Technical Report Documentation Page

1. Report No. FHWA/TX-95-1308-1F	2. Government Accession No.	3. Recipient's Catalog No.

4. Title and Subtitle INTERACTIVE GRAPHICS INTERSECTION DESIGN USER'S MANUAL	5. Report Date December 1994 6. Performing Organization Code
7. Author(s) Thomas W. Rioux, Robert F. Inman, Randy B. Machemehl, and Clyde E. Lee	8. Performing Organization Report No. Research Report 1308-1F
9. Performing Organization Name and Address Center for Transportation Research The University of Texas at Austin 3208 Red River, Suite 200 Austin, Texas 78705-2650	10. Work Unit No. (TRAIS) 11. Contract or Grant No. Research Study 0-1308
12. Sponsoring Agency Name and Address Texas Department of Transportation Research and Technology Transfer Office P. O. Box 5051 Austin, Texas 78763-5051	13. Type of Report and Period Covered Final 14. Sponsoring Agency Code

15. Supplementary Notes

Study conducted in cooperation with the U.S. Department of Transportation, Federal Highway Administration.
Research study title: "Interactive Graphics Intersection Design"

16. Abstract

The Interactive Graphics Intersection Design System (IGIDS) has been developed at the Center for Transportation Research at The University of Texas at Austin in cooperation with the Texas Department of Transportation and the Federal Highway Administration. IGIDS is a package of drawing, analysis, and data manipulation tools for use by the designer of street intersections. These tools are aids to the designer in reviewing and revising existing designs or in the design of new intersections. This report serves as the IGIDS reference manual. It describes the structure of the IGIDS database and contains descriptions of each IGIDS command.

17. Key Words Interactive graphics, intersection design, street intersections, Interactive Graphics Intersection Design System (IGIDS)	18. Distribution Statement No restrictions. This document is available to the public through the National Technical Information Service, Springfield, Virginia 22161.

19. Security Classif. (of this report) Unclassified	20. Security Classif. (of this page) Unclassified	21. No. of Pages 249	22. Price

INTERACTIVE GRAPHICS INTERSECTION DESIGN USER'S MANUAL

Thomas W. Rioux
Robert F. Inman
Randy B. Machemehl
and
Clyde E. Lee

Research Report Number 1308-1F

Research Project 0-1308
Interactive Graphics Intersection Design

conducted for the
Texas Department of Transportation
in cooperation with the
U.S. Department of Transportation
Federal Highway Administration

by the

CENTER FOR TRANSPORTATION RESEARCH
Bureau of Engineering Research
THE UNIVERSITY OF TEXAS AT AUSTIN

December 1994

IMPLEMENTATION

The Interactive Graphics Intersection Design System (IGIDS) has been developed and is recommended for implementation by the Texas Department of Transportation.

It is suggested that implementation be in two stages. First, IGIDS should be routinely used as part of the day to day operations in design offices. The available tools will be extremely useful to the Design Engineer. It is recommended that IGIDS become part of the series of training sessions supported by TxDOT.

Secondly, additional tools and standards should be cataloged for addition to IGIDS, with the goal of making all aspects of intersection design a part of IGIDS. With this done, IGIDS may be adapted as the standard for TxDOT intersection design.

Prepared in cooperation with the Texas Department of Transportation
and the U.S. Department of Transportation, Federal Highway Administration

DISCLAIMERS

The contents of this report reflect the views of the authors, who are responsible for the facts and the accuracy of the data presented herein. The contents do not necessarily reflect the official views or policies of the Federal Highway Administration or the Texas Department of Transportation. This report does not constitute a standard, specification, or regulation. There was no invention or discovery conceived or first actually reduced to practice in the course of or under this contract, including any art, method, process, machine, manufacture, design or composition of matter, or any new and useful improvement thereof, or any variety of plant which is or may be patentable under the patent laws of the United States of America or any foreign country.

NOT INTENDED FOR CONSTRUCTION,
PERMIT, OR BIDDING PURPOSES

Clyde E. Lee, P.E. (Texas No. 20512)
Thomas W. Rioux, P.E. (Texas No. 48008)
Randy B. Machemehl, P.E. (Texas No. 41921)

Research Supervisors

TABLE OF CONTENTS

LIST OF FIGURES

LIST OF TABLES

SUMMARY

The Interactive Graphics Intersection Design System (IGIDS) has been developed at the Center for Transportation Research at the University of Texas at Austin in cooperation with the Texas Departmant of Transportation and the Federal Highway Administration.

IGIDS is a package of drawing, analysis, and data manipulation tools for use by the designer of street intersections. These tools are aids to the designer in reviewing and revising existing designs or in the design of new intersections.

This report is the IGIDS reference manual. It describes the structure of the IGIDS database and contains descriptions of each IGIDS command.

CHAPTER 1 INTRODUCTION

BACKGROUND

Intersection design is a complex process that involves a number of different transportation engineering skills. Elements of transportation planning, traffic engineering, and geometric design contribute to the process of designing the best practicable traffic handling facility. Traditionally, the design engineer has relied upon the application of manual, or sometimes computer-aided, procedures to determine the most appropriate alternative that satisfies the objectives. Intersection design involves geometric layout, traffic analysis, channelization, selection and placement of traffic control devices, timing of traffic signals, lighting, drainage, fuel-consumption evaluation, pollution analysis, cost estimating and many other engineering functions which are eventually reduced to practice in a set of plans and specifications. Virtually all the engineering requirements and analysis procedures needed to complete the intersection design process are well known and documented.

OBJECTIVES

The ultimate objective of this study is the development of an Interactive Graphics Intersection Design System (IGIDS) which assists engineers in the analysis and design of individual, at-grade intersections and which operates on personal computers and workstations. The role of IGIDS is to provide the Intersection Design Engineer with suitable tools to assist with each process stage. The available tools may be loosely divided into three groups.

First are the drawing tools. IGIDS uses MicroStation™[1] as a graphics engine to perform all graphics input and output. The user may select from a library one of several typical intersection designs to be considered and then modify it as needed to define the particular intersection of interest. Alternately, the user may define the intersection geometry by pointing at selected elements in a reference file created by another source, such as a topographical map of the area (with optional use of a superimposed aerial photo). Finally, the user can create the key components of the intersection geometry on the scratch level using MicroStation™ commands (with optional use of a superimposed aerial photo) and then define the intersection geometry by pointing at these elements. After the centerline of each leg is defined, IGIDS has several commands to place lanes and curb returns and manipulate the intersection elements. The elements of the design can be specified by the most convenient method for the particular situation. The width of a lane may sometimes be specified by keyboard entry of a numerical value. At times, it may be more convenient to identify lane edge graphics that exist in a file that is being viewed concurrently with the

[1] MicroStation™ is a trademark of Bentley Systems, Inc.

IGIDS graphics file. Alternately, requesting that the lane edges be located by identifying existing landmarks on a superimposed aerial photo may be most convenient.

Second are the built-in analysis tools. Both graphical aids and computational analysis procedures are incorporated. Vehicle turning templates can be quickly drawn for the standard American Association of State Highway and Transportation Officials (AASHTO) [REF 1] design vehicles to a user-specified turn radius for the turn angle defined by the user-selected legs. These templates can be moved dynamically over the intersection geometry to evaluate pavement edge and channelization requirements. For checking sight distance restrictions, sight lines for stopped vehicles, yielding vehicles, or vehicles approaching an uncontrolled intersection can be drawn. The analysis detailed in Chapter 9 of the Highway Capacity Manual [REF 2] can be used to find v/c ratios and delay values for intersections with pretimed controllers and are displayed in bar chart format for each leg.

Finally, there are data-manipulation tools that prepare data files for analyses; these are executed outside IGIDS and retrieve the results back into the IGIDS-compatible context. The TEXAS Model for Intersection Traffic (TEXAS) [REF 3] and the Signal Operations Analysis Package (SOAP) [REF 4] are supported by IGIDS. The TEXAS Model provides simulation of vehicular traffic flow through a single intersection or a diamond interchange and generates both a statistical summary and animated graphics which show drawn-to-scale, color-coded vehicle types moving through the intersection geometry. Selected TEXAS Model statistics can be displayed in bar chart format for each leg and for the total intersection. SOAP develops and assesses single intersection signal timing plans.

Data for both built-in and external-analysis computer programs are drawn from a common database that is maintained by IGIDS. Many of these data are extracted from the graphical intersection geometry that has been constructed by the user. Some non-graphical data, such as traffic volumes, must be entered through the keyboard.

The development of IGIDS is discussed in the next three chapters, then a scripted beginning example is presented, and finally a scripted example of entering traffic signal data is given. The Appendices contain the IGIDS installation instructions, command menus, command descriptions in alphabetical order, tables of the parameters used for the AASHTO vehicles in the Vehicle Turn Template commands, a list of abbreviations, a definition of terminology, and a command index in alphabetical order with page numbers.

CHAPTER 2 IGIDS SYSTEM DEVELOPMENT

OBJECTIVE

The goal for this study was the development of an Interactive Graphics Intersection Design System (IGIDS) which operates on personal computers and workstations and assists engineers in the analysis and design of individual, at-grade intersections. The functions IGIDS performs were identified early in the study. That identification evolved into definition of the requirements and capabilities of selected functions that the automated system would support. Modular computer applications were structured to permit staged system development. Candidate computer hardware, operating systems, and software applications were selected after a thorough investigation was made to determine capabilities, limitations, and compatibility with other system components. Conceptual development then proceeded to writing the computer code necessary for managing the various IGIDS components so that the desired functions could be realized.

OVERVIEW

Determination of functional performance was achieved through examination of the conventional intersection design procedure. Analysis of the different activities involved in intersection design, together with the general order in which they are accomplished, identified functions which should be incorporated into the system.

The selection of system hardware and software components was based on how each component relates to the intersection design process in addition to ensuring access to IGIDS through the type of hardware currently used by many public and private design professionals.

IGIDS initially required an operating system which allows multi-tasking (the concurrent operation of multiple programs). With the development by Bentley Systems, Incorporated (BSI) of the MicroStation™ Development Language (MDL) in MicroStation™ Version 4, IGIDS could operate in an MS-DOS environment and could be easily ported to other operating and hardware systems supported by MicroStation™ Version 4.

Generic requirements for the system hardware components were designed to permit use of equipment from different manufactures. Development of the first-stage version of IGIDS was performed on Intergraph Corporation's workstation hardware. Intergraph equipment was selected for its versatility and widespread use among public and private engineering professionals. Final development was performed on an IBM compatible personal computer using an Intel Pentium processor and the MS-DOS operating system.

The software needed to perform the graphics operations of IGIDS was selected for it's availability on different types of equipment on which the system would likely be used. The first-stage version of

IGIDS supported the MicroStation™ graphics application from Bentley Systems, Incorporated using the workstation version of MicroStation™ 3 and using the "ms_app_login" type of MicroCSL programming. All programming was performed in the ANSI Standard C language, except for the Texas Truck Off-Tracking Model [REF 5], which was in Fortran. Final development employed the workstation and PC version of MicroStation™ Version 4 using the MDL. All previous software was converted to MDL including the Texas Truck Off-Tracking Model.

The selected combination of hardware components, operating system, and interactive graphics applications, together with design and analysis software which support intersection design, provides a convenient, useful tool for the transportation engineer.

CONCEPTUAL DESIGN

IGIDS is intended to assist the engineer in analyzing and designing individual, at-grade, vehicular-traffic intersections. This involves the development of computer-aided tools for defining the geometry of the intersection, the type and location of traffic control devices, and the traffic flow conditions. IGIDS provides a convenient and user-friendly interface for storing and accessing data that are needed to execute several analysis and design software packages, and also allows for adding or developing related expert systems.

IGIDS uses a graphics engine (software) to perform all interactive graphics operations and to maintain the graphics-engine database. IGIDS software operates above and drives the graphics engine through a higher-level language interface. IGIDS allows the user to switch easily between executing IGIDS commands and graphics-engine commands. The commands available within the graphics engine are used for this purpose as much as possible. IGIDS does not provide any plotting capabilities, but relies upon the graphics engine to perform these operations.

IGIDS accommodates up to 15 alternative designs for one intersection. Existing intersection conditions normally constitute one alternative. Each alternative, and its major graphical component groupings, is placed on separate graphical levels, or planes, so that it can be displayed independently, or not displayed in a particular view, by the graphics engine. IGIDS allocates a user graphical level, or plane, and a scratch graphical level, or plane. The user graphical level is allocated to level 1 in MicroStation™ and all graphics can be viewed but not accessed by IGIDS or MicroStation™. The scratch graphical level is allocated to level 2 in MicroStation™ and all graphics can be viewed and accessed by IGIDS and MicroStation™. All, or part, of an intersection alternative can be copied to another alternative, and all, or part, of an intersection alternative can be modified by IGIDS commands. In addition, any number of reference files can be attached to the master design file by the graphics engine. IGIDS can locate elements in these reference files to be added as graphics for IGIDS. Finally, a raster image, scanned photograph or other raster data, may be displayed by the graphics engine.

In its first-stage development, IGIDS graphics provides a two-dimensional in plan view and uses a state plane coordinate system. There are no programming impediments to a future three-dimensional system. Coordinates, distance, and other real numeric data are stored as 16 significant digit, 64 bit, double precision, floating-point variables in the master units of the graphics engine (normally feet or meters). All angular data are stored as the same type variables, but in degrees. All counter, or indexing-type, numbers are stored as 10 significant digit, 32 bit, integer variables. All other integer numbers with no perceived possibility of exceeding several hundred are stored as 5 significant digit, 16 bit, integer variables.

IGIDS uses relational, hierarchical geometry. Relational geometry refers to the fact that the only absolute coordinate needed by IGIDS is the center of the intersection. A leg of the intersection is defined relative to the center of the intersection; then, the lanes of a leg are defined relative to the leg's centerline, and so on. IGIDS data are stored as objects and IGIDS maintains the parent-child relationships among IGIDS objects. IGIDS works by manipulating a defined set of objects. There is a strict set of rules used by IGIDS that fixes the way in which these IGIDS objects are related to each other. The task of complying with these rules is assigned to IGIDS, so the user need not be too concerned with the relationship details. By using the IGIDS drawing tools to specify and group IGIDS objects, one or more alternative designs can be developed. Both the absolute and relative definitions of an item are calculated and stored. The data can be presented using a relative definition, and the absolute definition will be automatically calculated; or the data can be presented using the absolute definition, and the relative definition will be automatically calculated. IGIDS defines the relative object for each type of IGIDS object. IGIDS calculates the station and offset of a coordinate from the leg centerline for all items that are children of the leg. Only IGIDS commands can be used to manipulate the geometry because of the need to update the data in the IGIDS database. IGIDS generally does not require the user to enter data in a defined order or sequence. To accomplish this objective, IGIDS automatically sorts each list of children IGIDS objects as new children IGIDS objects are added to the list; so, the user can enter geometry data items in any order. IGIDS automatically sets the direction of any entered graphical IGIDS object so that it will be in conformance with the sorted direction of the list of which it is a part.

Hierarchical geometry refers to the fact that the IGIDS objects are related in a parent-child relationship. Each IGIDS object comprises only one parent IGIDS object and can have zero or more children IGIDS objects. An IGIDS object can have more than one parent IGIDS object type with the type of parent being an attribute of the IGIDS object. Each IGIDS object knows the type of its parent IGIDS object and which specific IGIDS object entry is its parent. An IGIDS object can have more than one category of child IGIDS object. The number of children IGIDS objects accommodated by IGIDS is virtually infinite. Each parent IGIDS object maintains the current number of children IGIDS objects and has a pointer to the beginning and ending children IGIDS objects for each category of children IGIDS objects. Each IGIDS object has a pointer to the previous and to the next IGIDS object on the list. An IGIDS object with a null previous-pointer is the first IGIDS object on the list and an IGIDS object with a null next-pointer is the last IGIDS object on the list. An IGIDS object with a null previous-pointer and a null next-pointer is the only IGIDS object on the list. Most higher-level IGIDS objects serve to group the children IGIDS objects, and only the lowest-level IGIDS objects have a graphical representation. Any procedure applied to an IGIDS object is automatically applied to all children of the IGIDS object.

