TRANSIMS Input Editor System for IOC-1

B. W. Bush
Energy and Environmental Analysis Group
Los Alamos National Laboratory

20 June 1997

Abstract

The TRANSIMS input editor provides a means for managing the TRANSIMS database, editing road network data, and setting up scenarios for simulation via its graphical user interface (GUI). It separates the user from the lower-level layers of TRANSIMS software involved with data management. It has functions for manipulating data in the TRANSIMS database; for creating, importing, altering, validating, and viewing road network data; and for setting up simulation output tables. The input editor is integrated into the ArcView geographic information system (GIS) and the Oracle relational database. One can also customize or extend the input editor using the Avenue programming language.

I. Introduction 4
II. Design 6
A. Concepts 6
1. Oracle Database Tables 6
2. ArcView Files 6
B. Script Groups 6
1. Ined 6
2. InedDatabase 6
3. InedBuild 8
4. InedGeography 9
5. InedTables 10
6. InedValidate 11
7. InedNetwork 12
8. InedIntersection 12
III. Implementation 13
A. ArcView 13
B. Oracle 13
IV. Usage 14
A. Menus 14
B. Tutorials 19
1. Viewing Oracle TRANSIMS Data 19

2. Creating a New Data Table 20
3. Network Data Validation 21
4. Creating Network Maps (Shape Files) 24
5. Viewing Intersections 25
V. Future Work 26
VI. References 26
VII. APPENDIX: Source Code 27
A. General Scripts 27
1. Ined.DumpScripts.ave 27
2. Ined.LoadScripts.ave 28
3. Ined.Shutdown.ave 28
4. Ined.Startup.ave 29
B. Data Table Building Scripts 29
1. InedBuild.GetConnectivity.ave 29
2. InedBuild.GetLaneUse.ave 29
3. InedBuild.GetLink.ave 30
4. InedBuild.GetNode.ave 30
5. InedBuild.GetOutput.ave 30
6. InedBuild.GetOutputLink.ave 30
7. InedBuild.GetOutputNode.ave 31
8. InedBuild.GetParking.ave 31
9. InedBuild.GetPhasing.ave 31
10. InedBuild.GetPocket.ave 31
11. InedBuild.GetPrototype.ave 32
12. InedBuild.GetSignalized.ave 32
13. InedBuild.GetStudyLink.ave 32
14. InedBuild.GetTiming.ave 32
15. InedBuild.GetUnsignalized.ave 33
C. Database Access Scripts 33
1. InedDatabase.AddDependency.ave 33
2. InedDatabase.ChooseSource.ave 34
3. InedDatabase.ChooseSourceAndTable.ave 34
4. InedDatabase.ChooseTable.ave 34
5. InedDatabase.CloseDirectory.ave 35
6. InedDatabase.Commit.ave 35
7. InedDatabase.CreateOracleTable.ave 35
8. InedDatabase.CreateSource.ave 36
9. InedDatabase.DeleteOracleTable.ave 37
10. InedDatabase.DeleteSource.ave 37
11. InedDatabase.DeleteTable.ave 38
12. InedDatabase.ExecuteSQL.ave 38
13. InedDatabase.ExecuteSQLQuery.ave 39
14. InedDatabase.GetMaster.ave 39
15. InedDatabase.GetTable.ave 40
16. InedDatabase.GetTableName.ave 40

17. InedDatabase.OpenDirectory.ave 40
18. InedDatabase.RegisterTable.ave 41
19. InedDatabase.RemoveDependency.ave 41
20. InedDatabase.UnregisterTable.ave 42
21. InedDatabase.UpdateDependencies.ave 43
22. InedDatabase.UpdateSources.ave 43
23. InedDatabase.UpdateTables.ave 44
24. InedDatabase.ViewDependents.ave 45
25. InedDatabase.ViewDependentTables.ave 45
26. InedDatabase.ViewMasters.ave 45
27. InedDatabase.ViewSourceIndex.ave 46
28. InedDatabase.ViewTable.ave 46
29. InedDatabase.ViewTableIndex.ave 47
D. Geography Creation Scripts 47
1. InedGeography.CreateLanes.ave 47
2. InedGeography.CreateParkings.ave 51
3. InedGeography.MakeLinks.ave 52
4. InedGeography.MakeNodes.ave 53
5. InedGeography.MakeShapeFile.ave 53
E. Intersection Viewing Scripts 55
1. InedIntersection.NextPhase.ave 55
2. InedIntersection.PreviousPhase.ave 55
3. InedIntersection.RestoreLanes.ave 55
4. InedIntersection.SaveLanes.ave 56
5. InedIntersection.SetIntersection.ave 56
6. InedIntersection.SetPhase.ave 58
7. InedIntersection.SetProtections.ave 58
8. InedIntersection.ShowIncoming.ave 59
9. InedIntersection.ShowIntersection.ave 60
10. InedIntersection.ShowOutGoing.ave 60
11. InedIntersection.ShowPhase.ave 61
12. InedIntersection.ShowProtection.ave 61
13. InedNetwork.GenerateLines.ave 62
F. Network Data Table Scripts 63
1. InedNetwork.GeneratePoints.ave 63
2. InedNetwork.GenerateRectangles.ave 63
3. InedNetwork.MakeLanes.ave 65
4. InedNetwork.MakeLinks.ave 67
5. InedNetwork.MakeNodes.ave 68
6. InedNetwork.MakeParkings.ave 69
7. InedNetwork.WriteHOOPS.ave 70
G. Table Definition Scripts 78
1. InedTables.SetConnectivity.ave 78
2. InedTables.SetLanes.ave 78
3. InedTables.SetLinks.ave 78

4. InedTables.SetNodes.ave 78
5. InedTables.SetPhasing.ave 79
6. InedTables.SetPockets.ave 79
7. InedTables.SetSignalized.ave 79
8. InedTables.SetTable.ave 79
9. InedTables.SetTiming.ave 80
10. InedTables.SetUnsignalized.ave 80
H. Network Validation Scripts 80
1. InedValidate.CheckField.ave 80
2. InedValidate.CheckPositive.ave 81
3. InedValidate.CheckRange.ave 82
4. InedValidate.CheckValues.ave 82
5. InedValidate.ShowMessage.ave 83
6. InedValidate.ValidateConnectivity.ave 83
7. InedValidate.ValidateLinks.ave 86
8. InedValidate.ValidateNodes.ave 90
9. InedValidate.ValidatePhasing.ave 91
10. InedValidate.ValidatePockets.ave 95
11. InedValidate.Validate.Signalized.ave 98
12. InedValidate.ValidateTiming.ave 101
13. InedValidate.ValidateUnsignalized.ave 103

I. Introduction

The TRANSIMS input editor provides a means for managing the TRANSIMS database, editing road network data, and setting up scenarios for simulation via its graphical user interface (GUI). It separates the user from the lower-level layers of TRANSIMS software involved with data management. Figure 1 shows the position of the database subsystem within the TRANSIMS software architecture.

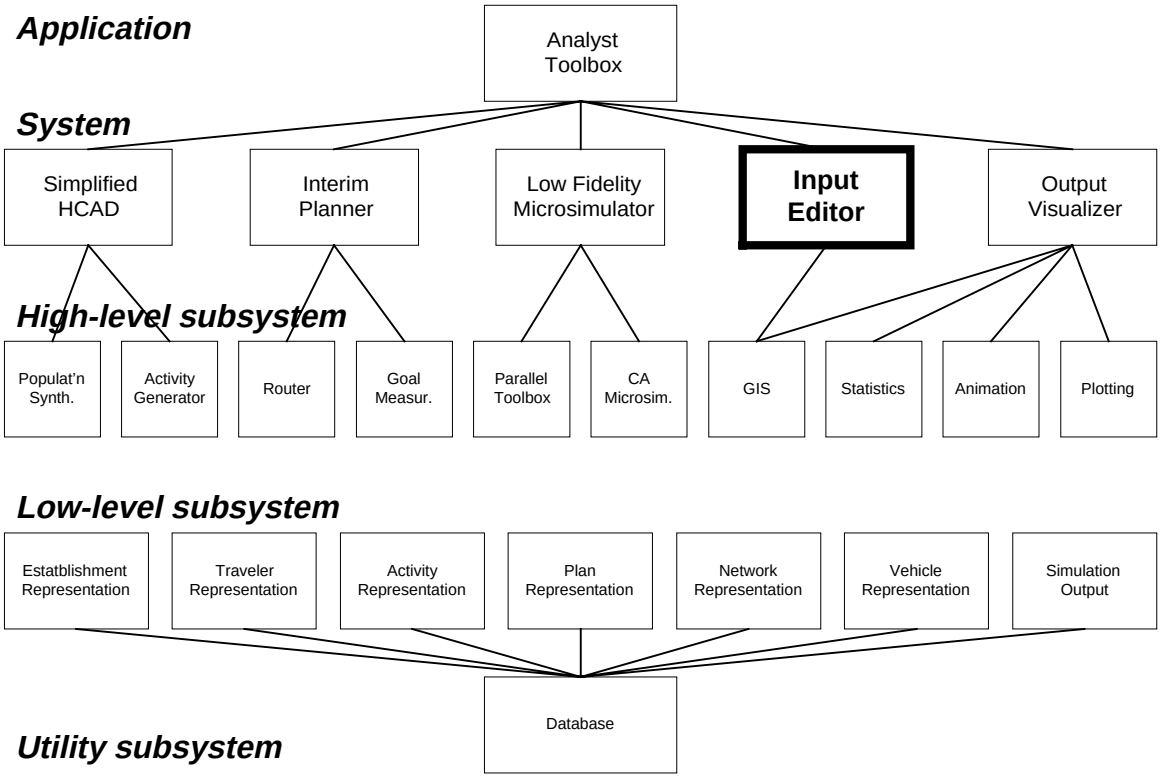

Figure 1. Location of the input editor system in the TRANSIMS software architecture.

The input editor provides a variety of data editing functions: it accepts standard file formats (dBASE, delimited text, and ArcInfo) and it allows form-, map- and spreadsheet-style editing of data tables. Functions exist for the generation of maps from tabular network data; such maps can be exported in several formats (ArcView shapes, ArcInfo coverages, vector graphics, and bitmapped graphics). There is also a graphical intersection viewer that allows a user to visualize the allowed lane-to-lane vehicle movements in the various phases of a traffic signal's operation. Additional tools support the validation of network data so that problems in network data (errors, anomalies, and inconsistencies) are quickly identified. The input editor database management functions permit the storage, retrieval, viewing, and deletion of TRANSIMS data tables in the Oracle database. Additional facilities allow the creation of simulation output specification tables.

The input editor is integrated into the ArcView geographic information system (GIS) and the Oracle relational database. It seamlessly connects to the ArcInfo product and other relational databases, too. One can also customize or extend the input editor using the Avenue programming language.

The body of this document outlines the design, implementation, and usage of the system. The appendices contain the complete Avenue source code for the system.

II. Design

A. Concepts

1. Oracle Database Tables

The primary repository for TRANSIMS data is an Oracle database managed by the TRANSIMS database subsystem. One can edit these data either through SQL commands sent to the Oracle server or by importing the data into ArcView, editing it within ArcView, and then exporting the data back to Oracle.

2. ArcView Files

ArcView primarily uses the Shape and dBASE file formats for data storage. The shape file format is a portable open standard for files that contain geographic data. Shape files are typically paired with dBASE files containing the non-geographic attributes of the geographic objects in the shape file. The dBASE format is a portable industry standard for files that contain tabular data. Both formats can easily be imported or exported between a variety of commercial products.

B. Script Groups

The input editor system has scripts for a manipulating, editing, and viewing TRANSIMS data. We discuss the scripts by functional group below.

1. Ined

These scripts perform general input editor functions.

`Ined.DumpScripts`
: This script dumps all of the user scripts to text files.

`Ined.LoadScripts`
: This script loads all of the user scripts from text files.

`Ined.Shutdown`
: This script shuts down the input editor.

`Ined.Startup`
: This script starts up the input editor.

2. InedDatabase

These scripts perform TRANSIMS database subsystem functions.

`InedDatabase.AddDependency`
: This script lets the user make one table dependent upon another.

`InedDatabase.ChooseSource`
: This script prompts the user for a data source. The argument is the prompter caption.

`InedDatabase.ChooseSourceAndTable`

This script prompts the user for a data source and then a data table within the source. The argument is the prompter caption.

`InedDatabase.ChooseTable`

This script prompts the user for a data table in the specified data source (second argument). The first argument is the prompter caption.

`InedDatabase.CloseDirectory`

This script closes a connection to the current data directory in Oracle.

`InedDatabase.Commit`

This script commits the Oracle database.

`InedDatabase.CreateOracleTable`

This script puts an ArcView table into Oracle.

`InedDatabase.CreateSource`

This script lets the user create a data source.

`InedDatabase.DeleteOracleTable`

This script lets the user delete an Oracle table.

`InedDatabase.DeleteSource`

This script lets the user delete a data source.

`InedDatabase.DeleteTable`

This script lets the user delete an Oracle table.

`InedDatabase.ExecuteSQL`

This script executes a user-input SQL statement.

`InedDatabase.ExecuteSQLQuery`

This script executes a user-input SQL query.

`InedDatabase.GetMaster`

This script gets the master table name for a table. The first parameter is the master source, the second parameter is the dependent source, and the third parameter is the dependent table.

`InedDatabase.GetTable`

This script returns a `VTab` for the specified data source and table. The first argument is the source name and the second argument is the table name.

`InedDatabase.GetTableName`

This script returns the database table name for a table. The first parameter is the source, and the second is the table.

`InedDatabase.OpenDirectory`
This script opens a connection to a data directory in Oracle.

`InedDatabase.RegisterTable`
This script lets the user register a data table.

`InedDatabase.RemoveDependency`
This script lets the user remove a dependency between data tables.

`InedDatabase.UnregisterTable`
This script lets the user unregister a data table.

`InedDatabase.UpdateDependencies`
This script updates the data dependencies from the database.

`InedDatabase.UpdateSources`
This script updates the dictionary of data sources for the current data directory.

`InedDatabase.UpdateTables`
This script updates the dictionary of data tables for the specified data sources.

`InedDatabase.ViewDependents`
This script lets the user view data table dependencies.

`InedDatabase.ViewDependentTables`
This script displays the dependent tables.

`InedDatabase.ViewMasters`
This script displays the master tables for a given table.

`InedDatabase.ViewSourceIndex`
This script displays the source index.

`InedDatabase.ViewTable`
This script prompts the user for a data source and table, and opens a view of the table.

`InedDatabase.ViewTableIndex`
This script displays the table index.

3. InedBuild

These scripts supply functions for creating new network data tables from prototypes.

`InedBuild.GetConnectivity`
This script gets a prototype for the lane connectivity table.

`InedBuild.GetLaneUse`
This script gets a prototype for the lane use table.

`InedBuild.GetLink`
This script gets a prototype for the link table.

`InedBuild.GetNode`
This script gets a prototype for the node table.

`InedBuild.GetOutput`
This script gets a prototype for the output specification table.

`InedBuild.GetOutputLink`
This script gets a prototype for the output link specification table.

`InedBuild.GetOutputNode`
This script gets a prototype for the output node specification table.

`InedBuild.GetParking`
This script gets a prototype for the parking table.

`InedBuild.GetPhasing`
This script gets a prototype for the phasing plan table.

`InedBuild.GetPocket`
This script gets a prototype for the pocket lane table.

`InedBuild.GetPrototype`
This script gets a prototype for a table. The first argument is the source name and the second argument is the prototype name.

`InedBuild.GetSignalized`
This script gets a prototype for the signalized node table.

`InedBuild.GetStudyLink`
This script gets a prototype for the study area link table.

`InedBuild.GetTiming`
This script gets a prototype for the timing plan table.

`InedBuild.GetUnsignalized`
This script gets a prototype for the unsignalized node table.

4. InedGeography

These scripts create network shape files from dBASE files.

`InedGeography.CreateLanes`
This script creates a lane table with geographic attributes.

`InedGeography.CreateParkings`
This script makes a parking shape table.

`InedGeography.MakeLinks`
This script adds geographic attributes to a link table.

`InedGeography.MakeNodes`
This script adds geographic attributes to a node table.

`InedGeography.MakeShapeFile`
This script makes a dBASE file into a shape file.

5. InedTables

These scripts select network subsystem tables.

`InedTables.SetConnectivity`
This script sets the project's lane connectivity table.

`InedTables.SetLanes`
This script sets the project's lane table.

`InedTables.SetLinks`
This script sets the project's link table.

`InedTables.SetNodes`
This script sets the project's node table.

`InedTables.SetPhasing`
This script sets the project's phasing plan table.

`InedTables.SetPockets`
This script sets the project's pocket lane table.

`InedTables.SetSignalized`
This script sets the project's signalized node table.

`InedTables.SetTable`
This script sets any table in the project. The first argument is the requested document name.

`InedTables.SetTiming`
This script sets the project's timing plan table.

`InedTables.SetUnsignalized`
This script sets the project's unsignalized node table.

6. InedValidate

These scripts perform validation of network subsystem tables.

`InedValidate.CheckField`

This script checks the presence and type of a field. The first argument is the `VTab`, the second is the field name, the third is the log file, and the fourth is a flag indicating a numeric field. Return the field.

`InedValidate.CheckPositive`

This script checks that a field value is positive. The first argument is the `VTab`, the second is the field, and the third is the log file. Return whether the check failed.

`InedValidate.CheckRange`

This script checks that a field value is within a specified range. The first argument is the `VTab`, the second is the field, the third is the log file, the fourth is the lowest legal value, and the fifth is the highest legal value. Return whether the check failed.

`InedValidate.CheckValues`

This script checks that a field value is in a set of values. The first argument is the `VTab`, the second is the field, the third is the log file, and the fourth is the list of legal values. Returns whether the check failed.

`InedValidate.ShowMessage`

This script shows a message on a log file and in the status bar. The first argument is the message string and the second is the log file.

`InedValidate.ValidateConnectivity`

This script checks to see that a lane connectivity table is valid.

`InedValidate.ValidateLinks`

This script checks to see that a link table is valid.

`InedValidate.ValidateNodes`

This script checks to see that a node table is valid.

`InedValidate.ValidatePhasing`

This script checks to see that a phasing plan table is valid.

`InedValidate.ValidatePockets`

This script checks to see that a pocket lane table is valid.

`InedValidate.Validate.Signalized`

This script checks to see that a signalized node table is valid.

`InedValidate.ValidateTiming`

This script checks to see that a timing plan table is valid.

`InedValidate.ValidateUnsignalized`

This script checks to see that an unsignalized node table is valid.

7. InedNetwork

These scripts create shape files for database subsystem tables.

`InedNetwork.GenerateLines`

This script makes lines for a table. The first parameter is the input `VTab`, the second parameter is the output `FTab`, the third parameter is a list of fields for which to copy attribute data, the fourth parameter is the name of the `x0` field, the fifth parameter is the name of the `y0` field, the sixth parameter is the name of the `x1` field, and the seventh parameter is the name of the `y1` field.

`InedNetwork.GeneratePoints`

This script makes points for a table. The first parameter is the input `VTab`, the second parameter is the output `FTab`, the third parameter is a list of fields for which to copy attribute data, the fourth parameter is the name of the `x` field, and the fifth parameter is the name of the `y` field.

`InedNetwork.GenerateRectangles`

This script makes rectangles for a table. The first parameter is the input `VTab`, the second parameter is the output `FTab`, the third parameter is a list of fields for which to copy attribute data, the fourth parameter is the name of the `x0` field, the fifth parameter is the name of the `y0` field, the sixth parameter is the name of the `x1` field, the seventh parameter is the name of the `y1` field, etc.

`InedNetwork.MakeLanes`

This script generates lane shapes from data tables.

`InedNetwork.MakeLinks`

This script makes a link shape table.

`InedNetwork.MakeNodes`

This script makes a node shape table.

`InedNetwork.MakeParkings`

This script makes a parking shape table.

`InedNetwork.WriteHOOPS`

This script writes a HOOPS metafile for a network.

8. InedIntersection

These scripts perform rudimentary intersection and signal viewing functions.

`InedIntersection.NextPhase`
This script sets the phase for analysis to the next phase.

`InedIntersection.PreviousPhase`
This script sets the phase for analysis to the previous phase.

`InedIntersection.RestoreLanes`
This script restores a lane selection.

`InedIntersection.SaveLanes`
This script saves a lane selection.

`InedIntersection.SetIntersection`
This script sets the intersection for analysis.

`InedIntersection.SetPhase`
This script sets the phase for analysis.

`InedIntersection.SetProtections`
This script sets the protections and signs for analysis.

`InedIntersection.ShowIncoming`
This script shows the incoming movements for the selected lanes.

`InedIntersection.ShowIntersection`
This script shows the intersection for analysis.

`InedIntersection.ShowOutGoing`
This script shows the outgoing movements for the selected lanes.

`InedIntersection.ShowPhase`
This script shows the phase for analysis.

`InedIntersection.ShowProtection`
This script shows the protections and signs for analysis.

III. Implementation

A. ArcView

The input editor uses the ArcView product [ES 96a] as its primary platform. All of the scripts are in the Avenue programming language (an object-oriented scripting language used by ArcView) [ES 96b].

B. Oracle

The input editor accesses TRANSIMS data via ArcView's services for connection to Oracle databases [KL 95]. Some of the Avenue scripts execute Oracle SQL commands [La 92].

IV. Usage

A. Menus

The input editor provides several custom menus (see Table 1), in addition to the standard ArcView menus, that allow access to the input editor functions. Table 2 through Table 8 below list the commands and describe their functions.

Table 1. Input editor menus.

Menu	***Description***	***Availability***
Database	Performs TRANSIMS database subsystem functions	when project window is active
Build	Creates empty network or output specification tables from templates	when project window is active
Geography	Generates maps from network tables in dBASE format	when project window is active
Tables	Identifies network tables	when project window is active
Validation	Validates network tables in dBASE format	when project window is active
Network	Generates maps from network tables in Oracle	when view window is active
Intersection	Views allowed movements at an intersection	when view window is active

Table 2. Database menu items (available when a project window is active).

Menu Item	*Description*	*Avenue Script*
Open Directory	Opens the data directory	InedDatabase.OpenDirectory
Close Directory	Closes the data directory	InedDatabase.CloseDirectory
Create Source	Creates a new data source	InedDatabase.CreateSource
Delete Source	Deletes a data source	InedDatabase.DeleteSource
Update Sources	Rereads the data source metadata from the data directory	InedDatabase.UpdateSources
Source Index	Views the source index	InedDatabase.ViewSourceIndex
Register Table	Alters the database subsystem metadata so that a table already in the Oracle database is a database subsystem data table	InedDatabase.RegisterTable
Unregister Table	Alters the database subsystem metadata so that an Oracle database table is no longer a database subsystem data table	InedDatabase.UnregisterTable
View Table	Views a data table	InedDatabase.ViewTable
Table Index	Views the table index	InedDatabase.ViewTableIndex
Add Dependency	Makes one data table depend upon another data table	InedDatabase.AddDependency
Remove Dependency	Makes a data table no longer dependent upon another	InedDatabase.RemoveDependency
View Masters	Views the list of data tables that are masters for a given data table	InedDatabase.ViewMasters
View Dependents	Views the list of data tables which are dependents for a given data table	InedDatabase.ViewDependents
Update Dependencies	Rereads the data table dependencies metadata from the data directory	InedDatabase.UpdateDependencies
Table Dependencies	Views the table dependencies	InedDatabase.ViewDependent Tables
Create Oracle Table	Makes an ArcView table into an Oracle one	InedDatabase.CreateOracleTable
Delete Oracle Table	Deletes a table from the Oracle database	InedDatabase.DeleteOracleTable
Execute SQL	Executes an SQL statement in Oracle	InedDatabase.ExecuteSQL
Execute SQL Query	Executes an SQL query in Oracle	InedDatabase.ExecuteSQLQuery
Commit	Commits the database	InedDatabase.Commit

Table 3. Build menu items (available when a project window is active).

Menu Item	*Description*	*Avenue Script*
Node	Creates a prototype node table	InedBuild.GetNode
Link	Creates a prototype link table	InedBuild.GetLink
Pocket Lane	Creates a prototype pocket lane table	InedBuild.GetPocket
Parking	Creates a prototype parking table	InedBuild.GetParking
Lane Use	Creates a prototype lane use table	InedBuild.GetLaneUse
Connectivity	Creates a prototype lane connectivity table	InedBuild.GetConnectivity
Unsignalized Node	Creates a prototype unsignalized node table	InedBuild.GetUnsignalized
Signalized Node	Creates a prototype signalized node table	InedBuild.GetSignalized
Phasing Plan	Creates a prototype phasing plan table	InedBuild.GetPhasing
Timing Plan	Creates a prototype timing plan table	InedBuild.GetTiming
Output Specification	Creates a prototype output specification table	InedBuild.GetOutput
Output Node Specification	Creates a prototype output node specification table	InedBuild.GetOutputNode
Output Link Specification	Creates a prototype output link specification	InedBuild.GetOutputLink
Study Area Link	Creates a prototype study area link table	InedBuild.GetStudyLink

Table 4. Geography menu items (available when a project window is active).

Menu Item	*Description*	*Avenue Script*
Make Shape File	Creates a shape file from a dBASE file	InedGeography.MakeShapeFile
Add Nodes	Adds points for nodes to a shape file	InedGeography.MakeNodes
Add Links	Adds lines for links to a shape file	InedGeography.MakeLinks
Create Lanes	Creates a shape file with lane rectangles	InedGeography.CreateLanes
Create Parkings	Creates a shape file with parking points	InedGeography.CreateParkings

Table 5. Tables menu items (available when a project window is active).

Menu Item	*Description*	*Avenue Script*
Set Nodes	Specifies the node table to use for analyses	InedTables.SetNodes
Set Links	Specifies the link table to use for analyses	InedTables.SetLinks
Set Pocket Lanes	Specifies the pocket lane table to use for analyses	InedTables.SetPockets
Set Lanes	Specifies the lane table to use for analyses	InedTables.SetLanes
Set Connectivity	Specifies the lane connectivity table to use for analyses	InedTables.SetConnectivity
Set Unsignalized Nodes	Specifies the unsignalized node table to use for analyses	InedTables.SetUnsignalized
Set Timing Plans	Specifies the timing plan table to use for analyses	InedTables.SetTiming
Set Signalized Nodes	Specifies the signalized node table to use for analyses	InedTables.SetSignalized
Set Phasing Plans	Specifies the phasing plan table to use for analyses	InedTables.SetPhasing

Table 6. Validate menu items (available when a project window is active).

Menu Item	*Description*	*Corresponding Avenue Script*
Nodes	Validates the contents of the node table	InedValidate.ValidateNodes
Links	Validates the contents of the link table	InedValidate.ValidateLinks
Pocket Lanes	Validates the contents of the pocket lane table	InedValidate.ValidatePockets
Connectivity	Validates the contents of the lane connectivity table	InedValidate.Validate Connectivity
Unsignalized Nodes	Validates the contents of the unsignalized node table	InedValidate.Validate Unsignalized
Timing Plans	Validates the contents of the timing plan table	InedValidate.ValidateTiming
Signalized Nodes	Validates the contents of the signalized node table	InedValidate.Validate Signalized
Phasing Plans	Validates the contents of the phasing plan table	InedValidate.ValidatePhasing

Table 7. Network menu items (available when a view window is active).

Menu Item	***Description***	***Avenue Script***
Shape Nodes	Creates a shape file for a Node data table	InedNetwork.MakeNodes
Shape Links	Creates a shape file for a Link data table	InedNetwork.MakeLinks
Shape Parkings	Creates a shape file for a Parking data table	InedNetwork.MakeParkings
Shape Lanes	Creates a shape file showing the lanes in a network	InedNetwork.MakeLanes
Export to HOOPS	Exports the network in HOOPS metafile format	InedNetwork.WriteHOOPS

Table 8. Intersection menu items (available when a view window is active).

Menu Item	*Description*	*Avenue Script*
Set Intersection	Makes the currently selected node the current intersection	InedIntersection.SetIntersection
Show Intersection	Shows the current intersection	InedIntersection.ShowIntersection
Set Protections/Signs	Specifies the protection and sign codes to be viewed for the current intersection	InedIntersection.SetProtections
Show Protections/Signs	Shows the protection and sign codes to be viewed for the current intersection	InedIntersection.ShowProtections
Set Phase	Specifies the phase to be viewed for the current intersection	InedIntersection.SetPhase
Show Phase	Shows the phase to be viewed for the current intersection	InedIntersection.ShowPhase
Next Phase	Specifies the phase to be viewed for the current intersection as one of the phases immediately after the current phase	InedIntersection.NextPhase
Previous Phase	Specifies the phase to be viewed for the current intersection as one of the phases immediately before the current phase	InedIntersection.PreviousPhase
Show Incoming Lanes	Selects the lanes from which movements are allowed to the currently selected lanes	InedIntersection.ShowIncoming
Show Outgoing Lanes	Selects the lanes to which movements are allowed from the currently selected lanes	InedIntersection.ShowOutgoing
Save Lane Selection	Remembers the currently selected lanes	InedIntersection.SaveLanes
Restore Lane Selection	Restores the lanes selection to the last one remembered	InedIntersection.RestoreLanes

B. Tutorials

1. Viewing Oracle TRANSIMS Data

The following discussion outlines how to access TRANSIMS data residing in Oracle.

Connect to an Oracle Database:

Before proceeding further, make sure the Project window is active. Use the Database | Open Directory menu command to connect to one of the data directories in the TRANSIMS database subsystem. This sets the database used by the rest of the input editor commands.

View the list of data sources:
Choosing the Database | Source Index menu command will open an ArcView table window listing all of the available data sources in the current TRANSIMS data directory. Each source has an annotation.

View the list of data tables:
Choosing the Database | Table Index menu command will open an ArcView table window listing all of the available data tables in the current TRANSIMS data directory. Each table belongs to a source and has an annotation.

View a data table:
Choosing the Database | View Table menu command opens a dialog box that allows one to select the data source to which the data table belongs. Select a `Node` data source, for example—another dialog box will appear listing all of the node data tables in the current TRANSIMS data directory. Select one of the tables so that the input editor will retrieve the data and display it in an ArcView table window.

View the list of tables dependent on a data table:
Choosing the Database | View Dependents menu command opens a dialog box that allows one to select the data source to which the data table belongs. Select a `Node` data source, for example—another dialog box will appear listing all of the node data tables in the current TRANSIMS data directory. Select one of the tables so that the input editor will display in an ArcView table window a list of the other TRANSIMS data tables dependent on the selected table.

2. Creating a New Data Table

The following discussion outlines how to create a new TRANSIMS data table and put it into the Oracle database.

Connect to an Oracle Database:
Before proceeding further, make sure the Project window is active. Use Database | Open Directory menu command to connect to one of the data directories in the TRANSIMS database subsystem. This sets the database used by the rest of the input editor commands.

Create an empty table:
Choosing the Build | Nodes menu command opens a dialog box that lets one select the file name for a new dBASE file containing a TRANSIMS network node table. Enter a name into this dialog box so that the input editor will display an ArcView table window containing an empty node table.

Add data to the table:
Make sure that the new node table is the active window. Use the Table | Start Editing menu command to begin editing the table and add some rows to the table using the Edit | Add record menu command. After filling the cells of the table, use the Table | Stop Editing menu command to finish editing the table.

Transfer the data into Oracle:
Make sure that the Project window is active. Choose the Database | Create Oracle Table menu command and select the dBASE file for the new table in the dialog box that appears. Next assign the table a name in the Oracle database using the dialog box that appears.

Register the table as a TRANSIMS data table:
Choose the Database | Register Table menu command and select the Node data source in the dialog box that appears. Next fill in the metadata for the new TRANSIMS data table; be sure to fill-in the Oracle table name exactly as previously chosen.

3. Network Data Validation

The procedure for constructing shape files (i.e., maps) from network data and validating network data is outlined below. Before starting the validation process, all of the data must be in ArcView dBASE table format. The steps below must be performed in sequence.

Validate the node table:

- Add the node table to the project.
- Set the node table using the Tables | Set Nodes* menu command.
- Run the validation script using the Validate | Nodes menu command. The log file will contain a description of the problems found and the corresponding records of the tables involved will be selected. The following checks are performed:
  - The field names and types (character vs. numeric) are correct.
  - The IDs have legal values.
  - The IDs are unique.
- Fix any errors that were detected, and then re-run the validation script.
- Remove the node table from the project.

