This class of functions forms the basis for all communication with the relational database. The errors defined by the mnemonic are defined in TDErrors.h.	

ODBCFetch

Summary	This function sends an ODBC Fetch command with statement handle to the ODBC database that is hstmt is connected to. The FETCH command requests that a row of data be sent back.	
Syntax	Int ODBCFetch (SQLHSTMT hstmt)	
Arguments	hstmt	SQL Fetch command
Returns	0 Not 0	no errors Errors
Comments	This function is one of the TDR database class functions. This class of functions forms the basis for all communication with the relational database. The errors defined by the mnemonic are defined in TDErrors.h.	

ODBCSelectFetch

Summary	This function combines both the ODBCSelect and the ODBCFetch functions.
---------	---

Syntax	Int ODBCSelectFetch (SQLHSTMT hstmt, SQLCHAR* selectxx)	
Arguments	hstmt selectxx	pointer to SQL statement handle SQL SELECT data command
Returns	0 Not 0	no errors Errors
Comments	This function is one of the TDR database class functions. This class of functions forms the basis for all communication with the relational database. The errors defined by the mnemonic are defined in TDErrors.h.	

ODBCTableSize

Summary	This function gets the number of records in the specified table in the database that hstmt is connected to.	
Syntax	Int ODBCTableSize(SQLHSTMT hstmt, const char* table, int* nrecords)	
Arguments	hstmt table nrecords	Pointer to SQL statement handle The table to count records from Number of records in the table (returned)
Returns	0 Not 0	no errors Errors
Comments	This function is one of the TDR database class functions. This class of functions forms the basis for all communication with the relational database. The errors defined by the mnemonic are defined in TDErrors.h.	

Appendix C

Sample of Traffic Volumes Provided by Mn/DOT

Appendix C: Sample of the traffic volumes provided by Mn/DOT

The data in the following table is a sampling of the data used. This is sample simulation data for detector number 8 using an ACCESS database to hold it. Data was provided by Mn/DOT . The columns are identified as:

- DetectorID: the Mn/DOT identification for this detector
- Time: the number of the 5 minute time period past midnight. For example, a time of 72 is $72*5=360$ minutes = 6 hours past midnight or 6 a.m.
- Date: the day of the month that the data was recorded
- Volume the number of vehicles in the last five (5) minutes. Note for simulation volume needs to be converted to vehicles/hour.
- Occupancy percentage of time the detector is occupied (range 0-100%)
- Status communication status of the detector. A value other than zero indicates a fault (range(-4 to 3)).
- Validity validity of the detector. A value other than zero indicates a fault (range: -3 to 4).

DetectorID	Time	Date	Volume	Occupancy	Status	Validity
8	72	12	17	1	0	0
8	73	12	21	1	0	0
8	74	12	16	1	0	0
8	75	12	23	2	0	0
8	76	12	30	2	0	0
8	77	12	22	2	0	0
8	78	12	34	2	0	0
8	79	12	34	2	0	0
8	80	12	32	3	0	0
8	81	12	40	3	0	0
8	82	12	39	3	0	0
8	83	12	39	3	0	0
8	84	12	37	4	0	0
8	85	12	22	2	0	0
8	86	12	28	2	0	0
8	87	12	32	3	0	0
8	88	12	39	4	0	0
8	89	12	33	3	0	0
8	90	12	41	3	0	0
8	91	12	38	3	0	0
8	92	12	50	4	0	0
8	93	12	37	3	0	0
8	94	12	44	4	0	0
8	95	12	44	4	0	0
8	96	12	39	3	0	0
8	97	12	48	4	0	0