The six IGIDS objects are Intersection, Alternative, Leg, Lane, Seg (Segment), and Text. Figure 2.1 shows the hierarchical relationship. Figures 2.2 through 2.10 show the geometry typically associated with most of the IGIDS objects using the Standard 4X4 Intersection; the IGIDS object is in bold. There is only one intersection IGIDS object, and it has intersection data and a list of up to 15 alternatives. Each alternative IGIDS object has alternative data, an intersection parent pointer, a list of legs, and a list of text (see Figure 2.2). Each leg IGIDS object has leg data, an alternative parent pointer, a list of centerline segments, a list of inbound lanes, a list of outbound lanes, a list of inner edge curb return segments, and a list of outer edge curb return segments (see Figures 2.4 and 2.5). Each lane IGIDS object has lane data, a leg parent pointer, a list of inner edge segments, a list of outer edge segments, and a list of stop line segments (see Figure 2.6). Each segment IGIDS object has data for either an arc of a circle or a line, a leg/lane parent pointer, and a list of texts (see Figures 2.7 through 2.10). Each text IGIDS object has text data and an alternative/segment parent pointer (see Figure 2.3). Each IGIDS object may have either a parent or child relationship with other different-type IGIDS objects. Any IGIDS object may have only one parent, but a parent may have none, one, or more children. In Figure 2.1, for each class of IGIDS object the suitable parent IGIDS object type is found by following a line to the immediate left. Likewise, suitable child IGIDS object types(s) are found on a line to the immediate right. Only Segs and Text have displayable graphic elements. All others have graphical visibility due solely to the attached child IGIDS objects. Although Figure 2.1 shows each IGIDS object having only one child of each type, each IGIDS object can have a virtually unlimited number of children. The one exception to this is that the number of Alternatives is limited to 15. There is always only one Intersection. The leg centerline must be entered and completed before any lanes can be attached. An IGIDS command applied to an IGIDS object is automatically applied by IGIDS to all child IGIDS objects of the selected IGIDS object. The IGIDS "rotate-leg" command thus causes each centerline segment, each inbound lane, each outbound lane, each inner edge curb return segment, and each outer edge curb return segment to be rotated. Then, each lane causes each inner edge, each outer edge, and each stopline segment to be rotated. Finally, each segment causes each text object to be rotated. It is convenient to sub-classify some IGIDS object types. The IGIDS object type Text can possibly have more than one type of parent. When each Text object is created, it is sub-classified to be either Text on a Seg, or Text on an Alternative. This designation remains unchanged for the life of the Text object. Both Lane and Seg can also have sub-classifications. A Lane is either an Inbound Lane or an Outbound Lane. A Seg is either a Lane Inner Edge Seg, Lane Outer Edge

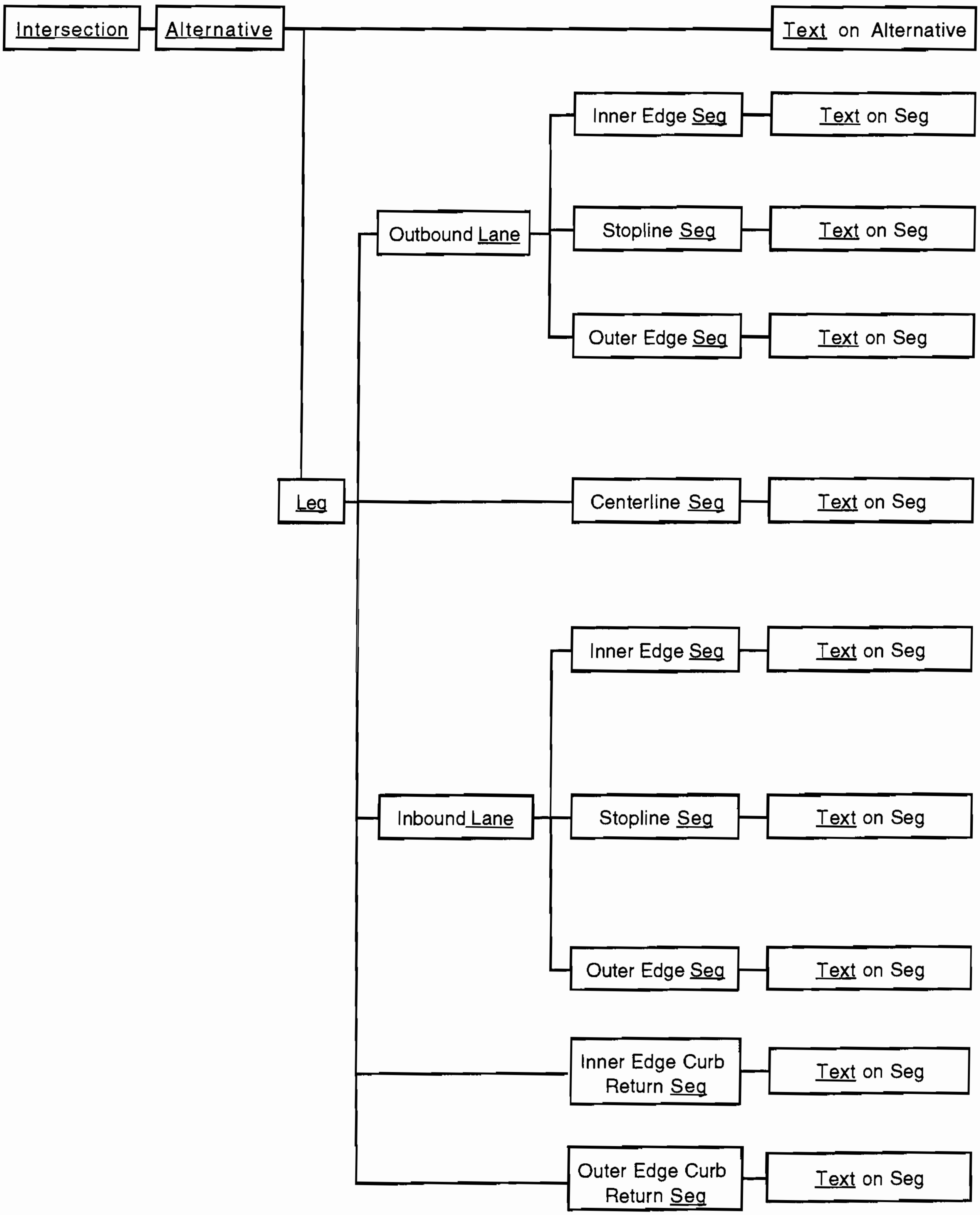


Figure 2.1 Object Hierarchy

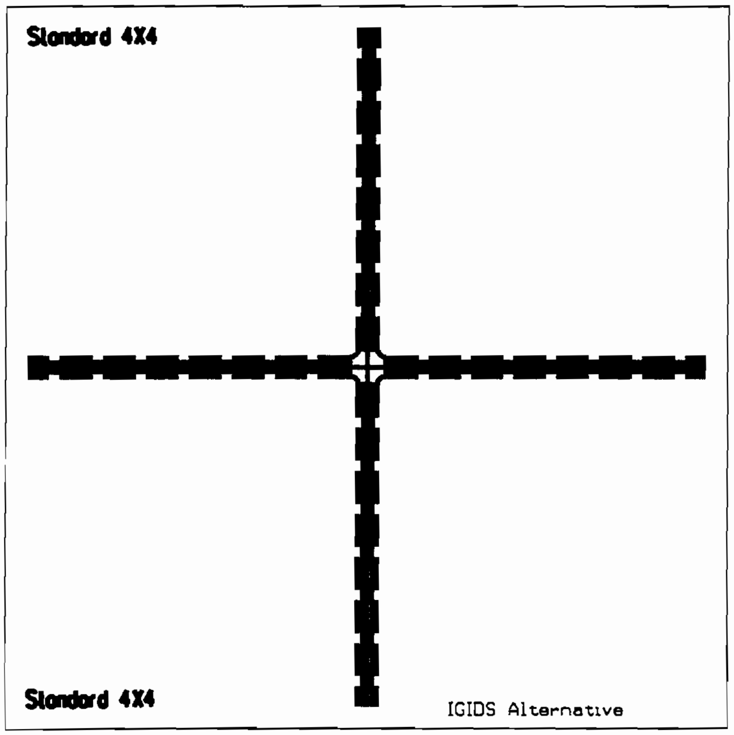


Figure 2.2 IGIDS Alternative

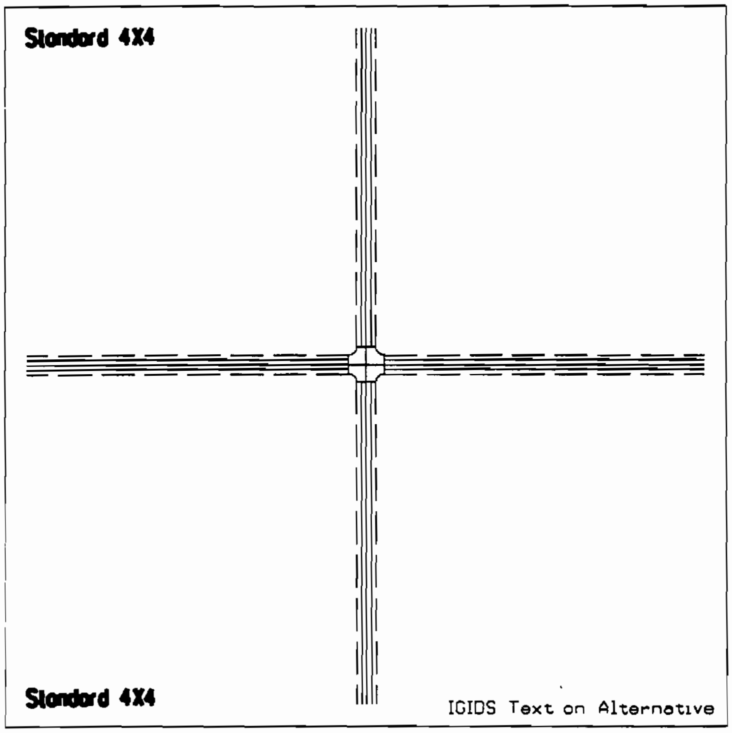


Figure 2.3 IGIDS Text on Alternative

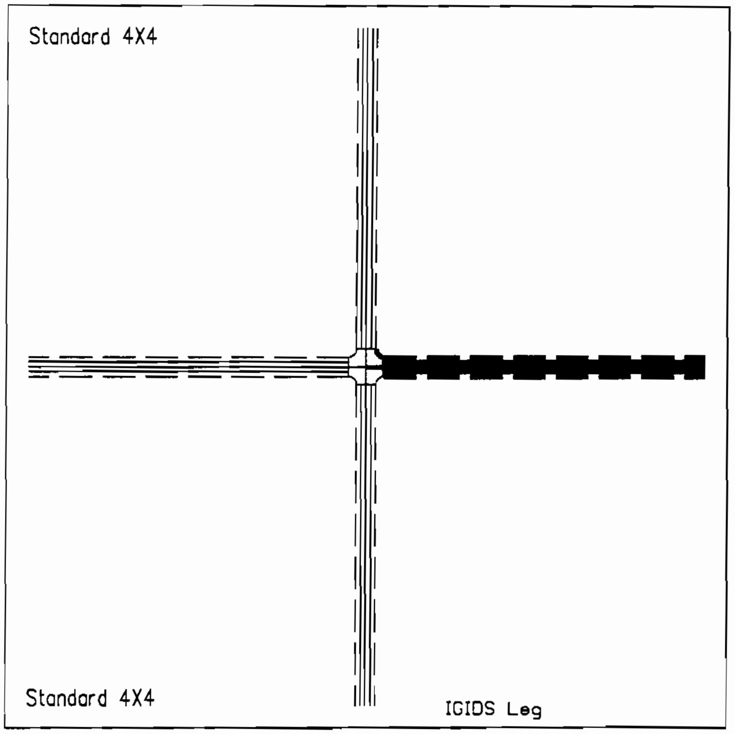


Figure 2.4 IGIDS Leg

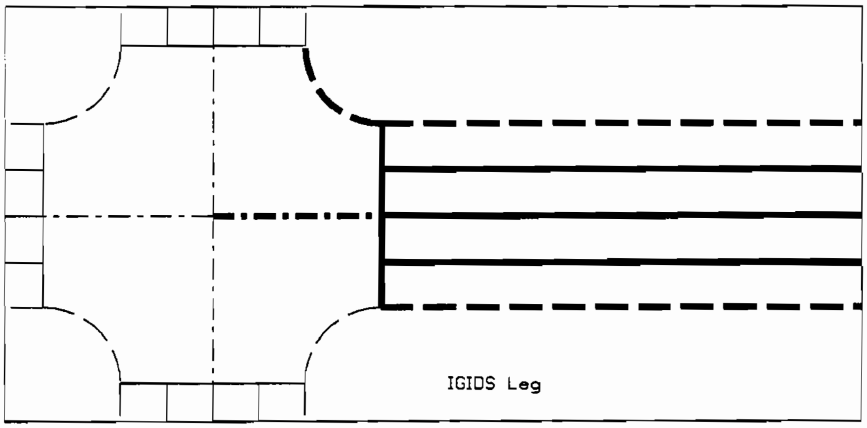


Figure 2.5 IGIDS Leg (enlarged)

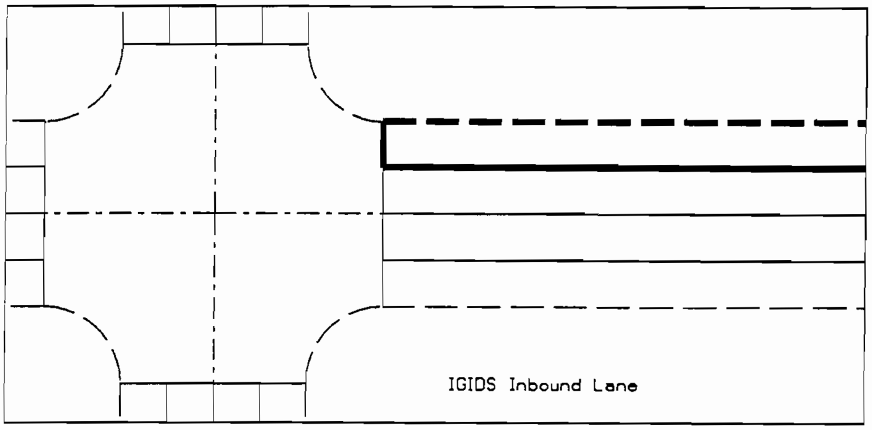


Figure 2.6 IGIDS Inbound Lane

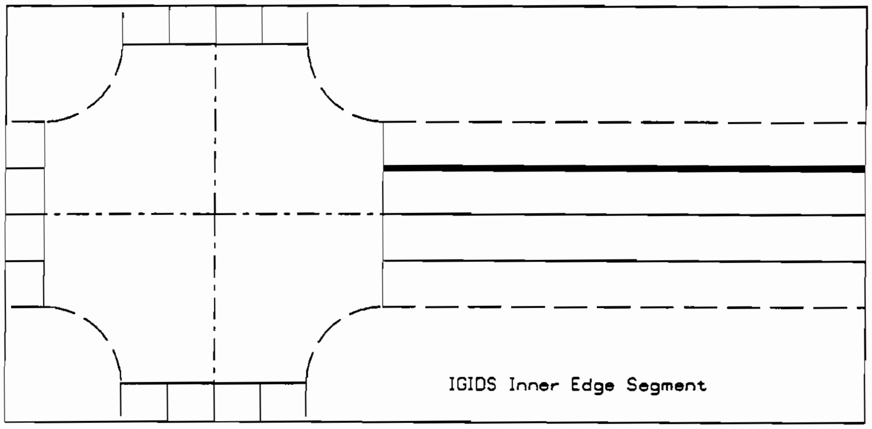


Figure 2.7 IGIDS Inner Edge Segment

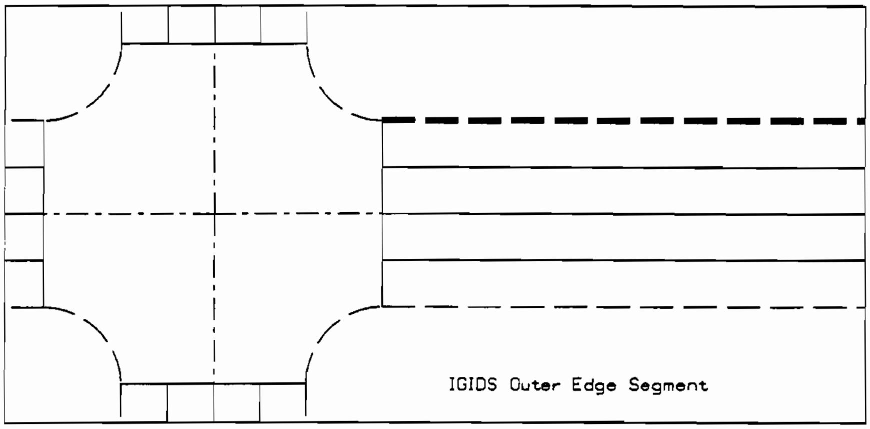


Figure 2.8 IGIDS Outer Edge Segment

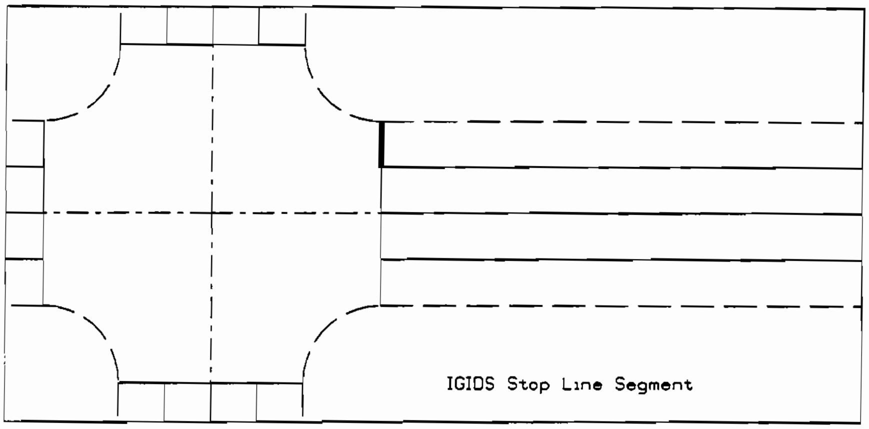


Figure 2.9 IGIDS Stop Line Segment

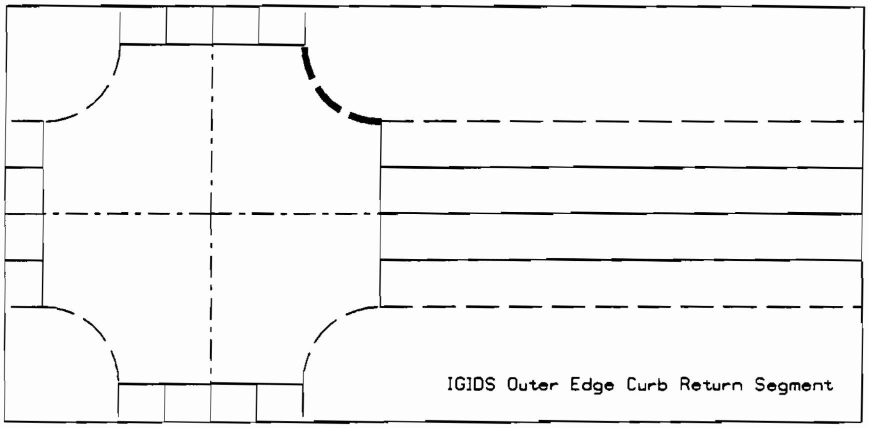


Figure 2.10 IGIDS Outer Edge Curb Return Segment

Seg, Lane Stopline Seg, Leg Centerline Seg, Curb Return Inner Edge Seg, or Curb Return Outer Edge Seg. Note also that Segs can have either Lane or Leg parents.