Add geography to the nodes:

- Use the Geography | Make Shape File menu command to convert the node dBASE file into a node shape file (of a different name); choose Point for the type of shape. The shape file that is created will contain all of the same data that the original dBASE file contained; this new node shape file will be used from here onwards.
- Use the Geography | Add Nodes menu command to create the points for the node shape file.
- Create a view in the project, and add the node shape file as a theme to this view.
- Use the Theme | Table menu command to add the attribute table for the node shape file to the project.
- Set the node table again using the Tables | Set Nodes menu command.
- It is probably a good idea to validate the nodes again at this point (to ensure that the data is still valid).

---

* Note that the Geography, Tables, and Validate drop-down menus are only accessible when the Project window is active.

Validate the link table:

- Add the link table to the project.
- Set the link table using the Tables | Set Links menu command.
- Run the validation script using the Validate | Links menu command. The log file will contain a description of the problems found and the corresponding records of the tables involved will be selected. The following checks are performed:
  - The field names and types (character vs. numeric) are correct.
  - The IDs have legal values.
  - The IDs are unique.
  - The nodes at the endpoints exist.
  - All nodes have at least one incoming and one outgoing link.
  - The values of all of the fields are valid.
- Fix any errors that were detected, and then re-run the validation script.
- Remove the link table from the project.

Add geography to the links:

- Use the Geography | Make Shape File menu command to convert the link dBASE file into a link shape file (of a different name); choose Polyline for the type of shape. The shape file that is created will contain all of the same data that the original dBASE file contained; this new link shape file will be used from here onwards.
- Use the Geography | Add Links menu command to create the lines for the link shape file.
- Add the link shape file as a theme to the view created earlier.
- Use the Theme | Table menu command to add the attribute table for the link shape file to the project.
- Set the link table again using the Tables | Set Links menu command.
- It is probably a good idea to validate the links again at this point (to ensure that the data is still valid).

Validate the pocket lane table:

- Add the pocket lane table to the project.
- Set the pocket lane table using the Tables | Set Pocket Lanes menu command.
- Run the validation script using the Validate | Pocket Lanes menu command. The log file will contain a description of the problems found and the corresponding records of the tables involved will be selected. The following checks are performed:
  - The field names and types (character vs. numeric) are correct.
  - The IDs have legal values.
  - The IDs are unique.
  - The styles are valid.
  - The node and link references are correct.
  - The lane number is that of a valid pocket lane.
  - The offset and length are consistent with the setbacks and length of the link.
  - All of the pocket lanes specified in the link table are present.

- Fix any errors that were detected, and then re-run the validation script.

Create the lane table:

- Use the Geography | Create Lanes menu command to create a lane shape file corresponding to the data in the node, link, and pocket lane tables.
- Add the lane theme to the view.
- Add the lane shape file as a theme to the view created earlier.
- Use the Theme | Table menu command to add the attribute table for the lane shape file to the project.
- Set the lane table using the Tables | Set Lanes menu command.

Validate the lane connectivity table:

- Add the lane connectivity table to the project.
- Set the lane connectivity table using the Tables | Set Connectivity menu command.
- Run the validation script using the Validate | Connectivity menu command. The log file will contain a description of the problems found and the corresponding records of the tables involved will be selected. The following checks are performed:
  - The field names and types (character vs. numeric) are correct.
  - The node, link, and lane references are correct.
  - Each lane has at least one incoming and at least one outgoing connection.
- Fix any errors that were detected, and then re-run the validation script.

Validate the unsignalized node table:

- Add the unsignalized node table to the project.
- Set the unsignalized node table using the Tables | Set Unsignalized Nodes menu command.
- Run the validation script using the Validate | Unsignalized Nodes menu command. The log file will contain a description of the problems found and the corresponding records of the tables involved will be selected. The following checks are performed:
  - The field names and types (character vs. numeric) are correct.
  - The signs are valid.
  - The node and link references are correct.
  - Each incoming link is controlled.
- Fix any errors that were detected, and then re-run the validation script.

Validate the timing plan table:

- Add the timing plan table to the project.
- Set the timing plan table using the Tables | Set Timing Plans menu command.
- Run the validation script using the Validate | Timing Plans menu command. The log file will contain a description of the problems found and the corresponding records of the tables involved will be selected. The following checks are performed:
  - The field names and types (character vs. numeric) are correct.
  - The plan and the phase values are legal.
  - The (plan, phase) pairs are duplicated.

  - The time values are legal and consistent.
  - The phase sequence references existent phases.
- Fix any errors that were detected, and then re-run the validation script.

Validate the signalized node table:

- Add the signalized node table to the project.
- Set the signalized node table using the Tables | Set Signalized Nodes menu command.
- Run the validation script using the Validate | Signalized Nodes menu command. The log file will contain a description of the problems found and the corresponding records of the tables involved will be selected. The following checks are performed:
  - The field names and types (character vs. numeric) are correct.
  - The types are valid.
  - The node references are correct.
  - Each node has at most one signalized control (or zero if there is an unsignalized control at the node).
  - The plan references are correct.
  - Each node has either a signalized or unsignalized control.
  - All plans are used.
  - The start times are valid.
- Fix any errors that were detected, and then re-run the validation script.

Validate the phasing plan table:

- Add the phasing plan table to the project.
- Set the phasing plan table using the Tables | Set Phasing Plans menu command.
- Run the validation script using the Validate | Phasing Plans menu command. The log file will contain a description of the problems found and the corresponding records of the tables involved will be selected. The following checks are performed:
  - The field names and types (character vs. numeric) are correct.
  - The protections are valid.
  - The plan, phase, node, and link references are correct.
  - Each incoming and outgoing link is controlled.
- Fix any errors that were detected, and then re-run the validation script.

4. Creating Network Maps (Shape Files)

The following discussion outlines the steps to create maps of a road network from TRANSIMS data in the Oracle database.

Create a new view:

Use the ArcView project window controls to create a new view window.

Create the node shape file:

Choose the Network | Shape Nodes menu command and select a node table from the dialog box that appears. Choose a file name for the shape file when the next dialog box

appears. After the shape file is created, one has the opportunity to display it in the ArcView view window.

Create the link shape file:
Choose the Network | Shape Links menu command and select a link table from the dialog box that appears. You may also have to select a node table if there is insufficient table dependency information for the link table in the TRANSIMS data directory for the input editor to determine the appropriate node table automatically. Choose a file name for the shape file when prompted. After the shape file is created, one has the opportunity to display it in the ArcView view window.

Create the parking shape file:
Choose the Network | Shape Parkings menu command and select a parking table from the dialog box that appears. You may also have to select a link and possibly a node table. Choose a file name for the shape file when prompted. After the shape file is created, one has the opportunity to display it in the ArcView view window.

Create the lane shape file:
Choose the Network | Shape Lanes menu command and select a pocket lane table from the dialog box that appears. You may also have to select a link and possibly a node table. Choose a file name for the shape file when prompted, and then choose the width for lanes. After the shape file is created, one has the opportunity to display it in the ArcView view window.

5. Viewing Intersections

Before starting the viewing, all of the network data tables must be loaded as specified in the network data validation tutorial above. A view containing themes named Nodes, Links, and Lanes must be the active document.

Select the intersection:
In the view, select the node containing the intersection of interest. This may be a signalized or an unsignalized intersection. Use the Intersection | Set Intersection* menu command to set this selected node as the current intersection to be viewed.

Select the phase:
Use the Intersection | Set Phase menu command to choose the phase to be viewed at this intersection. (The Intersection | Next Phase and Intersection | Previous Phase menu commands can be used to move to adjacent phases.)

* Note that the Intersection drop-down menu is only accessible when the View window is active.

Select the protections or signs of interest:
If only certain protections or signs are of interest, use the Intersection | Set Protections/Signs menu command to specify which should be considered.

Select the lanes of interest and view the allowed movements:
In the view, select one or more lanes for which the allowed movements are to be viewed. If these are incoming lanes, use the Intersection | Show Outgoing Lanes to see to which lanes these lead; if they are outgoing lanes, use the Intersection | Show Incoming Lanes menu command to see which lanes lead to them.

For convenience, menu commands are available for saving (Intersection | Save Lane Selection) and restoring (Intersection | Restore Lane Selection) the selection state of the lanes.

V. Future Work

Future work planned for the TRANSIMS input editor system will focus on migrating its functionality from Avenue to C++. This will reduce its dependence on commercial products such as ArcView and Oracle, and will also improve its performance.

The validation code currently has the following known limitations.

- The user is not able to select which validations to perform on a given table.
- The user cannot specify a subset of table records to be validated.
- The case where a pocket lane is a combination of pull-outs, merges, and turns is not adequately accounted for.
- Duplicate lane connectivity records are not detected.
- Duplicate phasing plan records are not detected.

The intersection viewing code currently has the following known limitations.

- The mechanism for setting the protection and sign codes is clumsy.
- It would be useful to color-code the links to show more data about connectivity, and to draw graphic symbols on the view indicating the allowed movements.
- It would be useful to see all of the connectivity, regardless of phase.
- Invalid input data is not detected—it is assumed that the data is valid.
- A timing diagram is not displayed.
- It is inconvenient not to have buttons for the commonly used commands.

VI. References

[ES 96a] *ArcView GIS: The Geographic Information System for Everyone*, (Readlands, California: Environmental Systems Research Institute, 1996).

[ES 96b] *Avenue: Customization and Application Development for ArcView*, (Readlands, California: Environmental Systems Research Institute, 1996).

[KL 95] G. Koch and K. Loney, *ORACLE: The Complete Reference*, Third Edition, (Berkeley, California: McGraw-Hill, 1995).

[La 92] R. F. van der Lans, *The SQL Guide to Oracle*, (Reading, Massachusetts: Addison-Wesley, 1992).

VII. APPENDIX: Source Code

This appendix contains the complete Avenue source code for the input editor system.

A. General Scripts

1. Ined.DumpScripts.ave

```
'   Project: TRANSIMS
'   Subsystem: Input Editor
'   $RCSfile: Ined.DumpScripts.ave,v $
'   $Revision: 1.0 $
'   $Date: 1996/04/25 18:20:38 $
'   $State: Rel $
'   $Author: bwb $
'   U.S. Government Copyright 1995
'   All rights reserved


'   This script dumps all of the user scripts to text files.


saveDir = av.GetProject.GetWorkDir
saveDirStr = av.GetProject.GetWorkDir.GetFullName

if (File.Exists(saveDir).Not) then
    MsgBox.Error("Working directory:" ++ saveDirStr ++ "does not exist", "")
    exit
end

theScriptList = List.Make

for each d in av.GetProject.GetDocs
    if (d.Is(SEd)) then
      theScriptList.Add(d)
    end
end

ok = MsgBox.YesNo("Are you sure you want to write all scripts to:" + NL + saveDirStr +
"?", "Save Scripts", true)
if (ok.Not) then
    exit
end


for each scriptName in theScriptList

    theFile = scriptName.AsString'.Substitute(".", "_")

    outFile = FileName.Merge(saveDirStr, theFile + ".ave")
    'outFile.SetExtension("ave")
    if (outFile.IsFile) then
        ok = MsgBox.YesNoCancel("Overwrite" ++ outFile.GetFullName + "?", "Save Scripts",
true)
        if (ok = Nil) then
            exit
        elseif (Not ok) then
            continue
        end
    end

    lf = LineFile.Make(outFile, #FILE_PERM_WRITE)
    if (lf <> nil) then
        lf.WriteElt(scriptName.GetScript.AsString)
        lf.Close
    else
        MsgBox.Warning("Unable to write:"++outFile.GetFullName,"")
    end
```

```
end

av.ShowMsg("Scripts written to:" ++ saveDirStr)
```

2. Ined.LoadScripts.ave

```
'   Project: TRANSIMS
'   Subsystem: Input Editor
'   $RCSfile: Ined.LoadScripts.ave,v $
'   $Revision: 1.0 $
'   $Date: 1996/04/25 18:20:38 $
'   $State: Rel $
'   $Author: bwb $
'   U.S. Government Copyright 1995
'   All rights reserved


'   This script loads all of the user scripts from text files.


saveDir = av.GetProject.GetWorkDir
saveDirStr = av.GetProject.GetWorkDir.GetFullName

if (File.Exists(saveDir).Not) then
    MsgBox.Error("Working directory:" ++ saveDirStr ++ "does not exist", "")
    exit
end

theScriptList = List.Make

for each d in av.GetProject.GetDocs
    if (d.Is(SEd)) then
      theScriptList.Add(d)
    end
end

ok = MsgBox.YesNo("Are you sure you want to read all scripts from:" + NL + saveDirStr +
"?", "Restore Scripts", true)
if (ok.Not) then
    exit
end


for each scriptName in theScriptList

    theFile = scriptName.AsString'.Substitute(".", "_")

    outFile = FileName.Merge(saveDirStr, theFile + ".ave")
    'outFile.SetExtension("ave")
    if (outFile.IsFile.Not) then
        continue
    end

    lf = TextFile.Make(outFile, #FILE_PERM_READ)
    if (lf <> nil) then
        source = ""
        source = source + lf.Read(1000000)
        scriptName.SetSource(source)
        lf.Close
        scriptName.Compile
    else
        MsgBox.Warning("Unable to read:"++outFile.GetFullName,"")
    end
end

av.ShowMsg("Scripts read from:" ++ saveDirStr)
```

3. Ined.Shutdown.ave

```
'   Project: TRANSIMS
'   Subsystem: Input Editor
'   $RCSfile: Ined.Shutdown.ave,v $
'   $Revision: 1.0 $
'   $Date: 1996/04/25 18:20:38 $
'   $State: Rel $
```

```
'   $Author: bwb $
'   U.S. Government Copyright 1995
'   All rights reserved


'   This script shuts down the input editor.


'   Close data directory.
av.Run("InedDatabase.CloseDirectory", NIL)

'   Free global variables.
av.ClearGlobals
```

4. Ined.Startup.ave

```
'   Project: TRANSIMS
'   Subsystem: Input Editor
'   $RCSfile: Ined.Startup.ave,v $
'   $Revision: 1.2 $
'   $Date: 1996/09/04 18:00:02 $
'   $State: Rel $
'   $Author: bwb $
'   U.S. Government Copyright 1995
'   All rights reserved


'   This script starts up the input editor.


'   Create global variables.
_directory = SQLCon.Find("Oracle")
_sources = Dictionary.Make(50)
_tables = Dictionary.Make(50)
_dependencies = List.Make

'   Open data directory, if necessary.
av.Run("InedDatabase.OpenDirectory", NIL)
```

B. Data Table Building Scripts

1. InedBuild.GetConnectivity.ave

```
'   $RCSfile: InedBuild.GetConnectivity.ave,v $
'   $Revision: 1.0 $
'   $Date: 1996/04/25 18:20:38 $
'   $State: Rel $
'   $Author: bwb $
'   U.S. Government Copyright 1995
'   All rights reserved


'   This script gets a prototype for the lane connectivity table.


'   Set the node table.
av.Run("InedBuild.GetPrototype", {"Lane Connectivity", "Empty Lane Connectivity Table"})
```

2. InedBuild.GetLaneUse.ave

```
'   $RCSfile: InedBuild.GetLaneUse.ave,v $
'   $Revision: 1.0 $
'   $Date: 1996/04/25 18:20:38 $
'   $State: Rel $
'   $Author: bwb $
'   U.S. Government Copyright 1995
'   All rights reserved


'   This script gets a prototype for the lane use table.


'   Set the node table.
```

```
av.Run("InedBuild.GetPrototype", {"Lane Use", "Empty Lane Use Table"})
```

3. InedBuild.GetLink.ave

```
'   Project: TRANSIMS
'   Subsystem: Input Editor
'   $RCSfile: InedBuild.GetLink.ave,v $
'   $Revision: 1.0 $
'   $Date: 1996/04/25 18:20:38 $
'   $State: Rel $
'   $Author: bwb $
'   U.S. Government Copyright 1995
'   All rights reserved


'   This script gets a prototype for the link table.


'   Set the node table.
av.Run("InedBuild.GetPrototype", {"Link", "Empty Link Table"})
```

4. InedBuild.GetNode.ave

```
'   Project: TRANSIMS
'   Subsystem: Input Editor
'   $RCSfile: InedBuild.GetNode.ave,v $
'   $Revision: 1.0 $
'   $Date: 1996/04/25 18:20:38 $
'   $State: Rel $
'   $Author: bwb $
'   U.S. Government Copyright 1995
'   All rights reserved


'   This script gets a prototype for the node table.


'   Set the node table.
av.Run("InedBuild.GetPrototype", {"Node", "Empty Node Table"})
```

5. InedBuild.GetOutput.ave

```
'   Project: TRANSIMS
'   Subsystem: Input Editor
'   $RCSfile: InedBuild.GetOutput.ave,v $
'   $Revision: 1.0 $
'   $Date: 1996/04/25 18:20:38 $
'   $State: Rel $
'   $Author: bwb $
'   U.S. Government Copyright 1995
'   All rights reserved


'   This script gets a prototype for the output specification table.


'   Set the node table.
av.Run("InedBuild.GetPrototype", {"Output Specification", "Empty Output Specification
Table"})
```

6. InedBuild.GetOutputLink.ave

```
'   Project: TRANSIMS
'   Subsystem: Input Editor
'   $RCSfile: InedBuild.GetOutputLink.ave,v $
'   $Revision: 1.0 $
'   $Date: 1996/04/25 18:20:38 $
'   $State: Rel $
'   $Author: bwb $
'   U.S. Government Copyright 1995
'   All rights reserved


'   This script gets a prototype for the output link specification table.
```

```
'   Set the node table.
av.Run("InedBuild.GetPrototype", {"Output Link Specification", "Empty Output Link
Specification Table"})
```

7. InedBuild.GetOutputNode.ave

```
'   Project: TRANSIMS
'   Subsystem: Input Editor
'   $RCSfile: InedBuild.GetOutputNode.ave,v $
'   $Revision: 1.0 $
'   $Date: 1996/04/25 18:20:38 $
'   $State: Rel $
'   $Author: bwb $
'   U.S. Government Copyright 1995
'   All rights reserved


'   This script gets a prototype for the output node specification table.


'   Set the node table.
av.Run("InedBuild.GetPrototype", {"Output Node Specification", "Empty Output Node
Specification Table"})
```

8. InedBuild.GetParking.ave

```
'   $RCSfile: InedBuild.GetParking.ave,v $
'   $Revision: 1.0 $
'   $Date: 1996/04/25 18:20:38 $
'   $State: Rel $
'   $Author: bwb $
'   U.S. Government Copyright 1995
'   All rights reserved


'   This script gets a prototype for the parking table.


'   Set the node table.
av.Run("InedBuild.GetPrototype", {"Parking", "Empty Parking Table"})
```

9. InedBuild.GetPhasing.ave

```
'   Project: TRANSIMS
'   Subsystem: Input Editor
'   $RCSfile: InedBuild.GetPhasing.ave,v $
'   $Revision: 1.0 $
'   $Date: 1996/04/25 18:20:38 $
'   $State: Rel $
'   $Author: bwb $
'   U.S. Government Copyright 1995
'   All rights reserved


'   This script gets a prototype for the phasing plan table.


'   Set the node table.
av.Run("InedBuild.GetPrototype", {"Phasing Plan", "Empty Phasing Plan Table"})
```

10. InedBuild.GetPocket.ave

```
'   Project: TRANSIMS
'   Subsystem: Input Editor
'   $RCSfile: InedBuild.GetPocket.ave,v $
'   $Revision: 1.0 $
'   $Date: 1996/04/25 18:20:38 $
'   $State: Rel $
'   $Author: bwb $
'   U.S. Government Copyright 1995
'   All rights reserved
```

```
'   This script gets a prototype for the pocket lane table.


'   Set the node table.
av.Run("InedBuild.GetPrototype", {"Pocket Lane", "Empty Pocket Lane Table"})
```

11. InedBuild.GetPrototype.ave

```
'   Project: TRANSIMS
'   Subsystem: Input Editor
'   $RCSfile: InedBuild.GetPrototype.ave,v $
'   $Revision: 1.0 $
'   $Date: 1996/04/25 18:20:38 $
'   $State: Rel $
'   $Author: bwb $
'   U.S. Government Copyright 1995
'   All rights reserved


'   This script gets a prototype for a table.  The first argument is the source name and
the second argument is the prototype name.


'   Get the output file.
answer = FileDialog.Put((SELF.Get(0) + ".dbf").AsFileName, "*.dbf", "dBASE Table")
if (answer = NIL) then
    exit
end

'   Get the table.
document = Table.Make(av.Run("InedDatabase.GetTable", SELF).Export(answer, dBASE, FALSE))
document.SetName(answer.GetBaseName.AsString)
document.GetWin.Open
```

12. InedBuild.GetSignalized.ave

```
'   $RCSfile: InedBuild.GetSignalized.ave,v $
'   $Revision: 1.0 $
'   $Date: 1996/04/25 18:20:38 $
'   $State: Rel $
'   $Author: bwb $
'   U.S. Government Copyright 1995
'   All rights reserved


'   This script gets a prototype for the signalized node table.


'   Set the node table.
av.Run("InedBuild.GetPrototype", {"Signalized Node", "Empty Signalized Node Table"})
```

13. InedBuild.GetStudyLink.ave

```
'   Project: TRANSIMS
'   Subsystem: Input Editor
'   $RCSfile: InedBuild.GetStudyLink.ave,v $
'   $Revision: 1.0 $
'   $Date: 1996/04/25 18:20:38 $
'   $State: Rel $
'   $Author: bwb $
'   U.S. Government Copyright 1995
'   All rights reserved


'   This script gets a prototype for the study area link table.


'   Set the node table.
av.Run("InedBuild.GetPrototype", {"Study Area Link", "Empty Study Area Link Table"})
```

14. InedBuild.GetTiming.ave

```
'   Project: TRANSIMS
```

```
'   Subsystem: Input Editor
'   $RCSfile: InedBuild.GetTiming.ave,v $
'   $Revision: 1.0 $
'   $Date: 1996/04/25 18:20:38 $
'   $State: Rel $
'   $Author: bwb $
'   U.S. Government Copyright 1995
'   All rights reserved


'   This script gets a prototype for the timing plan table.


'   Set the node table.
av.Run("InedBuild.GetPrototype", {"Timing Plan", "Empty Timing Plan Table"})
```

15. InedBuild.GetUnsignalized.ave

```
'   $RCSfile: InedBuild.GetUnsignalized.ave,v $
'   $Revision: 1.0 $
'   $Date: 1996/04/25 18:20:38 $
'   $State: Rel $
'   $Author: bwb $
'   U.S. Government Copyright 1995
'   All rights reserved


'   This script gets a prototype for the unsignalized node table.


'   Set the node table.
av.Run("InedBuild.GetPrototype", {"Unsignalized Node", "Empty Unsignalized Node Table"})
```

C. Database Access Scripts

1. InedDatabase.AddDependency.ave

```
'   Project: TRANSIMS
'   Subsystem: Input Editor
'   $RCSfile: InedDatabase.AddDependency.ave,v $
'   $Revision: 1.0 $
'   $Date: 1996/04/25 18:20:38 $
'   $State: Rel $
'   $Author: bwb $
'   U.S. Government Copyright 1995
'   All rights reserved


'   This script lets the user make one table dependent upon another.


'   Choose tables.
first = av.Run("InedDatabase.ChooseSourceAndTable", "Master Table")
if (first = NIL) then
    exit
end
second = av.Run("InedDatabase.ChooseSourceAndTable", "Dependent Table")
if (second = NIL) then
    exit
end
if (MsgBox.YesNo("Make """ + second.Get(1) + """ dependent on """ + first.Get(1) + """?",
"Add Data Dependency", TRUE).not) then
    exit
end

'   Add the dependencies.
if (_directory.ExecuteSQL("INSERT INTO DEPENDENT_TABLES VALUES('" +
av.Run("InedDatabase.GetTableName", first) + "', '" + av.Run("InedDatabase.GetTableName",
second) + "')")) then
    MsgBox.Info("Success", "Add Data Dependency")
else
    MsgBox.Warning("Failure", "Add Data Dependency")
end
```

```
'   Update the dependencies.
av.Run("InedDatabase.UpdateDependencies", NIL)
```

2. InedDatabase.ChooseSource.ave

```
'   Project: TRANSIMS
'   Subsystem: Input Editor
'   $RCSfile: InedDatabase.ChooseSource.ave,v $
'   $Revision: 1.0 $
'   $Date'
'   $State: Rel $
'   $Author: bwb $
'   U.S. Government Copyright 1995
'   All rights reserved


'   This script prompts the user for a data source.  The argument is the prompter caption.


'   Prompt the user for a source.
sources = _sources.ReturnKeys
sources.Sort(true)
return MsgBox.ListAsString(sources, "Choose a data source:", SELF)
```

3. InedDatabase.ChooseSourceAndTable.ave

```
'   Project: TRANSIMS
'   Subsystem: Input Editor
'   $RCSfile: InedDatabase.ChooseSourceAndTable.ave,v $
'   $Revision: 1.0 $
'   $Date: 1996/04/25 18:20:38 $
'   $State: Rel $
'   $Author: bwb $
'   U.S. Government Copyright 1995
'   All rights reserved


'   This script prompts the user for a data source and then a data table within the
source.  The argument is the prompter caption.


'   Prompt the user for a source.
sourceName = av.Run("InedDatabase.ChooseSource", SELF)
if (sourceName = NIL) then
    return NIL
end

'   Prompt the user for a table in the source.
tableName = av.Run("InedDatabase.ChooseTable", {SELF, sourceName})
if (tableName = NIL) then
    return NIL
end

'   Return the result.
return {sourceName, tableName}
```

4. InedDatabase.ChooseTable.ave

```
'   Project: TRANSIMS
'   Subsystem: Input Editor
'   $RCSfile: InedDatabase.ChooseTable.ave,v $
'   $Revision: 1.0 $
'   $Date: 1996/04/25 18:20:38 $
'   $State: Rel $
'   $Author: bwb $
'   U.S. Government Copyright 1995
'   All rights reserved


'   This script prompts the user for a data table in the specified data source (second
argument).  The first argument is the prompter caption.
```

```
'   Prompt the user for a table.
tables = _tables.Get(SELF.Get(1)).ReturnKeys
tables.Sort(true)
return MsgBox.ListAsString(tables, "Choose a data table in the " + SELF.Get(1) + " data
source:", SELF.Get(0))
```

5. InedDatabase.CloseDirectory.ave

```
'   Project: TRANSIMS
'   Subsystem: Input Editor
'   $RCSfile: InedDatabase.CloseDirectory.ave,v $
'   $Revision: 1.0 $
'   $Date: 1996/04/25 18:20:38 $
'   $State: Rel $
'   $Author: bwb $
'   U.S. Government Copyright 1995
'   All rights reserved


'   This script closes a connection to the current data directory in Oracle.


'   Make sure we are logged in already.
if (_directory.IsLogin.not) then
    exit
end

'   Log out.
_directory.Logout
```

6. InedDatabase.Commit.ave

```
'   Project: TRANSIMS
'   Subsystem: Input Editor
'   $RCSfile: InedDatabase.Commit.ave,v $
'   $Revision: 1.0 $
'   $Date: 1996/04/25 18:20:38 $
'   $State: Rel $
'   $Author: bwb $
'   U.S. Government Copyright 1995
'   All rights reserved


'   This script commits the database.


'   Execute the statement.
if (_directory.ExecuteSQL("commit")) then
    MsgBox.Info("Success", "Execute SQL")
else
    MsgBox.Warning("Failure", "Execute SQL")
end
```

7. InedDatabase.CreateOracleTable.ave

```
'   Project: TRANSIMS
'   Subsystem: Input Editor
'   $RCSfile: InedDatabase.CreateOracleTable.ave,v $
'   $Revision: 1.1 $
'   $Date: 1996/07/29 18:25:05 $
'   $State: Stab $
'   $Author: bwb $
'   U.S. Government Copyright 1995
'   All rights reserved


'   This script puts an ArcView table into Oracle.


'   Get the input file.
inFile = FileDialog.Show("*.dbf", "*.dbf", "Input Data for Oracle Table")
if (inFile = NIL) then
    exit
end
```

```
inTable = VTab.Make(inFile, false, false)
fields = inTable.GetFields

'   Get the output table name.
outName = MsgBox.Input("Oracle name:", "Create Oracle Table", infile.GetBaseName)
if (outName = NIL) then
    exit
end

'   Create the table.
sql = "CREATE TABLE """ + outName + """ (" + 13.AsChar
first = true
for each fld in fields
    if (first) then
        first = false
    else
        sql = sql + "," + 13.AsChar
    end
    sql = sql + """" + fld.GetAlias.UCase + """ "
    if (fld.IsTypeNumber) then
        sql = sql + "NUMBER(" + fld.GetWidth.AsString + "," + fld.GetPrecision.AsString +
")"
    else
        sql = sql + "VARCHAR(" + fld.GetWidth.AsString + ")"
    end
end
sql = sql + ")"
if (_directory.ExecuteSQL(sql).not) then
    MsgBox.Warning("Table creation failed.", "Create Oracle Table")
    exit
end

'   Copy the data.
for each rec in inTable
    sql = "INSERT INTO """ + outName + """ VALUES("
    first = true
    for each fld in fields
        if (first) then
            first = false
        else
            sql = sql + ","
        end
        sql = sql + "'" + inTable.ReturnValueString(fld, rec) + "'"
    end
    sql = sql + ")"
    if (_directory.ExecuteSQL(sql).not) then
        MsgBox.Warning("Data insertion failed.", "Create Oracle Table")
        exit
    end
end

'   Return result.
MsgBox.Info("Success", "Create Oracle Table")
```

8. InedDatabase.CreateSource.ave

```
'   Project: TRANSIMS
'   Subsystem: Input Editor
'   $RCSfile: InedDatabase.CreateSource.ave,v $
'   $Revision: 1.0 $
'   $Date: 1996/04/25 18:20:38 $
'   $State: Rel $
'   $Author: bwb $
'   U.S. Government Copyright 1995
'   All rights reserved


'   This script lets the user create a data source.


'   Get the source description.
response = MsgBox.MultiInput("Enter data source description:", "Create Data Source",
{"Name", "Comment"}, {"", ""})
if (response.Count <> 2) then
    exit
```

```
end

'   Create the source.
if (_directory.ExecuteSQL("INSERT INTO SOURCE_INDEX (NAME, COMMENT_TEXT) VALUES('" +
response.Get(0) + "', '" + response.Get(1) + "')")) then
    MsgBox.Info("Success", "Create Data Source")
else
    MsgBox.Warning("Failure", "Create Data Source")
    exit
end

'   Update the sources.
av.Run("InedDatabase.UpdateSources", NIL)
```

9. InedDatabase.DeleteOracleTable.ave

```
'   Project: TRANSIMS
'   Subsystem: Input Editor
'   $RCSfile: InedDatabase.DeleteOracleTable.ave,v $
'   $Revision: 1.0 $
'   $Date: 1996/04/25 18:20:38 $
'   $State: Rel $
'   $Author: bwb $
'   U.S. Government Copyright 1995
'   All rights reserved


'   This script lets the user delete an Oracle table.


'   Get the list of tables.
tableTable = VTab.MakeSQL(_directory, "SELECT TABLE_NAME FROM USER_TABLES")
if (tableTable.HasError) then
    MsgBox.Error("Cannot retrieve list of tables.", "Delete Oracle Table")
    exit
end
tableField = tableTable.FindField("TABLE_NAME")
tables = List.Make
for each rec in tableTable
    tables.Add(tableTable.ReturnValue(tableField, rec))
end

'   Get the user's choice.
response = MsgBox.ListAsString(tables, "Choose the table:", "Delete Oracle Table")
if (response = NIL) then
    exit
end
if (MsgBox.YesNo("Delete the table """ + response + """?", "Delete Oracle Table",
false).not) then
    exit
end

'   Delete the table.
if (_directory.ExecuteSQL("DROP TABLE """ + response + """")) then
    MsgBox.Info("Success", "Delete Oracle Table")
else
    MsgBox.Warning("Failure", "Delete Oracle Table")
end
```