DetectorID	Time	Date	Volume	Occupancy	Status	Validity
8	98	12	38	3	0	0
8	99	12	41	3	0	0
8	100	12	34	3	0	0
8	101	12	36	4	0	0
8	102	12	34	3	0	0
8	103	12	40	4	0	0
8	104	12	36	4	0	0
8	105	12	44	4	0	0
8	106	12	46	4	0	0
8	107	12	40	3	0	0
8	108	12	44	4	0	0
8	109	12	35	3	0	0
8	110	12	43	4	0	0
8	111	12	45	4	0	0
8	112	12	31	3	0	0
8	113	12	37	4	0	0
8	114	12	44	4	0	0
8	115	12	38	3	0	0
8	116	12	34	3	0	0
8	117	12	31	2	0	0
8	118	12	42	4	0	0
8	119	12	39	3	0	0
8	120	12	39	4	0	0
8	121	12	39	4	0	0
8	122	12	30	3	0	0
8	123	12	33	3	0	0
8	124	12	31	3	0	0
8	125	12	38	3	0	0
8	126	12	46	4	0	0
8	127	12	38	4	0	0
8	128	12	45	4	0	0
8	129	12	40	4	0	0
8	130	12	57	6	0	0
8	131	12	40	3	0	0
8	132	12	54	5	0	0
8	133	12	52	5	0	0
8	134	12	46	3	0	0
8	135	12	40	3	0	0
8	136	12	50	4	0	0
8	137	12	46	3	0	0
8	138	12	44	4	0	0
8	139	12	43	4	0	0
8	140	12	45	4	0	0
8	141	12	55	4	0	0
8	142	12	48	4	0	0
8	143	12	43	4	0	0
8	144	12	52	4	0	0
8	145	12	48	4	0	0

DetectorID	Time	Date	Volume	Occupancy	Status	Validity
8	146	12	60	6	0	0
8	147	12	48	4	0	0
8	148	12	42	4	0	0
8	149	12	46	4	0	0
8	150	12	50	4	0	0
8	151	12	50	4	0	0
8	152	12	53	5	0	0
8	153	12	47	4	0	0
8	154	12	46	4	0	0
8	155	12	55	5	0	0
8	156	12	53	5	0	0
8	157	12	51	5	0	0
8	158	12	52	4	0	0
8	159	12	43	3	0	0
8	160	12	58	5	0	0
8	161	12	46	4	0	0
8	162	12	39	4	0	0
8	163	12	51	5	0	0
8	163	13	56	4	0	0
8	164	12	56	5	0	0
8	164	13	60	5	0	0
8	165	12	51	5	0	0
8	165	13	59	5	0	0
8	166	12	50	4	0	0
8	166	13	42	4	0	0
8	167	12	38	3	0	0
8	167	13	56	4	0	0
8	168	12	51	4	0	0
8	168	13	46	5	0	0
8	169	12	64	6	0	0
8	169	13	66	5	0	0
8	170	12	54	5	0	0
8	170	13	57	5	0	0
8	171	12	71	6	0	0
8	171	13	65	6	0	0
8	172	12	57	5	0	0
8	172	13	57	5	0	0
8	173	12	58	5	0	0
8	173	13	46	3	0	0
8	174	12	52	4	0	0
8	174	13	47	3	0	0
8	175	12	65	5	0	0
8	175	13	58	4	0	0
8	176	12	68	5	0	0
8	176	13	65	5	0	0
8	177	12	64	6	0	0
8	177	13	64	6	0	0
8	178	12	61	5	0	0

DetectorID	Time	Date	Volume	Occupancy	Status	Validity
8	178	13	63	5	0	0
8	179	12	56	6	0	0
8	179	13	64	4	0	0
8	180	13	57	5	0	0
8	181	13	61	5	0	0
8	182	13	62	5	0	0
8	183	13	75	5	0	0
8	184	13	69	5	0	0
8	185	13	72	5	0	0
8	186	13	65	5	0	0
8	187	13	85	6	0	0
8	188	13	92	7	0	0
8	189	13	81	6	0	0
8	190	13	62	6	0	0
8	191	13	73	6	0	0
8	192	13	66	5	0	0
8	193	13	87	7	0	0
8	194	13	73	6	0	0
8	195	13	87	7	0	0
8	196	13	86	6	0	0
8	197	13	96	8	0	0
8	198	13	88	8	0	0
8	199	13	90	7	0	0
9	72	12	25	1	0	0
9	73	12	31	2	0	0
9	74	12	33	2	0	0