IGIDS automatically sorts an alternative's list of legs, a leg's list of inbound lanes, a leg's list of outbound lanes, and each list of segments. This automatic sorting by IGIDS allows the user to enter the elements of the alternative in any order. An alternative is considered to be complete when all its legs are completed. A leg is complete when its centerline, inbound lanes, outbound lanes, inner edge curb return, and outer edge curb return are all completed. A lane is complete when its inner edge, outer edge, and stopline are all completed. A list of segments is complete when (1) no segments are entered for optional elements such as curb returns, (2) one segment is entered, or (3) two or more segments are entered and there is no geometric gap between adjacent segments.

IGIDS maintains the design as descriptive data stored in the host computer's memory. This stored data is a complete record of the Intersection design. Included are the attributes of each IGIDS object and how IGIDS objects are related, data that have been calculated during the design process, and data that have been entered manually by the user. The data can be stored as a disk file and retrieved later. The IGIDS database is the master database. All graphics and attribute data items are contained in the IGIDS database, and the value stored there has precedence over any other value. The graphics engine database can be deleted and IGIDS can re-create the graphics previously entered into IGIDS. Coordinate, distance, angular, and other data in the IGIDS database are the definitive values. IGIDS uses the values in the IGIDS database for all calculations. IGIDS keeps the entire IGIDS database in memory within the IGIDS software so that no disk I/O is involved in reading a data item; this allows the software to operate as fast as possible.

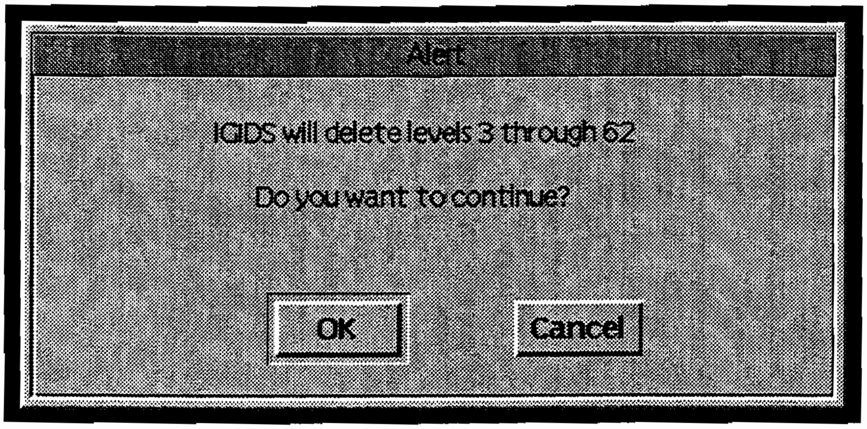
IGIDS presents a design to the user as graphics displayed by the graphics engine. The ID of each IGIDS object that is displayed (Segs and Text) is a part of the graphics engine's data and is used to link the graphics engine database with the IGIDS database. Each IGIDS graphical item in the graphics engine database contains the ID of the corresponding item in the appropriate IGIDS database where the attribute data are stored. The type of the graphics engine element (arc, line, or text) is used to determine the item type (segment or text) and, therefore, relates it to the appropriate IGIDS database item. The ID is a unique number defined by IGIDS and is the entry number, the instance number, or the row number in the appropriate IGIDS database. Given an ID, IGIDS searches the graphics-engine database or accesses the appropriate IGIDS database for the specified item. The higher-level (grouping) objects cannot have a graphical representation.

During the design process, the user can interact with IGIDS to modify the design as desired. This interaction is through the graphics engine's user interface. The user can identify existing graphical elements, specify geometric points, and key-in alphanumeric data, all in response to IGIDS prompts. All usual graphics engine functions are always available. Graphics engine and IGIDS functions can be used in any desired sequence. If there is a need to construct a feature that is beyond IGIDS's capability, the graphics engine's tools can be used to create the feature as "scratch" graphics. IGIDS can then inspect these scratch graphics and add the desired feature to the design. Graphics on an existing drawing can be processed in a similar manner.

Intersection analysis and design software packages are executed when the user selects from a menu the software package to be run. IGIDS checks the IGIDS relational database for appropriate data and prompts the user for any missing data. IGIDS then extracts data from the IGIDS database and builds the required input files for the software package that was selected. The software package is executed by the operating system as an external (or background) process. The user can then use graphics engine commands to present the output from the executed software package for review. When appropriate, the output from the software package can be displayed by IGIDS.

CAUTION

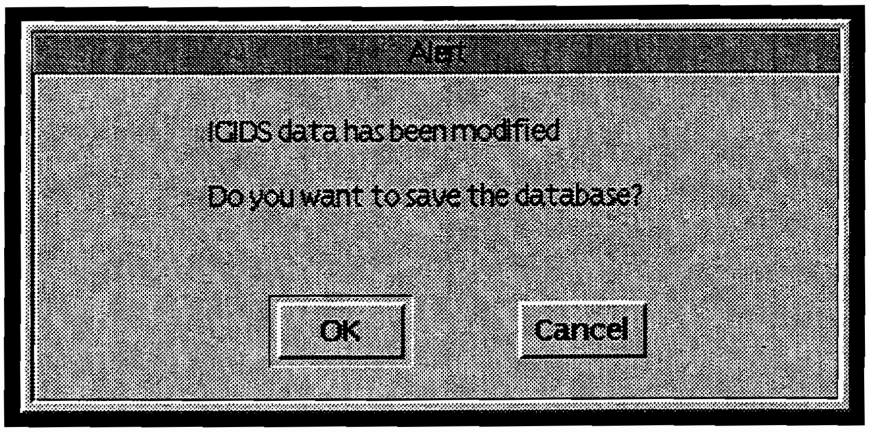
IGIDS takes control of the active graphics file and deletes everything except what is recognized as scratch-graphics (Level 2 is the scratch level, and all graphics on Levels 3 through 62 are controlled by IGIDS). IGIDS presents to the user an Alert Box (see Figure 2.11). Pressing the "OK" push button allows IGIDS to continue whereas pressing the "Cancel" push button causes IGIDS to exit without deleting any data. IGIDS recreates graphics using IGIDS data rather than relying on stored graphics files. Therefore, at start-up, any graphics stored in the active file is deleted. The IGIDS design is created by user interaction or by the importation of a previously created design that is stored in a file. All needed graphics are drawn as a part of this process. Existing non-IGIDS graphics should be accessed as a reference file. Reference files should be attached so that their elements can be snapped-to and located. Upon ending IGIDS or ending MicroStation™, IGIDS determines whether any data have been modified since the last SAVETO DATABASE command, if any, and presents to the user an Alert Box (see Figure 2.12). Pressing the "OK" push button allows IGIDS to perform a SAVETO DATABASE command whereas pressing the "Cancel" push button causes IGIDS to exit without saving any data.



IGIDS will delete levels 3 through 62

Do you want to continue?

Figure 2.11 IGIDS Beginning Alert Box



IGIDS data has been modified

Do you want to save the database?

Figure 2.12 IGIDS Ending Alert Box

OBJECTS

Each IGIDS object has a set of attributes and characteristics that define the IGIDS object. Many of these are of no practical use to the user and are not discussed here. Those that may be of interest are described.

INTERSECTION

The Intersection actually stores very little data that is of user interest. The user-specified description, the identities of each child Alternative, and the available ID numbers for each IGIDS object type fill the list. Most Intersection data is used internally by IGIDS.

ALT

Attributes of Alternatives are:

1) X and Y coordinates of the Intersection center,
2) ID number assigned by the user,
3) a list of attached Legs and Text children,
4) ID of parent Intersection,
5) ID of the selected Leg and Text,
6) user-supplied description, and
7) data needed by analysis procedures.

IGIDS automatically sorts the list of Legs of an Alternative using the centerline angle of the Leg converted to an azimuth (north is 0 and clockwise is positive), starting at zero and traversing clockwise. The Alternative itself has no displayable graphics.

LEG

Each Leg is described by:

1) angle of the centerline,
2) distance and direction from center of Intersection to start of centerline ,
3) station number at start of centerline and direction of stationing (increasing or decreasing).
4) ID number assigned by the user,
5) a list of attached Centerline Seg children,
6) lists of attached Inbound Lane and Outbound Lane children,
7) a list of attached Inner Edge Curb Return Seg children,
8) a list of attached Outer Edge Curb Return Seg children,
9) ID of the selected Lane and Centerline Seg,
10) ID of parent Alternative,
11) user-supplied description, and
12) data needed by analysis procedures.

The Leg Centerline is a connected list of Segs. IGIDS automatically sorts the list of Centerline Segs, starting at the Seg end nearest the Intersection center, and traversing away from the Intersection center. IGIDS automatically sorts the lists of Inbound and Outbound Lanes starting at the Lane nearest the median and farthest from the curb (lane 1) and traversing away from the median toward the curb (left to right in the direction of travel). The Inner Edge Curb Return and the Outer Edge Curb Return are each a connected list of Segs. The Inner Edge Curb Return Segs connect the inbound, median lane inner edge with the outbound, median lane inner edge of the same Leg. IGIDS automatically sorts the list of Inner Edge Curb Return Segs starting at the inbound, median lane inner edge and traversing to the outbound, median lane inner edge. The Outer Edge Curb Return Segs connect the inbound, curb lane outer edge with the outbound, curb lane outer edge of the adjacent, counter-clockwise Leg. IGIDS automatically sorts the list of Outer Edge Curb Return Segs starting at the inbound, curb lane outer edge and traversing to the outbound, curb lane outer edge of the adjacent, counter-clockwise Leg. The Leg itself has no displayable graphics.

LANE

Each Lane is described by:

1) the nominal width and length,
2) relative distance and direction from parent Leg's reference point, angle relative to parent Leg's absolute angle,
3) ID number, per sort as described above,
4) flag indicating that Leg is Inbound or Outbound,

5) lists of attached Inner Edge, Outer Edge and Stopline Seg children,
6) ID of parent Leg,
7) ID of the selected Seg, and
8) data needed by analysis procedures.

The Inner Edge, Outer Edge, and Stopline are connected Seg lists. IGIDS automatically sorts the Segs on each edge list using the distance from the Intersection center, starting at the Seg end nearest the Intersection center, and traversing away from the Intersection center. Stopline Segs are sorted starting with the Seg end with the minimum offset from the Leg Centerline and traversing away from the median toward the curb. The Lane itself has no displayable graphics.

SEG

A Seg is an arc of a circle or line string with these attributes:

1) for the beginning end of the first Seg in a list: relative distance and direction from parent's reference point, angle relative to parent's absolute angle, station and offset, absolute angle, and X and Y coordinates,
2) for the beginning end of all but the first Seg in a list: angle relative to previous Seg in the list, station and offset, absolute angle and X and Y coordinates,
3) for the ending end of all Segs: station and offset, absolute angle and X and Y coordinates,
4) flag indicating line or arc of a circle,
5) other geometric parameters (for line, length; for arc of a circle, X and Y coordinates of center, radius, and sweep angle),
6) flag indicating Inner Edge, Outer Edge, Stopline, Centerline, Inner Edge Curb Return, or Outer Edge Curb Return,
7) ID number, per sort of each list as described above,
8) lists of attached Text children,
9) ID of parent Leg or Lane, and
10) ID of the selected Text.

Segs may be displayed graphically.

TEXT

Text has these attributes:

1) angle, height, width, and display attributes (font, color, weight, etc.),
2) X and Y coordinates of center,
3) flag indicating parent is Seg or Alternative,
4) distance and direction from parents reference point,

5) if parent is Seg, flag indicating absolute/relative angle and if flag indicates relative, angle relative to parent's reference angle,

6) ID of parent.

Text may be displayed graphically.

CHAPTER 3 IGIDS SOFTWARE DEVELOPMENT

From a conceptual point of view, the design of IGIDS was based upon knowledge and experience of intersection analysis and design, computer software development, the TEXAS Model for Intersection Traffic, interactive graphics programming techniques, and other computer software systems.

IGIDS is designed to assist the traffic engineer in the analysis and design of single, at-grade, vehicular-traffic intersections including diamond interchanges. This involves the definition of the geometry of the intersection, the location and type of traffic control devices, and data describing the traffic flow conditions. IGIDS provides a convenient and user-friendly interface for the collection of data necessary to execute several intersection analysis and design software packages and later provides the environment to add or develop expert systems.

IGIDS uses a graphics engine to perform all interactive graphics operations and to maintain the graphics engine database. IGIDS software operates above and drives the graphics engine software through a graphics engine higher-level language interface. IGIDS takes control of the interactive graphics workstation and allows the user to easily switch between executing IGIDS commands and graphics engine commands. The graphics engine commands are used as much as possible. IGIDS provides no plotting capabilities but relies upon the graphics engine to perform these operations.

The role of IGIDS is to assist the traffic engineer in analysis and design of the intersection area available for vehicle travel as normally defined by pavement markings and stripings. This area includes the intersection, inbound, and outbound lanes. A lane may be 12 feet wide for analysis and design purposes but have additional pavement width to accommodate shoulders, curbs and gutters, and medians. IGIDS will not accommodate this additional pavement width. It is assumed that the intersection design developed by the user with IGIDS will become the starting point or input to a roadway design systems such as the IGrds or a CAD software package that will assist the roadway design engineer in designing the complete roadway.

GRAPHICS ENGINE

Several good graphics engines (CAD software packages) which operate on popular workstations are available from different manufacturers. There is a wide variety of good workstations for the user to choose from. Each workstation has its own screen size, screen resolution, pointing device (both the absolute positioning cursor and the relative movement mouse), windowing system, graphics device driver, optional input devices, operating system, and CPU. Each provides an interface with the hardware and numerous commands to create, display, modify, manipulate, delete, save, and plot the graphical data. CAD software packages generally maintain an external graphics engine database containing the graphics which may be saved from session to session. CAD software packages also provide functions for reading and writing various standard graphics engine database exchange formats. In addition, these CAD software packages are being constantly enhanced and corrected by a permanent staff of programmers in response to user requests. The IGIDS implementation decision was either:

(1) choose a small, limited set of workstations and develop extensive graphics software accommodating these workstations or

(2) choose two or more graphics engines and allow the graphics engines to perform all interactive graphics operations and to maintain the graphics engine database.

IGIDS is designed to use a graphics engine and to develop interfaces to multiple graphics engines. It is the responsibility of the user to choose and purchase/provide the graphics engine from the list of graphics engines interfaced to IGIDS.

Several criteria were established for choosing a graphics engine for interfacing with IGIDS. The graphics engine has to:

(1) offer a higher-level language, real-time interface so that the IGIDS software can operate above and drive the graphics engine software,

(2) allow IGIDS to take control of the interactive graphics workstation,

(3) allow the user to easily switch between executing IGIDS commands and graphics engine commands,

(4) provide plotting capabilities to numerous plotter devices,

(5) be used by many state DOTs and by many design professionals,

(6) operate on several moderately priced workstations from different manufacturers, and

(7) have a graphics engine reference database capability with numerous methods of creating graphics engine reference database data.

IGIDS makes calls to generic graphics functions developed specifically for IGIDS. A library of generic graphics functions which perform the requested graphics operation are developed for each graphics engine. The generic operations are: add a line, delete an arc, display a view or plane, etc. The graphics engines chosen are Intergraph's MicroStation™ and Autodesk's Autocad. Initial development was on Intergraph's MicroStation™. Later, IGIDS was converted to the MicroStation™ Development Language (MDL).

OPERATING SYSTEM

Many operating systems alternatives are currently available. Most large computer companies offer proprietary operating systems developed by them to operate only on their computers. They often offer operating systems that adhere to international standards, government standards, or standards developed by a consortium of many competing computer companies. The operating systems may be designed for batch operation, on-line transaction processing, real-time process control, and/or interactive operation. The operating systems may support a single user or multiple users. They may allow only one process to be memory resident and active at one time (example: MS-DOS), allow many processes to be memory resident and allow the user to designate which process is active (example: Apple's Macintosh MultiFinder), or allow many processes to be memory resident and allow the operating system to designate which process is active through a scheduling algorithm so that the processes share the CPU through multi-tasking (example: DEC's VMS and AT&T's Unix). The operating system may support virtual memory addressing or direct memory addressing.

The criteria for choosing an operating system for use by IGIDS included the fact that it would have to:

(1) be designed for batch operation and interactive operation,
(2) support multiple users,
(3) allow many processes to be memory resident and allow the operating system to designate which process is active through a scheduling algorithm so that the processes share the CPU through multi-tasking,
(4) support virtual memory addressing with large real memory,
(5) support all graphics engines interfaced to IGIDS,
(6) support higher-level language interfaces used by the graphics and database engines interfaced to IGIDS,
(7) support computer software languages used by analysis programs interfaced to IGIDS,
(8) support the computer software language chosen for IGIDS,
(9) be used by many state DOTs and by many design professionals, and
(10) operate on several moderately priced workstations from different manufacturers.

The operating system initially chosen was AT&T's Unix or Unix clone and OSF's OS1 for the future. With the development of the MicroStation™ Development Language, the choice of operating system became a non-issue. MDL allows large applications to be developed even in the MS-DOS environment and isolates the operating system from the application.