10. InedDatabase.DeleteSource.ave

```
'   Project: TRANSIMS
'   Subsystem: Input Editor
'   $RCSfile: InedDatabase.DeleteSource.ave,v $
'   $Revision: 1.0 $
'   $Date: 1996/04/25 18:20:38 $
'   $State: Rel $
'   $Author: bwb $
'   U.S. Government Copyright 1995
'   All rights reserved


'   This script lets the user delete a data source.
```

```
'   Get the source.
response = av.Run("InedDatabase.ChooseSource", "Delete Data Source")
if (response = NIL) then
    exit
end

'   Confirm.
if (MsgBox.YesNo("Are you sure you want to delete the """ + response + """ data source?",
"Delete Data Source", FALSE).not) then
    exit
end

'   Delete the source.
if (_directory.ExecuteSQL("DELETE FROM SOURCE_INDEX WHERE NAME = '" + response + "'"))
then
    MsgBox.Info("Success", "Delete Data Source")
else
    MsgBox.Warning("Failure", "Delete Data Source")
    exit
end

'   Update the sources.
av.Run("InedDatabase.UpdateSources", NIL)
```

11. InedDatabase.DeleteTable.ave

```
'   Project: TRANSIMS
'   Subsystem: Input Editor
'   $RCSfile: InedDatabase.DeleteTable.ave,v $
'   $Revision: 1.0 $
'   $Date: 1996/04/25 18:20:38 $
'   $State: Rel $
'   $Author: bwb $
'   U.S. Government Copyright 1995
'   All rights reserved


'   This script lets the user delete an Oracle table.


'   Get the list of tables.
tableTable = VTab.MakeSQL(_directory, "SELECT TABLE_NAME FROM USER_TABLES")
if (tableTable.HasError) then
    MsgBox.Error("Cannot retrieve list of tables.", "Delete Oracle Table")
    exit
end
tableField = tableTable.FindField("TABLE_NAME")
tables = List.Make
for each rec in tableTable
    tables.Add(tableTable.ReturnValue(tableField, rec))
end

'   Get the user's choice.
response = MsgBox.ListAsString(tables, "Choose the table:", "Delete Oracle Table")
if (response = NIL) then
    exit
end
if (MsgBox.YesNo("Delete the table """ + response + """?", "Delete Oracle Table",
false).not) then
    exit
end

'   Delete the table.
if (_directory.ExecuteSQL("DROP TABLE """ + response + """")) then
    MsgBox.Info("Success", "Delete Oracle Table")
else
    MsgBox.Warning("Failure", "Delete Oracle Table")
end
```

12. InedDatabase.ExecuteSQL.ave

```
'   Project: TRANSIMS
'   Subsystem: Input Editor
'   $RCSfile: InedDatabase.ExecuteSQL.ave,v $
'   $Revision: 1.0 $
```

```
'   $Date: 1996/04/25 18:20:38 $
'   $State: Rel $
'   $Author: bwb $
'   U.S. Government Copyright 1995
'   All rights reserved


'   This script executes a user-input SQL statement.


'   Get the statement.
response = MsgBox.Input("", "Execute SQL", "")
if (response = NIL) then
    exit
end

'   Execute the statement.
if (_directory.ExecuteSQL(response)) then
    MsgBox.Info("Success", "Execute SQL")
else
    MsgBox.Warning("Failure", "Execute SQL")
end
```

13. InedDatabase.ExecuteSQLQuery.ave

```
'   Project: TRANSIMS
'   Subsystem: Input Editor
'   $RCSfile: InedDatabase.ExecuteSQLQuery.ave,v $
'   $Revision: 1.0 $
'   $Date: 1996/04/25 18:20:38 $
'   $State: Rel $
'   $Author: bwb $
'   U.S. Government Copyright 1995
'   All rights reserved


'   This script executes a user-input SQL query.


'   Get the statement.
response = MsgBox.Input("", "Execute SQL Query", "")
if (response = NIL) then
    exit
end

'   Execute the statement.
answer = VTab.MakeSQL(_directory, response)
if (answer.HasError.Not) then
    MsgBox.Info("Success", "Execute SQL Query")
else
    MsgBox.Warning("Failure", "Execute SQL Query")
    exit
end

'   Display the table.
document = Table.Make(answer)
document.SetName(response)
document.GetWin.Open
```

14. InedDatabase.GetMaster.ave

```
'   Project: TRANSIMS
'   Subsystem: Input Editor
'   $RCSfile: InedDatabase.GetMaster.ave,v $
'   $Revision: 1.0 $
'   $Date: 1996/04/25 18:20:38 $
'   $State: Rel $
'   $Author: bwb $
'   U.S. Government Copyright 1995
'   All rights reserved


'   This script gets the master table name for a table.  The first parameter is the master
source, the second parameter is the dependent source, and the third parameter is the
dependent table.
```

```
'   Check the dependency information.
for each tuple in _dependencies
    if ((tuple.Get(0) = SELF.Get(0)) and (tuple.Get(2) = SELF.Get(1)) and (tuple.Get(3) =
SELF.Get(2))) then
        return tuple.Get(1)
    end
end

'   Get the dependent table from the user.
return av.Run("InedDatabase.ChooseTable" , {SELF.Get(0) + " Table on Which " + SELF.Get(1)
+ " Table Depends", SELF.Get(0)})
```

15. InedDatabase.GetTable.ave

```
'   Project: TRANSIMS
'   Subsystem: Input Editor
'   $RCSfile: InedDatabase.GetTable.ave,v $
'   $Revision: 1.0 $
'   $Date: 1996/04/25 18:20:38 $
'   $State: Rel $
'   $Author: bwb $
'   U.S. Government Copyright 1995
'   All rights reserved


'   This script returns a VTab for the specified data source and table.  The first
argument is the source name and the second argument is the table name.


'   Find the table name.
tableName = av.Run("InedDatabase.GetTableName", SELF)

'   Open the table.
answer = VTab.MakeSQL(_directory, "SELECT * FROM """ + tableName + """")
if (answer.HasError) then
    MsgBox.Error("Cannot retrieve table.", "Get Table")
    exit
end

return answer
```

16. InedDatabase.GetTableName.ave

```
'   Project: TRANSIMS
'   Subsystem: Input Editor
'   $RCSfile: InedDatabase.GetTableName.ave,v $
'   $Revision: 1.0 $
'   $Date: 1996/04/25 18:20:38 $
'   $State: Rel $
'   $Author: bwb $
'   U.S. Government Copyright 1995
'   All rights reserved


'   This script returns the database table name for a table.  The first parameter is the
source, and the second is the table.


'   Look up the database table name.
return _tables.Get(SELF.Get(0)).Get(SELF.Get(1)).Get(0)
```

17. InedDatabase.OpenDirectory.ave

```
'   Project: TRANSIMS
'   Subsystem: Input Editor
'   $RCSfile: InedDatabase.OpenDirectory.ave,v $
'   $Revision: 1.1 $
'   $Date: 1996/07/02 18:58:22 $
'   $State: Rel $
'   $Author: bwb $
'   U.S. Government Copyright 1995
'   All rights reserved
```

```
'   This script opens a connection to a data directory in Oracle.


'   Make sure we are not logged in already.
if (_directory.IsLogin) then
    _directory.Logout
end

'   Log in.
name = MsgBox.ListAsString({"IOC-1", "Case Study 1"}, "Choose a data directory:", "Open
Data Directory")
if (name = "IOC-1") then
    _directory.Login("ioc1/lanl")
elseif (name = "Case Study 1") then
    _directory.Login("cs1/lanl")
else
    exit
end

'   Read sources.
av.Run("InedDatabase.UpdateSources", NIL)

'   Read dependencies.
av.Run("InedDatabase.UpdateDependencies", NIL)
```

18. InedDatabase.RegisterTable.ave

```
'   Project: TRANSIMS
'   Subsystem: Input Editor
'   $RCSfile: InedDatabase.RegisterTable.ave,v $
'   $Revision: 1.0 $
'   $Date: 1996/04/25 18:20:38 $
'   $State: Rel $
'   $Author: bwb $
'   U.S. Government Copyright 1995
'   All rights reserved


'   This script lets the user register a data table.


'   Get the source.
source = av.Run("InedDatabase.ChooseSource", "Register Data Table")
if (source = NIL) then
    exit
end

'   Get the table description.
response = MsgBox.MultiInput("Enter data table description:", "Register Data Table",
{"Name", "Oracle Name", "Comment"}, {"", "", ""})
if (response.Count <> 3) then
    exit
end

'   Create the table.
if (_directory.ExecuteSQL("INSERT INTO TABLE_INDEX (SOURCE, NAME, TABLE_NAME,
COMMENT_TEXT) VALUES('" + source + "', '" + response.Get(0) + "', '" + response.Get(1) +
"', '" + response.Get(2) + "')")) then
    MsgBox.Info("Success", "Register Data Table")
else
    MsgBox.Warning("Failure", "Register Data Table")
    exit
end

'   Update the sources.
av.Run("InedDatabase.UpdateTables", source)
```

19. InedDatabase.RemoveDependency.ave

```
'   Project: TRANSIMS
'   Subsystem: Input Editor
'   $RCSfile: InedDatabase.RemoveDependency.ave,v $
'   $Revision: 1.0 $
```

```
'   $Date: 1996/04/25 18:20:38 $
'   $State: Rel $
'   $Author: bwb $
'   U.S. Government Copyright 1995
'   All rights reserved


'   This script lets the user remove a dependency between data tables.


'   Choose dependent.
dependent = av.Run("InedDatabase.ChooseSourceAndTable", "Dependent Table")
if (dependent = NIL) then
    exit
end

'   Choose master.
masters = List.Make
for each tuple in _dependencies
    if ((tuple.Get(2) = dependent.Get(0)) and (tuple.Get(3) = dependent.Get(1))) then
        masters.Add(tuple.Get(1))
    end
end
response = MsgBox.ListAsString(masters, "Master table:", "Remove Data Dependency")
if (response = NIL) then
    exit
end
for each tuple in _dependencies
    if ((tuple.Get(1) = response) and (tuple.Get(2) = dependent.Get(0)) and (tuple.Get(3)
= dependent.Get(1))) then
        master = {tuple.Get(0), tuple.Get(1)}
    end
end
if (MsgBox.YesNo("Remove the dependence of  """ + dependent.Get(1) + """ on """ +
master.Get(1) + """?", "Remove Data Dependency", TRUE).not) then
    exit
end

'   Eliminate the dependency.
if (_directory.ExecuteSQL("DELETE FROM DEPENDENT_TABLES WHERE TABLE_NAME = '" +
av.Run("InedDatabase.GetTableName", master) + "' AND DEPENDENT_NAME = '" +
av.Run("InedDatabase.GetTableName", dependent) + "'")) then
    MsgBox.Info("Success", "Remove Data Dependency")
else
    MsgBox.Warning("Failure", "Remove Data Dependency")
end

'   Update the dependencies.
av.Run("InedDatabase.UpdateDependencies", NIL)
```

20. InedDatabase.UnregisterTable.ave

```
'   Project: TRANSIMS
'   Subsystem: Input Editor
'   $RCSfile: InedDatabase.UnregisterTable.ave,v $
'   $Revision: 1.0 $
'   $Date: 1996/04/25 18:20:38 $
'   $State: Rel $
'   $Author: bwb $
'   U.S. Government Copyright 1995
'   All rights reserved


'   This script lets the user unregister a data table.


'   Get the table.
response = av.Run("InedDatabase.ChooseSourceAndTable", "Unregister Data Table")
if (response = NIL) then
    exit
end

'   Confirm.
```

```
if (MsgBox.YesNo("Are you sure you want to unregister the """ + response.Get(1) + """ data
table in the """ + response.Get(0) + """ data source?", "Unregister Data Table",
FALSE).not) then
    exit
end

'   Delete the source.
if (_directory.ExecuteSQL("DELETE FROM TABLE_INDEX WHERE SOURCE = '" + response.Get(0) +
"' AND NAME = '" + response.Get(1) + "'")) then
    MsgBox.Info("Success", "Unregister Data Table")
else
    MsgBox.Warning("Failure", "Unregister Data Table")
    exit
end

'   Update the tables.
av.Run("InedDatabase.UpdateTables", response.Get(0))
```

21. InedDatabase.UpdateDependencies.ave

```
'   Project: TRANSIMS
'   Subsystem: Input Editor
'   $RCSfile: InedDatabase.UpdateDependencies.ave,v $
'   $Revision: 1.0 $
'   $Date: 1996/04/25 18:20:38 $
'   $State: Rel $
'   $Author: bwb $
'   U.S. Government Copyright 1995
'   All rights reserved


'   This script updates the data dependencies from the database.


'   Clear the dictionary of tables for the specified sources.
_dependencies.Empty

'   Build joined table.
_directory.ExecuteSQL(
        "CREATE TABLE AV_INED_TEMP AS" + 13.AsChar +
        "  SELECT A.SOURCE AS MAS_SOURCE, A.NAME AS MAS_TABLE, B.SOURCE AS DEP_SOURCE,
B.NAME AS DEP_TABLE" + 13.AsChar +
        "    FROM DEPENDENT_TABLES D, TABLE_INDEX A, TABLE_INDEX B" + 13.AsChar +
        "    WHERE A.TABLE_NAME = D.TABLE_NAME AND B.TABLE_NAME = D.DEPENDENT_NAME")
dependentTable = VTab.MakeSQL(_directory, "SELECT * FROM AV_INED_TEMP")
_directory.ExecuteSQL("DROP TABLE AV_INED_TEMP")
if (dependentTable.HasError) then
    MsgBox.Error("Could not retrieve dependents.", "Update Dependencies")
    exit
end

'   Read the result.
masterSourceField = dependentTable.FindField("MAS_SOURCE")
masterTableField = dependentTable.FindField("MAS_TABLE")
dependentSourceField = dependentTable.FindField("DEP_SOURCE")
dependentTableField = dependentTable.FindField("DEP_TABLE")
for each rec in dependentTable
    _dependencies.Add({dependentTable.ReturnValue(masterSourceField, rec),
dependentTable.ReturnValue(masterTableField, rec),
dependentTable.ReturnValue(dependentSourceField, rec),
dependentTable.ReturnValue(dependentTableField, rec)})
end
```

22. InedDatabase.UpdateSources.ave

```
'   Project: TRANSIMS
'   Subsystem: Input Editor
'   $RCSfile: InedDatabase.UpdateSources.ave,v $
'   $Revision: 1.0 $
'   $Date: 1996/04/25 18:20:38 $
'   $State: Rel $
'   $Author: bwb $
'   U.S. Government Copyright 1995
'   All rights reserved
```

```
'   This script updates the dictionary of data sources for the current data directory.


'   Clear the dictionary of sources.
_sources.Empty

'   Get the source index.
sourceIndex = VTab.MakeSQL(_directory, "SELECT NAME, COMMENT_TEXT FROM SOURCE_INDEX")
if (sourceIndex.HasError) then
    MsgBox.Error("Cannot retrieve source index.", "Update Data Sources")
    exit
end

'   Find the fields in the source index.
nameField = sourceIndex.FindField("NAME")
commentField = sourceIndex.FindField("COMMENT_TEXT")

'   Read the source index.
for each record in sourceIndex
    _sources.Add(sourceIndex.ReturnValueString(nameField, record),
sourceIndex.ReturnValueString(commentField, record))
end

'   Update the table indexes.
for each source in _sources.ReturnKeys
    if (av.Run("InedDatabase.UpdateTables", source).Not) then
          MsgBox.Error("Cannot retrieve table index for source " + source + ".", "Update
Data Tables")
    end
end
```

23. InedDatabase.UpdateTables.ave

```
'   Project: TRANSIMS
'   Subsystem: Input Editor
'   $RCSfile: InedDatabase.UpdateTables.ave,v $
'   $Revision: 1.0 $
'   $Date: 1996/04/25 18:20:38 $
'   $State: Rel $
'   $Author: bwb $
'   U.S. Government Copyright 1995
'   All rights reserved


'   This script updates the dictionary of data tables for the specified data sources.


'   Clear the dictionary of tables for the specified sources.
_tables.Remove(SELF)
_tables.Add(SELF, Dictionary.Make(25))
tables = _tables.Get(SELF)

'   Get the table index.
tableIndex = VTab.MakeSQL(_directory, "SELECT NAME, TABLE_NAME, COMMENT_TEXT FROM
TABLE_INDEX WHERE SOURCE = '" + SELF + "'")
if (tableIndex.HasError) then
    return FALSE
end

'   Find the fields in the table index.
nameField = tableIndex.FindField("NAME")
tableField = tableIndex.FindField("TABLE_NAME")
commentField = tableIndex.FindField("COMMENT_TEXT")

'   Read the table index.
for each record in tableIndex
    tables.Add(tableIndex.ReturnValueString(nameField, record),
{tableIndex.ReturnValueString(tableField, record),
tableIndex.ReturnValueString(commentField, record)})
end

return TRUE
```

24. InedDatabase.ViewDependents.ave

```
'   Project: TRANSIMS
'   Subsystem: Input Editor
'   $RCSfile: InedDatabase.ViewDependents.ave,v $
'   $Revision: 1.0 $
'   $Date: 1996/04/25 18:20:38 $
'   $State: Rel $
'   $Author: bwb $
'   U.S. Government Copyright 1995
'   All rights reserved


'   This script lets the user view data table dependencies.


'   Get source and table.
response = av.Run("InedDatabase.ChooseSourceAndTable", "Master")
if (response = NIL) then
    exit
end

'   Build joined table.
_directory.ExecuteSQL(
        "CREATE TABLE AV_INED_TEMP AS" + 13.AsChar +
        "  SELECT B.SOURCE AS SOURCE, B.NAME AS NAME" + 13.AsChar +
        "    FROM DEPENDENT_TABLES D, TABLE_INDEX A, TABLE_INDEX B" + 13.AsChar +
        "    WHERE A.NAME = '" + response.Get(1) + "' AND A.TABLE_NAME = D.TABLE_NAME AND
B.TABLE_NAME = D.DEPENDENT_NAME")
depTable = VTab.MakeSQL(_directory, "SELECT * FROM AV_INED_TEMP")
_directory.ExecuteSQL("DROP TABLE AV_INED_TEMP")
if (depTable.HasError) then
    MsgBox.Error("Cannot find dependents.", "View Dependencies")
    exit
end

'   Get the table.
document = Table.Make(depTable)
document.SetName("Tables dependent on " + response.Get(1))
document.GetWin.Open
```

25. InedDatabase.ViewDependentTables.ave

```
'   Project: TRANSIMS
'   Subsystem: Input Editor
'   $RCSfile: InedDatabase.ViewDependentTables.ave,v $
'   $Revision: 1.0 $
'   $Date: 1996/04/25 18:20:38 $
'   $State: Rel $
'   $Author: bwb $
'   U.S. Government Copyright 1995
'   All rights reserved


'   This script displays the dependent tables.


'   Display the table.
document = Table.Make(VTab.MakeSQL(_directory, "SELECT * FROM DEPENDENT_TABLES"))
document.SetName("Dependent Tables")
document.GetWin.Open
```

26. InedDatabase.ViewMasters.ave

```
'   Project: TRANSIMS
'   Subsystem: Input Editor
'   $RCSfile: InedDatabase.ViewMasters.ave,v $
'   $Revision: 1.0 $
'   $Date: 1996/04/25 18:20:38 $
'   $State: Rel $
'   $Author: bwb $
'   U.S. Government Copyright 1995
'   All rights reserved
```

```
'   This script displays the master tables for a given table.


'   Get source and table.
response = av.Run("InedDatabase.ChooseSourceAndTable", "Dependent Table")
if (response = NIL) then
    exit
end

'   Build joined table.
_directory.ExecuteSQL(
        "CREATE TABLE AV_INED_TEMP AS" + 13.AsChar +
        "  SELECT A.SOURCE AS SOURCE, A.NAME AS NAME" + 13.AsChar +
        "    FROM DEPENDENT_TABLES D, TABLE_INDEX A, TABLE_INDEX B" + 13.AsChar +
        "    WHERE B.NAME = '" + response.Get(1) + "' AND A.TABLE_NAME = D.TABLE_NAME AND
B.TABLE_NAME = D.DEPENDENT_NAME")
depTable = VTab.MakeSQL(_directory, "SELECT * FROM AV_INED_TEMP")
_directory.ExecuteSQL("DROP TABLE AV_INED_TEMP")
if (depTable.HasError) then
    MsgBox.Error("Cannot find dependents.", "View Dependencies")
    exit
end

'   Get the table.
document = Table.Make(depTable)
document.SetName("Tables on which " + response.Get(1) + " depends")
document.GetWin.Open
```

27. InedDatabase.ViewSourceIndex.ave

```
'   Project: TRANSIMS
'   Subsystem: Input Editor
'   $RCSfile: InedDatabase.ViewSourceIndex.ave,v $
'   $Revision: 1.0 $
'   $Date: 1996/04/25 18:20:38 $
'   $State: Rel $
'   $Author: bwb $
'   U.S. Government Copyright 1995
'   All rights reserved


'   This script displays the source index.


'   Display the table.
document = Table.Make(VTab.MakeSQL(_directory, "SELECT * FROM SOURCE_INDEX"))
document.SetName("Source Index")
document.GetWin.Open
```

28. InedDatabase.ViewTable.ave

```
'   Project: TRANSIMS
'   Subsystem: Input Editor
'   $RCSfile: InedDatabase.ViewTable.ave,v $
'   $Revision: 1.0 $
'   $Date: 1996/04/25 18:20:38 $
'   $State: Rel $
'   $Author: bwb $
'   U.S. Government Copyright 1995
'   All rights reserved


'   This script prompts the user for a data source and table, and opens a view of the
table.


'   Prompt the user for the source and table.
response = av.Run("InedDatabase.ChooseSourceAndTable", "View Data Table")
if (response = NIL) then
    exit
end

'   Make sure the document is not open already.'
document = av.GetProject.FindDoc(response.Get(1))
if (document <> NIL) then
```

```
    ''''document.GetVTab.Refresh
    document.GetWin.Open
    exit
end

'   Get the table.
document = Table.Make(av.Run("InedDatabase.GetTable", response))
document.SetName(response.Get(1))
document.GetWin.Open
```

29. InedDatabase.ViewTableIndex.ave

```
'   Project: TRANSIMS
'   Subsystem: Input Editor
'   $RCSfile: InedDatabase.ViewTableIndex.ave,v $
'   $Revision: 1.0 $
'   $Date: 1996/04/25 18:20:38 $
'   $State: Rel $
'   $Author: bwb $
'   U.S. Government Copyright 1995
'   All rights reserved


'   This script displays the table index.


'   Display the table.
document = Table.Make(VTab.MakeSQL(_directory, "SELECT * FROM TABLE_INDEX"))
document.SetName("Table Index")
document.GetWin.Open
```

D. Geography Creation Scripts

1. InedGeography.CreateLanes.ave

```
'   Project: TRANSIMS
'   Subsystem: Input Editor
'   $RCSfile: InedGeography.CreateLanes.ave,v $
'   $Revision: 1.4 $
'   $Date: 1997/02/27 14:25:33 $
'   $State: Rel $
'   $Author: bwb $
'   U.S. Government Copyright 1995
'   All rights reserved


'   This script creates a lane table with geographic attributes.


'   Get the input tables.
nodeDoc = av.GetProject.FindDoc("Nodes")
if (nodeDoc = NIL) then
    MsgBox.Error("Node table not found.", "Create Lane Shape File")
    exit
end
nodeVTab = nodeDoc.GetVTab
linkDoc = av.GetProject.FindDoc("Links")
if (linkDoc = NIL) then
    MsgBox.Error("Link table not found.", "Create Lane Shape File")
    exit
end
linkVTab = linkDoc.GetVTab
pocketDoc = av.GetProject.FindDoc("Pocket Lanes")
if (pocketDoc = NIL) then
    MsgBox.Error("Pocket Lane table not found.", "Create Lane Shape File")
    exit
end
pocketVTab = pocketDoc.GetVTab

'   Get the offset for lanes.
offset = MsgBox.Input("Lane offset/width in coordinate units:", "Create Lane Shape File",
"3.25")
if (offset = NIL) then
```

```
    exit
end
offset = offset.AsNumber
if ((offset > 0).Not) then
    MsgBox.Error("Invalid lane offset.", "Create Lane Shape File")
    exit
end
width = offset

'   Get the input file.
laneFile = FileDialog.Put("lanes.shp".AsFileName, "*.shp", "Lane Shape File")
if (laneFile = NIL) then
    exit
end
laneFTab = FTab.MakeNew(laneFile, Polygon)
laneFTab.AddFields({
    Field.Make("Towards", #FIELD_LONG, 10, 0),
    Field.Make("From", #FIELD_LONG, 10, 0),
    Field.Make("Link", #FIELD_LONG, 10, 0),
    Field.Make("Lane", #FIELD_BYTE, 3, 0),
    Field.Make("Style", #FIELD_CHAR, 1, 0),
    Field.Make("Length", #FIELD_FLOAT, 16, 9)
})

'   Retrieve the fields for the lane table.
shapeFieldOut = laneFTab.FindField("Shape")
towardsFieldOut = laneFTab.FindField("Towards")
fromFieldOut = laneFTab.FindField("From")
linkFieldOut = laneFTab.FindField("Link")
laneFieldOut = laneFTab.FindField("Lane")
styleFieldOut = laneFTab.FindField("Style")
lengthFieldOut = laneFTab.FindField("Length")

'   Retrieve the fields for the node table.
nodeField = nodeVTab.FindField("ID")
abscissaField = nodeVTab.FindField("ABSCISSA")
ordinateField = nodeVTab.FindField("ORDINATE")

'   Generate the points.
points = Dictionary.Make(nodeVTab.GetNumRecords)
for each i in nodeVTab
    id = nodeVTab.ReturnValueNumber(nodeField, i)
    x = nodeVTab.ReturnValueNumber(abscissaField, i)
    y = nodeVTab.ReturnValueNumber(ordinateField, i)
    points.Add(id, Point.Make(x, y))
end

'   Retrieve the fields for the link table.
linkField = linkVTab.FindField("ID")
nodeaField = linkVTab.FindField("NODEA")
nodebField = linkVTab.FindField("NODEB")
permanentlanesaField = linkVTab.FindField("PERMLANESA")
permanentlanesbField = linkVTab.FindField("PERMLANESB")
leftpocketsaField = linkVTab.FindField("LEFTPCKTSA")
leftpocketsbField = linkVTab.FindField("LEFTPCKTSB")
twowayturnField = linkVTab.FindField("TWOWAYTURN")
setbackaField = linkVTab.FindField("SETBACKA")
setbackbField = linkVTab.FindField("SETBACKB")
lengthField = linkVTab.FindField("LENGTH")

'   Create the permanent lanes.
lines = Dictionary.Make(linkVTab.GetNumRecords)
count = 0
av.ShowMsg("Creating permanent lanes . . .")
av.ShowStopButton
for each i in linkVTab

    '   Calculate the tangent and normal.
    id = linkVTab.ReturnValueNumber(linkField, i)
    nodea = linkVTab.ReturnValueNumber(nodeaField, i)
    nodeb = linkVTab.ReturnValueNumber(nodebField, i)
    a = points.Get(nodea)
    b = points.Get(nodeb)
    delta = a - b
    norm = ((delta.GetX * delta.GetX) + (delta.GetY * delta.GetY) + 0.000001).sqrt
    tangent = Point.Make(delta.GetX / norm, delta.GetY / norm)
```

```
    normal = Point.Make(tangent.GetY, - (tangent.GetX))

    ’   Adjust the endpoints for the setbacks.
    setbacka = linkVTab.ReturnValueNumber(setbackaField, i)
    setbackb = linkVTab.ReturnValueNumber(setbackbField, i)
    length = linkVTab.ReturnValueNumber(lengthField, i) - setbacka - setbackb
    a = Point.Make(a.GetX - (tangent.GetX * setbacka), a.GetY - (tangent.GetY * setbacka))
    b = Point.Make(b.GetX + (tangent.GetX * setbackb), b.GetY + (tangent.GetY * setbackb))

    ’   Store some key information.
    lines.Add(id, {nodea, nodeb, setbacka, setbackb})

    ’   Make the permanent lanes.
    if (linkVTab.ReturnValue(twowayturnField, i) = "Y") then
        outer = Point.Make(0.5 * width * normal.GetX, 0.5 * width * normal.GetY)
        j = laneFTab.AddRecord
        laneFTab.SetValue(towardsFieldOut, j, nodea)
        laneFTab.SetValue(fromFieldOut, j, nodeb)
        laneFTab.SetValue(linkFieldOut, j, id)
        laneFTab.SetValue(laneFieldOut, j, 0)
        laneFTab.SetValue(styleFieldOut, j, "X")
        laneFTab.SetValue(lengthFieldOut, j, length)
        laneFTab.SetValue(shapeFieldOut, j, Polygon.Make({{a - outer, a + outer, b +
outer, b - outer}}))
    end
    lane0 = linkVTab.ReturnValueNumber(leftpocketsaField, i) + 1
    lane1 = lane0 - 1 + linkVTab.ReturnValueNumber(permanentlanesaField, i)
    if (lane0 <= lane1) then
        for each lane in lane0..lane1
            inner = Point.Make((lane - 0.5) * offset * normal.GetX, (lane - 0.5) * offset
* normal.GetY)
            outer = inner + Point.Make(width * normal.GetX, width * normal.GetY)
            j = laneFTab.AddRecord
            laneFTab.SetValue(towardsFieldOut, j, nodea)
            laneFTab.SetValue(fromFieldOut, j, nodeb)
            laneFTab.SetValue(linkFieldOut, j, id)
            laneFTab.SetValue(laneFieldOut, j, lane)
            laneFTab.SetValue(styleFieldOut, j, "X")
            laneFTab.SetValue(lengthFieldOut, j, length)
            laneFTab.SetValue(shapeFieldOut, j, Polygon.Make({{a + inner, a + outer, b +
outer, b + inner}}))
        end
    end
    lane0 = linkVTab.ReturnValueNumber(leftpocketsbField, i) + 1
    lane1 = lane0 - 1 + linkVTab.ReturnValueNumber(permanentlanesbField, i)
    if (lane0 <= lane1) then
        for each lane in lane0..lane1
            inner = Point.Make((lane - 0.5) * offset * normal.GetX, (lane - 0.5) * offset
* normal.GetY)
            outer = inner + Point.Make(width * normal.GetX, width * normal.GetY)
            j = laneFTab.AddRecord
            laneFTab.SetValue(towardsFieldOut, j, nodeb)
            laneFTab.SetValue(fromFieldOut, j, nodea)
            laneFTab.SetValue(linkFieldOut, j, id)
            laneFTab.SetValue(laneFieldOut, j, lane)
            laneFTab.SetValue(styleFieldOut, j, "X")
            laneFTab.SetValue(lengthFieldOut, j, length)
            laneFTab.SetValue(shapeFieldOut, j, Polygon.Make({{a - inner, a - outer, b -
outer, b - inner}}))
        end
    end

    ’   Update the status.
    count = count + 1
    if (av.SetStatus(100 * count / linkVTab.GetNumRecords).Not) then
        break
    end

end
av.PurgeObjects

’   Retrieve the fields for the pocket lane table.
linkField = pocketVTab.FindField("LINK")
nodeField = pocketVTab.FindField("NODE")
positionField = pocketVTab.FindField("OFFSET")
laneField = pocketVTab.FindField("LANE")
```

```
styleField = pocketVTab.FindField("STYLE")
lengthField = pocketVTab.FindField("LENGTH")

'   Create the pocket lanes.
count = 0
av.ShowMsg("Creating pocket lanes . . .")
av.ShowStopButton
for each i in pocketVTab

    '   Calculate the tangent and normal.
    id = pocketVTab.ReturnValueNumber(linkField, i)
    node = pocketVTab.ReturnValueNumber(nodeField, i)
    nodeData = lines.Get(id)
    if (node = nodeData.Get(0)) then
        nodea = nodeData.Get(0)
        nodeb = nodeData.Get(1)
        setbacka = nodeData.Get(2)
        setbackb = nodeData.Get(3)
    else
        nodea = nodeData.Get(1)
        nodeb = nodeData.Get(0)
        setbacka = nodeData.Get(3)
        setbackb = nodeData.Get(2)
    end
    a = points.Get(nodea)
    b = points.Get(nodeb)
    delta = a - b
    norm = ((delta.GetX * delta.GetX) + (delta.GetY * delta.GetY) + 0.000001).sqrt
    tangent = Point.Make(delta.GetX / norm, delta.GetY / norm)
    normal = Point.Make(tangent.GetY, - (tangent.GetX))

    '   Adjust the endpoints for the setbacks.
    style = pocketVTab.ReturnValueString(styleField, i)
    position = pocketVTab.ReturnValueNumber(positionField, i)
    length = pocketVTab.ReturnValueNumber(lengthField, i)
    if (style = "T") then
        a = Point.Make(a.GetX - (tangent.GetX * setbacka), a.GetY - (tangent.GetY *
setbacka))
        b = Point.Make(a.GetX - (tangent.GetX * length), a.GetY - (tangent.GetY * length))
    elseif (style = "M") then
        b = Point.Make(b.GetX + (tangent.GetX * setbackb), b.GetY + (tangent.GetY *
setbackb))
        a = Point.Make(b.GetX + (tangent.GetX * length), b.GetY + (tangent.GetY * length))
    elseif (style = "P") then
        b = Point.Make(b.GetX + (tangent.GetX * position), b.GetY + (tangent.GetY *
position))
        a = Point.Make(b.GetX + (tangent.GetX * length), b.GetY + (tangent.GetY * length))
    end

    '   Make the pocket lane.
    lane = pocketVTab.ReturnValueNumber(laneField, i)
    inner = Point.Make((lane - 0.5) * offset * normal.GetX, (lane - 0.5) * offset *
normal.GetY)
    outer = inner + Point.Make(width * normal.GetX, width * normal.GetY)
    j = laneFTab.AddRecord
    laneFTab.SetValue(towardsFieldOut, j, nodea)
    laneFTab.SetValue(FromFieldOut, j, nodeb)
    laneFTab.SetValue(linkFieldOut, j, id)
    laneFTab.SetValue(laneFieldOut, j, lane)
    laneFTab.SetValue(styleFieldOut, j, style)
    laneFTab.SetValue(lengthFieldOut, j, length)
    laneFTab.SetValue(shapeFieldOut, j, Polygon.Make({{a + inner, a + outer, b + outer, b
+ inner}}))

    '   Update the status.
    count = count + 1
    if (av.SetStatus(100 * count / pocketVTab.GetNumRecords).Not) then
        break
    end

end
av.PurgeObjects

'   Done.
av.ClearMsg
av.SetStatus(100)
```

```
laneFTab.Flush
```