COMPUTER SOFTWARE LANGUAGE FOR IGIDS

A computer software language may be

(1) an assembly language where one statement in the computer software language is translated into one computer machine instruction, takes the most computer software language statements to accomplish a given task, is designed to operate on one computer's machine instruction set, and is generally defined by the computer company manufacturing the computer;
(2) a macro language where one statement in the computer software language is translated into one or more computer machine instructions, takes the second most computer software language statements to accomplish a given task, is designed to operate on

one computer's machine instruction set, and is generally defined by the computer company manufacturing the computer; or

(3) a higher-level language where one statement in the computer software language is translated into many computer machine instructions, takes the least computer software language statements to accomplish a given task, is designed to operate on many computer machine instruction sets, and is generally defined by an international standard.

IGIDS uses a higher-level computer software language.

Some higher-level computer software languages are ADA, ALGOL, Basic, C, COBOL, Fortran, LISP, Pascal, or Snobol. Each higher-level language was generally designed to make the solution of a particular class of problem easier for the programmer: COBOL was designed for business applications, Fortran was designed for engineering and scientific applications, and LISP was designed for artificial intelligence applications. Some higher-level languages require declaration of all variables at compile time and have a fixed maximum problem size. Others allow allocation and use of dynamic memory at execution time through pointers. The maximum problem size is limited only by virtual address space available from the operating system.

Several criteria were established for initially choosing a higher-level computer software language for use by IGIDS. The higher-level language has to:

(1) be supported by all graphics engines interfaced to IGIDS,

(2) be supported by the operating system,

(3) be applicable to engineering and geometric calculations,

(4) be a state-of-the-art computer software language,

(5) be defined by an international standard,

(6) allow the allocation and use of dynamic memory at execution time through the use of pointers,

(7) support 16 significant digits, 64 bit, double precision floating point arithmetic,

(8) support 10 significant digits, 32 bit, integer arithmetic,

(9) support 5 significant digits, 16 bit, integer arithmetic,

(10) allow the inclusion of global variable declarations and definitions from an external file,

(11) allow upper and lower case characters in input and output records, and

(12) be used by many state DOTs and by many computer software development professionals.

The higher-level computer software language initially chosen was C. Later, IGIDS was converted to the MicroStation™ Development Language (MDL) developed by Bentley Systems, Inc., which is C plus one extension. MDL has some 3,000 documented subroutines to create, modify, and delete graphical elements and dialog boxes.

COMPUTER SOFTWARE CODING STANDARDS

Computer software coding standards are strongly recommended as they result in computer code that is consistent, easily understood by another programmer, and easier to maintain. Once the coding standard becomes familiar, the programmer develops an expectation of the manner in which the computer code should look and non-conforming items are easily detected.

The standards adopted for IGIDS specify that all items should be listed in alphabetical order. This standard applies to "#include" statements, in-line macro definitions, "#define" statements, function prototypes, "typedef" statements, variable declarations and definitions, structure declarations and definitions, union declarations and definitions, and most other list items. The exception to this standard is to list items in hierarchical order where appropriate. This standard has the advantages that items can be found more easily and logically.

The standard also specifies that each logical grouping of functions should be contained in its own source file and that the source file name should be an expression of the logical grouping. This standard has the advantages that the size of each source file is minimized making backup and editing easier, a programmer can modify a single function without disturbing other functions, the function can be individually compiled, there is a corresponding object file for each source file, the programmer can easily test a single function, and searching for a string within all functions will display the source file name (function name) for each string match indicating the usage of the string. This standard has the disadvantages that function names are limited in size by the largest unique file name size (currently 14 characters on AT&T's Unix) and the programmer must potentially edit multiple source files to change a variable used in multiple functions.

All IGIDS core software global declarations and definitions are contained in a single include source file. A declaration refers to places where the nature of the variable is stated but no storage is allocated while a definition refers to the place where the variable is created or assigned storage. This standard has the advantage that the programmer must maintain only one include source file. It eliminates errors caused by differences in declarations, searching for a string within a single function or all functions will display only where the string is referenced or used as opposed to where the string is also declared or defined. Making a change in the single include source file will cause all functions which use the include file to be compiled when the make command is issued. All graphics engine and database software global declarations and definitions are contained in a single include source file.

All global declarations, definitions, and variables are contained in the include source files. The include source file may contain: "#include" statements for other include source files needed by the functions, in-line macro definitions, "#define" statements, function prototypes, "typedef" statements, variable declarations and definitions, structure declarations and definitions, and union declarations and definitions. Once included in a compile, an include source file is not included again even though it is requested additional times.

All compiled IGIDS functions are contained in a single object library. This standard has the advantages that a single function can be easily modified and tested without changing the function in the archive library, and the linked image contains only the functions called. This standard has the disadvantages that the archive command must be used to replace object files in the object library.

Each function name begins with "igids_" and contains a maximum of 18 upper and lower case characters (6 characters for "igids_" and 12 characters for the rest of the function name). Finally, there are 2 consecutive blank lines at the end of the source file and there is no other occurrence of 2 consecutive blank lines in the source file. This standard has the advantages that the function name can be somewhat descriptive of the operations performed, there should be minimal conflicts with function names from other software, a programmer can readily distinguish the IGIDS functions from other functions, the source files may be concatenated into one large source file and later broken up into individual source files by a program, and the end of each function is standard and can be easily located while editing. Example function names are shown in Table 3.1.

Each variable name begins with a 5 character prefix, followed by the name, optionally ends with a 4 character suffix, and contains a maximum of 31 upper and lower case characters. The 5 character prefix is a single character indicating the scope of the variable, followed by 2 characters indicating the type of the variable, followed by a single character indicating the dimension, and terminated with an "_" character. Each variable that is a pointer has the mandatory 4 character suffix "_ptr" for pointer to a variable or "_pfn" for pointer to a function. The single character scope is: "g" for global static, "l" for local dynamic, "p" for parameter dynamic (parameters to the function), "s" for local static, "s" for structure member, "u" for union member, or "z" for zone. The 2 character type is: "ch" for a single character, "cz" for character string zero (null) byte terminated, "df" for double floating point, "fs" for file stream, "si" for signed int, "sl" for signed long, "ss" for signed short, "td" for typedef variables, "uc" for unsigned char, "ui" for unsigned int, "ul" for unsigned long, "us" for unsigned short, or "vo" for void. The single character dimension is "a" for array or "v" for single variable. This standard has several advantages. The variable name can be descriptive of the data storage allowing a programmer to readily distinguish the scope, type, and dimension of the variable and whether the variable is a pointer. Parameters to a function are treated in a special manner, and errors in formal parameter types not matching actual parameter types is reduced. Example variable names are presented in Table 3.2.

Table 3.1 Example Function Names

igids_addLegClKey

igids_dbload

igids_delSegCntrln

igids_drwarc

igids_errmsg

igids_lansrt

igids_loadAltLib

igids_moveLegLatrl

igids_openInterNew

igids_segswp

igids_selLanOutNxt

igids_turnTemplate

igids_tx_mdl

igids_vnumdf

Table 3.2 Example Variable Names

gslv_dest_seg_id_num

gstv_inter_entry_ptr->sslv_num_alters

gstv_leg_ent_ptr->ssla_num_lanes[LANE_INB]

gstv_lane_ent_ptr->sdfv_width

gstv_seg_ent_ptr->sslv_iosc_flag

gstv_text_ent_ptr->sslv_font

ldfv_initial_angle

lfsv_file_ptr

lslv_return_code

lstv_previous_leg_ptr

pczv_error_message_ptr

pucv_bit_mask_ptr

scza_file_name[FILE_NAME_NC+1]

sslv_resetfunc_pfn

suna_element[2]

ztdv_statedata

Each function that can directly detect an error or call a function that can return an error must return a signed long error code indicating the success or failure of the function. All other functions can return nothing (void) or can return a value. A function that properly completed returns an error code of "RETURN_SUCCESS", while a function that detected an error returns an error code of "RETURN_NON_FATAL_ERROR" for non-fatal (user recoverable) errors (warning messages are issued) or return an error code of "RETURN_FATAL_ERROR" for fatal (programming) errors (error messages are issued), while a function that calls a function that returns an error deals with the error or returns the error code. It is the responsibility of the function detecting the error to issue an appropriate error message and the detecting function name. It is the responsibility of a function that calls a function that returns an error and decides to return the error to issue the calling function name (i.e. provide a trace back for the error).

Virtually all constants are coded using the symbolic name capability within C. The syntax of the symbolic name statement is "#define", followed by the name, and followed by the replacement string. Subsequent occurrences of the name in the source code (not in quotes and not part of another name) are substituted with the replacement string before compilation of the source code is initiated. Additionally, names used in the "#define" statement are upper case characters only, all constants used in more than one function for the same purpose are defined in the global include source file, and local "#define" statement names are prefixed with the name of the function in upper-case characters plus an "_" character. Uses of this feature include dimensions for arrays, index values for arrays, switch case statement values, function return values, in-line macro definitions, and state or stage values. This standard has the advantages that errors caused by different values being used for the same purpose are greatly reduced or eliminated, the meaning of constants is more readily conveyed, and the source code becomes more self-documenting. Additionally, the symbolic name capability can also be used for in-line macros. The syntax of the symbolic macro statement is "#define"; followed by the macro name, "(", the macro parameter(s), and ")"; and followed by the replacement string. IGIDS allows both upper and lower case characters in the name for the symbolic macro statement. Example "#define" statements are in Table 3.3.

Table 3.3 Example "#define" Statements

#define	ALTER_DESC_NC	(size_t)	80
#define	ALTER_PROC_ALT	(long)	1
#define	ALTER_PROC_LEG	(long)	2
#define	ALTER_PROC_TXT	(long)	4
#define	ALTER_PROC_ALL	(long)	7
#define	CONCENTRIC_RADIUS_MINIMUM		50.0
#define	FILE_NAME_NC	(size_t)	127
#define	ID_NULL	(long)	-1
#define	LANE_INB	(long)	0
#define	LANE_OUT	(long)	1
#define	LANE_LENGTH_MINIMUM		10.0
#define	LANE_WIDTH_MINIMUM		8.0
#define	LANE_WIDTH_MAXIMUM		16.0
#define	LOCAL_FILE_NAME_NC	(size_t)	127
#define	RETURN_SUCCESS	(long)	0
#define	RETURN_NON_FATAL_ERROR	(long)	1
#define	RETURN_FATAL_ERROR	(long)	2
#define	max(a,b)	(((a)>(b))?(a):(b))	

The following statement order and syntax was chosen for all functions:

(1) "#include" statement(s) starting in column 1 and followed by 1 blank line

(2) "type function_name (function_parameter(s))" statement starting in column 1

(3) "type function_parameter; /* comment */" statement(s) starting in column 1

(4) "{" in column 1

(5) comment statement(s) starting in column 3 describing the purpose of the function and followed by 1 blank line

(6) comment statement(s) starting in column 3 describing the global input requirements and followed by 1 blank line, if required

(7) comment statement(s) starting in column 3 listing the function(s) which call the function and followed by 1 blank line

(8) comment statement(s) starting in column 3 listing the function(s) called by the function and followed by 1 blank line, if required

(9) local "#define" statement(s) starting in column 3 and followed by 1 blank line, if required

(10) local structure(s), union(s), and variable(s) in alphabetical order starting in column 3 and followed by 1 blank line, if required

(11) function code block(s) starting in column 3; a function code block includes (a) comment statement(s) starting in column 3 describing the purpose of the following code, (b) computer code starting in column 3, and (c) 1 blank line

(12) "return;" or "return (RETURN_SUCCESS);" statement starting in column 3 if not conditionally returned in the preceding code

(13) 1 blank line, "return_fatal_error:" starting in column 3, 1 blank line, "igids_detmsg (" function_name ");" starting in column 3, and "return (RETURN_FATAL_ERROR);" starting in column 3, if required

(14) 1 blank line, "return_non_fatal_error:" starting in column 3, 1 blank line, "igids_detmsg (" function_name ");" starting in column 3, and "return (RETURN_NON_FATAL_ERROR);" starting in column 3, if required

(15) 1 blank line, "return_returned_error:" starting in column 3, 1 blank line, "igids_trcmsg (" function_name ");" starting in column 3, and "return (lslv_return_code);" starting in column 3, if required

(16) "}" in column 1

(17) 2 blank lines

Within the function code, 2 spaces are used for indention. No tab characters are in the code. The standard maximum line length is 80 columns for functions and include source files. The comments for lines within include source files may be extended to 132 columns. The opening "{" and closing "}" for the "do", "for", "if", "switch", and "while" statements start in the same column directly below the first character of the "do", "for", "if", "switch", or "while" statement and all code between the opening "{" and the closing "}" is indented 2 spaces. The opening "{" and the closing "}" is used even when there is only one statement between them except an "if" statement without an "else" that does not exceed the standard maximum line length. The "else" for the "if" statement starts in the same column directly below the first character of the "if". An example of a nested "if" and "else" statement is presented in Table 3.4.

The "switch" and "case" statement is preferred over the "if" and "else" statement when there are more than 2 choices. The "case" and "default" for the "switch" statement is indented 2 columns below the first character of the "switch" statement. The code and any "break;" statement(s) are indented 2 columns below the first character of the "case" or "default" statement. An example of a "switch" and "case" is shown in Table 3.5.

Table 3.4 Example of a Nested "if" and "else" Statement

```
if ( aaa == bbb )
{
    if ( ccc == ddd )
    {
      www = xxx;
    }
    else
    {
      www = yyy;
    }
}
```

Table 3.5 Example of a "switch" and "case" Statement

```
switch ( pslv_function )
{
    case FUNCT_ADD_DBE:
        lslv_return_code = igids_altadddbe ();
        if ( lslv_return_code != RETURN_SUCCESS )  goto return_returned_error;
        break;

    case FUNCT_CPY_DBE:
        lslv_return_code = igids_altcpydbe ();
        if ( lslv_return_code != RETURN_SUCCESS )  goto return_returned_error;
        break;

    default:
        sprintf ( gcza_err_msg,"Undefined function = %ld",pslv_function );
        igids_error ( gcza_err_msg );
        goto return_fatal_error;
}
```

For statements within a block of code, the "="s, ";"s, and any comments at the end of a line are aligned if reasonable. There is no blank space before the ";" statement terminator unless needed for alignment. For a statement continued onto another line, the continued line(s) are indented a minimum of 2 columns and preferably lined up with the first parameter following a "(", "=", or "+". Table 3.6 presents examples of these computer software coding standards.

Table 3.6 Examples of Statement Alignment

```
lslv_seg_id_num     = gslv_seg_id_num ;     /* save global id num  */
lslv_id_num_blink   = lslv_seg_id_num  ;    /* set .local blink    */
sprintf (  gcza_err_msg,"Undefined function = %ld",
           pslv_function                         );
ldfv_dx =  gstv_seg_ptr[lslv_seg_id_num].sdfv_beg_x   -
           ldfa_ref_end_x[lsiv_end]                   ;
```

There are no blank space(s) at the end of any line. For comma separated lists of arguments, there are no blank spaces before or after the comma unless needed for alignment purposes whereas for comma separated statements, there is 1 blank space before and after the comma. For the separating ";" within the "for" statement, there is 1 blank space before and after the ";". Finally, there is 1 blank space before and after an opening "(" and the closing ")" while there are no blank space(s) before or after the opening "[" or the closing "]" unless needed for alignment. Function references which do not have parameters have no space between the "(" and the ")" as in "()". Table 3.7 shows examples of these computer software coding standards.

Table 3.7 Examples of Statement Spacing

```
lslv_return_code = igids_intchk ( 0l,gslv_dim_inters-1 );
for ( lsiv_i=0 , lsiv_j=0 ; lsiv_i<lsiv_n ; lsiv_i++ , lsiv_j-- )
lsla_ref_id_num[LOCAL_SEG_BEG] = gslv_seg_id_num;
lslv_return_code = igids_segsta ();
```

IGIDS keeps the entire IGIDS database in memory within the IGIDS software so that no disk I/O is involved in reading a data item thus making the software operate as fast as possible. Each structure contains an attribute which indicates whether the entry has been modified since the last time that the internal copy of the IGIDS database was written to the external copy.

Many computer software languages, including C, support the concept of a structure. A structure is a collection of one or more variables, possibly of different types, grouped together under a single name for convenient handling. Structures are called "records" in some computer software languages. Structures help to organize complicated data because they permit a group of related variables to be treated as a unit instead of as separate entities. The variables named in a structure are called members. The liberal use of structures is strongly supported in the IGIDS development.

Some higher-level computer software languages require the declaration of all variables at compile time and have a fixed maximum problem size whereas others, including C and MDL, allow the allocation and use of dynamic memory at execution time through the use of pointers and the maximum problem size is limited only by the virtual address space available from the operating system. This execution-time allocation and use of memory was established as one of the criteria for choosing a higher-level computer software language for IGIDS. The liberal use of execution-time allocation and use of memory is strongly supported to generate what might be called dimensionless programming.

At compile time, IGIDS declares a pointer to a structure, a variable for the current allocation of instances of the structure, and a variable for the current use of instances of the structure. Since all references to the structure are through the pointer, IGIDS can change the value of the pointer at execution time to reference different areas of memory. During the initialization phase at execution time, IGIDS requests the operating system to allocate memory for an initial number of structure instances, say 20, set the pointer to the address of this initial memory, set the current allocation to 20, and set the current use to 0. During normal execution, IGIDS uses these instances until there is a need for the 21st instance. IGIDS then requests the operating system to allocate memory for a new number of structure instances, say 30, copy the data from the 20 instances into the first 20 instances of the 30 instances, set the pointer to the address of this new memory, return the memory occupied by the original 20 instances to the operating system for further allocation, set the current allocation to 30, and allow the program to continue. This process is repeated until the virtual address space available to IGIDS is exhausted. The amount of virtual address space available to a process can normally be set within the operating system and the maximum is generally 1 to 4 billion bytes. The advantage of this technique is having virtually no limits within the program. The disadvantages of this technique are the increased execution time because of pointer addressing as opposed to direct addressing, the increased execution time because of the need to check the allocation each time another instance is needed, the increased execution time because of allocation and reallocation of dynamic memory, the need to initialize the allocated memory at execution time, and the increased risk of program failure by referencing memory outside the allocation range of the structures. The single advantage outweighs the disadvantages for IGIDS.

To overcome the increased risk of program failure by referencing memory outside the allocation range of the structures, a function (named "igids_xxxchk" where "xxx" is the 3 character structure name and "chk" stands for "check") has been developed for each structure. It checks to ensure that an index number is in the range from zero through the current number of structure instances allocated minus one and in the range from a minimum value (normally zero) through a maximum value (normally the current number of structure instances in use minus one). Additionally, when an instance of a structure is no longer needed and can normally be deleted, that instance is added to a linked list of deleted instances for the structure (last added is first used and first added is last used) and the instance is marked as deleted. When a new instance of a structure is needed, first the deleted instances are used, then the allocated but unused instances are used, and finally additional instances are allocated from the operating system.

If possible, IGIDS is limited only by the graphical engines. The graphical engines have a finite number of graphical levels or planes for separating graphics. The number of alternatives is limited by the minimum number of graphical levels or planes available on all graphics engines divided by the number of graphical levels or planes allocated per alternative. Currently, MicroStation™ has a limit of 63 graphical levels or planes and IGIDS has allocated 4 graphical levels or planes per alternative, therefore IGIDS is currently limited to 15 alternatives.

COMPUTER HARDWARE

Many workstations are available on today's market and the choice of a particular vendor's workstation is one of the least important considerations in influencing the IGIDS system design. The system design of IGIDS attempts to avoid anything that might exclude any vendor's workstation.

Several criteria were established for assisting the user in choosing a workstation for IGIDS. The workstation should:

(1) support the user's choice for a graphics engine,

(2) support the operating system,

(3) have a minimum performance of an Intel 486DX based computer system,

(4) have a minimum of 1 screen with a preference for 2 screens,

(5) have a minimum screen resolution of 72 dots-per-inch with a preference for 100 dots-per-inch,

(6) have a minimum screen size of 15 inches diagonal with a preference for 19 inches diagonal,

(7) have a minimum of 32 colors for the largest screen with a preference for 256 colors for the largest screen,

(8) have a pointing device (either the absolute positioning cursor or the relative movement mouse),

(9) have a minimum of 6 megabytes of physical memory with a preference for 8 megabytes of physical memory (the user should acquire as much physical memory as can be afforded and justified to keep the graphics engine database and the IGIDS database memory resident),

(10) have a minimum of 40 megabytes of hard disk with a preference for 300+ megabytes of hard disk (this will depend upon the size and quantity of the user's files),

(11) have a 1.44 megabyte floppy disk drive,

(12) optionally have networking hardware and software,

(13) be moderately priced, and

(14) be used by many state DOTs and by many design professionals.

ANALYSIS PROGRAMS

Engineers sometimes use computer software packages in the analysis and design of single, at-grade, vehicular-traffic intersections including diamond interchanges. Intersection geometry, location and type of traffic control devices, and traffic flow conditions must be defined in order to use these computer aids. These software packages are executed as external programs by the user selecting the software package. IGIDS checks the IGIDS database for the appropriate data and prompts the user for any missing data. IGIDS extracts the appropriate data from the IGIDS database and builds the input for the software package. IGIDS causes the software package to be executed by the operating system as an external or background process, and IGIDS allows graphics engine commands to be used by the user to review the output from the software package. When appropriate, the output from the software package may be displayed by IGIDS. It is the responsibility of the user to purchase/provide the external program software package.

The intersection analysis and design software packages chosen as external programs should:

(1) be supported by the operating system,

(2) accept all necessary input from input file(s) without user interaction,

(3) generate results into output file(s),

(4) be moderately priced,

(5) be readily available, and

(6) be used by many state DOTs and by many design professionals.

The external programs chosen for IGIDS are the Texas Department of Transportation Automated Plan Preparation System, Texas Model for Intersection Traffic, and the Signal Operations Analysis Program.

The intersection analysis and design software packages chosen as internal programs should:

(1) benefit from a closer relationship with IGIDS,

(2) be available in source code without copyright infringement,

(3) be supported by the operating system,

(4) be supported by the computer software language, and

(5) be used by many state DOTs and by many design professionals.

The internal programs chosen for IGIDS are the Texas Truck Off-tracking Model and the Highway Capacity Manual Chapter 9 procedures.

IGIDS MAIN STRUCTURES

IGIDS uses hierarchical, relational geometry and keeps the entire IGIDS database in memory. Additionally, a criteria established for the operating system is that it support virtual memory addressing with large real memory. A criteria established for the computer software language for IGIDS is that it allow the allocation and use of dynamic memory at execution time through the use of pointers. Finally, computer software coding standards require that each IGIDS database table contain an attribute which indicates whether the entry has been modified since the last time the internal copy of the IGIDS database was written to the external copy and that the liberal use of structures is strongly supported.

The primary structures are *intersection* (inter or int), *alternative* (alter or alt), *leg*, *lane* (lan), *segment* (Seg), and *text* (Txt). The ID is the number that is used to index the particular entry of the array of structures of the same type. The IDs start at 0 and are positive integers. The "#define" constant ID_NULL stands for an invalid ID and has a value of -1. The hierarchical relationships developed for these structures or IGIDS database tables are shown in Table 3.8.

The intersection, alternative, leg, lane ,segment, and text structures each contain a flag to indicate whether the instance has been modified since the last time the internal copy of the IGIDS database was written to the external copy. Each list of instances has the ID of the beginning instance or ID_NULL if there are no instances on the list, the ID of the ending instance or ID_NULL if there are no instances on

Table 3.8 Hierarchical Relationships of IGIDS Objects

- **Intersection:**
  - list of alternative IDs
  - other intersection attributes
  - **Alternative:**
    - parent intersection ID
    - list of leg IDs
    - list of text IDs
    - other alternative attributes
    - **Leg:**
      - parent alternative ID
      - list of centerline segment IDs
      - list of inner edge curb return segment IDs
      - list of outer edge curb return segment IDs
      - list of inbound lane IDs
      - list of outbound lane IDs
      - other leg attributes
      - **Lane:**
        - parent leg ID
        - list of inner edge segment IDs
        - list of outer edge segment IDs
        - list of stop line segment IDs
        - other lane attributes
        - **Segment:**
          - parent leg/lane ID
          - list of text IDs
          - other segment attributes
          - **Text:**
            - parent alternative/segment ID
            - other text attributes

the list, and the number of instances on the list. This doubly linked list allows IGIDS to traverse a list from beginning to ending or from ending to beginning, allows a list to be traversed from any instance forward to the ending or backward to the beginning, and allows for easy insertion/deletion of an instance to/from a list. For each structure, a global variable is defined for a single temporary instance or entry of a structure and a single input instance or entry of a structure. Additionally, for each structure, a global pointer is defined for the array of structures (this pointer is only modified by the allocation and reallocation functions), for a single instance or entry of a structure (usually the instance or entry to work with), for a single temporary instance or entry of a structure (initialized to point to the single temporary instance or entry of a structure), and for a single input instance or entry of a structure (initialized to point to the single input instance or entry of a structure. The input instance or entry of a structure is used by IGIDS to store data until all necessary data is defined by the user and then the input instance or entry is added to the main group. Finally, a "#define" constant is defined for the size in bytes of a single instance or entry of a structure.

The alternative, leg, lane, segment, and text structures each contain a flag to indicate whether the instance or entry is in-use or deleted, the ID of the backward link (blink) (the previous entry on the same list) or ID_NULL if the instance or entry is first on the list, the ID of the forward link (flink) (the next entry on the same list) or ID_NULL if the instance or entry is last on the list, and the ID of the parent.

There is only one instance of the intersection structure and its ID is 0. In addition to the attributes listed above, the intersection structure contains many one-of-a-kind attributes such as:

(1) the maximum gap between segments,
(2) the selected alternative ID,
(3) the global next ID (available for use) for alternative, leg, lane, segment, and text,
(4) the global maximum ID (memory allocated but unused) for alternative, leg, lane, segment, and text,
(5) the global number of instances (currently in use) for alternative, leg, lane, segment, and text,
(6) the graphics level or plane for the user and scratch,
(7) the beginning level or plane for alternatives,
(8) the number of levels or planes for alternatives,
(9) the relative level or plane for centerlines,
(10) the relative level or plane for lanes,
(11) the relative level or plane for traffic control,
(12) the relative level or plane for texts, and
(13) the intersection description provided by the use,

The intersection structure itself has no displayable graphics.

The number of alternatives is limited by the minimum number of graphical levels or planes available on all graphics engines divided by the number of graphical levels or planes allocated per alternative. Currently, MicroStation™ has a limit of 63 graphical levels or planes and IGIDS has allocated 4 graphical levels or planes per alternative, therefore IGIDS is currently limited to 15 alternatives. Therefore, there are zero to 15 instances of the alternative structure. In addition to the attributes listed above, the alternative structure contains many attributes such as:

(1) the center X and y coordinate,
(2) the alternative number (1->2 billion) provided by the user,
(3) the beginning level or plane for the alternative,
(4) the selected leg ID,
(5) the selected lane ID,
(6) the selected segment ID,
(7) the selected text ID,
(8) the alternative description provided by the user, and
(9) the alternative application data.

The alternative structure itself has no displayable graphics. IGIDS automatically sorts the list of Legs of an alternative using the centerline absolute angle of the Leg converted to an azimuth (north is zero degrees and clockwise is positive), starting at north, and traversing clockwise.

There are zero to any number of instances of theLeg structure. In addition to the attributes listed above, the Leg structure contains many attributes such as:

(1) the centerline absolute angle,
(2) the distance and offset from the intersection center,
(3) the station number at intersection center,
(4) the direction for station numbers (increasing or decreasing),
(5) the tie point x and y coordinate,
(6) the leg number (1->2 billion) provided by the user,
(7) a flag indicating whether the centerline segments are completed,
(8) the selected centerline segment ID,
(9) the selected lane ID,
(10) the Leg description provided by the user, and
(11) the Leg application data.

The Leg structure itself has no displayable graphics. The centerline is a connected list of segments. IGIDS automatically sorts the list of centerline segments of a Leg using the distance from the intersection center, starting at the segment end nearest the intersection center, and traversing away from the intersection center. IGIDS automatically sorts the list of inbound lanes and the list of outbound lanes of a Leg using the beginning offset from the centerline of the leg of the first segment of the inner edge of the lane, starting at the lane nearest the median and farthest from the curb (lane 1), and traversing away from the median toward the curb (lane N) (left to right in the direction of travel). IGIDS also assigns the lane number (1->N) to each lane after sorting the list of inbound lanes and the list of outbound lanes.

There are zero to any number of instances of the lane structure. In addition to the attributes listed above, the lane structure contains many attributes such as:

(1) the lane width at the intersection,

(2) the lane length,

(3) the centerline station for the intersection end of the lane, the beginning of the blockage, the ending of the blockage, and the end of the lane,

(7) the relative distance and direction from the tie point,

(8) the tie point x and y coordinate and absolute angle,

(9) the lane number (1->N) calculated by IGIDS from median lane (lane 1) to the curb lane (lane N),

(10) a flag indicating whether the lane is inbound or outbound,

(11) 3 flags indicating whether the inner edge, outer edge, and stop line edge are completed,

(12) the selected segment ID, and

(13) the lane application data.

The lane structure itself has no displayable graphics. The inner edge, outer edge, and stop line are connected lists of segments. IGIDS automatically sorts the list of inner edge segments and the list of outer edge segments of a lane using the distance from the intersection center, starting at the segment end nearest the intersection center, and traversing away from the intersection center. IGIDS automatically sorts the list of stop line segments of a lane using the distance from the stop line segment end with the minimum offset from the centerline of the leg (the stop line segment end nearest the median and farthest from the curb), starting at the stop line segment end with the minimum offset from the centerline of the Leg (the stop line segment end nearest the median and farthest from the curb), and traversing away from the median toward the curb (the stop line segment end farthest from the median and nearest the curb) (left to right in the direction of travel).

There are zero to any number of instances of the segment structure. A segment is an arc or a line. In addition to the attributes listed above, the segment structure contains many attributes such as:

(1) the relative distance and direction from the tie point,

(2) for the beginning end of the segment, the relative angle, the centerline station and offset, the absolute angle, and the x and y coordinate,

(3) for the ending end of the segment, the centerline station and offset, the absolute angle, and the x and y coordinate,

(4) the 4 parameters defining the segment geometry,

(5) the segment number (1->N) calculated by IGIDS from the beginning end (1) to the ending end (N) depending on the type of segment list (lane inner edge, lane outer edge, lane stop line, centerline, curb return inner edge, and curb return outer edge),

(6) a flag indicating whether the segment is an arc or a line,

(7) a flag indicating whether the segment is an lane inner edge, lane outer edge, lane stop line, centerline, curb return inner edge, or curb return outer edge (which defines whether the segment parent is a leg or a lane), and

(8) the selected text ID.

For an arc segment, the 4 parameters are the radius, the sweep angle, the center x coordinate, and the center y coordinate. For a line, the first parameter is the length of the line and the remaining 3 parameters are not used. The segment structure itself has displayable graphics.

There are zero to any number of instances of the text structure. In addition to the attributes listed above, the text structure contains many attributes such as:

(1) the text angle,

(2) the text height,

(3) the text width,

(4) the lower left x and y coordinate,

(5) the 4 parameters for the text,

(6) a flag indicating whether the text parent is an alternative or a segment,

(7) the number of characters,

(8) the text font, and

(9) the text string.

For alternative text, the 4 parameters are the absolute angle for the lower left X/Y coordinate, the distance for the lower left X/Y coordinate, the absolute text angle, and parameter 4 is not used. For segment text, the 4 parameters are the relative angle for the lower left X/Y coordinate, the distance for the lower left X/Y coordinate, the absolute/relative text angle, and a flag indicating whether the text is at an absolute angle or at an angle relative to the segment. The text structure itself has displayable graphics.

OBJECT-ORIENTED PROGRAMMING TECHNIQUES

Object-oriented programming techniques have been used throughout IGIDS development. Although C++ is available, it was not used because it is not defined by an international standard.

IGIDS uses hierarchical, relational geometry. The primary structures or IGIDS database tables are *intersection* (Int), *alternative* (Alt), *leg* (Leg), *lane* (Lan), *segment* (Seg), and *text* (Txt). There is a global variable for each structure indicating the current instance. Each structure has a list of IDs of its children and each child structure has the ID of its parent. This feature allows traversal of the structure hierarchy.

IGIDS has a function (named "igids_xxxspr" where "xxx" is the 3 character structure name and "spr" stands for "set parent") which sets and checks the parent for the structure instance and which calls the appropriate "igids_xxxspr" function for the parent, automatically propagating up the hierarchy toward the intersection structure. Every structure except intersection has a parent. There is only one parent for a structure instance. In the case of a structure where there may be more than one parent structure, there is a flag indicating which is the parent.

Also, IGIDS has a function (named "igids_xxxsch" where "xxx" is the 3 character structure name and "sch" stands for "set child") which sets and checks the child for the structure instance and which calls the appropriate "igids_xxxsch" function for the child if a child instance exists, automatically propagating down the hierarchy toward the text structure. Every structure except text has a child structure. There may be no children instances even though there is a child structure. There may be more than one child structure for a structure. The function checks each list of children until if finds a child instance.

Finally, IGIDS has a function (named "igids_xxxsdb" where "xxx" is the 3 character structure name and "sdb" stands for "set database entry") which sets and checks the parent and the child for the structure instance by calling the appropriate "igids_xxxspr" and "igids_xxxsch" functions. This function is normally called after the user has selected an object by pointing at a segment or text.

After calling the appropriate "igids_xxxspr" or "igids_xxxsdb" function, all the attributes of the parent structures can be easily accessed. The programmer must know the structure that contains the attribute that is needed. As an example, suppose that "gslv_seg_id_num" is set to the ID of a segment and that the programmer calls the "igids_segspr" function, then the x coordinate of the center of the intersection for the segment's parent alternative is referenced by "gstv_alter_ent_ptr->sdfv_center_x".

IGIDS has structures and operations that it wants to perform on those structures. The primary structures or IGIDS database tables are *intersection*, *alternative*, *leg*, *lane* , *segment*, and *text*. The primary operations to be performed on those structures are:

(1) add a database entry (FUNCT_ADD_DBE),

(2) copy a database entry (FUNCT_CPY_DBE),

(3) delete a database entry (FUNCT_DEL_DBE),

(4) initialize the database (FUNCT_INI_DBE) (only valid for the intersection structure),

(5) select a database entry (FUNCT_SEL_DBE),

(6) calculate the graphics for a database entry (FUNCT_CAL_GRA),

(7) draw the graphics for a database entry (FUNCT_DRW_GRA),

(8) erase the graphics for a database entry (FUNCT_ERS_GRA),

(9) hilite the graphics for a database entry (FUNCT_HIL_GRA), and

(10) calculate the station and offset for a database entry (FUNCT_CAL_STA) (only valid for a leg, a lane, and a segment structure).

The only operations that can be performed on the intersection structure are

(1) initialize the database (FUNCT_INI_DBE),

(2) draw the graphics for a database entry (FUNCT_DRW_GRA), and

(3) erase the graphics for a database entry (FUNCT_ERS_GRA).

All operations can be used with all structures except as noted. Table 3.9 summarizes the relationship between the primary operation and the IGIDS objects.