2. InedGeography.CreateParkings.ave

```
'   Project: TRANSIMS
'   Subsystem: Input Editor
'   $RCSfile: InedGeography.CreateParkings.ave,v $
'   $Revision: 1.1 $
'   $Date: 1997/02/27 14:25:33 $
'   $State: Rel $
'   $Author: bwb $
'   U.S. Government Copyright 1995
'   All rights reserved


'   This script makes a parking shape table.


'   Get the input table.
inName = av.Run("InedDatabase.ChooseTable", {"Make Parking Shape Table", "Parking"})
if (inName = NIL) then
    exit
end
inTableName = av.Run("InedDatabase.GetTableName", {"Parking", inName})

'   Get the node table.
nodeName = av.Run("InedDatabase.ChooseTable", {"Choose Node Table", "Node"})
if (nodeName = NIL) then
    exit
end
nodeTableName = av.Run("InedDatabase.GetTableName", {"Node", nodeName})

'   Get the link table.
linkName = av.Run("InedDatabase.ChooseTable", {"Choose Link Table", "Link"})
if (linkName = NIL) then
    exit
end
linkTableName = av.Run("InedDatabase.GetTableName", {"Link", linkName})

'   Build joined table.
_directory.ExecuteSQL(
        "CREATE TABLE AV_INED_TEMP AS" + 13.AsChar +
        "  SELECT P.*," + 13.AsChar +
        "      A.ABSCISSA + (B.ABSCISSA - A.ABSCISSA) * P.OFFSET / (L.LENGTH + 0.01) AS
ABSCISSA, " + 13.AsChar +
        "      A.ORDINATE + (B.ORDINATE - A.ORDINATE) * P.OFFSET / (L.LENGTH + 0.01) AS
ORDINATE" + 13.AsChar +
        "    FROM """ + inTableName + """ P, """ + linkTableName + """ L, """ +
nodeTableName + """ A, """ + nodeTableName + """ B" + 13.AsChar +
        "    WHERE P.LINK = L.ID AND P.NODE = A.ID AND L.NODEA = A.ID AND L.NODEB = B.ID"
+ 13.AsChar +
        "  UNION SELECT P.*," + 13.AsChar +
        "      B.ABSCISSA + (A.ABSCISSA - B.ABSCISSA) * P.OFFSET / (L.LENGTH + 0.01) AS
ABSCISSA, " + 13.AsChar +
        "      B.ORDINATE + (A.ORDINATE - B.ORDINATE) * P.OFFSET / (L.LENGTH + 0.01) AS
ORDINATE" + 13.AsChar +
        "    FROM """ + inTableName + """ P, """ + linkTableName + """ L, """ +
nodeTableName + """ A, """ + nodeTableName + """ B" + 13.AsChar +
        "    WHERE P.LINK = L.ID AND P.NODE = B.ID AND L.NODEA = A.ID AND L.NODEB = B.ID")
inTable = VTab.MakeSQL(_directory, "SELECT * FROM AV_INED_TEMP")
_directory.ExecuteSQL("DROP TABLE AV_INED_TEMP")
if (inTable.HasError) then
    MsgBox.Error("Cannot join parkings, nodes, and links.", "Make Parking Shape Table")
    exit
end

'   Get the fields.
copyFields = inTable.GetFields.clone
xField = inTable.FindField("ABSCISSA")
yField = inTable.FindField("ORDINATE")

'   Get the output table.
response = FileDialog.Put("parkings.shp".AsFileName, "*.shp", "Make Parking Shape Table")
if (response = NIL) then
    exit
```

```
end
outTable = FTab.MakeNew(response, Point)

'   Generate the points.
av.Run("InedNetwork.GeneratePoints", {inTable, outTable, copyFields, xField, yField})
outTable.Flush
```

3. InedGeography.MakeLinks.ave

```
'   Project: TRANSIMS
'   Subsystem: Input Editor
'   $RCSfile: InedGeography.MakeLinks.ave,v $
'   $Revision: 1.1 $
'   $Date: 1997/02/27 14:25:33 $
'   $State: Rel $
'   $Author: bwb $
'   U.S. Government Copyright 1995
'   All rights reserved


'   This script adds geographic attributes to a link table.


'   Get the input tables.
nodeDoc = av.GetProject.FindDoc("Nodes")
if (nodeDoc = NIL) then
    MsgBox.Error("Node table not found.", "Add Link Shapes")
    exit
end
nodeVTab = nodeDoc.GetVTab

'   Get the input file.
linkFile = FileDialog.Show("*.shp", "*.shp", "Link Shape File")
if (linkFile = NIL) then
    exit
end
linkFTab = FTab.Make(SrcName.Make(linkFile.AsString))
linkFTab.SetEditable(TRUE)

'   Retrieve the fields.
idField = nodeVTab.FindField("ID")
abscissaField = nodeVTab.FindField("ABSCISSA")
ordinateField = nodeVTab.FindField("ORDINATE")
shapeField = linkFTab.FindField("Shape")
nodeaField = linkFTab.FindField("NODEA")
nodebField = linkFTab.FindField("NODEB")

'   Generate the points.
points = Dictionary.Make(nodeVTab.GetNumRecords)
for each i in nodeVTab
    id = nodeVTab.ReturnValueNumber(idField, i)
    x = nodeVTab.ReturnValueNumber(abscissaField, i)
    y = nodeVTab.ReturnValueNumber(ordinateField, i)
    points.Add(id, Point.Make(x, y))
end

'   Show the abort button.
av.ShowStopButton

'   Create the points.
count = 0
for each i in linkFTab

    '   Make the point.
    a = points.Get(linkFTab.ReturnValueNumber(nodeaField, i))
    b = points.Get(linkFTab.ReturnValueNumber(nodebField, i))
    linkFTab.SetValue(shapeField, i, PolyLine.Make({{a, b}}))

    '   Update the status.
    count = count + 1
    if (av.SetStatus(100 * count / linkFTab.GetNumRecords).Not) then
        exit
    end

end
```

```
'   Done.
av.SetStatus(100)
linkFTab.SetEditable(FALSE)
linkFTab.Flush
```

4. InedGeography.MakeNodes.ave

```
'   Project: TRANSIMS
'   Subsystem: Input Editor
'   $RCSfile: InedGeography.MakeNodes.ave,v $
'   $Revision: 1.1 $
'   $Date: 1997/02/27 14:25:33 $
'   $State: Rel $
'   $Author: bwb $
'   U.S. Government Copyright 1995
'   All rights reserved


'   This script adds geographic attributes to a node table.


'   Get the input file.
nodeFile = FileDialog.Show("*.shp", "*.shp", "Node Shape File")
if (nodeFile = NIL) then
    exit
end
nodeFTab = FTab.Make(SrcName.Make(nodeFile.AsString))
nodeFTab.SetEditable(TRUE)

'   Retrieve the fields.
shapeField = nodeFTab.FindField("Shape")
abscissaField = nodeFTab.FindField("ABSCISSA")
ordinateField = nodeFTab.FindField("ORDINATE")

'   Show the abort button.
av.ShowStopButton

'   Create the points.
count = 0
for each i in nodeFTab

    '   Make the point.
    x = nodeFTab.ReturnValueNumber(abscissaField, i)
    y = nodeFTab.ReturnValueNumber(ordinateField, i)
    nodeFTab.SetValue(shapeField, i, Point.Make(x, y))

    '   Update the status.
    count = count + 1
    if (av.SetStatus(100 * count / nodeFTab.GetNumRecords).Not) then
        exit
    end

end

'   Done.
av.SetStatus(100)
nodeFTab.SetEditable(FALSE)
nodeFTab.Flush
```

5. InedGeography.MakeShapeFile.ave

```
'   Project: TRANSIMS
'   Subsystem: Input Editor
'   $RCSfile: InedGeography.MakeShapeFile.ave,v $
'   $Revision: 1.1 $
'   $Date: 1997/02/27 14:25:33 $
'   $State: Rel $
'   $Author: bwb $
'   U.S. Government Copyright 1995
'   All rights reserved


'   This script makes a DBF file into a SHP file.
```

```
'   Get the input file.
inFile = FileDialog.Show("*.dbf", "*.dbf", "Input Table for Making Shape File")
if (inFile = NIL) then
    exit
end
inVTab = VTab.Make(inFile, FALSE, FALSE)

'   Get type of geographic object.
shapeClass = MsgBox.ListAsString({"Point", "PolyLine", "Polygon"}, "Type of shape:", "Make
Shape File")
if (shapeClass = NIL) then
    exit
elseif (shapeClass = "Point") then
    shapeClass = Point
    shapePrototype = Point.MakeNull
elseif (shapeClass = "PolyLine") then
    shapeClass = PolyLine
    shapePrototype = PolyLine.MakeNull
elseif (shapeClass = "Polygon") then
    shapeClass = Polygon
    shapePrototype = Polygon.MakeNull
end

'   Get the output table.
outFile = inFile.Clone
outFile.SetExtension("shp")
outFile = FileDialog.Put(outFile, "*.shp", "Make Shape Table")
if (outFile = NIL) then
    exit
end
outFTab = FTab.MakeNew(outFile, shapeClass)

'   Get the fields.
inFields = inVTab.GetFields
outFTab.AddFields(inFields.DeepClone)
xrefFields = Dictionary.Make(inFields.Count)
for each fld in inFields
    xrefFields.Add(fld, outFTab.FindField(fld.GetAlias))
end
shapeField = outFTab.FindField("Shape")

'   Show the abort button.
av.ShowStopButton

'  Cycle through records.
count = 0
for each inRecord in inVTab

    '   Add record.
    outRecord = outFTab.AddRecord

    '   Create shapes.
    outFTab.SetValue(shapeField, outRecord, shapePrototype)

    '   Copy attributes.
    for each fld in xrefFields.ReturnKeys
        outFTab.SetValue(xrefFields.Get(fld), outRecord, inVTab.ReturnValue(fld,
inRecord))
    end

    '   Update the status.
    count = count + 1
    if (av.SetStatus(100 * count / inVTab.GetNumRecords).Not) then
        exit
    end

end

'   Done.
av.SetStatus(100)
outFTab.Flush
```

E. Intersection Viewing Scripts

1. InedIntersection.NextPhase.ave

```
'   Project: TRANSIMS
'   Subsystem: Input Editor
'   $RCSfile: InedIntersection.NextPhase.ave,v $
'   $Revision: 1.0 $
'   $Date: 1996/04/25 18:20:38 $
'   $State: Rel $
'   $Author: bwb $
'   U.S. Government Copyright 1995
'   All rights reserved


'   This script sets the phase for analysis to the next phase.


'   Choose the next phase.
phase = MsgBox.ListAsString(_phases.Get(_phase), "Choose a phase after phase " +
_phase.AsString + ":", "Next Phase")
if (phase <> NIL) then
    _phase = phase
end
```

2. InedIntersection.PreviousPhase.ave

```
'   Project: TRANSIMS
'   Subsystem: Input Editor
'   $RCSfile: InedIntersection.PreviousPhase.ave,v $
'   $Revision: 1.0 $
'   $Date: 1996/04/25 18:20:38 $
'   $State: Rel $
'   $Author: bwb $
'   U.S. Government Copyright 1995
'   All rights reserved


'   This script sets the phase for analysis to the previous phase.


'   Choose the previous phase.
possible = List.Make
for each phase in _phases.ReturnKeys
    if (_phases.Get(phase).FindByValue(_phase) <> -1) then
        possible.Add(phase)
    end
end
phase = MsgBox.ListAsString(possible, "Choose a phase before phase " + _phase.AsString +
":", "Next Phase")
if (phase <> NIL) then
    _phase = phase
end
```

3. InedIntersection.RestoreLanes.ave

```
'   Project: TRANSIMS
'   Subsystem: Input Editor
'   $RCSfile: InedIntersection.RestoreLanes.ave,v $
'   $Revision: 1.0 $
'   $Date: 1996/04/25 18:20:38 $
'   $State: Rel $
'   $Author: bwb $
'   U.S. Government Copyright 1995
'   All rights reserved


'   This script restores a lane selection.


'   Get the input tables.
theView = av.GetActiveDoc
if (theView = NIL) then
```

```
    MsgBox.Error("Benchmark network not found.", "Restore Lane Selection")
    exit
end
laneTheme = theView.FindTheme("Lanes")
if (laneTheme = NIL) then
    MsgBox.Error("Lane table not found.", "Restore Lane Selection")
    exit
end
laneFTab = laneTheme.GetFTab

'   Save the selection.
laneFTab.SetSelection(_laneSelection.Clone)
laneFTab.UpdateSelection
```

4. InedIntersection.SaveLanes.ave

```
'   Project: TRANSIMS
'   Subsystem: Input Editor
'   $RCSfile: InedIntersection.SaveLanes.ave,v $
'   $Revision: 1.0 $
'   $Date: 1996/04/25 18:20:38 $
'   $State: Rel $
'   $Author: bwb $
'   U.S. Government Copyright 1995
'   All rights reserved


'   This script saves a lane selection.


'   Get the input tables.
theView = av.GetActiveDoc
if (theView = NIL) then
    MsgBox.Error("Benchmark network not found.", "Save Lane Selection")
    exit
end
laneTheme = theView.FindTheme("Lanes")
if (laneTheme = NIL) then
    MsgBox.Error("Lane table not found.", "Save Lane Selection")
    exit
end
laneFTab = laneTheme.GetFTab

'   Save the selection.
_laneSelection = laneFTab.GetSelection.Clone
```

5. InedIntersection.SetIntersection.ave

```
'   Project: TRANSIMS
'   Subsystem: Input Editor
'   $RCSfile: InedIntersection.SetIntersection.ave,v $
'   $Revision: 1.0 $
'   $Date: 1996/04/25 18:20:38 $
'   $State: Rel $
'   $Author: bwb $
'   U.S. Government Copyright 1995
'   All rights reserved


'   This script sets the intersection for analysis.


'   Get the input tables.
theView = av.GetActiveDoc
if (theView = NIL) then
    MsgBox.Error("Benchmark network not found.", "Set Intersection")
    exit
end
nodeTheme = theView.FindTheme("Nodes")
if (nodeTheme = NIL) then
    MsgBox.Error("Node table not found.", "Set Intersection")
    exit
end
nodeFTab = nodeTheme.GetFTab
connectivityDoc = av.GetProject.FindDoc("Lane Connectivity")
```

```
if (connectivityDoc = NIL) then
    MsgBox.Error("Lane connectivity table not found.", "Set Intersection")
    exit
end
connectivityVTab = connectivityDoc.GetVTab
unsignalizedDoc = av.GetProject.FindDoc("Unsignalized Nodes")
if (unsignalizedDoc = NIL) then
    MsgBox.Error("Unsignalized node table not found.", "Set Intersection")
    exit
end
unsignalizedVTab = unsignalizedDoc.GetVTab
signalizedDoc = av.GetProject.FindDoc("Signalized Nodes")
if (signalizedDoc = NIL) then
    MsgBox.Error("Signalized node table not found.", "Set Intersection")
    exit
end
signalizedVTab = signalizedDoc.GetVTab
phasingDoc = av.GetProject.FindDoc("Phasing Plans")
if (phasingDoc = NIL) then
    MsgBox.Error("Phasing plans table not found.", "Set Intersection")
    exit
end
phasingVTab = phasingDoc.GetVTab
timingDoc = av.GetProject.FindDoc("Timing Plans")
if (timingDoc = NIL) then
    MsgBox.Error("Timing plan table not found.", "Set Intersection")
    exit
end
timingVTab = timingDoc.GetVTab

'   Find the selected node.
selection = nodeFTab.GetSelection
if (selection.Count <> 1) then
    MsgBox.Error("One and only one node must be selected.", "Set Intersection")
    exit
end
for each i in selection
    _node = nodeFTab.ReturnValueNumber(nodeFTab.FindField("ID"), i)
end

'   Collect the lane connectivity.
selection = connectivityVTab.GetSelection
connectivityVTab.Query("[NODE] = " + _node.SetFormat("d").AsString, selection,
#VTAB_SELTYPE_NEW)
connectivityVTab.UpdateSelection
_connections = List.Make
for each i in selection
    inlink = connectivityVTab.ReturnValueNumber(connectivityVTab.FindField("INLINK"), i)
    inlane = connectivityVTab.ReturnValueNumber(connectivityVTab.FindField("INLANE"), i)
    outlink = connectivityVTab.ReturnValueNumber(connectivityVTab.FindField("OUTLINK"), i)
    outlane = connectivityVTab.ReturnValueNumber(connectivityVTab.FindField("OUTLANE"), i)
    _connections.Add({inlink, inlane, outlink, outlane})
end

'   Find the plan.
selection = signalizedVTab.GetSelection
signalizedVTab.Query("[NODE] = " + _node.SetFormat("d").AsString, selection,
#VTAB_SELTYPE_NEW)
signalizedVTab.UpdateSelection
for each i in selection
    _plan = signalizedVTab.ReturnValueNumber(signalizedVTab.FindField("PLAN"), i)
end

'   Collect the phase information.
_phase = 0
selection = timingVTab.GetSelection
timingVTab.Query("[PLAN] = " + _plan.AsString, selection, #VTAB_SELTYPE_NEW)
timingVTab.UpdateSelection
_phases = Dictionary.Make(selection.Count)
for each i in selection
    _phase = timingVTab.ReturnValueNumber(timingVTab.FindField("PHASE"), i)
    nextphases = List.Make
    for each j in timingVTab.ReturnValueString(timingVTab.FindField("NEXTPHASES"),
i).AsTokens("/")
        nextphases.Add(j.AsNumber)
    end
```

```
    _phases.Set(_phase, nextphases)
end

'   Collect the allowed movement information for signalized intersections.
selection = phasingVTab.GetSelection
phasingVTab.Query("([NODE] = " + _node.SetFormat("d").AsString + ") and ([PLAN] = " +
_plan.AsString + ")", selection, #VTAB_SELTYPE_NEW)
phasingVTab.UpdateSelection
_movements = Dictionary.Make(_phases.GetSize + 1)
for each i in selection
    phase = phasingVTab.ReturnValueNumber(phasingVTab.FindField("PHASE"), i)
    inlink = phasingVTab.ReturnValueNumber(phasingVTab.FindField("INLINK"), i)
    outlink = phasingVTab.ReturnValueNumber(phasingVTab.FindField("OUTLINK"), i)
    protection = phasingVTab.ReturnValueString(phasingVTab.FindField("PROTECTION"), i)
    if (_movements.Get(phase) = NIL) then
        _movements.Set(phase, List.Make)
    end
    for each connection in _connections
        if ((connection.Get(0) = inlink) and (connection.Get(2) = outlink)) then
            _movements.Get(phase).Add({inlink, connection.Get(1), outlink,
connection.Get(3), protection})
        end
    end
end

'   Collect the allowed movement information for unsignalized intersections.
selection = unsignalizedVTab.GetSelection
unsignalizedVTab.Query("[NODE] = " + _node.SetFormat("d").AsString, selection,
#VTAB_SELTYPE_NEW)
unsignalizedVTab.UpdateSelection
_movements.Set(0, List.Make)
for each i in selection
    _phases.Set(0, {0})
    inlink = unsignalizedVTab.ReturnValueNumber(unsignalizedVTab.FindField("INLINK"), i)
    sign = unsignalizedVTab.ReturnValueString(unsignalizedVTab.FindField("SIGN"), i)
    for each connection in _connections
        if (connection.Get(0) = inlink) then
            _movements.Get(0).Add({inlink, connection.Get(1), connection.Get(2),
connection.Get(3), sign})
        end
    end
end

'   Set the default protections and signs.
_protections = {"S", "Y", "N", "P", "U"}

'   Done.
MsgBox.Info("The node " + _node.SetFormat("d").AsString + " is now the current
intersection.", "Set Intersection")
```

6. InedIntersection.SetPhase.ave

```
'   Project: TRANSIMS
'   Subsystem: Input Editor
'   $RCSfile: InedIntersection.SetPhase.ave,v $
'   $Revision: 1.0 $
'   $Date: 1996/04/25 18:20:38 $
'   $State: Rel $
'   $Author: bwb $
'   U.S. Government Copyright 1995
'   All rights reserved


'   This script sets the phase for analysis.


'   Ask for the phase.
phase = MsgBox.ListAsString(_phases.ReturnKeys, "Phase:", "Set Phase")
if (phase <> NIL) then
    _phase = phase
end
```

7. InedIntersection.SetProtections.ave

```
'   Project: TRANSIMS
```

```
'   Subsystem: Input Editor
'   $RCSfile: InedIntersection.SetProtections.ave,v $
'   $Revision: 1.0 $
'   $Date: 1996/04/25 18:20:38 $
'   $State: Rel $
'   $Author: bwb $
'   U.S. Government Copyright 1995
'   All rights reserved


'   This script sets the protections and signs for analysis.


'   Ask for the phase.
answer = MsgBox.MultiInput("Choose the protections and signs of interest:", "Set
Protections/Signs", {"Protection codes:", "Sign codes:"}, {"PU", "SYN"})
if (answer = NIL) then
    exit
end
_protections = List.Make
for each i in 0..(answer.Get(0).Count - 1)
    c = answer.Get(0).Middle(i, 1)
    if ((c = "P") or (c = "U")) then
        _protections.Add(c)
    else
        MsgBox.Warning("""" + c + """ is not a valid protection code.", "Set
Protections/Signs")
    end
end
for each i in 0..(answer.Get(1).Count - 1)
    c = answer.Get(1).Middle(i, 1)
    if ((c = "S") or (c = "Y") or (c = "N")) then
        _protections.Add(c)
    else
        MsgBox.Warning("""" + c + """ is not a valid sign code.", "Set Protections/Signs")
    end
end
```

8. InedIntersection.ShowIncoming.ave

```
'   Project: TRANSIMS
'   Subsystem: Input Editor
'   $RCSfile: InedIntersection.ShowIncoming.ave,v $
'   $Revision: 1.0 $
'   $Date: 1996/04/25 18:20:38 $
'   $State: Rel $
'   $Author: bwb $
'   U.S. Government Copyright 1995
'   All rights reserved


'   This script shows the incoming movements for the selected lanes.


'   Get the input tables.
theView = av.GetActiveDoc
if (theView = NIL) then
    MsgBox.Error("Benchmark network not found.", "Show Incoming Lanes")
    exit
end
laneTheme = theView.FindTheme("Lanes")
if (laneTheme = NIL) then
    MsgBox.Error("Lane table not found.", "Show Incoming Lanes")
    exit
end
laneFTab = laneTheme.GetFTab

'   Find the movements that are allowed.
selection = laneFTab.GetSelection
allowed = List.Make
for each i in selection
    if (laneFTab.ReturnValueNumber(laneFTab.FindField("From"), i) <> _node) then
        continue
    end
    outlink = laneFTab.ReturnValueNumber(laneFTab.FindField("Link"), i)
```

```
    outlane = laneFTab.ReturnValueNumber(laneFTab.FindField("Lane"), i)
    for each movement in _movements.Get(_phase)
        if ((movement.Get(2) = outlink) and (movement.Get(3) = outlane)) then
            allowed.Add(movement)
        end
    end
end

'   Select the allowed movements.
selection = laneFTab.GetSelection
selection.ClearAll
possible = selection.Clone
laneFTab.Query("[Towards] = " + _node.SetFormat("d").AsString, possible,
#VTAB_SELTYPE_NEW)
for each i in possible
    inlink = laneFTab.ReturnValueNumber(laneFTab.FindField("Link"), i)
    inlane = laneFTab.ReturnValueNumber(laneFTab.FindField("Lane"), i)
    for each movement in allowed
        if ((movement.Get(0) = inlink) and (movement.Get(1) = inlane) and
(_protections.FindByValue(movement.Get(4)) <> -1)) then
            selection.Set(i)
        end
    end
end
laneFTab.UpdateSelection
```

9. InedIntersection.ShowIntersection.ave

```
'   Project: TRANSIMS
'   Subsystem: Input Editor
'   $RCSfile: InedIntersection.ShowIntersection.ave,v $
'   $Revision: 1.0 $
'   $Date: 1996/04/25 18:20:38 $
'   $State: Rel $
'   $Author: bwb $
'   U.S. Government Copyright 1995
'   All rights reserved


'   This script shows the intersection for analysis.


'   Show the intersection.
MsgBox.Info("The current intersection is at node " + _node.SetFormat("d").AsString + ".",
"Show Intersection")
```

10. InedIntersection.ShowOutGoing.ave

```
'   Project: TRANSIMS
'   Subsystem: Input Editor
'   $RCSfile: InedIntersection.ShowOutgoing.ave,v $
'   $Revision: 1.0 $
'   $Date: 1996/04/25 18:20:38 $
'   $State: Rel $
'   $Author: bwb $
'   U.S. Government Copyright 1995
'   All rights reserved


'   This script shows the outgoing movements for the selected lanes.


'   Get the input tables.
theView = av.GetActiveDoc
if (theView = NIL) then
    MsgBox.Error("Benchmark network not found.", "Show Outgoing Lanes")
    exit
end
laneTheme = theView.FindTheme("Lanes")
if (laneTheme = NIL) then
    MsgBox.Error("Lane table not found.", "Show Outgoing Lanes")
    exit
end
laneFTab = laneTheme.GetFTab
```

```
'   Find the movements that are allowed.
selection = laneFTab.GetSelection
allowed = List.Make
for each i in selection
    if (laneFTab.ReturnValueNumber(laneFTab.FindField("Towards"), i) <> _node) then
        continue
    end
    inlink = laneFTab.ReturnValueNumber(laneFTab.FindField("Link"), i)
    inlane = laneFTab.ReturnValueNumber(laneFTab.FindField("Lane"), i)
    for each movement in _movements.Get(_phase)
        if ((movement.Get(0) = inlink) and (movement.Get(1) = inlane)) then
            allowed.Add(movement)
        end
    end
end

'   Select the allowed movements.
selection = laneFTab.GetSelection
selection.ClearAll
possible = selection.Clone
laneFTab.Query("[From] = " + _node.SetFormat("d").AsString, possible, #VTAB_SELTYPE_NEW)
for each i in possible
    outlink = laneFTab.ReturnValueNumber(laneFTab.FindField("Link"), i)
    outlane = laneFTab.ReturnValueNumber(laneFTab.FindField("Lane"), i)
    for each movement in allowed
        if ((movement.Get(2) = outlink) and (movement.Get(3) = outlane) and
(_protections.FindByValue(movement.Get(4)) <> -1)) then
            selection.Set(i)
        end
    end
end
laneFTab.UpdateSelection
```

11. InedIntersection.ShowPhase.ave

```
'   Project: TRANSIMS
'   Subsystem: Input Editor
'   $RCSfile: InedIntersection.ShowPhase.ave,v $
'   $Revision: 1.0 $
'   $Date: 1996/04/25 18:20:38 $
'   $State: Rel $
'   $Author: bwb $
'   U.S. Government Copyright 1995
'   All rights reserved


'   This script shows the phase for analysis.


'   Show the phase.
MsgBox.Info("The current phase is " + _phase.AsString + ".", "Show Phase")
```

12. InedIntersection.ShowProtection.ave

```
'   Project: TRANSIMS
'   Subsystem: Input Editor
'   $RCSfile: InedIntersection.ShowProtections.ave,v $
'   $Revision: 1.0 $
'   $Date: 1996/04/25 18:20:38 $
'   $State: Rel $
'   $Author: bwb $
'   U.S. Government Copyright 1995
'   All rights reserved


'   This script shows the protections and signs for analysis.


'   Show the protections and signs.
protections =  ""
for each c in _protections
    protections = protections + c
end
MsgBox.Info("The current protection and sign codes are """ + protections + """.", "Show
Protections/Signs")
```

13. InedNetwork.GenerateLines.ave

```
'   Project: TRANSIMS
'   Subsystem: Input Editor
'   $RCSfile: InedNetwork.GenerateLines.ave,v $
'   $Revision: 1.1 $
'   $Date: 1997/02/27 14:25:33 $
'   $State: Rel $
'   $Author: bwb $
'   U.S. Government Copyright 1995
'   All rights reserved


'   This script makes lines for a table.  The first parameter is the input VTab, the
second parameter is the output FTab, the third parameter is a list of fields
'   for which to copy attribute data, the fourth parameter is the name of the x0 field,
the fifth parameter is the name of the y0 field, the sixth parameter is the
'   name of the x0 field, and the seventh parameter is the name of the y1 field.


'   Set up aliases.
inTable = SELF.Get(0)
outTable = SELF.Get(1)

'   Get the fields.
inFields = SELF.Get(2)
outTable.AddFields(inFields.DeepClone)
xrefFields = Dictionary.Make(inFields.Count)
for each fld in inFields
    xrefFields.Add(fld, outTable.FindField(fld.GetAlias))
end
x0Field = SELF.Get(3)
y0Field = SELF.Get(4)
x1Field = SELF.Get(5)
y1Field = SELF.Get(6)
shapeField = outTable.FindField("Shape")

'   Show the abort button.
av.ShowStopButton

'  Cycle through records.
for each inRecord in inTable

    '   Add record.
    outRecord = outTable.AddRecord

    '   Create line.
    pts = List.Make
    pts.Add(Point.Make(inTable.ReturnValue(x0Field, inRecord),
inTable.ReturnValue(y0Field, inRecord)))
    pts.Add(Point.Make(inTable.ReturnValue(x1Field, inRecord),
inTable.ReturnValue(y1Field, inRecord)))
    outTable.SetValue(shapeField, outRecord, PolyLine.Make({pts}))

    '   Copy attributes.
    for each fld in xrefFields.ReturnKeys
        outTable.SetValue(xrefFields.Get(fld), outRecord, inTable.ReturnValue(fld,
inRecord))
    end

    '   Update the status.
    more = av.SetStatus(100 * inRecord / inTable.GetNumRecords)
    if (not more) then
        exit
    end

end

'   Done.
av.SetStatus(100)
outTable.Flush
```

F. Network Data Table Scripts

1. InedNetwork.GeneratePoints.ave

```
'   Project: TRANSIMS
'   Subsystem: Input Editor
'   $RCSfile: InedNetwork.GeneratePoints.ave,v $
'   $Revision: 1.1 $
'   $Date: 1997/02/27 14:25:33 $
'   $State: Rel $
'   $Author: bwb $
'   U.S. Government Copyright 1995
'   All rights reserved


'   This script makes points for a table.  The first parameter is the input VTab, the
second parameter is the output FTab, the third parameter is a list of fields for which to
'   copy attribute data, the fourth parameter is the name of the x field, and the fifth
parameter is the name of the y field.


'   Set up aliases.
inTable = SELF.Get(0)
outTable = SELF.Get(1)

'   Get the fields.
inFields = SELF.Get(2)
outTable.AddFields(inFields.DeepClone)
xrefFields = Dictionary.Make(inFields.Count)
for each fld in inFields
    xrefFields.Add(fld, outTable.FindField(fld.GetAlias))
end
xField = SELF.Get(3)
yField = SELF.Get(4)
shapeField = outTable.FindField("Shape")

'   Show the abort button.
av.ShowStopButton

'  Cycle through records.
for each inRecord in inTable

    '   Add record.
    outRecord = outTable.AddRecord

    '   Create point.
    outTable.SetValue(shapeField, outRecord, Point.Make(inTable.ReturnValue(xField,
inRecord), inTable.ReturnValue(yField, inRecord)))

    '   Copy attributes.
    for each fld in xrefFields.ReturnKeys
        outTable.SetValue(xrefFields.Get(fld), outRecord, inTable.ReturnValue(fld,
inRecord))
    end

    '   Update the status.
    more = av.SetStatus(100 * inRecord / inTable.GetNumRecords)
    if (not more) then
        exit
    end

end

'   Done.
av.SetStatus(100)
outTable.Flush
```

2. InedNetwork.GenerateRectangles.ave

```
'   Project: TRANSIMS
'   Subsystem: Input Editor
'   $RCSfile: InedNetwork.GenerateRectangles.ave,v $
'   $Revision: 1.1 $
'   $Date: 1997/02/27 14:25:33 $
```

```
'   $State: Rel $
'   $Author: bwb $
'   U.S. Government Copyright 1995
'   All rights reserved


'   This script makes rectangles for a table.  The first parameter is the input VTab, the
second parameter is the output FTab, the third parameter is a list of fields for
'   which to copy attribute data, the fourth parameter is the name of the x0 field, the
fifth parameter is the name of the y0 field, the sixth parameter is the name of the x0
'   field, the seventh parameter is the name of the y1 field, etc.


'   Set up aliases.
inTable = SELF.Get(0)
outTable = SELF.Get(1)

'   Get the fields.
inFields = SELF.Get(2)
outTable.AddFields(inFields.DeepClone)
xrefFields = Dictionary.Make(inFields.Count)
for each fld in inFields
    xrefFields.Add(fld, outTable.FindField(fld.GetAlias))
end
x0Field = SELF.Get(3)
y0Field = SELF.Get(4)
x1Field = SELF.Get(5)
y1Field = SELF.Get(6)
x2Field = SELF.Get(7)
y2Field = SELF.Get(8)
x3Field = SELF.Get(9)
y3Field = SELF.Get(10)
shapeField = outTable.FindField("Shape")

'   Show the abort button.
av.ShowStopButton

'  Cycle through records.
for each inRecord in inTable

    '   Add record.
    outRecord = outTable.AddRecord

    '   Create line.
    pts = List.Make
    pts.Add(Point.Make(inTable.ReturnValue(x0Field, inRecord),
inTable.ReturnValue(y0Field, inRecord)))
    pts.Add(Point.Make(inTable.ReturnValue(x1Field, inRecord),
inTable.ReturnValue(y1Field, inRecord)))
    pts.Add(Point.Make(inTable.ReturnValue(x2Field, inRecord),
inTable.ReturnValue(y2Field, inRecord)))
    pts.Add(Point.Make(inTable.ReturnValue(x3Field, inRecord),
inTable.ReturnValue(y3Field, inRecord)))
    outTable.SetValue(shapeField, outRecord, Polygon.Make({pts}))

    '   Copy attributes.
    for each fld in xrefFields.ReturnKeys
        outTable.SetValue(xrefFields.Get(fld), outRecord, inTable.ReturnValue(fld,
inRecord))
    end

    '   Update the status.
    more = av.SetStatus(100 * inRecord / inTable.GetNumRecords)
    if (not more) then
        exit
    end

end

'   Done.
av.SetStatus(100)
outTable.Flush
```

3. InedNetwork.MakeLanes.ave

```
'   Project: TRANSIMS
'   Subsystem: Input Editor
'   $RCSfile: InedNetwork.MakeLanes.ave,v $
'   $Revision: 1.3 $
'   $Date: 1997/02/27 14:25:33 $
'   $State: Rel $
'   $Author: bwb $
'   U.S. Government Copyright 1995
'   All rights reserved


'   This script generates lane shapes from data tables.


'   Get the tables.
pocketName = av.Run("InedDatabase.ChooseTable", {"Make Lane Shape Table", "Pocket Lane"})
if (pocketName = NIL) then
    exit
end
pocketTableName = av.Run("InedDatabase.GetTableName", {"Pocket Lane", pocketName})
nodeName = av.Run("InedDatabase.GetMaster" , {"Node", "Pocket Lane", pocketName})
if (nodeName = NIL) then
    exit
end
nodeTableName = av.Run("InedDatabase.GetTableName", {"Node", nodeName})
linkName = av.Run("InedDatabase.GetMaster", {"Link", "Pocket Lane", pocketName})
if (linkName = NIL) then
    exit
end
linkTableName = av.Run("InedDatabase.GetTableName", {"Link", linkName})

'   Get the output table.
response = FileDialog.Put("lanes.shp".AsFileName, "*.shp", "Make Lane Shape Table")
if (response = NIL) then
    exit
end

'   Get the offset for lanes.
offset = MsgBox.Input("Lane offset/width in coordinate units:", "Create Lane Shape File",
"3.25")
if (offset = NIL) then
    exit
end
if ((offset.AsNumber > 0).Not) then
    MsgBox.Error("Invalid lane offset.", "Make Lane Shape Table")
    exit
end
laneWidth = offset

'  Make the link table.
sql = ""
sql = sql + "CREATE TABLE AV_INED_TEMP AS" + 13.AsChar
sql = sql + "  SELECT" + 13.AsChar
sql = sql + "      L.ID," + 13.AsChar
sql = sql + "      L.NODEA," + 13.AsChar
sql = sql + "      L.NODEB," + 13.AsChar
sql = sql + "      L.PERMLANESA," + 13.AsChar
sql = sql + "      L.PERMLANESB," + 13.AsChar
sql = sql + "      L.LEFTPCKTSA," + 13.AsChar
sql = sql + "      L.LEFTPCKTSB," + 13.AsChar
sql = sql + "      L.RGHTPCKTSA," + 13.AsChar
sql = sql + "      L.RGHTPCKTSB," + 13.AsChar
sql = sql + "      L.TWOWAYTURN," + 13.AsChar
sql = sql + "      L.LENGTH," + 13.AsChar
sql = sql + "      L.SETBACKA," + 13.AsChar
sql = sql + "      L.SETBACKB," + 13.AsChar
sql = sql + "      A.ABSCISSA AS A_X," + 13.AsChar
sql = sql + "      A.ORDINATE AS A_Y," + 13.AsChar
sql = sql + "      B.ABSCISSA AS B_X," + 13.AsChar
sql = sql + "      B.ORDINATE AS B_Y," + 13.AsChar
sql = sql + "      (B.ABSCISSA - A.ABSCISSA) AS D_X," + 13.AsChar
sql = sql + "      (B.ORDINATE - A.ORDINATE) AS D_Y," + 13.AsChar
sql = sql + "      SQRT((B.ABSCISSA - A.ABSCISSA) * (B.ABSCISSA - A.ABSCISSA) +
(B.ORDINATE - A.ORDINATE) * (B.ORDINATE - A.ORDINATE) + 0.01) AS NORM" + 13.AsChar
```

```
sql = sql + "    FROM """ + linkTableName + """ L, """ + nodeTableName + """ A, """ +
nodeTableName + """ B" + 13.AsChar
sql = sql + "    WHERE L.NODEA = A.ID AND L.NODEB = B.ID"
if (_directory.ExecuteSQL(sql).not) then
    MsgBox.Warning("Query failed.", "Make Lane Shape Table")
    _directory.ExecuteSQL("DROP TABLE AV_INED_TEMP")
    exit
end
sql = ""
sql = sql + "CREATE TABLE AV_INED_LINK AS" + 13.AsChar
sql = sql + "  SELECT" + 13.AsChar
sql = sql + "      ID AS LINK," + 13.AsChar
sql = sql + "      NODEA AS NODE," + 13.AsChar
sql = sql + "      PERMLANESA," + 13.AsChar
sql = sql + "      LEFTPCKTSA," + 13.AsChar
sql = sql + "      RGHTPCKTSA," + 13.AsChar
sql = sql + "      TWOWAYTURN," + 13.AsChar
sql = sql + "      SETBACKA," + 13.AsChar
sql = sql + "      SETBACKB," + 13.AsChar
sql = sql + "      LENGTH," + 13.AsChar
sql = sql + "      A_X," + 13.AsChar
sql = sql + "      A_Y," + 13.AsChar
sql = sql + "      B_X," + 13.AsChar
sql = sql + "      B_Y," + 13.AsChar
sql = sql + "      D_X / NORM AS T_X," + 13.AsChar
sql = sql + "      D_Y / NORM AS T_Y," + 13.AsChar
sql = sql + "      - D_Y / NORM AS N_X," + 13.AsChar
sql = sql + "      D_X / NORM AS N_Y" + 13.AsChar
sql = sql + "    FROM AV_INED_TEMP" + 13.AsChar
sql = sql + "  UNION SELECT" + 13.AsChar
sql = sql + "      ID," + 13.AsChar
sql = sql + "      NODEB," + 13.AsChar
sql = sql + "      PERMLANESB," + 13.AsChar
sql = sql + "      LEFTPCKTSB," + 13.AsChar
sql = sql + "      RGHTPCKTSB," + 13.AsChar
sql = sql + "      TWOWAYTURN," + 13.AsChar
sql = sql + "      SETBACKB," + 13.AsChar
sql = sql + "      SETBACKA," + 13.AsChar
sql = sql + "      LENGTH," + 13.AsChar
sql = sql + "      B_X," + 13.AsChar
sql = sql + "      B_Y," + 13.AsChar
sql = sql + "      A_X," + 13.AsChar
sql = sql + "      A_Y," + 13.AsChar
sql = sql + "      - D_X / NORM AS T_X," + 13.AsChar
sql = sql + "      - D_Y / NORM AS T_Y," + 13.AsChar
sql = sql + "      D_Y / NORM AS N_X," + 13.AsChar
sql = sql + "      - D_X / NORM AS N_Y" + 13.AsChar
sql = sql + "    FROM AV_INED_TEMP"
if (_directory.ExecuteSQL(sql).not) then
    MsgBox.Warning("Query failed.", "Make Lane Shape Table")
    MsgBox.Input("", "SQL", sql)
    exit
end
_directory.ExecuteSQL("DROP TABLE AV_INED_TEMP")

'   Make lane table.
sql = ""
sql = sql + "CREATE TABLE AV_INED_LANE AS" + 13.AsChar
sql = sql + "  SELECT" + 13.AsChar
sql = sql + "      L.LINK," + 13.AsChar
sql = sql + "      L.NODE," + 13.AsChar
sql = sql + "      N.I + L.LEFTPCKTSA AS LANE," + 13.AsChar
sql = sql + "      L.A_X + L.T_X * L.SETBACKA + L.N_X * " + laneWidth + " * (N.I +
L.LEFTPCKTSA - 0.5) AS ABSCISSAAI," + 13.AsChar
sql = sql + "      L.A_Y + L.T_Y * L.SETBACKA + L.N_Y * " + laneWidth + " * (N.I +
L.LEFTPCKTSA - 0.5) AS ORDINATEAI," + 13.AsChar
sql = sql + "      L.B_X - L.T_X * L.SETBACKB + L.N_X * " + laneWidth + " * (N.I +
L.LEFTPCKTSA - 0.5) AS ABSCISSABI," + 13.AsChar
sql = sql + "      L.B_Y - L.T_Y * L.SETBACKB + L.N_Y * " + laneWidth + " * (N.I +
L.LEFTPCKTSA - 0.5) AS ORDINATEBI," + 13.AsChar
sql = sql + "      L.A_X + L.T_X * L.SETBACKA + L.N_X * " + laneWidth + " * (N.I +
L.LEFTPCKTSA + 0.5) AS ABSCISSAAO," + 13.AsChar
sql = sql + "      L.A_Y + L.T_Y * L.SETBACKA + L.N_Y * " + laneWidth + " * (N.I +
L.LEFTPCKTSA + 0.5) AS ORDINATEAO," + 13.AsChar
sql = sql + "      L.B_X - L.T_X * L.SETBACKB + L.N_X * " + laneWidth + " * (N.I +
L.LEFTPCKTSA + 0.5) AS ABSCISSABO," + 13.AsChar
```

```
sql = sql + "      L.B_Y - L.T_Y * L.SETBACKB + L.N_Y * " + laneWidth + " * (N.I +
L.LEFTPCKTSA + 0.5) AS ORDINATEBO," + 13.AsChar
sql = sql + "      L.LENGTH - L.SETBACKA - L.SETBACKB AS LENGTH" + 13.AsChar
sql = sql + "    FROM AV_INED_LINK L, AV_INED_NUMBERS N" + 13.AsChar
sql = sql + "    WHERE L.PERMLANESA >= N.I" + 13.AsChar
sql = sql + "  UNION SELECT" + 13.AsChar
sql = sql + "      L.LINK," + 13.AsChar
sql = sql + "      L.NODE," + 13.AsChar
sql = sql + "      P.LANE," + 13.AsChar
sql = sql + "      L.A_X + L.T_X * L.SETBACKA + L.N_X * " + laneWidth + " * (P.LANE -
0.5)," + 13.AsChar
sql = sql + "      L.A_Y + L.T_Y * L.SETBACKA + L.N_Y * " + laneWidth + " * (P.LANE -
0.5)," + 13.AsChar
sql = sql + "      L.A_X + L.T_X * (L.SETBACKA + P.LENGTH) + L.N_X * " + laneWidth + " *
(P.LANE - 0.5)," + 13.AsChar
sql = sql + "      L.A_Y + L.T_Y * (L.SETBACKA + P.LENGTH) + L.N_Y * " + laneWidth + " *
(P.LANE - 0.5)," + 13.AsChar
sql = sql + "      L.A_X + L.T_X * L.SETBACKA + L.N_X * " + laneWidth + " * (P.LANE +
0.5)," + 13.AsChar
sql = sql + "      L.A_Y + L.T_Y * L.SETBACKA + L.N_Y * " + laneWidth + " * (P.LANE +
0.5)," + 13.AsChar
sql = sql + "      L.A_X + L.T_X * (L.SETBACKA + P.LENGTH) + L.N_X * " + laneWidth + " *
(P.LANE + 0.5)," + 13.AsChar
sql = sql + "      L.A_Y + L.T_Y * (L.SETBACKA + P.LENGTH) + L.N_Y * " + laneWidth + " *
(P.LANE + 0.5)," + 13.AsChar
sql = sql + "      P.LENGTH" + 13.AsChar
sql = sql + "    FROM """ + pocketTableName + """ P, AV_INED_LINK L" + 13.AsChar
sql = sql + "    WHERE P.LINK = L.LINK AND P.NODE = L.NODE"
if (_directory.ExecuteSQL(sql).not) then
    MsgBox.Warning("Query failed.", "Make Lane Shape Table")
    _directory.ExecuteSQL("DROP TABLE AV_INED_LINK")
    exit
end
_directory.ExecuteSQL("DROP TABLE AV_INED_LINK")

'   Import the lane data.
inTable = VTab.MakeSQL(_directory, "SELECT * FROM AV_INED_LANE")
_directory.ExecuteSQL("DROP TABLE AV_INED_LANE")
outTable = FTab.MakeNew(response, Polygon)

'   Get the fields.
copyFields = inTable.GetFields.clone
x0Field = inTable.FindField("ABSCISSAAI")
y0Field = inTable.FindField("ORDINATEAI")
x1Field = inTable.FindField("ABSCISSAAO")
y1Field = inTable.FindField("ORDINATEAO")
x2Field = inTable.FindField("ABSCISSABO")
y2Field = inTable.FindField("ORDINATEBO")
x3Field = inTable.FindField("ABSCISSABI")
y3Field = inTable.FindField("ORDINATEBI")


'   Generate the rectangles.
av.Run("InedNetwork.GenerateRectangles", {inTable, outTable, copyFields, x0Field, y0Field,
x1Field, y1Field, x2Field, y2Field, x3Field, y3Field})
outTable.Flush

'   View the result if necessary.
if (MsgBox.YesNo("Do you want to view the result?", "Make Lane Shape Table", TRUE)) then
    outTheme = FTheme.Make(outTable)
    outTheme.SetName("Lanes")
    outView = av.GetActiveDoc
    outView.AddTheme(outTheme)
    outTheme.SetVisible(TRUE)
end
```

4. InedNetwork.MakeLinks.ave

```
'   Project: TRANSIMS
'   Subsystem: Input Editor
'   $RCSfile: InedNetwork.MakeLinks.ave,v $
'   $Revision: 1.1 $
'   $Date: 1997/02/27 14:25:33 $
'   $State: Rel $
'   $Author: bwb $
```

```
'   U.S. Government Copyright 1995
'   All rights reserved


'   This script makes a link shape table.


'   Get the input table.
inName = av.Run("InedDatabase.ChooseTable", {"Make Link Shape Table", "Link"})
if (inName = NIL) then
    exit
end
inTableName = av.Run("InedDatabase.GetTableName", {"Link", inName})

'   Get the node table.
nodeName = av.Run("InedDatabase.GetMaster" , {"Node", "Link", inName})
if (nodeName = NIL) then
    exit
end
nodeTableName = av.Run("InedDatabase.GetTableName", {"Node", nodeName})

'   Build joined table.
_directory.ExecuteSQL(
        "CREATE TABLE AV_INED_TEMP AS" + 13.AsChar +
        "  SELECT L.*, A.ABSCISSA AS ABSCISSAA, A.ORDINATE AS ORDINATEA, B.ABSCISSA AS
ABSCISSAB, B.ORDINATE AS ORDINATEB" + 13.AsChar +
        "    FROM """ + inTableName + """ L, """ + nodeTableName + """ A, """ +
nodeTableName + """ B" + 13.AsChar +
        "    WHERE L.NODEA = A.ID AND L.NODEB = B.ID")
inTable = VTab.MakeSQL(_directory, "SELECT * FROM AV_INED_TEMP")
_directory.ExecuteSQL("DROP TABLE AV_INED_TEMP")
if (inTable.HasError) then
    MsgBox.Error("Cannot join nodes and links.", "Make Link Shape Table")
    exit
end

'   Get the fields.
copyFields = inTable.GetFields.clone
x0Field = inTable.FindField("ABSCISSAA")
y0Field = inTable.FindField("ORDINATEA")
x1Field = inTable.FindField("ABSCISSAB")
y1Field = inTable.FindField("ORDINATEB")

'   Get the output table.
response = FileDialog.Put("links.shp".AsFileName, "*.shp", "Make Link Shape Table")
if (response = NIL) then
    exit
end
outTable = FTab.MakeNew(response, PolyLine)

'   Generate the points.
av.Run("InedNetwork.GenerateLines", {inTable, outTable, copyFields, x0Field, y0Field,
x1Field, y1Field})
outTable.Flush

'   View the result if necessary.
if (MsgBox.YesNo("Do you want to view the result?", "Make Link Shape Table", TRUE)) then
    outTheme = FTheme.Make(outTable)
    outTheme.SetName("Links")
    outView = av.GetActiveDoc
    outView.AddTheme(outTheme)
    outTheme.SetVisible(TRUE)
end
```

5. InedNetwork.MakeNodes.ave

```
'   Project: TRANSIMS
'   Subsystem: Input Editor
'   $RCSfile: InedNetwork.MakeNodes.ave,v $
'   $Revision: 1.1 $
'   $Date: 1997/02/27 14:25:33 $
'   $State: Rel $
'   $Author: bwb $
'   U.S. Government Copyright 1995
'   All rights reserved
```

```
'   This script makes a node shape table.


'   Get the input table.
inName = av.Run("InedDatabase.ChooseTable", {"Make Node Shape Table", "Node"})
if (inName = NIL) then
    exit
end
inTable = av.Run("InedDatabase.GetTable", {"Node", inName})

'   Get the fields.
copyFields = inTable.GetFields.clone
xField = inTable.FindField("ABSCISSA")
yField = inTable.FindField("ORDINATE")

'   Get the output table.
response = FileDialog.Put("nodes.shp".AsFileName, "*.shp", "Make Node Shape Table")
if (response = NIL) then
    exit
end
outTable = FTab.MakeNew(response, Point)

'   Generate the points.
av.Run("InedNetwork.GeneratePoints", {inTable, outTable, copyFields, xField, yField})
outTable.Flush

'   View the result if necessary.
if (MsgBox.YesNo("Do you want to view the result?", "Make Node Shape Table", TRUE)) then
    outTheme = FTheme.Make(outTable)
    outTheme.SetName("Nodes")
    outView = av.GetActiveDoc
    outView.AddTheme(outTheme)
    outTheme.SetVisible(TRUE)
end
```

6. InedNetwork.MakeParkings.ave

```
'   Project: TRANSIMS
'   Subsystem: Input Editor
'   $RCSfile: InedNetwork.MakeParkings.ave,v $
'   $Revision: 1.1 $
'   $Date: 1997/02/27 14:25:33 $
'   $State: Rel $
'   $Author: bwb $
'   U.S. Government Copyright 1995
'   All rights reserved


'   This script makes a parking shape table.


'   Get the input table.
inName = av.Run("InedDatabase.ChooseTable", {"Make Parking Shape Table", "Parking"})
if (inName = NIL) then
    exit
end
inTableName = av.Run("InedDatabase.GetTableName", {"Parking", inName})

'   Get the node table.
nodeName = av.Run("InedDatabase.GetMaster" , {"Node", "Parking", inName})
if (nodeName = NIL) then
    exit
end
nodeTableName = av.Run("InedDatabase.GetTableName", {"Node", nodeName})

'   Get the link table.
linkName = av.Run("InedDatabase.GetMaster", {"Link", "Parking", inName})
if (linkName = NIL) then
    exit
end
linkTableName = av.Run("InedDatabase.GetTableName", {"Link", linkName})

'   Build joined table.
```

```
_directory.ExecuteSQL(
        "CREATE TABLE AV_INED_TEMP AS" + 13.AsChar +
        "  SELECT P.*," + 13.AsChar +
        "      A.ABSCISSA + (B.ABSCISSA - A.ABSCISSA) * P.OFFSET / (L.LENGTH + 0.01) AS
ABSCISSA, " + 13.AsChar +
        "      A.ORDINATE + (B.ORDINATE - A.ORDINATE) * P.OFFSET / (L.LENGTH + 0.01) AS
ORDINATE" + 13.AsChar +
        "    FROM """ + inTableName + """ P, """ + linkTableName + """ L, """ +
nodeTableName + """ A, """ + nodeTableName + """ B" + 13.AsChar +
        "    WHERE P.LINK = L.ID AND P.NODE = A.ID AND L.NODEA = A.ID AND L.NODEB = B.ID"
+ 13.AsChar +
        "  UNION SELECT P.*," + 13.AsChar +
        "      B.ABSCISSA + (A.ABSCISSA - B.ABSCISSA) * P.OFFSET / (L.LENGTH + 0.01) AS
ABSCISSA, " + 13.AsChar +
        "      B.ORDINATE + (A.ORDINATE - B.ORDINATE) * P.OFFSET / (L.LENGTH + 0.01) AS
ORDINATE" + 13.AsChar +
        "    FROM """ + inTableName + """ P, """ + linkTableName + """ L, """ +
nodeTableName + """ A, """ + nodeTableName + """ B" + 13.AsChar +
        "    WHERE P.LINK = L.ID AND P.NODE = B.ID AND L.NODEA = A.ID AND L.NODEB = B.ID")
inTable = VTab.MakeSQL(_directory, "SELECT * FROM AV_INED_TEMP")
_directory.ExecuteSQL("DROP TABLE AV_INED_TEMP")
if (inTable.HasError) then
    MsgBox.Error("Cannot join parkings, nodes, and links.", "Make Parking Shape Table")
    exit
end

'   Get the fields.
copyFields = inTable.GetFields.clone
xField = inTable.FindField("ABSCISSA")
yField = inTable.FindField("ORDINATE")

'   Get the output table.
response = FileDialog.Put("parkings.shp".AsFileName, "*.shp", "Make Parking Shape Table")
if (response = NIL) then
    exit
end
outTable = FTab.MakeNew(response, Point)

'   Generate the points.
av.Run("InedNetwork.GeneratePoints", {inTable, outTable, copyFields, xField, yField})
outTable.Flush

'   View the result if necessary.
if (MsgBox.YesNo("Do you want to view the result?", "Make Parking Shape Table", TRUE))
then
    outTheme = FTheme.Make(outTable)
    outTheme.SetName("Parkings")
    outView = av.GetActiveDoc
    outView.AddTheme(outTheme)
    outTheme.SetVisible(TRUE)
end
```