Table 3.9 Relationship Between Primary Operations and IGIDS Objects

Primary Operation	int	alt	leg	lan	seg	txt
FUNCT_ADD_DBE	no	yes	yes	yes	yes	yes
FUNCT_CPY_DBE	no	yes	yes	yes	yes	yes
FUNCT_DEL_DBE	no	yes	yes	yes	yes	yes
FUNCT_INI_DBE	yes	no	no	no	no	no
FUNCT_SEL_DBE	no	yes	yes	yes	yes	yes
FUNCT_CAL_GRA	no	yes	yes	yes	yes	yes
FUNCT_DRW_GRA	yes	yes	yes	yes	yes	yes
FUNCT_ERS_GRA	yes	yes	yes	yes	yes	yes
FUNCT_HIL_GRA	no	yes	yes	yes	yes	yes
FUNCT_CAL_STA	no	no	yes	yes	yes	no

An object is a single instance of a structure and all its children (such as a leg or an inbound lane) or a list of children and all their children (such as the centerline segments of a leg). To allow the user to select an object by pointing at it:

(1) the user chooses the appropriate command which defines the operation to perform and the object type (such as rotate leg),

(2) the user points at some displayable graphics (arc, line, or text) for the object (such as the center line segments of the leg; the inner edge, outer edge, or stop line segments of an inbound or outbound lane for the leg; or text attached to a segment of the leg),

(3) the graphics engine searches the graphics engine database and gives IGIDS the ID from the graphics engine element that the user selected,

(4) based upon the ID and the type (arc, line, or text) of the graphics engine element, the appropriate "igids_xxxspr" function is called,

(5) IGIDS checks to confirm that the selected structure(s) refer to the requested object (automatically rejecting the graphics engine element if it does not refer to the requested object and repeating the process starting at (3)), and

(6) IGIDS hilites the selected object for acceptance or rejection by the user.

Table 3.10 is a list of the IGIDS objects and the corresponding "#define" object type.

In order to allow IGIDS to perform an operation on a structure or one or more selected children of the structure, a global processing mask (a bit mask; true = process, false = do not process) was created (gslv_proc_mask). Functions, like delete database entry for the leg, conditionally process the operation on each list of children for the specified structure instance or entry on the structure instance or entry itself based upon the value of a bit in the global processing mask . The first processing mask is for the structure instance or entry itself and uses bit 0 (value = 1). The "_ALL" processing mask is a bitwise "or" of all the possible values for a particular structure. Table 3.11 is a list of the IGIDS objects and the corresponding "#define" processing mask.

The traditional programming approach is to write a function where the operation is primary and the object is secondary. Each function knows how to perform the operation on every object. When the programmer wants to add another object, each function that performs an operation is modified to perform the operation on the new object. When the programmer wants to add another operation, a new function is developed that knows how to perform the operation on all objects.

The object-oriented programming approach is to write a function where the object is primary and the operation is secondary. Each function knows how to perform every operation on the object. When the programmer wants to add another object, a new function is developed that knows how to perform all operations on the object. When the programmer wants to add another operation, each function is modified to perform the new operation on the object.

The approach adopted in IGIDS is to develop a function for each object that dispatches the operation to the appropriate function(s) and develops a function for each object-operation combination where there is actually something to be done other than perform the operation on all children. IGIDS has a function (named "igids_xxx" where "xxx" is the 3 character structure name) which takes as a parameter the "FUNCT_" operation to be performed and dispatches the specified operation to the appropriate function(s). Additionally, IGIDS has a function (named "igids_xxxyyy" where "xxx" is the 3 character structure name and "yyy" is "adb" for FUNCT_ADD_DBE, "cdb" for FUNCT_CPY_DBE, "cgr" for FUNCT_CAL_GRA, "ddb" for FUNCT_DEL_DBE, "dgr" for FUNCT_DRW_GRA, "egr" for FUNCT_ERS_GRA, "hil" for FUNCT_HIL_GRA, "sdb" for FUNCT_SEL_DBE, and "sta" for FUNCT_CAL_STA) which performs operation "yyy" on structure "xxx". Finally, IGIDS has a function (named "igids_xxxpgf" where "xxx" is the 3 character structure name and "pgf" stands for "process generic function") which performs the specified "FUNCT_" operation on the selected "_PROC_" structure instance or entry. These "igids_xxxpgf" functions are used to perform an operation on all children. Currently, there is an "igids_xxxpgf" function for the *intersection, alternative, leg, lane*, and *segment* structures or IGIDS database tables.

Table 3.10 IGIDS Objects and Object Types

one	alt			OBJECT_ALT_ALL
all	legs		of an alt	OBJECT_ALT_LEGS
all	texts		of an alt	OBJECT_ALT_TXTS
one	leg			OBJECT_LEG_ALL
all	centerline	segs	of a leg	OBJECT_LEG_CNTRLN_SEGS
all	inbound	lanes	of a leg	OBJECT_LEG_INBLAN_LANS
all	outbound	lanes	of a leg	OBJECT_LEG_OUTLAN_LANS
all	in/out edge curb return	segs	of a leg	OBJECT_LEG_CR_SEGS
all	inner edge curb return	segs	of a leg	OBJECT_LEG_INN_CR_SEGS
all	outer edge curb return	segs	of a leg	OBJECT_LEG_OUT_CR_SEGS
one	lane			OBJECT_LAN_ALL
one	inbound	lane		OBJECT_LAN_INBLAN_ALL
one	outbound	lane		OBJECT_LAN_OUTLAN_ALL
all	inner edge	segs	of a lane	OBJECT_LAN_INNEDG_SEGS
all	outer edge	segs	of a lane	OBJECT_LAN_OUTEDG_SEGS
all	stop line	segs	of a lane	OBJECT_LAN_STOPLN_SEGS
one	seg			OBJECT_SEG_ALL
one	inner edge	seg		OBJECT_SEG_INNEDG_ALL
one	outer edge	seg		OBJECT_SEG_OUTEDG_ALL
one	stop line	seg		OBJECT_SEG_STOPLN_ALL
one	centerline	seg		OBJECT_SEG_CNTRLN_ALL
one	inner edge curb return	seg		OBJECT_SEG_INN_CR_ALL
one	outer edge curb return	seg		OBJECT_SEG_OUT_CR_ALL
all	texts		of a seg	OBJECT_SEG_TXTS
one	text			OBJECT_TXT_ALL
one	alt text			OBJECT_TXT_ALT_ALL
one	seg text			OBJECT_TXT_SEG_ALL
all	traffic control			OBJECT_TRAFFIC_CONTROL_ALL
all	traffic signs			OBJECT_TRAFFIC_SIGN
all	traffic signal faces			OBJECT_SIGNAL_FACE
all	traffic controllers			OBJECT_TRAFFIC_CONTROLLER
all	traffic channelization			OBJECT_CHANNELIZE
all	traffic signal phasing			OBJECT_SIGNAL_PHASING

Table 3.11 IGIDS Objects and Processing Masks

process intersection		only	INT_PROC_INT
process intersection		alts	INT_PROC_ALT
process intersection		and all children	INT_PROC_ALL
process alternative		only	ALTER_PROC_ALT
process alternative		legs	ALTER_PROC_LEG
process alternative		texts	ALTER_PROC_TXT
process alternative		and all children	ALTER_PROC_ALL
process leg		only	LEG_PROC_LEG
process leg centerline		segments	LEG_PROC_CNTRLN
process leg inbound		lanes	LEG_PROC_INBLAN
process leg outbound		lanes	LEG_PROC_OUTLAN
process leg inner edge	curb return	segments	LEG_PROC_INN_CR
process leg outer edge	curb return	segments	LEG_PROC_INN_CR
process leg	curb return	segments	LEG_PROC_CR
process leg		and all children	LEG_PROC_ALL
process lane		only	LANE_PROC_LANE
process lane inner edge		segments	LANE_PROC_INNEDG
process lane outer edge		segments	LANE_PROC_OUTEDG
process lane stop line		segments	LANE_PROC_STOPLN
process lane		and all children	LANE_PROC_ALL
process segment		only	SEG_PROC_SEG
process segment		texts	SEG_PROC_TXT
process segment		and all children	SEG_PROC_ALL
process text		only	TEXT_PROC_TXT
process text		and all children	TEXT_PROC_ALL

CHAPTER 4 IGIDS FUNCTIONAL DESIGN

MULTIPLE INTERSECTION ALTERNATIVES

IGIDS allows for the analysis and design of a minimum of 15 alternative concepts for each intersection. This capability is accomplished by ordering the hierarchical parent-child relationship as intersection-alternative-leg. Currently, MicroStation™ has a limit of 63 graphical levels or planes and IGIDS has allocated 4 graphical levels or planes per alternative. IGIDS is currently limited to 15 alternatives. Commands were developed to copy all or part of an alternative to another alternative and to modify all or part of an alternative. Additionally, commands were developed to display or not display an entire alternative.

LEVEL ASSIGNMENTS

Each alternative and the major graphical component groupings of the alternative are placed on separate graphical levels or planes so that they can be independently displayed or not displayed in a particular view by the graphics engine. IGIDS allocates a user graphical level or plane and a scratch graphical level or plane. IGIDS allows the user to display or not display graphics by alternative and items. These capabilities are accomplished by defining the following attributes in the intersection structure or IGIDS database table and by assigning them the values shown in Table 4.1.

Table 4.1 IGIDS Level Assignments

Variable Name	Description	Value
sslv_lev_scratch	Level for scratch graphics	2
sslv_beg_lev_alter	Beginning level for alternatives	3
sslv_num_lev_alter	Number of levels for alternatives	4
sslv_rel_lev_center	Relative level for centerlines	0
sslv_rel_lev_lanes	Relative level for lanes	1
sslv_rel_lev_tc	Relative level for traffic control	2
sslv_rel_lev_text	Relative level for texts	3

Additionally, commands were developed which allow the user to display or not display an entire alternative and to individually display or not display the centerline, lanes, traffic control, and text for an alternative.

USER INTERACTION WITH COMMANDS

IGIDS uses an interactive event-driven user interface. The various generic inputs available from the graphics engine are:

(1) command selection,
(2) coordinate entry,
(3) reset entry, and
(4) keyboard entry.

Command selection is made by:

(1) user keyin,
(2) function key activation,
(3) screen menu selection, or
(4) digitizer menu selection.

The net result of a command selection is that IGIDS can detect that a command has been selected, whether the command is a graphics engine command or an IGIDS command, and the command name or command number. The actual event causing the command selection is not important to IGIDS.

Coordinate entry is made by:

(1) user keyin,
(2) function key activation,
(3) mouse button activation, or
(4) cursor button activation.

The result of a coordinate entry is that IGIDS detects that a coordinate entry has been performed and obtains the coordinate value in real world coordinates. The actual event causing the coordinate entry is not important to IGIDS.

Reset entry is made by:

(1) user keyin,
(2) function key activation,
(3) screen menu selection,
(4) digitizer menu selection
(5) mouse button activation, or
(6) cursor button activation.

The result of a reset entry is that IGIDS detects that a reset entry has been performed. Keyboard entry is made by user keyin, command selection, or function key activation. The result of a keyboard entry is that

IGIDS detects that a keyboard entry has been performed and obtains the character string that was entered. The actual event causing the reset or keyboard entry is not important to IGIDS.

Each command is processed by a single function. To accomplish this goal, the concept of a processing stage was developed. The processing of a command proceeds from stage to stage until the command is completed or the user selects another command. Table 4.2 lists the global variables defined to implement processing of commands by stages. Table 4.3 enumerates the stages defined for IGIDS.

IGIDS knows the function to call (a large "switch" and "case" statement in function "igids_prccmd") and the stage number to set (STAGE_1) when the user performs a command selection and the command is an IGIDS command. IGIDS maintains a pointer to the function to call and the stage number to set when the user performs a coordinate entry, reset entry, or keyboard entry. The pointers to functions are paired with the previously discussed global stage variables. Table 4.4 lists the zone variables defined in the "ztdv_statedata" structure and the global variables defined to maintain a pointer to the function to call.

Basically, IGIDSs in an event loop waiting for the user to perform a:

(1) command selection,
(2) coordinate entry,
(3) reset entry, or
(4) keyboard entry.

When the graphics engine notifies IGIDS that the user has performed one of these events, IGIDS sets the global stage number to the appropriate value and calls the designated function. Upon return from the designated function, IGIDS waits for the user to perform another entry and the process begins again. MicroStation™ performs the event loop processing for IGIDS.

Table 4.2 IGIDS Global Variables to Implement Stages

Variable Name	Description		Comment	
gsiv_stage	current command	stage		
gsiv_stage_datapt	data button	stage	coordinate	entry
gsiv_stage_home	high-level return	stage		
gsiv_stage_keyin	keyin	stage	keyboard	entry
gsiv_stage_reenter	re-enter last data	stage		
gsiv_stage_reset	reset button	stage	reset	entry
gsiv_stage_return	low-level return	stage		

Table 4.3 IGIDS Stages

#define Name	Description
STAGE_0	Command Processing Stage Number 0 (special)
STAGE_2	Command Processing Stage Number 2
STAGE_3	Command Processing Stage Number 3
STAGE_4	Command Processing Stage Number 4
STAGE_5	Command Processing Stage Number 5
STAGE_6	Command Processing Stage Number 6
STAGE_7	Command Processing Stage Number 7
STAGE_8	Command Processing Stage Number 8
STAGE_9	Command Processing Stage Number 9
STAGE_10	Command Processing Stage Number 10
STAGE_11	Command Processing Stage Number 11
STAGE_12	Command Processing Stage Number 12
STAGE_13	Command Processing Stage Number 13
STAGE_14	Command Processing Stage Number 14
STAGE_15	Command Processing Stage Number 15
STAGE_16	Command Processing Stage Number 16
STAGE_17	Command Processing Stage Number 17
STAGE_18	Command Processing Stage Number 18
STAGE_19	Command Processing Stage Number 19
STAGE_20	Command Processing Stage Number 20
STAGE_END_COMMAND	Command Processing Stage Number 99

Table 4.4 IGIDS Function Pointers

Variable	Description	Comment
(*sslv_datafunc_pfn) ()	data button funct	coordinate entry
(*gslv_home_pfn) ()	high-level return funct	
(*sslv_keyinfunc_pfn) ()	keyin funct	keyboard entry
(*gslv_reenter_pfn) ()	re-enter last data funct	
(*sslv_resetfunc_pfn) ()	reset button funct	reset entry
(*gslv_return_pfn) ()	low-level return funct	

It is the responsibility of a command function to designate at each processing stage the appropriate function and stage for a coordinate entry, reset entry, and keyboard entry. Most of the time, in IGIDS, the reset entry is used to reject a selection. Normally, the coordinate entry and keyboard entry events are mutually exclusive at a given processing stage for a command so that a function and stage are designated to output a message to the user that invalid input has been received and dismissed. The general processing of a command is illustrated in Table 4.5.

Table 4.5 IGIDS Generalized Command Processing

```
long igids_command ()
{
    top:
        igids_cmdmsg  ( "command message" );
    switch ( gsiv_stage )
    {
        case STAGE_1:
            igids_askValSl (STAGE_1,"Keyin: Value",FALSE,0L,
                            igids_command          ,STAGE_END_COMMAND ,   /* reset */
                            NULL_FUNCTION_PFN,STAGE_0                 ,   /* reentry */
                            igids_command          ,STAGE_2                /* retask */  );
            break; /* wait for user input - keyin */
        case STAGE_2:
            if ( data_not_ok ) /* check data for acceptability */
            {
                igids_wrnmsg ( "data not ok" );
                gsiv_stage = STAGE_1;
                goto top;
            }
            /* save signed long data in static or global variable for later use */
        case STAGE_3:
            igids_askPoint (STAGE_1,"DataPt: Point",
                            igids_command          ,STAGE_1                ,   /* reset */
                            igids_command          ,STAGE_1                ,   /* reenter */
                            igids_command          ,STAGE_4                    /* retask */  );
            break; /* wait for user input - datapoint */
```

Table 4.5 IGIDS Generalized Command Processing (continued)

```
        case STAGE_4:
            if ( data_not_ok ) /* check data for acceptability */
            {
                igids_wrnmsg ( "data not ok" );
                gsiv_stage = STAGE_3;
                goto top;
            }
            /* save datapoint data in static or global variable for later use */
        case STAGE_5:
            /* process command with saved input */
            igids_infmsg ( "command completed" );
            gsiv_stage = STAGE_3; /* to re-cycle on last input */
            goto top;
        case STAGE_END_COMMAND:
            igids_endMdlPCmd ();
            break;
        default:
            igids_errmsg ( "programming error in command" );
            goto return_fatal_error;
    }

    return ( RETURN_SUCCESS );

    return_fatal_error:
    igids_detmsg ( "igids_command" );
    return ( RETURN_FATAL_ERROR );
}
```

IGIDS implements both noun/verb and verb/noun command processing. In noun/verb command processing, the user first selects the object to be operated upon (uses the selected object or executes the command to select a new object), then selects the operation to be performed by initiating the command, and finally accepts or rejects the hilited object. An example of noun/verb command processing is "leg rotate". In verb/noun command processing, the user first selects the operation to be performed by initiating the command, then selects the object to be operated upon by pointing at the object, and finally accepts or rejects the hilited object. IGIDS provides a toggle between noun/verb and verb/noun command processing. Since the noun/verb command processing uses the selected object, more methods of choosing the object are available with noun/verb command processing. The generalized code to implement both noun/verb and verb/noun command processing is illustrated in Table 4.6.

IGIDS allows the user to back up to previous input entry. This is accomplished by defining the following global variables (as previously discussed) and by developing a command to set the global stage to the re-enter stage and executing the re-enter function:

gsiv_stage_reenter	re-enter last data stage
(*gslv_reenter_pfn) ()	re-enter last data funct

It is the responsibility of a command function to designate at each processing stage the appropriate function and stage for an input re-entry operation.