7. InedNetwork.WriteHOOPS.ave

```
'   Project: TRANSIMS
'   Subsystem: Output Visualizer
'   $RCSfile: InedNetwork.WriteHOOPS.ave,v $
'   $Revision: 1.11 $
'   $Date: 1997/02/27 14:25:33 $
'   $State: Rel $
'   $Author: bwb $
'   U.S. Government Copyright, 1995
'   All rights reserved


'   This script writes a HOOPS metafile for a network.


'   Get the input tables.
theGUI = av.GetActiveDoc
if (theGUI = NIL) then
    MsgBox.Error("TRANSIMS network not found.", "Write HOOPS Metafile")
    exit
end
```

```
nodeTheme = theGUI.FindTheme("Nodes")
if (nodeTheme = NIL) then
    MsgBox.Error("Node theme not found.", "Write HOOPS Metafile")
    exit
end
nodeFTab = nodeTheme.GetFTab

linkTheme = theGUI.FindTheme("Links")
if (linkTheme = NIL) then
    MsgBox.Error("Link theme not found.", "Write HOOPS Metafile")
    exit
end
linkFTab = linkTheme.GetFTab

laneTheme = theGUI.FindTheme("Lanes")
if (laneTheme = NIL) then
    MsgBox.Error("Lane theme not found.", "Write HOOPS Metafile")
    exit
end
laneFTab = laneTheme.GetFTab

parkingTheme = theGUI.FindTheme("Parkings")
if (parkingTheme = NIL) then
    MsgBox.Error("Parking theme not found." + NL + "NOT Outputing Parking Places", "Write
HOOPS Metafile")
    outputParking = FALSE
else
    outputParking = TRUE
    parkingFTab = parkingTheme.GetFTab
end

'   Get the box length for links.
boxlength = MsgBox.Input("Box length in link units:", "Write HOOPS Metafile", "150.0")
if (boxlength = NIL) then
    exit
end
if ((boxlength.AsNumber > 0).Not) then
    MsgBox.Error("Invalid box length.", "Write HOOPS Metafile")
    exit
end
boxlength = boxlength.AsNumber

'   Get the lane width for links.
width = MsgBox.Input("Lane Width in link units:", "Write HOOPS Metafile", "3.25")
if (width = NIL) then
        exit
end
if ((width.AsNumber > 0).Not) then
        MsgBox.Error("Invalid lane width.", "Write HOOPS Metafile")
        exit
end
width = width.AsNumber

' Set cell length
celllength = 7.5

'   Get output file.
outFileName = FileDialog.Put("network.hmf".AsFileName, "*.hmf", "HOOPS Metafile")
if (outFileName = NIL) then
    exit
end
outFile = LineFile.Make(outFileName, #FILE_PERM_CLEARMODIFY)
outFile.WriteELT(";; HMF V1.1 TEXT")
outFile.WriteELT(";; cell size:" ++ celllength.AsString ++
           "meters   box size:" ++ boxlength.AsString ++
           "meters   lane width:" ++ width.AsString ++ "meters")

'   Prepare Dictionary of lanes for use later on
links = Dictionary.Make(linkFTab.GetNumRecords)
towardsField = laneFTab.FindField("Towards")
if (towardsField = NIL) then
    towardsField = laneFTab.FindField("Node")
end
linkField = laneFTab.FindField("Link")
laneField = laneFTab.FindField("Lane")
lengthField = laneFTab.FindField("Length")
```

```
shapeField = laneFTab.FindField("Shape")
count = 0
av.ShowMsg("Reading Lane Table . . .")
av.ShowStopButton
for each i in laneFTab
    node = laneFTab.ReturnValueNumber(towardsField, i).Round
    id = laneFTab.ReturnValueNumber(linkField, i).Round
    lane = laneFTab.ReturnValueNumber(laneField, i).Round
    length = laneFTab.ReturnValueNumber(lengthField, i)
    thePolygon = laneFTab.ReturnValue(shapeField, i)
    if (links.Get(id) = NIL) then
        links.Set(id, List.Make)
    end
    lanes = links.Get(id)
    lanes.Add({thePolygon, lane, node, length})

    ’   Update the status.
    count = count + 1
    if (av.SetStatus(count/laneFTab.GetNumRecords * 100).Not) then   ’ user clicked the
stop button
        break
    end
end

’ Determine bounding area (Min X, Max X, Min Y, and Max Y) for the metafile.
’   Create a temp field so when summarize will end up with only one record
’   in the summary table (summarize creates a diff record for each unique value)
’ Note: node table needs ALL records unselected or only thoses selected will be
’   included in the summary
nodeFTab.SetEditable(TRUE)
nodeFTab.GetSelection.ClearAll
tempField = Field.Make("temp", #FIELD_BYTE, 8, 0)
nodeFTab.AddFields({tempField})
idField = nodeFTab.FindField("ID")
xField = nodeFTab.FindField("Abscissa")
yField = nodeFTab.FindField("Ordinate")
summaryVTab = nodeFTab.Summarize("/tmp/animate.tmp.txt".AsFileName,
    DText, tempField, {xField, xField, yField, yField},
    {#VTAB_SUMMARY_MIN, #VTAB_SUMMARY_MAX, #VTAB_SUMMARY_MIN, #VTAB_SUMMARY_MAX})
nodeFTab.RemoveFields({tempField})
nodeFTab.SetEditable(FALSE)
nodeFTab.Flush
File.Delete("/tmp/animate.tmp.txt".AsFileName)

minx = summaryVTab.ReturnValue(summaryVTab.FindField("Min_Abscissa"),0)
maxx = summaryVTab.ReturnValue(summaryVTab.FindField("Max_Abscissa"),0)
miny = summaryVTab.ReturnValue(summaryVTab.FindField("Min_Ordinate"),0)
maxy = summaryVTab.ReturnValue(summaryVTab.FindField("Max_Ordinate"),0)

’ Write bounding area and width to metafile
outFile.WriteELT("(Segment ""?Picture/scene"" (")
outFile.WriteELT("    (User_Options ""minx=" + minx.AsString +
        ",miny=" + miny.AsString + ",maxx=" + maxx.AsString +
        ",maxy=" + maxy.AsString + ",width=" + width.AsString + """)")
outFile.WriteELT("))")

’   Write nodes to the file.
idField = nodeFTab.FindField("ID")
shapeField = nodeFTab.FindField("Shape")
outFile.WriteELT("(Segment ""Nodes"" (")
count = 0
av.ShowMsg("Writing out Nodes . . .")
av.ShowStopButton
for each i in nodeFTab
    id = nodeFTab.ReturnValueNumber(idField, i).Round
    thePoint = nodeFTab.ReturnValue(shapeField, i)
    outFile.WriteELT("  (Segment """ + id.SetFormat("d").AsString + """ (")
    outFile.WriteELT("    (Marker " + thePoint.GetX.AsString + " " +
thePoint.GetY.AsString + " 0)")
    outFile.WriteELT("  ))")

    ’   Update the status.
    count = count + 1
    if (av.SetStatus(count/nodeFTab.GetNumRecords * 100).Not) then   ’ user clicked the
stop button
        break
```

```
    end
end
outFile.WriteELT("))")

'   Write Parking Places to the file.
if (outputParking = TRUE) then
    idField = parkingFTab.FindField("ID")
    shapeField = parkingFTab.FindField("Shape")
    outFile.WriteELT("(Segment ""Parking"" (")
    count = 0
    av.ShowMsg("Writing out Parking Places . . .")
    av.ShowStopButton
    for each i in parkingFTab
        id = parkingFTab.ReturnValueNumber(idField, i).Round
        thePoint = parkingFTab.ReturnValue(shapeField, i)
        outFile.WriteELT("  (Segment """ + id.SetFormat("d").AsString + """ (")
        outFile.WriteELT("    (Marker " + thePoint.GetX.AsString + " " +
thePoint.GetY.AsString + " 0)")
        outFile.WriteELT("  ))")

        '   Update the status.
        count = count + 1
        if (av.SetStatus(count/parkingFTab.GetNumRecords * 100).Not) then   ' user clicked
the stop button
            break
        end
    end
    outFile.WriteELT("))")
end

'   Write links and lanes to the file.
idField = linkFTab.FindField("ID")
nodeaField = linkFTab.FindField("NODEA")
nodebField = linkFTab.FindField("NODEB")
permlanesaField = linkFTab.FindField("PERMLANESA")
permlanesbField = linkFTab.FindField("PERMLANESB")
setbackaField = linkFTab.FindField("SETBACKA")
setbackbField = linkFTab.FindField("SETBACKB")
lengthField = linkFTab.FindField("LENGTH")
functionField = linkFTab.FindField("FCLASS")
shapeField = linkFTab.FindField("Shape")
outFile.WriteELT("(Segment ""Links"" (")
count = 0
av.ShowMsg("Writing out Links . . .")
av.ShowStopButton
for each i in linkFTab
    id = linkFTab.ReturnValueNumber(idField, i).Round
    nodea = linkFTab.ReturnValueNumber(nodeaField, i).Round
    nodeb = linkFTab.ReturnValueNumber(nodebField, i).Round
    permlanesa = linkFTab.ReturnValueNumber(permlanesaField, i).Round
    permlanesb = linkFTab.ReturnValueNumber(permlanesbField, i).Round
    setbacka = linkFTab.ReturnValueNumber(setbackaField, i)
    setbackb = linkFTab.ReturnValueNumber(setbackbField, i)
    totalsetback = setbacka + setbackb
    length = linkFTab.ReturnValueNumber(lengthField, i)
    if (functionField = NIL) then
        function = "-1"
    else
        function = linkFTab.ReturnValueNumber(functionField, i)
    end
    thePolyline = linkFTab.ReturnValue(shapeField, i)
    a = thePolyline.AsList.Get(0).Get(0)    'nodeA
    b = thePolyline.AsList.Get(0).Get(1)    'nodeB
    delta = a - b
    norm = ((delta.GetX * delta.GetX) + (delta.GetY * delta.GetY) + 0.000001).sqrt
    tangent = Point.Make(delta.GetX / norm, delta.GetY / norm)
    normal = Point.Make(tangent.GetY, - (tangent.GetX))
    delta = a - b - Point.Make(tangent.GetX * (setbackb + setbacka), tangent.GetY *
(setbackb + setbacka))
    scale = ((delta.GetX * delta.GetX) + (delta.GetY * delta.GetY) + 0.000001).sqrt /
(length - setbacka - setbackb + 0.000001)

          ' write out lanes going from nodeB toward nodeA
    if (permlanesa > 0) then
        aax = b.GetX + (tangent.GetX * setbackb)
        aay = b.GetY + (tangent.GetY * setbackb)
```

```
        bbx = tangent.GetX * scale
        bby = tangent.GetY * scale
        ccx = normal.GetX * width
        ccy = normal.GetY * width
        outFile.WriteELT("  (Segment """ + id.SetFormat("d").AsString + ":" +
            nodeb.SetFormat("d").AsString + """ (")
        outFile.WriteELT("    (User_Options ""ax=" + aax.AsString +
            ",ay=" + aay.AsString + ",bx=" + bbx.AsString +
            ",by=" + bby.AsString + ",cx=" + ccx.AsString +
            ",cy=" + ccy.AsString + ",setback=" + totalsetback.AsString +
            ",length=" + length.AsString + ",fclass=" + function.AsString + """)")
        lanes = links.Get(id)
        if (lanes <> NIL) then
        for each datum in lanes
            if (datum.Get(2) <> nodea) then
                continue
            end
            outFile.WriteELT("    ;;  Lane " + datum.Get(1).AsString)
            outFile.WriteELT("    (Polygon (")
            for each thePoint in datum.Get(0).AsList.Get(0)
                outFile.WriteELT("      (" + thePoint.GetX.AsString + " " +
thePoint.GetY.AsString + " 0)")
            end
            outFile.WriteELT("    ))")
        end
        end
        outFile.WriteELT("  ))")
    end

          ' write out lanes going from nodeA toward nodeB
    if (permlanesb > 0) then
        aax = a.GetX - (tangent.GetX * setbacka)
        aay = a.GetY - (tangent.GetY * setbacka)
        bbx = tangent.GetX * (- scale)
        bby = tangent.GetY * (- scale)
        ccx = normal.GetX * (- width)
        ccy = normal.GetY * (- width)
        outFile.WriteELT("  (Segment """ + id.SetFormat("d").AsString + ":" +
            nodea.SetFormat("d").AsString + """ (")
        outFile.WriteELT("    (User_Options ""ax=" + aax.AsString +
             ",ay=" + aay.AsString + ",bx=" + bbx.AsString +
             ",by=" + bby.AsString + ",cx=" + ccx.AsString +
             ",cy=" + ccy.AsString + ",setback=" + totalsetback.AsString +
             ",length=" + length.AsString + ",fclass=" + function.AsString + """)")
        lanes = links.Get(id)
        if (lanes <> NIL) then
        for each datum in lanes
            if (datum.Get(2) <> nodeb) then
                continue
            end
            outFile.WriteELT("    ;;  Lane " + datum.Get(1).AsString)
            outFile.WriteELT("    (Polygon (")
            for each thePoint in datum.Get(0).AsList.Get(0)
                outFile.WriteELT("      (" + thePoint.GetX.AsString + " " +
thePoint.GetY.AsString + " 0)")
            end
            outFile.WriteELT("    ))")
        end
        end
        outFile.WriteELT("  ))")
    end

    '   Update the status.
    count = count + 1
    if (av.SetStatus(count/linkFTab.GetNumRecords * 100).Not) then   ' user clicked the
stop button
        break
    end
end
outFile.WriteELT("))")


'   Write boxes to the file.
leftpocketsaField = linkFTab.FindField("LEFTPCKTSA")
leftpocketsbField = linkFTab.FindField("LEFTPCKTSB")
rightpocketsaField = linkFTab.FindField("RGHTPCKTSA")
```

```
rightpocketsbField = linkFTab.FindField("RGHTPCKTSB")
outFile.WriteELT("(Segment ""Boxes"" (")
outFile.WriteELT("    (User_Options ""cell_length=" + celllength.AsString + ",box_length="
+ boxlength.AsString + """)")
count = 0
av.ShowMsg("Writing out Boxes . . .")
av.ShowStopButton
for each i in linkFTab
    id = linkFTab.ReturnValueNumber(idField, i).Round
    nodea = linkFTab.ReturnValueNumber(nodeaField, i).Round
    nodeb = linkFTab.ReturnValueNumber(nodebField, i).Round
    permlanesa = linkFTab.ReturnValueNumber(permlanesaField, i).Round
    permlanesb = linkFTab.ReturnValueNumber(permlanesbField, i).Round
    lanesa = (permlanesa + linkFTab.ReturnValueNumber(leftpocketsaField, i) +
linkFTab.ReturnValueNumber(rightpocketsaField, i)).Round
    lanesb = (permlanesb + linkFTab.ReturnValueNumber(leftpocketsbField, i) +
linkFTab.ReturnValueNumber(rightpocketsbField, i)).Round
    setbacka = linkFTab.ReturnValueNumber(setbackaField, i)
    setbackb = linkFTab.ReturnValueNumber(setbackbField, i)
    totalsetback = setbacka + setbackb
    length = linkFTab.ReturnValueNumber(lengthField, i)
    linklength = celllength * ((length - totalsetback) / celllength).Floor
    if (linklength = 0) then
        linklength = celllength
    end
    numberboxes = (linklength / boxlength).Ceiling
    if (functionField = NIL) then
        function = "-1"
    else
        function = linkFTab.ReturnValueNumber(functionField, i)
    end
    thePolyline = linkFTab.ReturnValue(shapeField, i)
    a = thePolyline.AsList.Get(0).Get(0)    'nodeA
    b = thePolyline.AsList.Get(0).Get(1)    'nodeB
    delta = a - b
    norm = ((delta.GetX * delta.GetX) + (delta.GetY * delta.GetY) + 0.000001).sqrt
    tangent = Point.Make(delta.GetX / norm, delta.GetY / norm)
    normal = Point.Make(tangent.GetY, - (tangent.GetX))
    delta = a - b - Point.Make(tangent.GetX * (setbackb + setbacka), tangent.GetY *
(setbackb + setbacka))
    scale = ((delta.GetX * delta.GetX) + (delta.GetY * delta.GetY) + 0.000001).sqrt /
(linklength + 0.000001)

          ' write out boxes going from nodeB toward nodeA
    if (lanesa > 0) then
        aax = b.GetX + (tangent.GetX * setbackb)
        aay = b.GetY + (tangent.GetY * setbackb)
        bbx = tangent.GetX * scale
        bby = tangent.GetY * scale
        ccx = normal.GetX * width
        ccy = normal.GetY * width
        for each j in 1..numberboxes
            'ISSUE(bwb): This only works for turn pockets and permanent lanes.
            if (j < numberboxes) then
                boxbegin = linklength - (j * boxlength)
            else
                boxbegin = 0
            end
            boxend = linklength - ((j - 1) * boxlength)
            numbercells = 0
            boxpolygon = List.Make
            lanes = links.Get(id)
            if (lanes <> NIL) then
            for each datum in lanes
                if (datum.Get(2) <> nodea) then
                    continue
                end
                lanenumber = datum.Get(1)
                lanelength = celllength * (datum.Get(3) / celllength).Floor
                'ISSUE(bwb): This will not work for zero-cell pocket lanes.
                if (lanelength = 0) then
                    lanelength = celllength
                end
                laneend = linklength
                lanebegin = linklength - lanelength
                if (lanebegin < boxend) then
```

```
                    lanebegin = lanebegin.Max(boxbegin)
                    laneend = laneend.Min(boxend)
                    numbercells = numbercells + ((boxend - lanebegin) / celllength).Round
                    boxpolygon.Add({
                        Point.Make(aax + (bbx * laneend) + (ccx * (lanenumber - 0.5)), aay
+ (bby * laneend) + (ccy * (lanenumber - 0.5))),
                        Point.Make(aax + (bbx * lanebegin) + (ccx * (lanenumber - 0.5)),
aay + (bby * lanebegin) + (ccy * (lanenumber - 0.5))),
                        Point.Make(aax + (bbx * lanebegin) + (ccx * (lanenumber + 0.5)),
aay + (bby * lanebegin) + (ccy * (lanenumber + 0.5))),
                        Point.Make(aax + (bbx * laneend) + (ccx * (lanenumber + 0.5)), aay
+ (bby * laneend) + (ccy * (lanenumber + 0.5)))
                    })
                end
            end
            end
            outFile.WriteELT("  (Segment """ + id.SetFormat("d").AsString + ":" +
nodeb.SetFormat("d").AsString + ":" + boxend.Floor.AsString + """ (")
            outFile.WriteELT("    (User_Options ""box_end=" + boxend.AsString +
",num_cells=" + numbercells.AsString + ",length_cells=" + ((boxend - boxbegin) /
celllength).AsString + ",perm_lanes=" + permlanesa.AsString + ",total_lanes=" +
lanesa.AsString + ",fclass=" + function.AsString + """)")
            'if (boxpolygon.IsEmpty.Not) then
            '    outFile.WriteELT("    (")
            '    for each boxsegment in boxpolygon
            '        outFile.WriteELT("      Polygon (")
            '        for each boxpoint in boxsegment
            '            outFile.WriteELT("        (" + boxpoint.GetX.AsString + " " +
boxpoint.GetY.AsString + " 0)")
            '        end
            '        outFile.WriteELT("      )")
            '    end
            '    outFile.WriteELT("    )")
            'else
                outFile.WriteELT("    (Polygon (")
                outFile.WriteELT("      ;;  Default")
                outFile.WriteELT("      (" + (aax + (bbx * boxbegin)).AsString + " " +
(aay + (bby * boxbegin)).AsString + " 0)")
                outFile.WriteELT("      (" + (aax + (bbx * boxbegin) + (ccx *
permlanesa)).AsString + " " + (aay + (bby * boxbegin) + (ccy * permlanesa)).AsString + "
0)")
                outFile.WriteELT("      (" + (aax + (bbx * boxend) + (ccx *
permlanesa)).AsString + " " + (aay + (bby * boxend) + (ccy * permlanesa)).AsString + "
0)")
                outFile.WriteELT("      (" + (aax + (bbx * boxend)).AsString + " " + (aay
+ (bby * boxend)).AsString + " 0)")
                outFile.WriteELT("    ))")
            'end
            outFile.WriteELT("  ))")
        end
    end

           ' write out boxes going from nodeA toward nodeB
    if (lanesb > 0) then
        aax = a.GetX - (tangent.GetX * setbacka)
        aay = a.GetY - (tangent.GetY * setbacka)
        bbx = tangent.GetX * (- scale)
        bby = tangent.GetY * (- scale)
        ccx = normal.GetX * (- width)
        ccy = normal.GetY * (- width)
        for each j in 1..numberboxes
            'ISSUE(bwb): This only works for turn pockets and permanent lanes.
            if (j < numberboxes) then
                boxbegin = linklength - (j * boxlength)
            else
                boxbegin = 0
            end
            boxend = linklength - ((j - 1) * boxlength)
            numbercells = 0
            boxpolygon = List.Make
            lanes = links.Get(id)
            if (lanes <> NIL) then
            for each datum in lanes
                if (datum.Get(2) <> nodeb) then
                    continue
                end
```

```
                lanenumber = datum.Get(1)
                lanelength = celllength * (datum.Get(3) / celllength).Floor
                'ISSUE(bwb): This will not work for zero-cell pocket lanes.
                if (lanelength = 0) then
                    lanelength = celllength
                end
                laneend = linklength
                lanebegin = linklength - lanelength
                if (lanebegin < boxend) then
                    lanebegin = lanebegin.Max(boxbegin)
                    laneend = laneend.Min(boxend)
                    numbercells = numbercells + ((boxend - lanebegin) / celllength).Round
                    boxpolygon.Add({
                        Point.Make(aax + (bbx * laneend) + (ccx * (lanenumber - 0.5)), aay
+ (bby * laneend) + (ccy * (lanenumber - 0.5))),
                        Point.Make(aax + (bbx * lanebegin) + (ccx * (lanenumber - 0.5)),
aay + (bby * lanebegin) + (ccy * (lanenumber - 0.5))),
                        Point.Make(aax + (bbx * lanebegin) + (ccx * (lanenumber + 0.5)),
aay + (bby * lanebegin) + (ccy * (lanenumber + 0.5))),
                        Point.Make(aax + (bbx * laneend) + (ccx * (lanenumber + 0.5)), aay
+ (bby * laneend) + (ccy * (lanenumber + 0.5)))
                    })
                end
            end
            end
            outFile.WriteELT("  (Segment """ + id.SetFormat("d").AsString + ":" +
nodea.SetFormat("d").AsString + ":" + boxend.Floor.AsString + """ (")
            outFile.WriteELT("    (User_Options ""box_end=" + boxend.AsString +
",num_cells=" + numbercells.AsString + ",length_cells=" + ((boxend - boxbegin) /
celllength).AsString + ",perm_lanes=" + permlanesb.AsString + ",total_lanes=" +
lanesb.AsString + ",fclass=" + function.AsString + """)")
            'if (boxpolygon.IsEmpty.Not) then
            '    outFile.WriteELT("    (")
            '    for each boxsegment in boxpolygon
            '        outFile.WriteELT("      Polygon (")
            '        for each boxpoint in boxsegment
            '            outFile.WriteELT("        (" + boxpoint.GetX.AsString + " " +
boxpoint.GetY.AsString + " 0)")
            '        end
            '        outFile.WriteELT("      )")
            '    end
            '    outFile.WriteELT("    )")
            'else
                outFile.WriteELT("    (Polygon (")
                outFile.WriteELT("      ;;  Default")
                outFile.WriteELT("      (" + (aax + (bbx * boxbegin)).AsString + " " +
(aay + (bby * boxbegin)).AsString + " 0)")
                outFile.WriteELT("      (" + (aax + (bbx * boxbegin) + (ccx *
permlanesb)).AsString + " " + (aay + (bby * boxbegin) + (ccy * permlanesb)).AsString + "
0)")
                outFile.WriteELT("      (" + (aax + (bbx * boxend) + (ccx *
permlanesb)).AsString + " " + (aay + (bby * boxend) + (ccy * permlanesb)).AsString + "
0)")
                outFile.WriteELT("      (" + (aax + (bbx * boxend)).AsString + " " + (aay
+ (bby * boxend)).AsString + " 0)")
                outFile.WriteELT("    ))")
            'end
            outFile.WriteELT("  ))")
        end
    end

    '   Update the status.
    count = count + 1
    if (av.SetStatus(count/linkFTab.GetNumRecords * 100).Not) then   ' user clicked the
stop button
        break
    end
end
outFile.WriteELT("))")

'   Done.
av.ClearMsg
av.SetStatus(100)
```

G. Table Definition Scripts

1. InedTables.SetConnectivity.ave

```
'   Project: TRANSIMS
'   Subsystem: Input Editor
'   $RCSfile: InedTables.SetConnectivity.ave,v $
'   $Revision: 1.0 $
'   $Date: 1996/04/25 18:20:38 $
'   $State: Rel $
'   $Author: bwb $
'   U.S. Government Copyright 1995
'   All rights reserved


'   This script sets the project's lane connectivity table.


'   Set the node table.
av.Run("InedTables.SetTable", "Lane Connectivity")
```

2. InedTables.SetLanes.ave

```
'   Project: TRANSIMS
'   Subsystem: Input Editor
'   $RCSfile: InedTables.SetLanes.ave,v $
'   $Revision: 1.0 $
'   $Date: 1996/04/25 18:20:38 $
'   $State: Rel $
'   $Author: bwb $
'   U.S. Government Copyright 1995
'   All rights reserved


'   This script sets the project's lane table.


'   Set the node table.
av.Run("InedTables.SetTable", "Lanes")
```

3. InedTables.SetLinks.ave

```
'   Project: TRANSIMS
'   Subsystem: Input Editor
'   $RCSfile: InedTables.SetLinks.ave,v $
'   $Revision: 1.0 $
'   $Date: 1996/04/25 18:20:38 $
'   $State: Rel $
'   $Author: bwb $
'   U.S. Government Copyright 1995
'   All rights reserved


'   This script sets the project's link table.


'   Set the node table.
av.Run("InedTables.SetTable", "Links")
```

4. InedTables.SetNodes.ave

```
'   Project: TRANSIMS
'   Subsystem: Input Editor
'   $RCSfile: InedTables.SetNodes.ave,v $
'   $Revision: 1.0 $
'   $Date: 1996/04/25 18:20:38 $
'   $State: Rel $
'   $Author: bwb $
'   U.S. Government Copyright 1995
'   All rights reserved


'   This script sets the project's node table.
```

```
'   Set the node table.
av.Run("InedTables.SetTable", "Nodes")
```

5. InedTables.SetPhasing.ave

```
'   Project: TRANSIMS
'   Subsystem: Input Editor
'   $RCSfile: InedTables.SetPhasing.ave,v $
'   $Revision: 1.0 $
'   $Date: 1996/04/25 18:20:38 $
'   $State: Rel $
'   $Author: bwb $
'   U.S. Government Copyright 1995
'   All rights reserved


'   This script sets the project's phasing plan table.


'   Set the node table.
av.Run("InedTables.SetTable", "Phasing Plans")
```

6. InedTables.SetPockets.ave

```
'   Project: TRANSIMS
'   Subsystem: Input Editor
'   $RCSfile: InedTables.SetPockets.ave,v $
'   $Revision: 1.0 $
'   $Date: 1996/04/25 18:20:38 $
'   $State: Rel $
'   $Author: bwb $
'   U.S. Government Copyright 1995
'   All rights reserved


'   This script sets the project's pocket lane table.


'   Set the node table.
av.Run("InedTables.SetTable", "Pocket Lanes")
```

7. InedTables.SetSignalized.ave

```
'   Project: TRANSIMS
'   Subsystem: Input Editor
'   $RCSfile: InedTables.SetSignalized.ave,v $
'   $Revision: 1.0 $
'   $Date: 1996/04/25 18:20:38 $
'   $State: Rel $
'   $Author: bwb $
'   U.S. Government Copyright 1995
'   All rights reserved


'   This script sets the project's signalized node table.


'   Set the node table.
av.Run("InedTables.SetTable", "Signalized Nodes")
```

8. InedTables.SetTable.ave

```
'   Project: TRANSIMS
'   Subsystem: Input Editor
'   $RCSfile: InedTables.SetTable.ave,v $
'   $Revision: 1.0 $
'   $Date: 1996/04/25 18:20:38 $
'   $State: Rel $
'   $Author: bwb $
'   U.S. Government Copyright 1995
'   All rights reserved
```

```
'   This script sets any table in the project.  The first argument is the requested
document name.


'   Make sure the document name is not already in use.
if (av.GetProject.FindDoc(self) <> NIL) then
    MsgBox.Error("Sorry.  There is already a document named """ + self + """.", "Set
Table")
    exit
end

'   Make a list of table documents.
tables = List.Make
for each i in av.GetProject.GetDocs
    if (i.Is(Table)) then
        tables.Add(i)
    end
end

'   Ask the user which one to use.
answer = MsgBox.ListAsString(tables, "Choose the table for " + self + ":", "Set Table")
if (answer = NIL) then
    exit
end

'   Change the document name.
answer.SetName(self)
```

9. InedTables.SetTiming.ave

```
'   Project: TRANSIMS
'   Subsystem: Input Editor
'   $RCSfile: InedTables.SetTiming.ave,v $
'   $Revision: 1.0 $
'   $Date: 1996/04/25 18:20:38 $
'   $State: Rel $
'   $Author: bwb $
'   U.S. Government Copyright 1995
'   All rights reserved


'   This script sets the project's timing plan table.


'   Set the node table.
av.Run("InedTables.SetTable", "Timing Plans")
```

10. InedTables.SetUnsignalized.ave

```
'   Project: TRANSIMS
'   Subsystem: Input Editor
'   $RCSfile: InedTables.SetUnsignalized.ave,v $
'   $Revision: 1.0 $
'   $Date: 1996/04/25 18:20:38 $
'   $State: Rel $
'   $Author: bwb $
'   U.S. Government Copyright 1995
'   All rights reserved


'   This script sets the project's unsignalized node table.


'   Set the node table.
av.Run("InedTables.SetTable", "Unsignalized Nodes")
```

H. Network Validation Scripts

1. InedValidate.CheckField.ave

```
'   Project: TRANSIMS
'   Subsystem: Input Editor
```

```
'   $RCSfile: InedValidate.CheckField.ave,v $
'   $Revision: 1.0 $
'   $Date: 1996/04/25 18:20:38 $
'   $State: Rel $
'   $Author: bwb $
'   U.S. Government Copyright 1995
'   All rights reserved


'   This script checks the presence and type of a field.  The first argument is the VTab,
the second is the field name, the third is the log file, and the fourth is a flag
indicating a numeric field.  Return the field.


'   Retreive arguments.
aVTab = self.Get(0)
aFieldName = self.Get(1)
aLogFile = self.Get(2)
aNumericFlag = self.Get(3)

'   Check for the field.
theField = aVTab.FindField(aFieldName)
if (theField = NIL) then
    aLogFile.WriteELT("FATAL: field " + aFieldName + " not found.")
    return NIL
else
    if (aNumericFlag and theField.IsTypeNumber.Not) then
        aLogFile.WriteELT("FATAL: field " + aFieldName + " is not numeric.")
        return NIL
    elseif (aNumericFlag.Not and theField.IsTypeString.Not) then
        aLogFile.WriteELT("FATAL: field " + aFieldName + " is not character.")
        return NIL
    end
end

'   Return result.
return theField
```

2. InedValidate.CheckPositive.ave

```
'   Project: TRANSIMS
'   Subsystem: Input Editor
'   $RCSfile: InedValidate.CheckPositive.ave,v $
'   $Revision: 1.0 $
'   $Date: 1996/04/25 18:20:38 $
'   $State: Rel $
'   $Author: bwb $
'   U.S. Government Copyright 1995
'   All rights reserved


'   This script checks that a field value is positive.  The first argument is the VTab,
the second is the field, and the third is the log file.  Return whether the check failed.


'   Retreive arguments.
aVTab = self.Get(0)
aField = self.Get(1)
aLogFile = self.Get(2)

'   Check the values.
fail = FALSE
count = 0
av.ShowStopButton
for each i in aVTab

    '   Check the value.
    theValue = aVTab.ReturnValueNumber(aField, i)
    if ((theValue < 0) or (theValue = NIL)) then
            aLogFile.WriteELT("ERROR: illegal value for field """ + aField.GetAlias + """
of record " + i.AsString + ".")
            aVTab.GetSelection.Set(i)
            fail = TRUE
    end
```

```
    ’   Update the status bar.
    count = count + 1
    if (av.SetStatus(100 * count / aVTab.GetNumRecords).Not) then
        break
    end
end

’    Return the result.
av.SetStatus(100)
return fail
```

3. InedValidate.CheckRange.ave

```
’   Project: TRANSIMS
’   Subsystem: Input Editor
’   $RCSfile: InedValidate.CheckRange.ave,v $
’   $Revision: 1.0 $
’   $Date: 1996/04/25 18:20:38 $
’   $State: Rel $
’   $Author: bwb $
’   U.S. Government Copyright 1995
’   All rights reserved


’   This script checks that a field value is within a specified range.  The first argument
is the VTab, the second is the field, the third is the log file, the fourth is the lowest
legal value, and the fifth is the highest legal value.  Return whether the check failed.


’   Retreive arguments.
aVTab = self.Get(0)
aField = self.Get(1)
aLogFile = self.Get(2)
aLowValue = self.Get(3)
aHighValue = self.Get(4)

’   Check the values.
fail = FALSE
count = 0
av.ShowStopButton
for each i in aVTab

    ’   Check the value.
    theValue = aVTab.ReturnValueNumber(aField, i)
    if ((theValue < aLowValue) or (theValue > aHighValue) or (theValue = NIL)) then
            aLogFile.WriteELT("ERROR: illegal value for field """ + aField.GetAlias + """
of record " + i.AsString + ".")
            aVTab.GetSelection.Set(i)
            fail = TRUE
    end

    ’   Update the status bar.
    count = count + 1
    if (av.SetStatus(100 * count / aVTab.GetNumRecords).Not) then
        break
    end
end

’    Return the result.
av.SetStatus(100)
return fail
```

4. InedValidate.CheckValues.ave

```
’   Project: TRANSIMS
’   Subsystem: Input Editor
’   $RCSfile: InedValidate.CheckValues.ave,v $
’   $Revision: 1.0 $
’   $Date: 1996/04/25 18:20:38 $
’   $State: Rel $
’   $Author: bwb $
’   U.S. Government Copyright 1995
’   All rights reserved
```

```
'   This script checks that a field value is in a set of values.  The first argument is
the VTab, the second is the field, the third is the log file, and the fourth is the list
of legal values.  Returns whether the check failed.


'   Retreive arguments.
aVTab = self.Get(0)
aField = self.Get(1)
aLogFile = self.Get(2)
aList = self.Get(3)

'   Check the values.
fail = FALSE
count = 0
av.ShowStopButton
for each i in aVTab

    '   Check the value.
    theValue = aVTab.ReturnValueString(aField, i)
    if ((aList.FindByValue(theValue) = -1) or (theValue = NIL)) then
            aLogFile.WriteELT("ERROR: illegal value for field """ + aField.GetAlias + """
of record " + i.AsString + ".")
            aVTab.GetSelection.Set(i)
            fail = TRUE
    end

    '   Update the status bar.
    count = count + 1
    if (av.SetStatus(100 * count / aVTab.GetNumRecords).Not) then
        break
    end
end

'    Return the result.
av.SetStatus(100)
return fail
```

5. InedValidate.ShowMessage.ave

```
'   Project: TRANSIMS
'   Subsystem: Input Editor
'   $RCSfile: InedValidate.ShowMessage.ave,v $
'   $Revision: 1.0 $
'   $Date: 1996/04/25 18:20:38 $
'   $State: Rel $
'   $Author: bwb $
'   U.S. Government Copyright 1995
'   All rights reserved


'   This script shows a message on a log file and in the status bar.  The first argument
is the message string and the second is the log file.


'   Retrieve arguments.
aMessage = SELF.Get(0)
aLogFile = SELF.Get(1)

'   Log the message.
aLogFile.WriteELT("")
aLogFile.WriteELT(aMessage)

'   Display the message.
av.ShowMsg(aMessage)
```

6. InedValidate.ValidateConnectivity.ave

```
'   Project: TRANSIMS
'   Subsystem: Input Editor
'   $RCSfile: InedValidate.ValidateConnectivity.ave,v $
'   $Revision: 1.3 $
'   $Date: 1996/09/04 17:59:36 $
'   $State: Rel $
'   $Author: bwb $
'   U.S. Government Copyright 1995
```

```
'   All rights reserved

'   This script checks to see that a lane connectivity table is valid.

'   Get the input tables.
MsgBox.Info("Only connectivity at the selected nodes will be validated.", "Validate Lane
Connectivity Table")
nodeDoc = av.GetProject.FindDoc("Nodes")
if (nodeDoc = NIL) then
    MsgBox.Error("Node table not found.", "Validate Lane Connectivity Table")
    exit
end
nodeVTab = nodeDoc.GetVTab
laneDoc = av.GetProject.FindDoc("Lanes")
if (laneDoc = NIL) then
    MsgBox.Error("Lane table not found.", "Validate Lane Connectivity Table")
    exit
end
laneVTab = laneDoc.GetVTab
connectivityDoc = av.GetProject.FindDoc("Lane Connectivity")
if (connectivityDoc = NIL) then
    MsgBox.Error("Lane Connectivity table not found.", "Validate Lane Connectivity Table")
    exit
end
connectivityVTab = connectivityDoc.GetVTab

'   Get the log file.
logFile = FileDialog.Put("connect.log".AsFileName, "*.log", "Log File for Validation
Results")
if (logFile = NIL) then
    exit
end
log = LineFile.Make(logFile, #FILE_PERM_CLEARMODIFY)
log.WriteELT("")
log.WriteELT("")
log.WriteELT("Lane Connectivity Table Validation Log  -  " + Date.Now.AsString)
fail = FALSE

'   ISSUE(bwb): allow the user to decide here which tests to perform

'   ISSUE(bwb): perform the tests only for the selected records

'   Validate.
while (TRUE)
    laneVTab.GetSelection.ClearAll
    connectivityVTab.GetSelection.ClearAll

    '   Check the field names.
    av.Run("InedValidate.ShowMessage", {"Checking field names . . .", log})
    nodeField = av.Run("InedValidate.CheckField", {connectivityVTab, "NODE", log, TRUE})
    fail = (nodeField = NIL) or fail
    inlinkField = av.Run("InedValidate.CheckField", {connectivityVTab, "INLINK", log,
TRUE})
    fail = (inlinkField = NIL) or fail
    inlaneField = av.Run("InedValidate.CheckField", {connectivityVTab, "INLANE", log,
TRUE})
    fail = (inlaneField = NIL) or fail
    outlinkField = av.Run("InedValidate.CheckField", {connectivityVTab, "OUTLINK", log,
TRUE})
    fail = (outlinkField = NIL) or fail
    outlaneField = av.Run("InedValidate.CheckField", {connectivityVTab, "OUTLANE", log,
TRUE})
    fail = (outlaneField = NIL) or fail

    '   Check for fatal error.
    if (fail) then
        break
    end

    '   Get the nodes of interest.
    nodes = Dictionary.Make(nodeVTab.GetNumRecords)
    idField = nodeVTab.FindField("ID")
    for each i in nodeVTab.GetSelection
        nodes.Add(nodeVTab.ReturnValueNumber(idField, i), TRUE)
```

```
    end

    ’   Check connectivity.
    av.Run("InedValidate.ShowMessage", {"Checking connectivity . . .", log})
    towardsLaneField = laneVTab.FindField("TOWARDS")
    fromLaneField = laneVTab.FindField("FROM")
    linkLaneField = laneVTab.FindField("LINK")
    laneLaneField = laneVTab.FindField("LANE")
    styleLaneField = laneVTab.FindField("STYLE")
    lanes = Dictionary.Make(laneVTab.GetNumRecords)
    for each i in laneVTab
        towardsNode = laneVTab.ReturnValueNumber(towardsLaneField, i)
        fromNode = laneVTab.ReturnValueNumber(fromLaneField, i)
        if ((nodes.Get(towardsNode) = NIL) and (nodes.Get(fromNode) = NIL)) then
            continue
        end
        id = laneVTab.ReturnValueNumber(linkLaneField, i)
        lane = laneVTab.ReturnValueNumber(laneLaneField, i)
        style = laneVTab.ReturnValueString(styleLaneField, i)
        if (lanes.Get(id) = NIL) then
            lanes.Add(id, List.Make)
        end
        lanes.Get(id).Add({towardsNode, fromNode, lane, style, FALSE, FALSE})
    end
    count = 0
    av.ShowStopButton
    for each i in connectivityVTab
        node = connectivityVTab.ReturnValueNumber(nodeField, i)
        inlink = connectivityVTab.ReturnValueNumber(inlinkField, i)
        inlane = connectivityVTab.ReturnValueNumber(inlaneField, i)
        outlink = connectivityVTab.ReturnValueNumber(outlinkField, i)
        outlane = connectivityVTab.ReturnValueNumber(outlaneField, i)
        if (nodes.Get(node) = NIL) then
            continue
        end
        laneData = lanes.Get(inLink)
        if (laneData = NIL) then
            log.WriteELT("ERROR: the link id " + inlink.SetFormat("dddddddddd").AsString +
" does not exist.")
            connectivityVTab.GetSelection.Set(i)
            fail = TRUE
        else
            found = FALSE
            for each datum in laneData
                if ((datum.Get(0) = node) and (datum.Get(2) = inlane) and ((datum.Get(3) =
"X") or (datum.Get(3) = "T"))) then
                    found = TRUE
                    datum.Set(4, TRUE)
                    break
                end
            end
            if (found.Not) then
                log.WriteELT("ERROR: there is no incoming lane " + inlane.AsString + " on
link id " + inlink.SetFormat("dddddddddd").AsString + " towards node id " +
node.SetFormat("dddddddddd").AsString + ".")
                connectivityVTab.GetSelection.Set(i)
                fail = TRUE
            end
        end
        laneData = lanes.Get(outLink)
        if (laneData = NIL) then
            log.WriteELT("ERROR: the link id " + outlink.SetFormat("dddddddddd").AsString
+ " does not exist.")
            connectivityVTab.GetSelection.Set(i)
            fail = TRUE
        else
            found = FALSE
            for each datum in laneData
                if ((datum.Get(1) = node) and (datum.Get(2) = outlane) and ((datum.Get(3)
= "X") or (datum.Get(3) = "M"))) then
                    found = TRUE
                    datum.Set(5, TRUE)
                    break
                end
            end
            if (found.Not) then
```

```
                log.WriteELT("ERROR: there is no outgoing lane " + outlane.AsString + " on
link id " + outlink.SetFormat("dddddddddd").AsString + " from node id " +
node.SetFormat("dddddddddd").AsString + ".")
                connectivityVTab.GetSelection.Set(i)
                fail = TRUE
            end
        end
        count = count + 1
        if (av.SetStatus(100 * count / connectivityVTab.GetNumRecords).Not) then
            break
        end
    end
    av.SetStatus(100)
    for each i in laneVTab
        towardsNode = laneVTab.ReturnValueNumber(towardsLaneField, i)
        fromNode = laneVTab.ReturnValueNumber(fromLaneField, i)
        id = laneVTab.ReturnValueNumber(linkLaneField, i)
        lane = laneVTab.ReturnValueNumber(laneLaneField, i)
        if (lanes.Get(id) = NIL) then
            continue
        end
        for each datum in lanes.Get(id)
            if ((towardsNode = datum.Get(0)) and (fromNode = datum.Get(1)) and (lane =
datum.Get(2))) then
                if ((nodes.Get(towardsNode) = TRUE) and ((datum.Get(3) = "X") or
(datum.Get(3) = "T")) and (datum.Get(4).Not)) then
                    log.WriteELT("WARNING: lane " + lane.AsString + " on link id " +
id.SetFormat("dddddddddd").AsString + " towards node id " +
towardsNode.SetFormat("dddddddddd").AsString + " has no incoming connections.")
                    laneVTab.GetSelection.Set(i)
                    fail = TRUE
                end
                if ((nodes.Get(fromNode) = TRUE) and ((datum.Get(3) = "X") or
(datum.Get(3) = "M")) and (datum.Get(5).Not)) then
                    log.WriteELT("WARNING: lane " + lane.AsString + " on link id " +
id.SetFormat("dddddddddd").AsString + " from node id " +
fromNode.SetFormat("dddddddddd").AsString + " has no outgoing connections.")
                    laneVTab.GetSelection.Set(i)
                    fail = TRUE
                end
                break
            end
        end
    end
        '   ISSUE(bwb): treat the case when a pocket lane is a combination of pull-outs,
merges, and turns

    '   ISSUE(bwb): check for duplicate records

    '   Cleanup and exit.
    laneVTab.UpdateSelection
    connectivityVTab.UpdateSelection
    break
end
if (fail) then
    message = "Errors have been detected and selected."
else
    message = "No errors have been detected."
end
av.Run("InedValidate.ShowMessage", {message, log})
log.Close
MsgBox.Info(message, "Validate Lane Connectivity Table")
if (fail) then
    TextWin.Make(logFile, "Lane Connectivity Table Validation Results")
end
```

7. InedValidate.ValidateLinks.ave

```
'   Project: TRANSIMS
'   Subsystem: Input Editor
'   $RCSfile: InedValidate.ValidateLinks.ave,v $
'   $Revision: 1.0 $
'   $Date: 1996/04/25 18:20:38 $
'   $State: Rel $
'   $Author: bwb $
```

```
'   U.S. Government Copyright 1995
'   All rights reserved


'   This script checks to see that a link table is valid.


'   Get the input tables.
nodeDoc = av.GetProject.FindDoc("Nodes")
if (nodeDoc = NIL) then
    MsgBox.Error("Node table not found.", "Validate Link Table")
    exit
end
nodeVTab = nodeDoc.GetVTab
linkDoc = av.GetProject.FindDoc("Links")
if (linkDoc = NIL) then
    MsgBox.Error("Link table not found.", "Validate Link Table")
    exit
end
linkVTab = linkDoc.GetVTab

'   Get the log file.
logFile = FileDialog.Put("link.log".AsFileName, "*.log", "Log File for Validation
Results")
if (logFile = NIL) then
    exit
end
log = LineFile.Make(logFile, #FILE_PERM_CLEARMODIFY)
log.WriteELT("")
log.WriteELT("")
log.WriteELT("Link Table Validation Log  -  " + Date.Now.AsString)
fail = FALSE

'   ISSUE(bwb): allow the user to decide here which tests to perform

'   ISSUE(bwb): perform the tests only for the selected records

'   Validate.
while (TRUE)
    nodeVTab.GetSelection.ClearAll
    linkVTab.GetSelection.ClearAll

    '   Check the field names.
    av.Run("InedValidate.ShowMessage", {"Checking field names . . .", log})
    idField = av.Run("InedValidate.CheckField", {linkVTab, "ID", log, TRUE})
    fail = (idField = NIL) or fail
    nodeaField = av.Run("InedValidate.CheckField", {linkVTab, "NODEA", log, TRUE})
    fail = (nodeaField = NIL) or fail
    nodebField = av.Run("InedValidate.CheckField", {linkVTab, "NODEB", log, TRUE})
    fail = (nodebField = NIL) or fail
    permanentlanesaField = av.Run("InedValidate.CheckField", {linkVTab, "PERMLANESA", log,
TRUE})
    fail = (permanentlanesaField = NIL) or fail
    permanentlanesbField = av.Run("InedValidate.CheckField", {linkVTab, "PERMLANESB", log,
TRUE})
    fail = (permanentlanesbField = NIL) or fail
    leftpocketsaField = av.Run("InedValidate.CheckField", {linkVTab, "LEFTPCKTSA", log,
TRUE})
    fail = (leftpocketsaField = NIL) or fail
    leftpocketsbField = av.Run("InedValidate.CheckField", {linkVTab, "LEFTPCKTSB", log,
TRUE})
    fail = (leftpocketsbField = NIL) or fail
    rightpocketsaField = av.Run("InedValidate.CheckField", {linkVTab, "RGHTPCKTSA", log,
TRUE})
    fail = (rightpocketsaField = NIL) or fail
    rightpocketsbField = av.Run("InedValidate.CheckField", {linkVTab, "RGHTPCKTSB", log,
TRUE})
    fail = (rightpocketsbField = NIL) or fail
    twowayturnField = av.Run("InedValidate.CheckField", {linkVTab, "TWOWAYTURN", log,
FALSE})
    fail = (twowayturnField = NIL) or fail
    lengthField = av.Run("InedValidate.CheckField", {linkVTab, "LENGTH", log, TRUE})
    fail = (lengthField = NIL) or fail
    gradeField = av.Run("InedValidate.CheckField", {linkVTab, "GRADE", log, TRUE})
    fail = (gradeField = NIL) or fail
    setbackaField = av.Run("InedValidate.CheckField", {linkVTab, "SETBACKA", log, TRUE})
```

```
    fail = (setbackaField = NIL) or fail
    setbackbField = av.Run("InedValidate.CheckField", {linkVTab, "SETBACKB", log, TRUE})
    fail = (setbackbField = NIL) or fail
    capacityaField = av.Run("InedValidate.CheckField", {linkVTab, "CAPACITYA", log, TRUE})
    fail = (capacityaField = NIL) or fail
    capacitybField = av.Run("InedValidate.CheckField", {linkVTab, "CAPACITYB", log, TRUE})
    fail = (capacitybField = NIL) or fail
    speedlimitaField = av.Run("InedValidate.CheckField", {linkVTab, "SPEEDLMTA", log,
TRUE})
    fail = (speedlimitaField = NIL) or fail
    speedlimitbField = av.Run("InedValidate.CheckField", {linkVTab, "SPEEDLMTB", log,
TRUE})
    fail = (speedlimitbField = NIL) or fail
    freespeedaField = av.Run("InedValidate.CheckField", {linkVTab, "FREESPDA", log, TRUE})
    fail = (freespeedaField = NIL) or fail
    freespeedbField = av.Run("InedValidate.CheckField", {linkVTab, "FREESPDB", log, TRUE})
    fail = (freespeedbField = NIL) or fail
    crawlspeedaField = av.Run("InedValidate.CheckField", {linkVTab, "CRAWLSPDA", log,
TRUE})
    fail = (crawlspeedaField = NIL) or fail
    crawlspeedbField = av.Run("InedValidate.CheckField", {linkVTab, "CRAWLSPDB", log,
TRUE})
    fail = (crawlspeedbField = NIL) or fail

    ’   Check for fatal error.
    if (fail) then
        break
    end

    ’   Check ids.
    av.Run("InedValidate.ShowMessage", {"Checking ids . . .", log})
    ids = Dictionary.Make(linkVTab.GetNumRecords)
    count = 0
    av.ShowStopButton
    for each i in linkVTab
        id = linkVTab.ReturnValueNumber(idField, i)
        if (ids.Get(id) = TRUE) then
            log.WriteELT("ERROR: the id " + id.SetFormat("dddddddddd").AsString + " is
duplicated.")
            linkVTab.GetSelection.Set(i)
            fail = TRUE
        else
            ids.Add(id, TRUE)
        end
        if ((id < 1) or (id > 4294967295)) then
            log.WriteELT("ERROR: illegal value for id " +
id.SetFormat("dddddddddd").AsString + ".")
            linkVTab.GetSelection.Set(i)
            fail = TRUE
        end
        count = count + 1
        if (av.SetStatus(100 * count / linkVTab.GetNumRecords).Not) then
            break
        end
    end
    av.SetStatus(100)

    ’   Check node ids.
    av.Run("InedValidate.ShowMessage", {"Checking endpoints . . .", log})
    idNodeField = nodeVTab.FindField("ID")
    ids = Dictionary.Make(nodeVTab.GetNumRecords)
    for each i in nodeVTab
        id = nodeVTab.ReturnValueNumber(idNodeField, i)
        ids.Add(id, {0, 0})
    end
    count = 0
    av.ShowStopButton
    for each i in linkVTab
        id = linkVTab.ReturnValueNumber(nodeaField, i)
        tally = ids.Get(id)
        if (tally = NIL) then
            log.WriteELT("ERROR: the id " + id.SetFormat("dddddddddd").AsString + " for
NODEA does not exist in the node table.")
            linkVTab.GetSelection.Set(i)
            fail = TRUE
        else
```

```
            if (linkVTab.ReturnValueNumber(permanentlanesaField, i) > 0) then
                tally.Set(0, tally.Get(0) + 1)
            end
            if (linkVTab.ReturnValueNumber(permanentlanesbField, i) > 0) then
                tally.Set(1, tally.Get(1) + 1)
            end
        end
        id = linkVTab.ReturnValueNumber(nodebField, i)
        tally = ids.Get(id)
        if (tally = NIL) then
            log.WriteELT("ERROR: the id " + id.SetFormat("dddddddddd").AsString + " for
NODEB does not exist in the node table.")
            linkVTab.GetSelection.Set(i)
            fail = TRUE
        else
            if (linkVTab.ReturnValueNumber(permanentlanesbField, i) > 0) then
                tally.Set(0, tally.Get(0) + 1)
            end
            if (linkVTab.ReturnValueNumber(permanentlanesaField, i) > 0) then
                tally.Set(1, tally.Get(1) + 1)
            end
        end
        count = count + 1
        if (av.SetStatus(100 * count / linkVTab.GetNumRecords).Not) then
            break
        end
    end
    av.SetStatus(100)
    for each i in nodeVTab
        id = nodeVTab.ReturnValueNumber(idNodeField, i)
        tally = ids.Get(id)
        if (tally.Get(0) = 0) then
            log.WriteELT("WARNING: the node id " + id.SetFormat("dddddddddd").AsString + "
has no incoming links.")
            nodeVTab.GetSelection.Set(i)
            fail = TRUE
        end
        if (tally.Get(1) = 0) then
            log.WriteELT("WARNING: the node id " + id.SetFormat("dddddddddd").AsString + "
has no outgoing links.")
            nodeVTab.GetSelection.Set(i)
            fail = TRUE
        end
    end

    ’   Check field values.
    av.Run("InedValidate.ShowMessage", {"Checking number of lanes . . .", log})
    fail = av.Run("InedValidate.CheckRange", {linkVTab, permanentlanesaField, log, 0,
255}) or fail
    fail = av.Run("InedValidate.CheckRange", {linkVTab, permanentlanesbField, log, 0,
255}) or fail
    fail = av.Run("InedValidate.CheckRange", {linkVTab, leftpocketsaField, log, 0, 255})
or fail
    fail = av.Run("InedValidate.CheckRange", {linkVTab, leftpocketsbField, log, 0, 255})
or fail
    fail = av.Run("InedValidate.CheckRange", {linkVTab, rightpocketsaField, log, 0, 255})
or fail
    fail = av.Run("InedValidate.CheckRange", {linkVTab, rightpocketsbField, log, 0, 255})
or fail
    av.Run("InedValidate.ShowMessage", {"Checking two way turns . . .", log})
    fail = av.Run("InedValidate.CheckValues", {linkVTab, twowayturnField, log, {"T",
"F"}}) or fail
    av.Run("InedValidate.ShowMessage", {"Checking lengths and setbacks.", log})
    fail = av.Run("InedValidate.CheckPositive", {linkVTab, lengthField, log}) or fail
    fail = av.Run("InedValidate.CheckPositive", {linkVTab, setbackaField, log}) or fail
    fail = av.Run("InedValidate.CheckPositive", {linkVTab, setbackbField, log}) or fail
    av.Run("InedValidate.ShowMessage", {"Checking capacities . . .", log})
    fail = av.Run("InedValidate.CheckPositive", {linkVTab, capacityaField, log}) or fail
    fail = av.Run("InedValidate.CheckPositive", {linkVTab, capacitybField, log}) or fail
    av.Run("InedValidate.ShowMessage", {"Checking speeds . . .", log})
    fail = av.Run("InedValidate.CheckPositive", {linkVTab, speedlimitaField, log}) or fail
    fail = av.Run("InedValidate.CheckPositive", {linkVTab, speedlimitbField, log}) or fail
    fail = av.Run("InedValidate.CheckPositive", {linkVTab, freespeedaField, log}) or fail
    fail = av.Run("InedValidate.CheckPositive", {linkVTab, freespeedbField, log}) or fail
    fail = av.Run("InedValidate.CheckPositive", {linkVTab, crawlspeedaField, log}) or fail
    fail = av.Run("InedValidate.CheckPositive", {linkVTab, crawlspeedbField, log}) or fail
```

```
    ’   Cleanup and exit.
    nodeVTab.UpdateSelection
    linkVTab.UpdateSelection
    break
end
if (fail) then
    message = "Errors have been detected and selected."
else
    message = "No errors have been detected."
end
av.Run("InedValidate.ShowMessage", {message, log})
log.Close
MsgBox.Info(message, "Validate Link Table")
if (fail) then
    TextWin.Make(logFile, "Link Table Validation Results")
end
```

8. InedValidate.ValidateNodes.ave

```
’   Project: TRANSIMS
’   Subsystem: Input Editor
’   $RCSfile: InedValidate.ValidateNodes.ave,v $
’   $Revision: 1.0 $
’   $Date: 1996/04/25 18:20:38 $
’   $State: Rel $
’   $Author: bwb $
’   U.S. Government Copyright 1995
’   All rights reserved


’   This script checks to see that a node table is valid.


’   Get the input table.
nodeDoc = av.GetProject.FindDoc("Nodes")
if (nodeDoc = NIL) then
    MsgBox.Error("Node table not found.", "Validate Node Table")
    exit
end
nodeVTab = nodeDoc.GetVTab

’   Get the log file.
logFile = FileDialog.Put("node.log".AsFileName, "*.log", "Log File for Validation
Results")
if (logFile = NIL) then
    exit
end
log = LineFile.Make(logFile, #FILE_PERM_CLEARMODIFY)
log.WriteELT("")
log.WriteELT("")
log.WriteELT("Node Table Validation Log  -  " + Date.Now.AsString)
fail = FALSE

’   ISSUE(bwb): allow the user to decide here which tests to perform

’   ISSUE(bwb): perform the tests only for the selected records

’   Validate.
while (TRUE)
    nodeVTab.GetSelection.ClearAll

    ’   Check the field names.
    av.Run("InedValidate.ShowMessage", {"Checking field names . . .", log})
    idField = av.Run("InedValidate.CheckField", {nodeVTab, "ID", log, TRUE})
    fail = (idField = NIL) or fail
    abscissaField = av.Run("InedValidate.CheckField", {nodeVTab, "ABSCISSA", log, TRUE})
    fail = (abscissaField = NIL) or fail
    ordinateField = av.Run("InedValidate.CheckField", {nodeVTab, "ORDINATE", log, TRUE})
    fail = (ordinateField = NIL) or fail

    ’   Check for fatal error.
    if (fail) then
        break
    end
```

```
    ’   Check ids.
    av.Run("InedValidate.ShowMessage", {"Checking ids . . .", log})
    ids = Dictionary.Make(nodeVTab.GetNumRecords)
    count = 0
    av.ShowStopButton
    for each i in nodeVTab
        id = nodeVTab.ReturnValueNumber(idField, i)
        if (ids.Get(id) = TRUE) then
            log.WriteELT("ERROR: the id " + id.SetFormat("dddddddddd").AsString + " is
duplicated.")
            nodeVTab.GetSelection.Set(i)
            fail = TRUE
        else
            ids.Add(id, TRUE)
        end
        if ((id < 1) or (id > 4294967295)) then
            log.WriteELT("ERROR: illegal value for id " +
id.SetFormat("dddddddddd").AsString + ".")
            nodeVTab.GetSelection.Set(i)
            fail = TRUE
        end
        count = count + 1
        if (av.SetStatus(100 * count / nodeVTab.GetNumRecords).Not) then
            break
        end
    end
    av.SetStatus(100)

’   Cleanup and exit.
    nodeVTab.UpdateSelection
    break
end
if (fail) then
    message = "Errors have been detected and selected."
else
    message = "No errors have been detected."
end
av.Run("InedValidate.ShowMessage", {message, log})
log.Close
MsgBox.Info(message, "Validate Node Table")
if (fail) then
    TextWin.Make(logFile, "Node Table Validation Results")
end
```

9. InedValidate.ValidatePhasing.ave

```
’   Project: TRANSIMS
’   Subsystem: Input Editor
’   $RCSfile: InedValidate.ValidatePhasing.ave,v $
’   $Revision: 1.0 $
’   $Date: 1996/04/25 18:20:38 $
’   $State: Rel $
’   $Author: bwb $
’   U.S. Government Copyright 1995
’   All rights reserved


’   This script checks to see that a phasing plan table is valid.


’   Get the input tables.
linkDoc = av.GetProject.FindDoc("Links")
if (linkDoc = NIL) then
    MsgBox.Error("Link table not found.", "Validate Phasing Plan Table")
    exit
end
linkVTab = linkDoc.GetVTab
timingDoc = av.GetProject.FindDoc("Timing Plans")
if (timingDoc = NIL) then
    MsgBox.Error("Timing plan table not found.", "Validate Phasing Plan Table")
    exit
end
timingVTab = timingDoc.GetVTab
signalizedDoc = av.GetProject.FindDoc("Signalized Nodes")
```

```
if (signalizedDoc = NIL) then
    MsgBox.Error("Signalized node table not found.", "Validate Phasing Plan Table")
    exit
end
signalizedVTab = signalizedDoc.GetVTab
phasingDoc = av.GetProject.FindDoc("Phasing Plans")
if (phasingDoc = NIL) then
    MsgBox.Error("Phasing plans table not found.", "Validate Phasing Plan Table")
    exit
end
phasingVTab = phasingDoc.GetVTab

'   Get the log file.
logFile = FileDialog.Put("phasing.log".AsFileName, "*.log", "Log File for Validation
Results")
if (logFile = NIL) then
    exit
end
log = LineFile.Make(logFile, #FILE_PERM_CLEARMODIFY)
log.WriteELT("")
log.WriteELT("")
log.WriteELT("Phasing Plan Table Validation Log  -  " + Date.Now.AsString)
fail = FALSE

'   ISSUE(bwb): allow the user to decide here which tests to perform

'   ISSUE(bwb): perform the tests only for the selected records

'   Validate.
while (TRUE)
    linkVTab.GetSelection.ClearAll
    timingVTab.GetSelection.ClearAll
    signalizedVTab.GetSelection.ClearAll
    phasingVTab.GetSelection.ClearAll

    '   Check the field names.
    av.Run("InedValidate.ShowMessage", {"Checking field names . . .", log})
    nodeField = av.Run("InedValidate.CheckField", {phasingVTab, "NODE", log, TRUE})
    fail = (nodeField = NIL) or fail
    planField = av.Run("InedValidate.CheckField", {phasingVTab, "PLAN", log, TRUE})
    fail = (planField = NIL) or fail
    phaseField = av.Run("InedValidate.CheckField", {phasingVTab, "PHASE", log, TRUE})
    fail = (phaseField = NIL) or fail
    inlinkField = av.Run("InedValidate.CheckField", {phasingVTab, "INLINK", log, TRUE})
    fail = (inlinkField = NIL) or fail
    outlinkField = av.Run("InedValidate.CheckField", {phasingVTab, "OUTLINK", log, TRUE})
    fail = (outlinkField = NIL) or fail
    protectionField = av.Run("InedValidate.CheckField", {phasingVTab, "PROTECTION", log,
FALSE})
    fail = (protectionField = NIL) or fail

    '   Check for fatal error.
    if (fail) then
        break
    end

    '   Check field values.
    av.Run("InedValidate.ShowMessage", {"Checking values . . .", log})
    fail = av.Run("InedValidate.CheckValues", {phasingVTab, protectionField, log, {"P",
"U"}}) or fail

    '   Check connectivity.
    av.Run("InedValidate.ShowMessage", {"Checking connectivity . . .", log})
    idLinkField = linkVTab.FindField("ID")
    nodeaLinkField = linkVTab.FindField("NODEA")
    nodebLinkField = linkVTab.FindField("NODEB")
    permanentlanesaLinkField = linkVTab.FindField("PERMLANESA")
    permanentlanesbLinkField = linkVTab.FindField("PERMLANESB")
    leftpocketsaLinkField = linkVTab.FindField("LEFTPCKTSA")
    leftpocketsbLinkField = linkVTab.FindField("LEFTPCKTSB")
    rightpocketsaLinkField = linkVTab.FindField("RGHTPCKTSA")
    rightpocketsbLinkField = linkVTab.FindField("RGHTPCKTSB")
    links = Dictionary.Make(linkVTab.GetNumRecords)
    for each i in linkVTab
        id = linkVTab.ReturnValueNumber(idLinkField, i)
        nodea = linkVTab.ReturnValueNumber(nodeaLinkField, i)
```

```
        nodeb = linkVTab.ReturnValueNumber(nodebLinkField, i)
        lanesa = linkVTab.ReturnValueNumber(permanentlanesaLinkField, i) +
linkVTab.ReturnValueNumber(leftpocketsaLinkField, i) +
linkVTab.ReturnValueNumber(rightpocketsaLinkField, i)
        lanesb = linkVTab.ReturnValueNumber(permanentlanesbLinkField, i) +
linkVTab.ReturnValueNumber(leftpocketsbLinkField, i) +
linkVTab.ReturnValueNumber(rightpocketsbLinkField, i)
        links.Add(id, {nodea, nodeb, FALSE, FALSE, FALSE, FALSE, lanesa > 0, lanesb > 0})
    end
    planTimingField = timingVTab.FindField("PLAN")
    phaseTimingField = timingVTab.FindField("PHASE")
    plans = Dictionary.Make(timingVTab.GetNumRecords)
    for each i in timingVTab
        plan = timingVTab.ReturnValueNumber(planTimingField, i)
        phase = timingVTab.ReturnValueNumber(phaseTimingFIeld, i)
        if (plans.Get(plan) = NIL) then
            plans.Add(plan, List.Make)
        end
        plans.Get(plan).Add(phase)
    end
    nodeSignalizedField = signalizedVTab.FindField("NODE")
    planSignalizedField = signalizedVTab.FindField("PLAN")
    nodes = Dictionary.Make(signalizedVTab.GetNumRecords)
    for each i in signalizedVTab
        node = signalizedVTab.ReturnValueNumber(nodeSignalizedField, i)
        plan = signalizedVTab.ReturnValueNumber(planSignalizedField, i)
        nodes.Add(node, plan)
    end
    count = 0
    av.ShowStopButton
    for each i in phasingVTab
        node = phasingVTab.ReturnValueNumber(nodeField, i)
        plan = phasingVTab.ReturnValueNumber(planField, i)
        phase = phasingVTab.ReturnValueNumber(phaseField, i)
        inlink = phasingVTab.ReturnValueNumber(inlinkField, i)
        outlink = phasingVTab.ReturnValueNumber(outlinkField, i)
        if (nodes.Get(node) <> plan) then
            log.WriteELT("ERROR: node id " + node.SetFormat("dddddddddd").AsString + "
does not have plan " + plan.AsString + ".")
            phasingVTab.GetSelection.Set(i)
            fail = TRUE
        elseif (plans.Get(plan).FindByValue(phase) = -1) then
            log.WriteELT("ERROR: plan " + plan.AsString + " does not have a phase " +
phase.AsString + ".")
            phasingVTab.GetSelection.Set(i)
            fail = TRUE
        else
            linkData = links.Get(inlink)
            if (linkData = NIL) then
                log.WriteELT("ERROR: incoming link id " +
inlink.SetFormat("dddddddddd").AsString + " does not exist.")
                phasingVTab.GetSelection.Set(i)
                fail = TRUE
            else
                if ((linkData.Get(0) = node) and linkData.Get(6)) then
                    linkData.Set(2, TRUE)
                elseif ((linkData.Get(1) = node) and linkData.Get(7)) then
                    linkData.Set(3, TRUE)
                else
                    log.WriteELT("ERROR: incoming link id " +
inlink.SetFormat("dddddddddd").AsString + " is not connected to node id " +
node.SetFormat("dddddddddd").AsString + ".")
                    phasingVTab.GetSelection.Set(i)
                end
            end
            linkData = links.Get(outlink)
            if (linkData = NIL) then
                log.WriteELT("ERROR: outgoing link id " +
outlink.SetFormat("dddddddddd").AsString + " does not exist.")
                phasingVTab.GetSelection.Set(i)
                fail = TRUE
            else
                if ((linkData.Get(0) = node) and linkData.Get(7)) then
                    linkData.Set(4, TRUE)
                elseif ((linkData.Get(1) = node) and linkData.Get(6)) then
                    linkData.Set(5, TRUE)
```

```
                else
                    log.WriteELT("ERROR: outgoing link id " +
outlink.SetFormat("dddddddddd").AsString + " is not connected to node id " +
node.SetFormat("dddddddddd").AsString + ".")
                    phasingVTab.GetSelection.Set(i)
                end
            end
        end
        count = count + 1
        if (av.SetStatus(100 * count / phasingVTab.GetNumRecords).Not) then
            break
        end
    end
    av.SetStatus(100)
    for each i in linkVTab
        id = linkVTab.ReturnValueNumber(idLinkField, i)
        nodea = linkVTab.ReturnValueNumber(nodeaLinkField, i)
        nodeb = linkVTab.ReturnValueNumber(nodebLinkField, i)
        lanesa = linkVTab.ReturnValueNumber(permanentlanesaLinkField, i) +
linkVTab.ReturnValueNumber(leftpocketsaLinkField, i) +
linkVTab.ReturnValueNumber(rightpocketsaLinkField, i)
        lanesb = linkVTab.ReturnValueNumber(permanentlanesbLinkField, i) +
linkVTab.ReturnValueNumber(leftpocketsbLinkField, i) +
linkVTab.ReturnValueNumber(rightpocketsbLinkField, i)
        linkData = links.Get(id)
        if (nodes.Get(nodea) <> NIL) then
            if ((linkData.Get(2) = FALSE) and (lanesa > 0)) then
                log.WriteELT("WARNING: incoming link id " +
id.SetFormat("dddddddddd").AsString + " has no allowed movements at node id " +
nodea.SetFormat("dddddddddd").AsString +  ".")
                linkVTab.GetSelection.Set(i)
                fail = TRUE
            end
            if ((linkData.Get(4) = FALSE) and (lanesb > 0)) then
                log.WriteELT("WARNING: outgoing link id " +
id.SetFormat("dddddddddd").AsString + " has no allowed movements at node id " +
nodea.SetFormat("dddddddddd").AsString +  ".")
                linkVTab.GetSelection.Set(i)
                fail = TRUE
            end
        end
        if (nodes.Get(nodeb) <> NIL) then
            if ((linkData.Get(3) = FALSE) and (lanesb > 0)) then
                log.WriteELT("WARNING: incoming link id " +
id.SetFormat("dddddddddd").AsString + " has no allowed movements at node id " +
nodeb.SetFormat("dddddddddd").AsString +  ".")
                linkVTab.GetSelection.Set(i)
                fail = TRUE
            end
            if ((linkData.Get(5) = FALSE) and (lanesa > 0)) then
                log.WriteELT("WARNING: outgoing link id " +
id.SetFormat("dddddddddd").AsString + " has no allowed movements at node id " +
nodeb.SetFormat("dddddddddd").AsString +  ".")
                linkVTab.GetSelection.Set(i)
                fail = TRUE
            end
        end
    end

    '   ISSUE(bwb): We need to check for duplicates, also.

    '   Cleanup and exit.
    linkVTab.UpdateSelection
    timingVTab.UpdateSelection
    signalizedVTab.UpdateSelection
    phasingVTab.UpdateSelection
    break
end
if (fail) then
    message = "Errors have been detected and selected."
else
    message = "No errors have been detected."
end
av.Run("InedValidate.ShowMessage", {message, log})
log.Close
MsgBox.Info(message, "Validate Phasing Plan Table")
```

```
if (fail) then
    TextWin.Make(logFile, "Phasing Plan Table Validation Results")
end
```