IGIDS allows the user to cancel an IGIDS command at any point and to choose another IGIDS command at any point. These features were accomplished by making all commands collect input and store the data in global or static variables until all necessary input is received before performing the operation. Additionally, upon receiving a request for a graphics engine immediate command or an IGIDS immediate command, IGIDS can save the state of IGIDS command processing, execute the immediate command, restore the state of IGIDS, and continue processing the interrupted IGIDS command. An immediate command is one that may interrupt another command. The state of IGIDS command processing is the current values for the stage and function variables as discussed earlier.

IGIDS allows the user to switch between IGIDS commands and graphics engine commands. This was accomplished by IGIDS filtering all event loop data and sending the graphics engine data to the graphics engine for final processing.

Table 4.6 Generalized Noun/Verb and Verb/Noun Command Processing

```
long igids_command ()
{
    if ( gsiv_noun_verb && ( gsiv_stage == STAGE_1 ) )
    {
        gsiv_stage = STAGE_2;
    }
    top:
        igids_cmdmsg  ( "command message" );
    switch ( gsiv_stage )
    {
        case STAGE_1:
            /* select object */
        case STAGE_2:
            /* hilite and accept selected object */
            if ( selected_object_not_accepted )
            {
                gsiv_stage = STAGE_1;
                goto top;
            }
        case STAGE_3:
            /* process remainder of command */
            break;
        default:
            igids_errmsg ( "programming error in command" );
            goto return_fatal_error;
    }

    return ( RETURN_SUCCESS );

    return_fatal_error:
    igids_detmsg ( "igids_command" );
    return ( RETURN_FATAL_ERROR );
}
```

DATABASE "SAVE" OPERATIONS

Each structure and IGIDS database table contains an attribute which indicates whether the entry has been modified since the last time that the internal copy of the IGIDS database was written to the external copy. Upon initial startup of IGIDS, after the user executes the IGIDS LOADFROM DATABASE command, and after the user executes the IGIDS SAVETO DATABASE command, IGIDS sets all IGIDS objects as nonmodified. Every addition or modification of an IGIDS object sets that object's attribute as modified. Upon ending IGIDS or ending MicroStation™, IGIDS determines whether any data has been modified since the last SAVETO DATABASE command and presents to the user an Alert Box. Pressing the "OK" push button will allow IGIDS to perform a SAVETO DATABASE command. Pressing the "Cancel" push button will cause IGIDS to exit without saving any data.

CHAPTER 5 GETTING STARTED

To become comfortable with IGIDS, stepping through a simple example will be useful. This example is for an Intergraph workstation running the Clix operating system. The user will be lead through an example which performs the following actions:

(1) login to the workstation,
(2) start MicroStation,
(3) start IGIDS,
(4) place an intersection that is typical of those found in many areas,
(5) adjust the angle of one leg by 3.25 degrees,
(6) place a standard Vehicle Turning Template,
(7) save the intersection to a database file for later use,
(8) end IGIDS,
(9) end MicroStation™, and
(10) logoff the workstation.

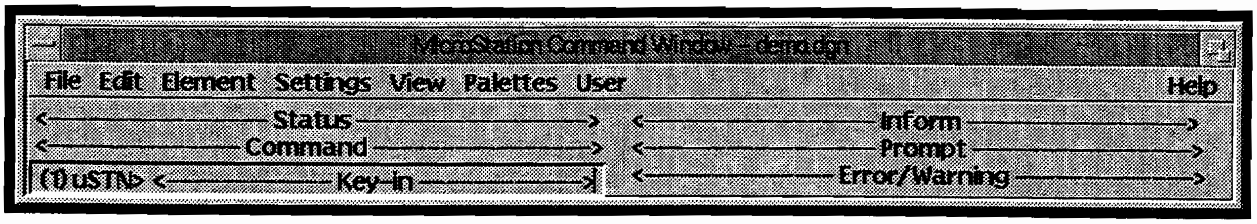
IGIDS has been installed for your use in the directory **/usr/igids**. If IGIDS has not been installed on your workstation, please refer to the Appendix entitled Installation Procedures.

Step 1 is to login to the workstation in the usual manner as indicated in Figure 5.1.

```
login: binman
Password:
CLIX System V Release 3.1 IA3260
ia3260nl
Copyright (c) 1984 AT&T
Copyright (c) 1992
Intergraph Corporation; All Rights Reserved
Including Application Programs, File Formats, and Visual Displays
********************************************************************************
*                                                                              *
*                                                                              *
*          Nasser I. Al-Rashid Transportation Engineering Laboratory           *
*                                                                              *
*                       Department of Civil Engineering                        *
*                                                                              *
*                      The University of Texas at Austin                       *
*                                                                              *
*                                                                              *
********************************************************************************
$ ustation_igids  /usr/igids/demo.dgn
Interactive Graphics Intersection Design System
COPYRIGHT 1993 The University of Texas at Austin
using default IGIDS path "/usr/igids".
```

Figure 5.1 Starting MicroStation™ for use with IGIDS

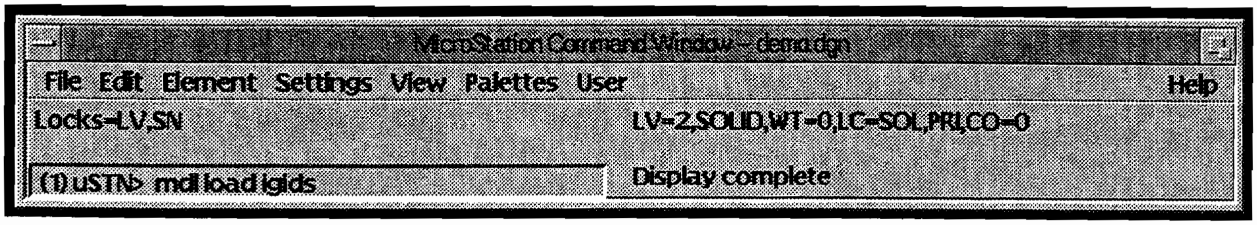
Step 2 is to start MicroStation™. IGIDS operation requires the support of a graphics engine (MicroStation™). This graphics engine must be up and running before starting IGIDS. To start MicroStation™ in a mode to work with IGIDS, the Unix command **ustation_igids** has been installed on your computer. A graphics file named **/usr/igids/demo.dgn** has been included in the IGIDS delivery for your use. To start MicroStation™ and use the above named file, at the UNIX prompt, type in "**ustation_igids /usr/igids/demo.dgn**" on one line. Once the MicroStation™ startup is complete, the MicroStation™ Command Window will appear (see Figure 5.2). The MicroStation™ field names (status, command, key-in, inform, prompt, and error/warning) have been added to this figure for your future reference. IGIDS places the command name chosen by the user in the command field, places informative messages in the inform field, places user prompts for data entry in the prompt field, and places error and warning messages in the error/warning field. The user enters keyboard input and MicroStation™ key-in commands in the key-in field. The user should set the MicroStation™working units (master units, sub units, and positional units) and set the global origin before starting IGIDS. IGIDS prohibits the user from changing the active level, the level lock, the level for an element, and the global origin. **Changing the working units after IGIDS has started would cause disastrous results.**



(1) uSTN>

Figure 5.2 MicroStation™ Command Window

Step 3 is to start IGIDS by keying in the MicroStation™ command "**mdl load igids**" in the key-in field of the MicroStation™ Command Window (see Figure 5.3).


MicroStation Command Window - demo.dgn
File Edit Element Settings View Palettes User
Help

Locks=LV,SN

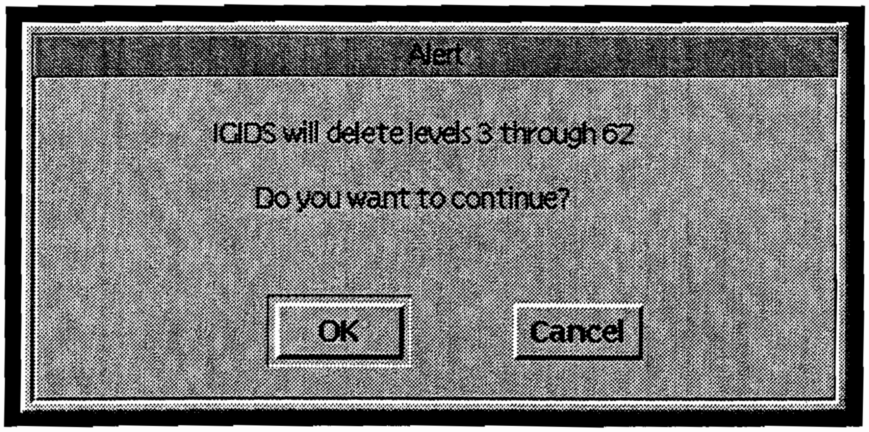
LV=2,SOLID,WT=0,LC=SOL,PRI,CO=0

(1) uSTN> mdl load igids

Display complete

Figure 5.3 Starting IGIDS in the MicroStation™ Command Window

IGIDS takes control of the active graphics file and deletes everything except what is recognized as scratch graphics (all graphics on levels 3 through 62 are controlled by IGIDS, while level 2 is the scratch level). IGIDS presents to the user an Alert Box (see Figure 5.4). Press the "OK" push button to allow IGIDS to continue.

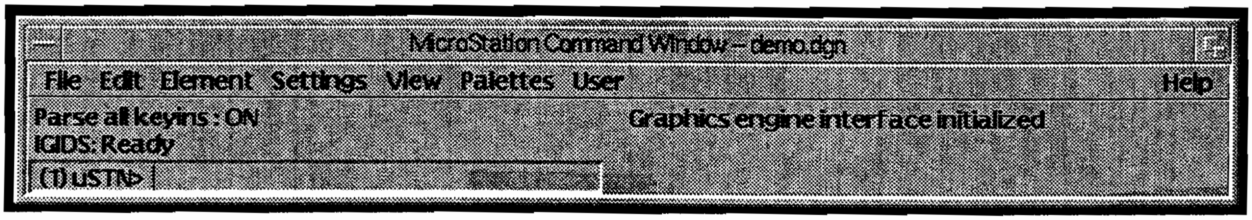


IGIDS will delete levels 3 through 62

Do you want to continue?

Figure 5.4 IGIDS Beginning Alert Box

When ready, IGIDS will report "**IGIDS: Ready**" in the command field of the MicroStation™ Command Window (see Figure 5.5).



```
Parse all keyins : ON                Graphics engine interface initialized
IGIDS: Ready
(1) uSTN>
```

Figure 5.5 MicroStation™ Command Window after Starting IGIDS

The IGIDS startup sequence will display the IGIDS menu at the upper left of the screen (see Figure 5.6).

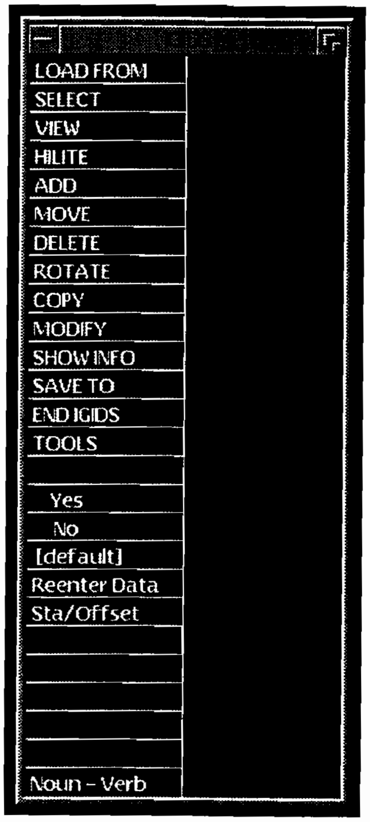
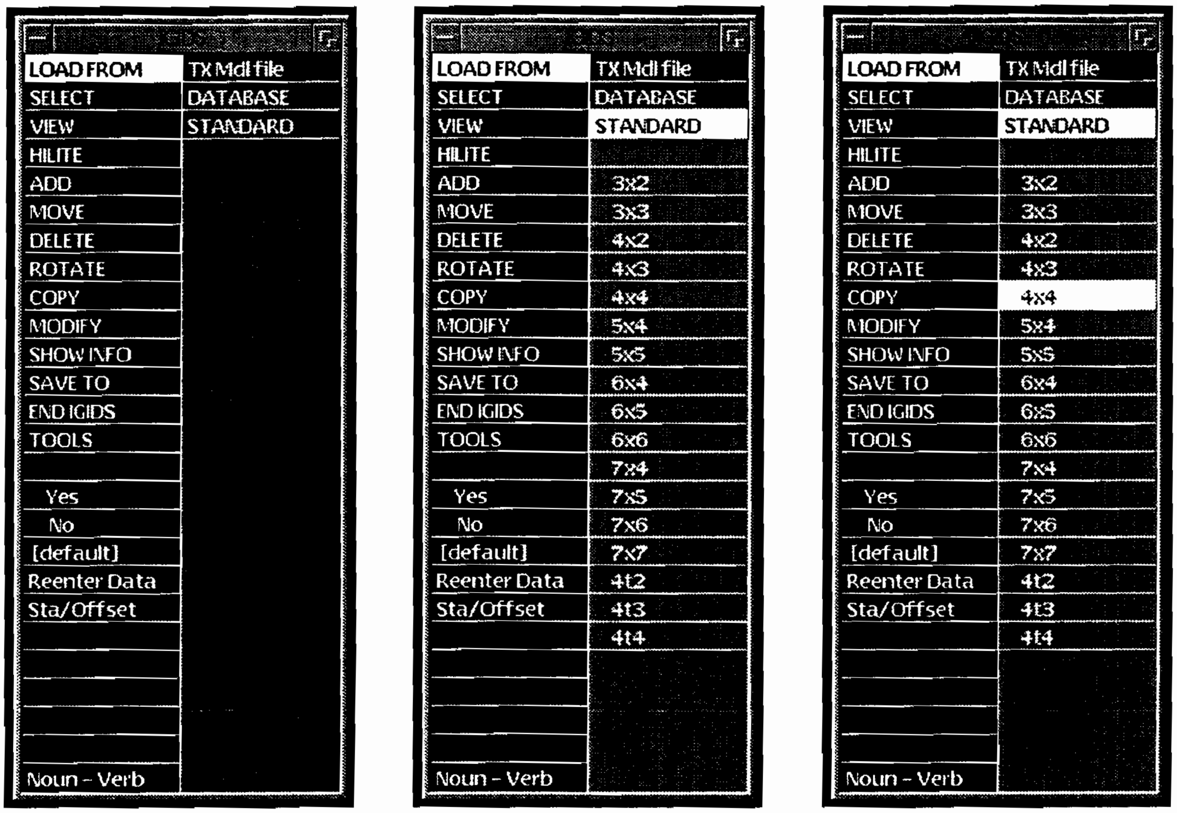


Figure 5.6 IGIDS menu

You may now execute the IGIDS commands of your choice. IGIDS commands are initiated by choosing one or a sequence of IGIDS menu items. These choices are made by placing the cursor on the desired menu item, then pressing and releasing the data button on the mouse. This sequence of moving the cursor, pressing, and then releasing the data button is called a click. If you are not sure which mouse button is designated as the data button, please consult your system administrator. The data button is usually the leftmost button on a 2-button or 3-button mouse.

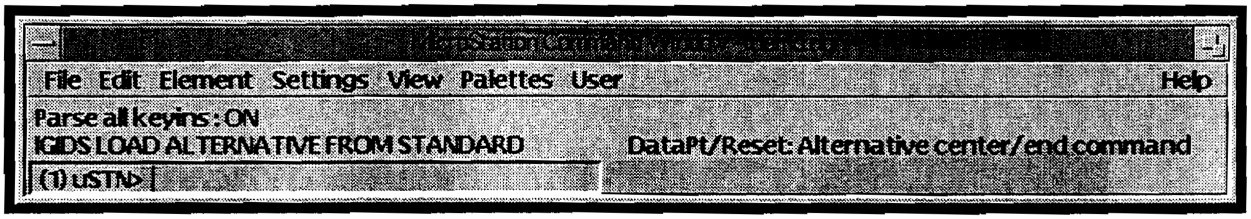
Step 4 is to place an intersection that is typical of those found in many areas by clicking on the **LOADFROM** menu item at the top left of the IGIDS menu (see Figure 5.7-a). The colors of the text and background of the **LOADFROM** menu item will be swapped and three new menu items will appear in the right column of the IGIDS menu. Next, click on the **STANDARD** menu item (see Figure 5.7-b). An additional set of menu items will appear in the right column of the menu below the currently displayed menu items. From these, click the **4x4** menu item (see Figure 5.7-c). This will start an IGIDS command to load the chosen standard Alternative.



(a) (b) (c)

Figure 5.7 Starting the IGIDS command *LOADFROM STANDARD 4x4*

The name of the active IGIDS command will be shown in the Command field of the MicroStation Command Window (see Figure 5.8). The Prompt field is used to inform the user of the next user action for IGIDS.



Parse all keyins : ON

IGDS LOAD ALTERNATIVE FROM STANDARD

DataPt/Reset: Alternative center/end command

(1) uSTN>

Figure 5.8 MicroStation Command Window with first prompt from the command *LOADFROM STANDARD 4x4*

The command **LOADFROM - STANDARD - 4x4** requires some further user input to complete the operation. The actions that are acceptable to IGIDS are indicated in the Prompt field. At this stage of the command, **"DataPt/Reset: Alternative center/end command"** is shown in the Prompt field (see Figure 5.8). This prompt has two major divisions separated by a colon (:). Each major division has some number of minor divisions which are separated by a slash (/). The major division to the left of the colon indicates the acceptable user input types ("DataPt", "Keyin", and/or "Reset"). The minor divisions to the right of the colon have a one-to-one relationship to those to the left and offer a brief description of the input or indicate the way that IGIDS will respond to each input.