10. InedValidate.ValidatePockets.ave

```
'   Project: TRANSIMS
'   Subsystem: Input Editor
'   $RCSfile: InedValidate.ValidatePockets.ave,v $
'   $Revision: 1.1 $
'   $Date: 1996/09/19 16:05:16 $
'   $State: Rel $
'   $Author: bwb $
'   U.S. Government Copyright 1995
'   All rights reserved


'   This script checks to see that a pocket lane table is valid.


'   Get the input tables.
linkDoc = av.GetProject.FindDoc("Links")
if (linkDoc = NIL) then
    MsgBox.Error("Link table not found.", "Validate Pocket Lane Table")
    exit
end
linkVTab = linkDoc.GetVTab
pocketDoc = av.GetProject.FindDoc("Pocket Lanes")
if (pocketDoc = NIL) then
    MsgBox.Error("Pocket lane table not found.", "Validate Pocket Lane Table")
    exit
end
pocketVTab = pocketDoc.GetVTab

'   Get the log file.
logFile = FileDialog.Put("pocket.log".AsFileName, "*.log", "Log File for Validation
Results")
if (logFile = NIL) then
    exit
end
log = LineFile.Make(logFile, #FILE_PERM_CLEARMODIFY)
log.WriteELT("")
log.WriteELT("")
log.WriteELT("Pocket Lane Table Validation Log  -  " + Date.Now.AsString)
fail = FALSE

'   ISSUE(bwb): allow the user to decide here which tests to perform

'   ISSUE(bwb): perform the tests only for the selected records

'   Validate.
while (TRUE)
    linkVTab.GetSelection.ClearAll
    pocketVTab.GetSelection.ClearAll

    '   Check the field names.
    av.Run("InedValidate.ShowMessage", {"Checking field names . . .", log})
    idField = av.Run("InedValidate.CheckField", {pocketVTab, "ID", log, TRUE})
    fail = (idField = NIL) or fail
    nodeField = av.Run("InedValidate.CheckField", {pocketVTab, "NODE", log, TRUE})
    fail = (nodeField = NIL) or fail
    linkField = av.Run("InedValidate.CheckField", {pocketVTab, "LINK", log, TRUE})
    fail = (linkField = NIL) or fail
    offsetField = av.Run("InedValidate.CheckField", {pocketVTab, "OFFSET", log, TRUE})
    fail = (offsetField = NIL) or fail
    laneField = av.Run("InedValidate.CheckField", {pocketVTab, "LANE", log, TRUE})
    fail = (laneField = NIL) or fail
    styleField = av.Run("InedValidate.CheckField", {pocketVTab, "STYLE", log, FALSE})
    fail = (styleField = NIL) or fail
    lengthField = av.Run("InedValidate.CheckField", {pocketVTab, "LENGTH", log, TRUE})
    fail = (lengthField = NIL) or fail

    '   Check for fatal error.
    if (fail) then
        break
```

```
    end

    ’   Check field values.
    av.Run("InedValidate.ShowMessage", {"Checking styles . . .", log})
    fail = av.Run("InedValidate.CheckValues", {pocketVTab, styleField, log, {"T", "P",
"M"}}) or fail

    ’   Check for fatal error.
    if (fail) then
        break
    end

    ’   Check ids.
    av.Run("InedValidate.ShowMessage", {"Checking ids . . .", log})
    ids = Dictionary.Make(pocketVTab.GetNumRecords)
    count = 0
    av.ShowStopButton
    for each i in pocketVTab
        id = pocketVTab.ReturnValueNumber(idField, i)
        if (ids.Get(id) = TRUE) then
            log.WriteELT("ERROR: the id " + id.SetFormat("dddddddddd").AsString + " is
duplicated.")
            pocketVTab.GetSelection.Set(i)
            fail = TRUE
        else
            ids.Add(id, TRUE)
        end
        if ((id < 1) or (id > 4294967295)) then
            log.WriteELT("ERROR: illegal value for id " +
id.SetFormat("dddddddddd").AsString + ".")
            pocketVTab.GetSelection.Set(i)
            fail = TRUE
        end
        count = count + 1
        if (av.SetStatus(100 * count / pocketVTab.GetNumRecords).Not) then
            break
        end
    end
    ids.Empty
    av.SetStatus(100)

    ’   Check connectivity.
    av.Run("InedValidate.ShowMessage", {"Checking connectivity . . .", log})
    idLinkField = linkVTab.FindField("ID")
    nodeaLinkField = linkVTab.FindField("NODEA")
    nodebLinkField = linkVTab.FindField("NODEB")
    permanentlanesaLinkField = linkVTab.FindField("PERMLANESA")
    permanentlanesbLinkField = linkVTab.FindField("PERMLANESB")
    leftpocketsaLinkField = linkVTab.FindField("LEFTPCKTSA")
    leftpocketsbLinkField = linkVTab.FindField("LEFTPCKTSB")
    rightpocketsaLinkField = linkVTab.FindField("RGHTPCKTSA")
    rightpocketsbLinkField = linkVTab.FindField("RGHTPCKTSB")
    lengthLinkField = linkVTab.FindField("LENGTH")
    setbackaLinkField = linkVTab.FindField("SETBACKA")
    setbackbLinkField = linkVTab.FindField("SETBACKB")
    links = Dictionary.Make(linkVTab.GetNumRecords)
    for each i in linkVTab
        id = linkVTab.ReturnValueNumber(idLinkField, i)
        length = linkVTab.ReturnValueNumber(lengthLinkField, i)
        setbacka = linkVTab.ReturnValueNumber(setbackaLinkField, i)
        setbackb = linkVTab.ReturnValueNumber(setbackbLinkField, i)
        nodea = linkVTab.ReturnValueNumber(nodeaLinkField, i)
        perm = linkVTab.ReturnValueNumber(permanentlanesaLinkField, i)
        left = linkVTab.ReturnValueNumber(leftpocketsaLinkField, i)
        right = linkVTab.ReturnValueNumber(rightpocketsaLinkField, i)
        pocketsa = List.Make
        if (left > 0) then
            for each lane in 1..left
                pocketsa.Add(lane)
            end
        end
        if (right > 0) then
            for each lane in (left + perm + 1)..(left + perm + right)
                pocketsa.Add(lane)
            end
        end
```

```
        nodeb = linkVTab.ReturnValueNumber(nodebLinkField, i)
        perm = linkVTab.ReturnValueNumber(permanentlanesbLinkField, i)
        left = linkVTab.ReturnValueNumber(leftpocketsbLinkField, i)
        right = linkVTab.ReturnValueNumber(rightpocketsbLinkField, i)
        pocketsb = List.Make
        if (left > 0) then
            for each lane in 1..left
                pocketsb.Add(lane)
            end
        end
        if (right > 0) then
            for each lane in (left + perm + 1)..(left + perm + right)
                pocketsb.Add(lane)
            end
        end
        links.Add(id, {nodea, nodeb, setbacka, setbackb, length, pocketsa, pocketsb})
    end
    count = 0
    av.ShowStopButton
    for each i in pocketVTab
        id = pocketVTab.ReturnValueNumber(idField, i)
        node = pocketVTab.ReturnValueNumber(nodeField, i)
        linkData = links.Get(pocketVTab.ReturnValueNumber(linkField, i))
        offset = pocketVTab.ReturnValueNumber(offsetField, i)
        lane = pocketVTab.ReturnValueNumber(laneField, i)
        style = pocketVTab.ReturnValueNumber(styleField, i)
        length = pocketVTab.ReturnValueNumber(lengthField, i)
        if (linkData = NIL) then
            log.WriteELT("ERROR: id " + id.SetFormat("dddddddddd").AsString + " references
a non-existent link.")
            pocketVTab.GetSelection.Set(i)
            fail = TRUE
            continue
        end
        if (node = linkData.Get(0)) then
            iNode = 0
        elseif (node = linkData.Get(1)) then
            iNode = 1
        else
            log.WriteELT("ERROR: id " + id.SetFormat("dddddddddd").AsString + " references
a non-existent node.")
            pocketVTab.GetSelection.Set(i)
            fail = TRUE
            continue
        end
        j = linkData.Get(5 + iNode).FindByValue(lane)
        if (j = -1) then
            log.WriteELT("INFO: id " + id.SetFormat("dddddddddd").AsString + " may be
inconsistent with link table or a duplicate--please check this manually.")
            pocketVTab.GetSelection.Set(i)
            fail = TRUE
        else
            linkData.Get(5 + iNode).Remove(j)
        end
        if (((style = "T") or (style = "M")) and (offset <> 0)) then
            log.WriteELT("WARNING: id " + id.SetFormat("dddddddddd").AsString + " does not
have zero offset.")
            pocketVTab.GetSelection.Set(i)
            fail = TRUE
        end
        if (((style = "T") or (style = "M")) and (length > (linkData.Get(4) -
linkData.Get(2) - linkData.Get(3)))) then
            log.WriteELT("WARNING: id " + id.SetFormat("dddddddddd").AsString + " is too
long.")
            pocketVTab.GetSelection.Set(i)
            fail = TRUE
        end
        if ((style = "P") and ((offset < linkData.Get(2)) or ((offset + length) >
(linkData.Get(4) - linkData.Get(3))))) then
            log.WriteELT("WARNING: id " + id.SetFormat("dddddddddd").AsString + " is too
long.")
            pocketVTab.GetSelection.Set(i)
            fail = TRUE
        end
        count = count + 1
        if (av.SetStatus(100 * count / pocketVTab.GetNumRecords).Not) then
```

```
            break
        end
    end
    av.SetStatus(100)
    for each i in linkVTab
        id = linkVTab.ReturnValueNumber(idLinkField, i)
        linkData = links.Get(id)
        for each j in linkData.Get(5)
            log.WriteELT("ERROR: no data for lane " + j.AsString + " on link id " +
id.SetFormat("dddddddddd").AsString + " towards node id " +
linkData.Get(0).SetFormat("dddddddddd").AsString + ".")
            linkVTab.GetSelection.Set(i)
            fail = TRUE
        end
        for each j in linkData.Get(6)
            log.WriteELT("ERROR: no data for lane " + j.AsString + " on link id " +
id.SetFormat("dddddddddd").AsString + " towards node id " +
linkData.Get(1).SetFormat("dddddddddd").AsString + ".")
            linkVTab.GetSelection.Set(i)
            fail = TRUE
        end
    end

    ’   Cleanup and exit.
    linkVTab.UpdateSelection
    pocketVTab.UpdateSelection
    break
end
if (fail) then
    message = "Errors have been detected and selected."
else
    message = "No errors have been detected."
end
av.Run("InedValidate.ShowMessage", {message, log})
log.Close
MsgBox.Info(message, "Validate Pocket Lane Table")
if (fail) then
    TextWin.Make(logFile, "Pocket Lane Table Validation Results")
end
```

11. InedValidate.Validate.Signalized.ave

```
’   Project: TRANSIMS
’   Subsystem: Input Editor
’   $RCSfile: InedValidate.ValidateSignalized.ave,v $
’   $Revision: 1.0 $
’   $Date: 1996/04/25 18:20:38 $
’   $State: Rel $
’   $Author: bwb $
’   U.S. Government Copyright 1995
’   All rights reserved


’   This script checks to see that a signalized node table is valid.


’   Get the input tables.
nodeDoc = av.GetProject.FindDoc("Nodes")
if (nodeDoc = NIL) then
    MsgBox.Error("Node table not found.", "Validate Signalized Node Table")
    exit
end
nodeVTab = nodeDoc.GetVTab
unsignalizedDoc = av.GetProject.FindDoc("Unsignalized Nodes")
if (unsignalizedDoc = NIL) then
    MsgBox.Error("Unsignalized node table not found.", "Validate Signalized Node Table")
    exit
end
unsignalizedVTab = unsignalizedDoc.GetVTab
timingDoc = av.GetProject.FindDoc("Timing Plans")
if (timingDoc = NIL) then
    MsgBox.Error("Timing plan table not found.", "Validate Signalized Node Table")
    exit
end
timingVTab = timingDoc.GetVTab
```

```
signalizedDoc = av.GetProject.FindDoc("Signalized Nodes")
if (signalizedDoc = NIL) then
    MsgBox.Error("Signalized node table not found.", "Validate Signalized Node Table")
    exit
end
signalizedVTab = signalizedDoc.GetVTab

'   Get the log file.
logFile = FileDialog.Put("signalized.log".AsFileName, "*.log", "Log File for Validation
Results")
if (logFile = NIL) then
    exit
end
log = LineFile.Make(logFile, #FILE_PERM_CLEARMODIFY)
log.WriteELT("")
log.WriteELT("")
log.WriteELT("Signalized Node Table Validation Log  -  " + Date.Now.AsString)
fail = FALSE

'   ISSUE(bwb): allow the user to decide here which tests to perform

'   ISSUE(bwb): perform the tests only for the selected records

'   Validate.
while (TRUE)
    nodeVTab.GetSelection.ClearAll
    timingVTab.GetSelection.ClearAll
    signalizedVTab.GetSelection.ClearAll

    '   Check the field names.
    av.Run("InedValidate.ShowMessage", {"Checking field names . . .", log})
    nodeField = av.Run("InedValidate.CheckField", {signalizedVTab, "NODE", log, TRUE})
    fail = (nodeField = NIL) or fail
    typeField = av.Run("InedValidate.CheckField", {signalizedVTab, "TYPE", log, FALSE})
    fail = (typeField = NIL) or fail
    planField = av.Run("InedValidate.CheckField", {signalizedVTab, "PLAN", log, TRUE})
    fail = (planField = NIL) or fail
    offsetField = av.Run("InedValidate.CheckField", {signalizedVTab, "OFFSET", log, TRUE})
    fail = (offsetField = NIL) or fail
    starttimeField = av.Run("InedValidate.CheckField", {signalizedVTab, "STARTTIME", log,
FALSE})
    fail = (starttimeField = NIL) or fail

    '   Check for fatal error.
    if (fail) then
        break
    end

    '   Check field values.
    av.Run("InedValidate.ShowMessage", {"Checking values . . .", log})
    fail = av.Run("InedValidate.CheckValues", {signalizedVTab, typeField, log, {"T",
"A"}}) or fail
    fail = av.Run("InedValidate.CheckPositive", {signalizedVTab, offsetField, log}) or
fail

    '   Check connectivity.
    av.Run("InedValidate.ShowMessage", {"Checking connectivity . . .", log})
    idNodeField = nodeVTab.FindField("ID")
    nodes = Dictionary.Make(nodeVTab.GetNumRecords)
    for each i in nodeVTab
        nodes.Add(nodeVTab.ReturnValueNumber(idNodeField, i), FALSE)
    end
    nodeUnsignalizedField = unsignalizedVTab.FindField("NODE")
    for each i in unsignalizedVTab
        node = unsignalizedVTab.ReturnValueNumber(nodeUnsignalizedField, i)
        nodes.Set(node, TRUE)
    end
    planTimingField = timingVTab.FindField("PLAN")
    plans = Dictionary.Make(timingVTab.GetNumRecords)
    for each i in timingVTab
        plans.Add(timingVTab.ReturnValueNumber(planTimingField, i), FALSE)
    end
    count = 0
    av.ShowStopButton
    for each i in signalizedVTab
        node = signalizedVTab.ReturnValueNumber(nodeField, i)
```

```
        plan = signalizedVTab.ReturnValueNumber(planField, i)
        if (nodes.Get(node) = NIL) then
            log.WriteELT("ERROR: node id " + node.SetFormat("dddddddddd").AsString + "
does not exist.")
            signalizedVTab.GetSelection.Set(i)
            fail = TRUE
        elseif (nodes.Get(node)) then
            log.WriteELT("ERROR: node id " + node.SetFormat("dddddddddd").AsString + " has
a duplicate signalized or unsignalized control.")
            signalizedVTab.GetSelection.Set(i)
            fail = TRUE
         else
            nodes.Set(node, TRUE)
        end
        if (plans.Get(plan) = NIL) then
            log.WriteELT("ERROR: node id " + node.SetFormat("dddddddddd").AsString + "
references non-existent plan " + plan.AsString + ".")
            signalizedVTab.GetSelection.Set(i)
            fail = TRUE
        else
            plans.Set(plan, TRUE)
        end
        count = count + 1
        if (av.SetStatus(100 * count / signalizedVTab.GetNumRecords).Not) then
            break
        end
    end
    av.SetStatus(100)
    for each i in nodeVTab
        node = nodeVTab.ReturnValueNumber(idNodeField, i)
        if (nodes.Get(node) = FALSE) then
            log.WriteELT("ERROR: node id " + node.SetFormat("dddddddddd").AsString + " has
no control.")
            nodeVTab.GetSelection.Set(i)
            fail = TRUE
        end
    end
    for each i in timingVTab
        plan = timingVTab.ReturnValueNumber(planTimingField, i)
        if (plans.Get(plan) = FALSE) then
            log.WriteELT("WARNING: plan " + plan.AsString + " has not been used.")
            timingVTab.GetSelection.Set(i)
            fail = TRUE
        end
    end

    '   Check start times.
    av.Run("InedValidate.ShowMessage", {"Checking start times . . .", log})
    count = 0
    av.ShowStopButton
    for each i in signalizedVTab
        node = signalizedVTab.ReturnValueNumber(nodeField, i)
        starttime = signalizedVTab.ReturnValueString(starttimeField, i)
        if (starttime.Count <> 8) then
            bad = TRUE
        elseif ({"SUN", "MON", "TUE", "WED", "THU", "FRI", "SAT", "WKE", "WKD",
"ALL"}.FindByValue(starttime.Left(3)) = -1) then
            bad = TRUE
        elseif (starttime.Middle(5, 1) <> ":") then
            log.WriteELT("8")
            bad = TRUE
        elseif ((starttime.Middle(3, 2).AsNumber > 24) and (starttime.Middle(6,
2).AsNumber >= 60)) then
            bad = TRUE
        else
            bad = FALSE
        end
        if (bad) then
            log.WriteELT("ERROR: invalid start time for node " +
node.SetFormat("dddddddddd").AsString + ".")
            signalizedVTab.GetSelection.Set(i)
            fail = TRUE
        end
        count = count + 1
        if (av.SetStatus(100 * count / signalizedVTab.GetNumRecords).Not) then
            break
```

```
        end
    end

    ’   Cleanup and exit.
    nodeVTab.UpdateSelection
    timingVTab.UpdateSelection
    signalizedVTab.UpdateSelection
    break
end
if (fail) then
    message = "Errors have been detected and selected."
else
    message = "No errors have been detected."
end
av.Run("InedValidate.ShowMessage", {message, log})
log.Close
MsgBox.Info(message, "Validate Signalized Node Table")
if (fail) then
    TextWin.Make(logFile, "Signalized Node Table Validation Results")
end
```

12. InedValidate.ValidateTiming.ave

```
’   Project: TRANSIMS
’   Subsystem: Input Editor
’   $RCSfile: InedValidate.ValidateTiming.ave,v $
’   $Revision: 1.0 $
’   $Date: 1996/04/25 18:20:38 $
’   $State: Rel $
’   $Author: bwb $
’   U.S. Government Copyright 1995
’   All rights reserved


’   This script checks to see that a timing plan table is valid.


’   Get the input table.
timingDoc = av.GetProject.FindDoc("Timing Plans")
if (timingDoc = NIL) then
    MsgBox.Error("Timing plan table not found.", "Validate Timing Plan Table")
    exit
end
timingVTab = timingDoc.GetVTab

’   Get the log file.
logFile = FileDialog.Put("timing.log".AsFileName, "*.log", "Log File for Validation
Results")
if (logFile = NIL) then
    exit
end
log = LineFile.Make(logFile, #FILE_PERM_CLEARMODIFY)
log.WriteELT("")
log.WriteELT("")
log.WriteELT("Timing Plan Table Validation Log  -  " + Date.Now.AsString)
fail = FALSE

’   ISSUE(bwb): allow the user to decide here which tests to perform

’   ISSUE(bwb): perform the tests only for the selected records

’   Validate.
while (TRUE)
    timingVTab.GetSelection.ClearAll

    ’   Check the field names.
    av.Run("InedValidate.ShowMessage", {"Checking field names . . .", log})
    planField = av.Run("InedValidate.CheckField", {timingVTab, "PLAN", log, TRUE})
    fail = (planField = NIL) or fail
    phaseField = av.Run("InedValidate.CheckField", {timingVTab, "PHASE", log, TRUE})
    fail = (phaseField = NIL) or fail
    nextphasesField = av.Run("InedValidate.CheckField", {timingVTab, "NEXTPHASES", log,
FALSE})
    fail = (nextphasesField = NIL) or fail
    greenminField = av.Run("InedValidate.CheckField", {timingVTab, "GREENMIN", log, TRUE})
```

```
    fail = (greenminField = NIL) or fail
    greenmaxField = av.Run("InedValidate.CheckField", {timingVTab, "GREENMAX", log, TRUE})
    fail = (greenmaxField = NIL) or fail
    greenextField = av.Run("InedValidate.CheckField", {timingVTab, "GREENEXT", log, TRUE})
    fail = (greenextField = NIL) or fail
    yellowField = av.Run("InedValidate.CheckField", {timingVTab, "YELLOW", log, TRUE})
    fail = (yellowField = NIL) or fail
    redclearField = av.Run("InedValidate.CheckField", {timingVTab, "REDCLEAR", log, TRUE})
    fail = (redclearField = NIL) or fail

    ’   Check for fatal error.
    if (fail) then
        break
    end

    ’   Check ids.
    av.Run("InedValidate.ShowMessage", {"Checking ids . . .", log})
    plans = Dictionary.Make(timingVTab.GetNumRecords)
    count = 0
    av.ShowStopButton
    for each i in timingVTab
        plan = timingVTab.ReturnValueNumber(planField, i)
        phase = timingVTab.ReturnValueNumber(phaseField, i)
        if ((plan < 1) or (plan > 255)) then
            log.WriteELT("ERROR: illegal value for plan " + plan.AsString + ".")
            timingVTab.GetSelection.Set(i)
            fail = TRUE
        end
        if ((phase < 1) or (phase > 255)) then
            log.WriteELT("ERROR: illegal value for phase " + phase.AsString + " in plan "
+ plan.AsString + ".")
            timingVTab.GetSelection.Set(i)
            fail = TRUE
        end
        if (plans.Get(plan) = NIL) then
            plans.Set(plan, List.Make)
        end
        phases = plans.Get(plan)
        if (phases.FindByValue(phase) = -1) then
            phases.Add(phase)
        else
            log.WriteELT("ERROR: the phase " + phase.AsString + " in plan " +
plan.AsString + " is duplicated.")
            timingVTab.GetSelection.Set(i)
            fail = TRUE
        end
        count = count + 1
        if (av.SetStatus(100 * count / timingVTab.GetNumRecords).Not) then
            break
        end
    end
    av.SetStatus(100)

    ’   Check field values.
    av.Run("InedValidate.ShowMessage", {"Checking values . . .", log})
    fail = av.Run("InedValidate.CheckPositive", {timingVTab, greenminField, log}) or fail
    fail = av.Run("InedValidate.CheckPositive", {timingVTab, greenmaxField, log}) or fail
    fail = av.Run("InedValidate.CheckPositive", {timingVTab, greenextField, log}) or fail
    fail = av.Run("InedValidate.CheckPositive", {timingVTab, yellowField, log}) or fail
    fail = av.Run("InedValidate.CheckPositive", {timingVTab, redclearField, log}) or fail

    ’   Check timings.
    av.Run("InedValidate.ShowMessage", {"Checking green timings . . .", log})
    count = 0
    av.ShowStopButton
    for each i in timingVTab
        plan = timingVTab.ReturnValueNumber(planField, i)
        phase = timingVTab.ReturnValueNumber(phaseField, i)
        greenmin = timingVTab.ReturnValueNumber(greenminField, i)
        greenmax = timingVTab.ReturnValueNumber(greenmaxField, i)
        greenext = timingVTab.ReturnValueNumber(greenextField, i)
        if (greenmax = 0) then
            if (greenext <> 0) then
                log.WriteELT("ERROR: the green interval extension is too large for phase "
+ phase.AsString + " of plan " + plan.AsString + ".")
                timingVTab.GetSelection.Set(i)
```

```
                fail = TRUE
            end
        elseif (greenmin > greenmax) then
            log.WriteELT("ERROR: the green interval maximum is too small for phase " +
phase.AsString + " of plan " + plan.AsString + ".")
            timingVTab.GetSelection.Set(i)
            fail = TRUE
        end
        count = count + 1
        if (av.SetStatus(100 * count / timingVTab.GetNumRecords).Not) then
            break
        end
    end
    av.SetStatus(100)

    '   Check sequence.
    av.Run("InedValidate.ShowMessage", {"Checking phase sequence . . .", log})
    count = 0
    for each i in timingVTab
        plan = timingVTab.ReturnValueNumber(planField, i)
        phase = timingVTab.ReturnValueNumber(phaseField, i)
        nextphases = timingVTab.ReturnValueString(nextphasesField, i).AsTokens("/")
        phases = plans.Get(plan)
        for each j in nextphases
            if (phases.FindByValue(j.AsNumber) = -1) then
                log.WriteELT("ERROR: " + j + " is not a valid next phase in phase " +
phase.AsString + " of plan " + plan.AsString + ".")
                timingVTab.GetSelection.Set(i)
                fail = TRUE
            end
        end
        count = count + 1
        if (av.SetStatus(100 * count / timingVTab.GetNumRecords).Not) then
            break
        end
    end
    av.SetStatus(100)

'   Cleanup and exit.
    timingVTab.UpdateSelection
    break
end
if (fail) then
    message = "Errors have been detected and selected."
else
    message = "No errors have been detected."
end
av.Run("InedValidate.ShowMessage", {message, log})
log.Close
MsgBox.Info(message, "Validate Timing Plan Table")
if (fail) then
    TextWin.Make(logFile, "Timing Plan Table Validation Results")
end
```

13. InedValidate.ValidateUnsignalized.ave

```
'   Project: TRANSIMS
'   Subsystem: Input Editor
'   $RCSfile: InedValidate.ValidateUnsignalized.ave,v $
'   $Revision: 1.1 $
'   $Date: 1996/09/19 16:03:28 $
'   $State: Rel $
'   $Author: bwb $
'   U.S. Government Copyright 1995
'   All rights reserved


'   This script checks to see that an unsignalized node table is valid.


'   Get the input tables.
linkDoc = av.GetProject.FindDoc("Links")
if (linkDoc = NIL) then
    MsgBox.Error("Link table not found.", "Validate Unsignalized Node Table")
    exit
```

```
end
linkVTab = linkDoc.GetVTab
unsignalizedDoc = av.GetProject.FindDoc("Unsignalized Nodes")
if (unsignalizedDoc = NIL) then
    MsgBox.Error("Unsignalized node table not found.", "Validate Unsignalized Node Table")
    exit
end
unsignalizedVTab = unsignalizedDoc.GetVTab

'   Get the log file.
logFile = FileDialog.Put("unsignalized.log".AsFileName, "*.log", "Log File for Validation
Results")
if (logFile = NIL) then
    exit
end
log = LineFile.Make(logFile, #FILE_PERM_CLEARMODIFY)
log.WriteELT("")
log.WriteELT("")
log.WriteELT("Unsignalized Node Table Validation Log  -  " + Date.Now.AsString)
fail = FALSE

'   ISSUE(bwb): allow the user to decide here which tests to perform

'   ISSUE(bwb): perform the tests only for the selected records

'   Validate.
while (TRUE)
    linkVTab.GetSelection.ClearAll
    unsignalizedVTab.GetSelection.ClearAll

    '   Check the field names.
    av.Run("InedValidate.ShowMessage", {"Checking field names . . .", log})
    nodeField = av.Run("InedValidate.CheckField", {unsignalizedVTab, "NODE", log, TRUE})
    fail = (nodeField = NIL) or fail
    inlinkField = av.Run("InedValidate.CheckField", {unsignalizedVTab, "INLINK", log,
TRUE})
    fail = (inlinkField = NIL) or fail
    signField = av.Run("InedValidate.CheckField", {unsignalizedVTab, "SIGN", log, FALSE})
    fail = (signField = NIL) or fail

    '   Check for fatal error.
    if (fail) then
        break
    end

    '   Check field values.
    av.Run("InedValidate.ShowMessage", {"Checking signs . . .", log})
    fail = av.Run("InedValidate.CheckValues", {unsignalizedVTab, signField, log, {"S",
"Y", "N"}}) or fail

    '   Check connectivity.
    av.Run("InedValidate.ShowMessage", {"Checking connectivity . . .", log})
    idLinkField = linkVTab.FindField("ID")
    nodeaLinkField = linkVTab.FindField("NODEA")
    nodebLinkField = linkVTab.FindField("NODEB")
    permanentlanesaLinkField = linkVTab.FindField("PERMLANESA")
    permanentlanesbLinkField = linkVTab.FindField("PERMLANESB")
    leftpocketsaLinkField = linkVTab.FindField("LEFTPCKTSA")
    leftpocketsbLinkField = linkVTab.FindField("LEFTPCKTSB")
    rightpocketsaLinkField = linkVTab.FindField("RGHTPCKTSA")
    rightpocketsbLinkField = linkVTab.FindField("RGHTPCKTSB")
    links = Dictionary.Make(linkVTab.GetNumRecords)
    for each i in linkVTab
        id = linkVTab.ReturnValueNumber(idLinkField, i)
        nodea = linkVTab.ReturnValueNumber(nodeaLinkField, i)
        nodeb = linkVTab.ReturnValueNumber(nodebLinkField, i)
        links.Add(id, {nodea, nodeb, FALSE, FALSE})
    end
    nodes = Dictionary.Make(unsignalizedVTab.GetNumRecords)
    count = 0
    av.ShowStopButton
    for each i in unsignalizedVTab
        node = unsignalizedVTab.ReturnValueNumber(nodeField, i)
        inlink = unsignalizedVTab.ReturnValueNumber(inlinkField, i)
        linkData = links.Get(inlink)
        if (linkData = NIL) then
```

```
            log.WriteELT("ERROR: link id " + inlink.SetFormat("dddddddddd").AsString + "
references a non-existent link.")
            unsignalizedVTab.GetSelection.Set(i)
            fail = TRUE
            continue
        end
        if (node = linkData.Get(0)) then
            iNode = 0
        elseif (node = linkData.Get(1)) then
            iNode = 1
        else
            log.WriteELT("ERROR: link id " + inlink.SetFormat("dddddddddd").AsString + "
references the non-existent node " + node.SetFormat("dddddddddd").AsString + ".")
            unsignalizedVTab.GetSelection.Set(i)
            fail = TRUE
            continue
        end
        if (linkData.Get(2 + iNode)) then
            log.WriteELT("ERROR: link id " + inlink.SetFormat("dddddddddd").AsString + "
duplicates a reference to the node " + node.SetFormat("dddddddddd").AsString + ".")
            unsignalizedVTab.GetSelection.Set(i)
            fail = TRUE
        else
            linkData.Set(2 + iNode, TRUE)
        end
        nodes.Add(node, TRUE)
        count = count + 1
        if (av.SetStatus(100 * count / unsignalizedVTab.GetNumRecords).Not) then
            break
        end
    end
    av.SetStatus(100)
    for each i in linkVTab
        id = linkVTab.ReturnValueNumber(idLinkField, i)
        nodea = linkVTab.ReturnValueNumber(nodeaLinkField, i)
        nodeb = linkVTab.ReturnValueNumber(nodebLinkField, i)
        lanesa = linkVTab.ReturnValueNumber(permanentlanesaLinkField, i) +
linkVTab.ReturnValueNumber(leftpocketsaLinkField, i) +
linkVTab.ReturnValueNumber(rightpocketsaLinkField, i)
        lanesb = linkVTab.ReturnValueNumber(permanentlanesbLinkField, i) +
linkVTab.ReturnValueNumber(leftpocketsbLinkField, i) +
linkVTab.ReturnValueNumber(rightpocketsbLinkField, i)
        linkData = links.Get(id)
        if ((linkData.Get(2) = FALSE) and (nodes.Get(nodea) = TRUE) and (lanesa > 0)) then
            log.WriteELT("WARNING: incoming link id " +
id.SetFormat("dddddddddd").AsString + " at node id " +
nodea.SetFormat("dddddddddd").AsString + " has no control.")
            linkVTab.GetSelection.Set(i)
            fail = TRUE
        end
        if ((linkData.Get(3) = FALSE) and (nodes.Get(nodeb) = TRUE) and (lanesb > 0)) then
            log.WriteELT("WARNING: incoming link id " +
id.SetFormat("dddddddddd").AsString + " at node id " +
nodeb.SetFormat("dddddddddd").AsString + " has no control.")
            linkVTab.GetSelection.Set(i)
            fail = TRUE
        end
    end

    '   Cleanup and exit.
    linkVTab.UpdateSelection
    unsignalizedVTab.UpdateSelection
    break
end
if (fail) then
    message = "Errors have been detected and selected."
else
    message = "No errors have been detected."
end
av.Run("InedValidate.ShowMessage", {message, log})
log.Close
MsgBox.Info(message, "Validate Unsignalized Node Table")
if (fail) then
    TextWin.Make(logFile, "Unsignalized Node Table Validation Results")
end
```