In this case, either a DataPt or a Reset is an acceptable user input. A DataPt is a specialized type of click that requires the cursor to be positioned on some portion of a graphical window and not on a menu item. A DataPt defines the X and Y coordinates of a point to IGIDS. A DataPt may also be input by entering any of the MicroStation™ precision input key-in commands. These commands are listed in the MicroStation™ Reference Guide in the Appendix and discussed under the topic Precision Input. The most commonly used entry is "XY=x,y". A Reset is the press and release of the mouse button that is designated as the reset button or other mechanism to generate a reset. The reset button is usually the rightmost button on a 2-button and 3-button mouse. A KeyIn is keyboard entry of alphanumeric characters terminated with a carriage return. The displayed prompt indicates that IGIDS will interpret a DataPt to indicate the coordinates of the center point for the Alternative that is being added. A Reset will indicate that the user wishes to end processing of the current IGIDS command.

Place a DataPt near the center of a window. IGIDS will process this input and report the actual coordinates in the Inform field. IGIDS has also put "**Keyin: Alternative number [1]**" into the Prompt field. This indicates that a Keyin is the only acceptable user input and that the keyin will be the number that the user wishes to assign to this Alternative. The "1" in square brackets is the default number to be used if the user input consists only of pressing the return key or the user clicks on the "[default]" command in the IGIDS menu. Press the return key. IGIDS will read an external file that describes the standard of interest and draw the graphics to represent this Alternative. Completion of the loading process will be reported in the Inform field. To properly view the graphics, it may be necessary to use a MicroStation™zoom or fit command to adjust the scale of the window. (Figure 5.9 depicts the alternative at this stage of this example (the text "Standard 4x4" has been moved in the plot).

Step 5 is to adjust the angle of one leg by 3.25 degrees by clicking the **ROTATE** command from the left column of the IGIDS menu and then the **LEG** command from the right column. This will start the **ROTATE - LEG** command. In response to the prompt "**Keyin: Rotation angle [1.0]**", enter "3.25" plus a carriage return through the keyboard in the key-in field. IGIDS will hilite the leg nearest the top of the screen and issue the prompt "**DataPt/Reset: accept & define dir. & rotate/reidentify**". Enter a Reset and the leg nearest the top of the screen will return to its normal

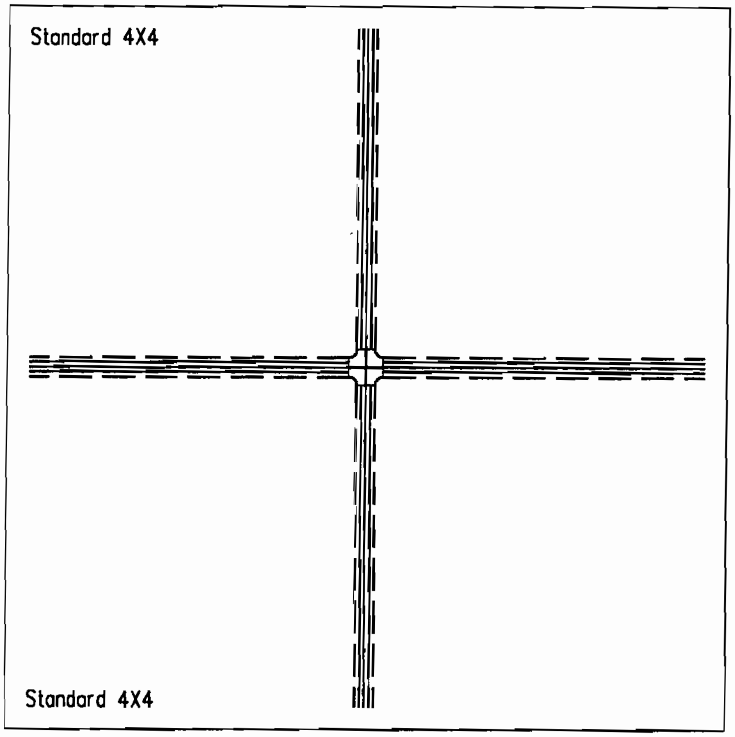


Figure 5.9 Standard 4x4 Intersection

color. In response to the prompt "**DataPt: identify a Leg**", place a DataPt on the leg nearest the bottom of the screen. In response to the prompt "**DataPt/Reset: accept & define dir. & rotate/reidentify**", place a DataPt to the right of the hilited Leg in a location that the perpendicular projection of the point falls on the hilited centerline of the Leg. IGIDS will understand this DataPt to mean: (1) the hilited leg has been accepted as the leg to rotate, (2) since the point was located in the counterclockwise direction from the leg, rotation is to be counterclockwise, and (3) the user is ready to rotate the leg. Note that IGIDS has rotated the leg through the 3.25 degrees and reported statistics in the Inform field. You have probably noticed that to start an IGIDS command, first click a menu item from the left column of the IGIDS menu then click one or a sequence of menu items from the right column.

Step 6 is to place a standard Vehicle Turning Template by clicking **TOOLS**, then **TurnTemplate**, and then **WB-60** to start the command that places Vehicle Turning Templates. The templates depict the area traversed by a vehicle turning through an angle defined by the directions of an Inbound and Outbound Leg. The outside, front bumper and the inside, rear tire locations are tracked while keeping the outside, front tire on the arc of a circle of radius defined by the user for the selected AASHTO vehicle.

The prompt "**DataPt: identify inbound leg**" will be displayed. In response, place a DataPt on any graphic element that is part of the Leg toward the bottom of the screen. IGIDS will hilite this leg and prompt "**DataPt/Reset: accept/reidentify**". Place a DataPt anywhere to confirm that this is the Inbound Leg of choice. IGIDS will prompt "**DataPt/Reset: identify outbound leg/reidentify inbound leg**". Place a DataPt on any graphic element that is part of the Leg to the right of the screen. IGIDS will hilite this Leg and prompt "**DataPt/Reset: accept/reidentify**". Place a DataPt anywhere to accept this as the desired Outbound Leg.

IGIDS will prompt "**Keyin/Reset: turn radius[45]/reidentify Outbound Leg**". Press the return key to use the default 45 feet turn radius. IGIDS will draw the requested template. MicroStation's window area or zoom in command may be used to better view the turn template while the move element command may be used to position the turn template as desired by (1) keying in "move element" in the MicroStation™ keyin field or clicking on the move element command in the MicroStation™ menu, (2) placing a DataPt on any of the graphics of the Vehicle Turning Template, (3) moving the cursor to dynamically move the Vehicle Turning Template to the desired position, (4) placing a DataPt at the desired position, and (5) entering a Reset to terminate the MicroStation™ move element command. The final intersection example graphics are shown in Figure 5.10 and 5.11.

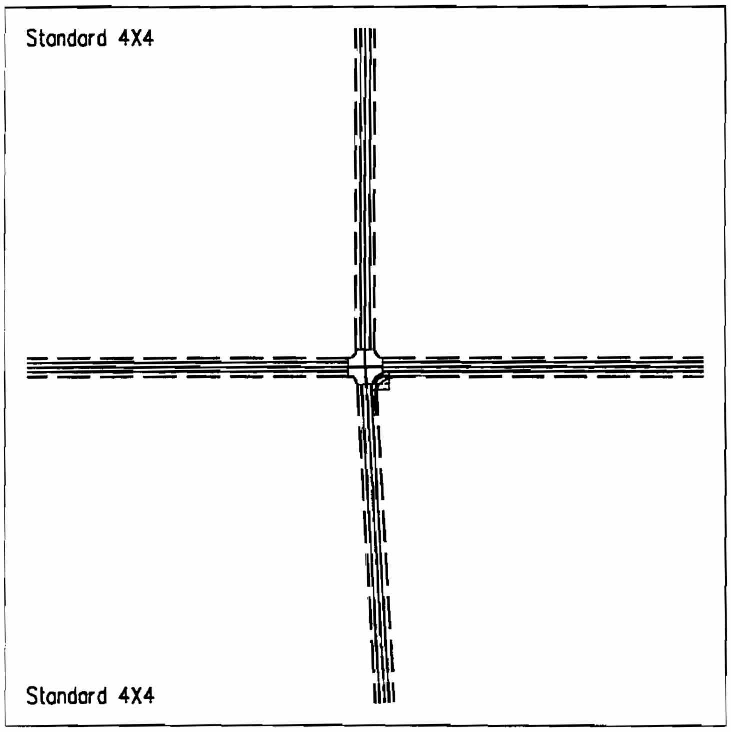


Figure 5.10 Example Intersection

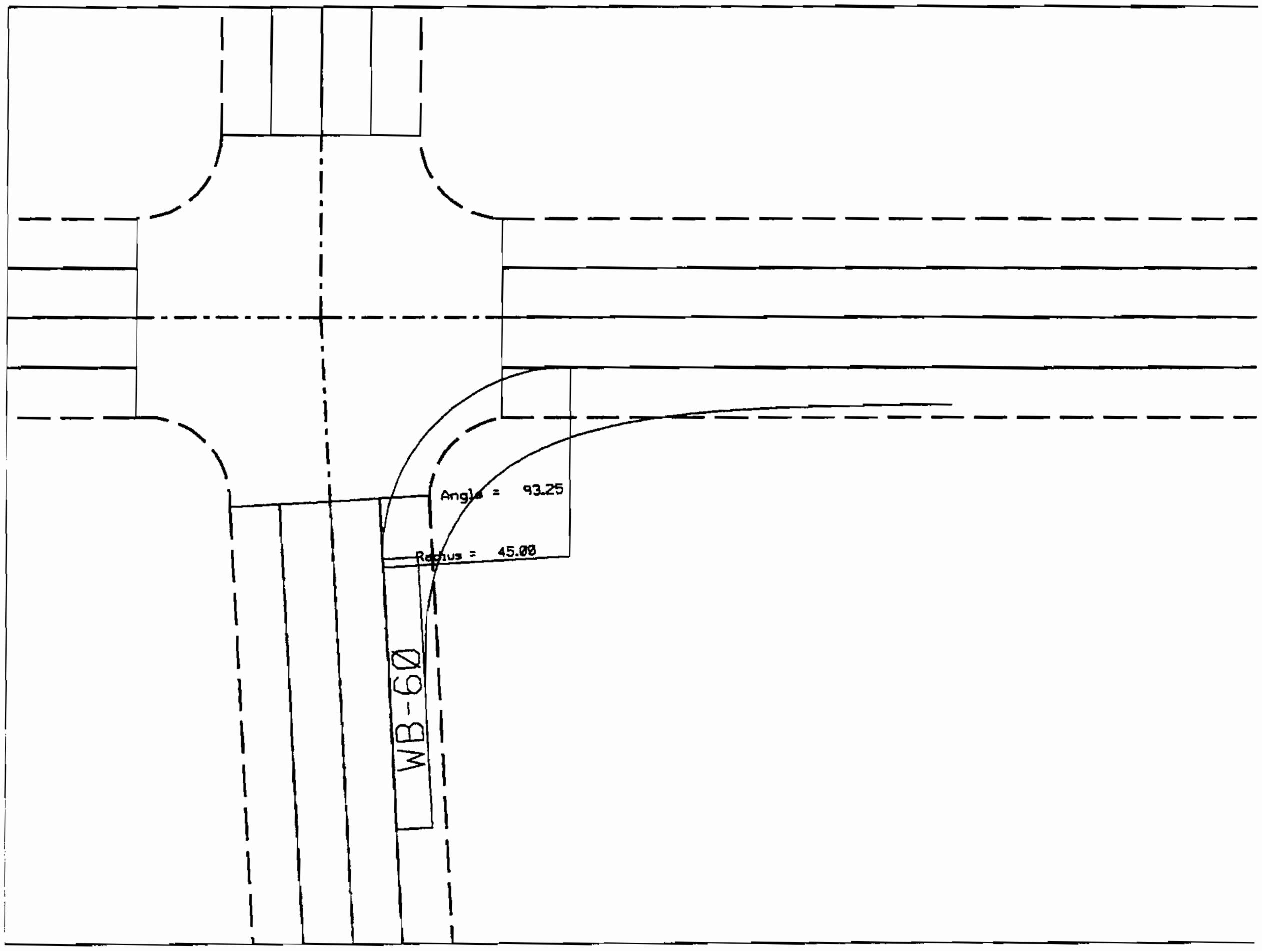


Figure 5.11 Enlarged Example Intersection

Step 7 is to save the intersection to a database file for later use by clicking on the **SAVE TO** command from the left column of the IGIDS menu and then the **DataBase** command from the right column. This will start the save alternative to database command. In response to the prompt "**Keyin: IGIDS database file name**", enter "**demo.dbs**" through the keyboard into the keyin field. IGIDS will save the intersection and all alternative data to the file "**demo.dbs**" and report "**Saved database file name: demo.dbs**" in the inform field. You may later load this intersection and all alternatives by issuing the **LOAD FROM - DATABASE** command at another time.

Step 8 is to end IGIDS by clicking on the "**END IGIDS**" command from the left column of the IGIDS menu. Upon ending IGIDS or ending MicroStation^{TM}, IGIDS determines whether any data has been modified since the last SAVETO DATABASE command, if any, and presents to the user an Alert Box. Pressing the "OK" push button will allow IGIDS to perform a SAVETO DATABASE command whereas pressing the "Cancel" push button will cause IGIDS to exit without saving any data. Since a SAVETO

DATABASE command was just completed, the Alert Box will not be presented to the user. When finished, IGIDS will report "**End IGIDS**" in the command field.

Step 9 is to end MicroStation™ by choosing the MicroStation™ **File** then **Exit** command from the MicroStation™ pull down menu or keying in "**exit**" in the keyin field.

Step 10 is to logoff the workstation by entering "**lo**" at the Unix prompt. (not applicable on DOS machines)

Now that you have become a novice IGIDS user, proceed to become an expert user. To do this, the following steps are suggested:

1) perform the scripted example of entering traffic signal data in the next chapter,
2) scan the menu tree in the Appendix entitled IGIDS Command Menus to become familiar with the location of commands on the IGIDS menu,
2) study the command descriptions and their processing diagrams in the Appendix IGIDS Command Descriptions, and
3) develop command sequences for the jobs that you are to do.

CHAPTER 6 TRAFFIC SIGNAL DATA EXAMPLE

Adding signalization to a 4 leg intersection with left turn bays on the north and south legs will serve as an advanced example of using IGIDS. The user should have already completed the scripted beginning example in the Chapter entitled Getting Started. This advanced example is for an Intergraph workstation running the Clix operating system. If using a DOS microcomputer steps 1 and 14 may not be necessary. The user will be lead through an example which performs the following actions:

(1) login to the workstation,
(2) start MicroStation™,
(3) start IGIDS,
(4) place a 4 leg intersection with left turn bays on the north and south legs,
(5) place a pretimed controller,
(6) place signal faces,
(7) specify signal phasing,
(8) specify signal timing,
(9) specify traffic volumes,
(10) save the intersection to Texas Model for Intersection Traffic files for later use,
(11) save the intersection to a database file for later use,
(12) end IGIDS,
(13) end MicroStation™, and
(14) logoff the workstation.

IGIDS has been installed for your use in the directory **/usr/igids**. If IGIDS has not been installed on your workstation, please refer to the Appendix entitled Installation Procedures.

Step 1 is to login to the workstation in the usual manner.

Step 2 is to start MicroStation™ by entering "**ustation_igids /usr/igids/demo.dgn**" on one line at the Unix prompt.

Step 3 is to start IGIDS by keying in the MicroStation™ command "**mdl load igids**" in the key-in field of the MicroStation™ Command Window.

Step 4 is to place a 4 leg intersection with left turn bays on the north and south legs by executing the **LOADFROM - STANDARD - 5x4** menu item. Place a DataPt near the center of a window in response to the prompt "**DataPt/Reset: Alternative center/end command**". Press the return key in response to the prompt "**Keyin: Alternative number [1]**". Figure 6.1 and 6.2 depicts the alternative at this stage of this example (the text "Standard 5x4" has been moved in the plot).

Step 5 is to place a pretimed controller. Start the **TOOLS - TRAFFIC - CONTROLLER - PRETIMED** command. In response to the prompt "**DataPt: locate and place Pretimed**

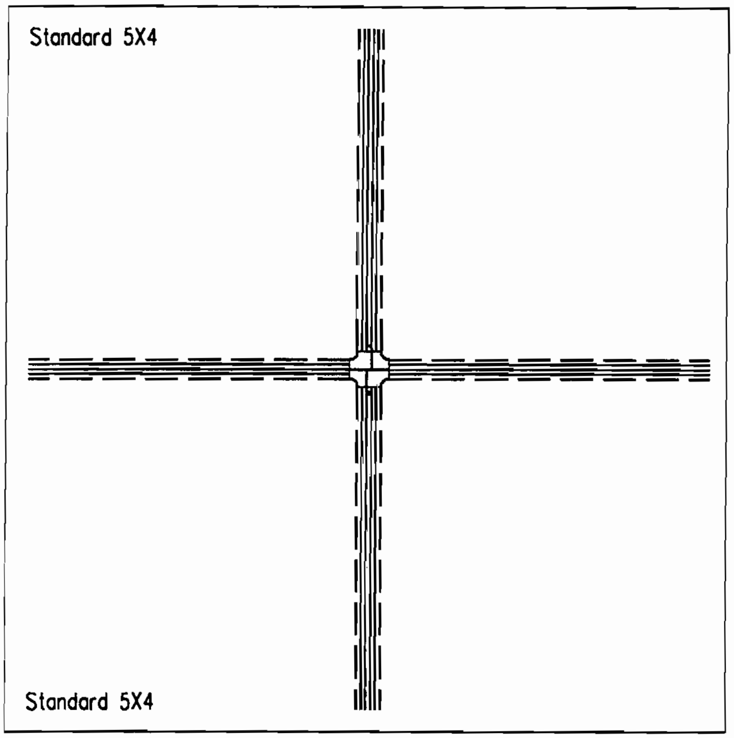


Figure 6.1 Standard 5x4 Intersection

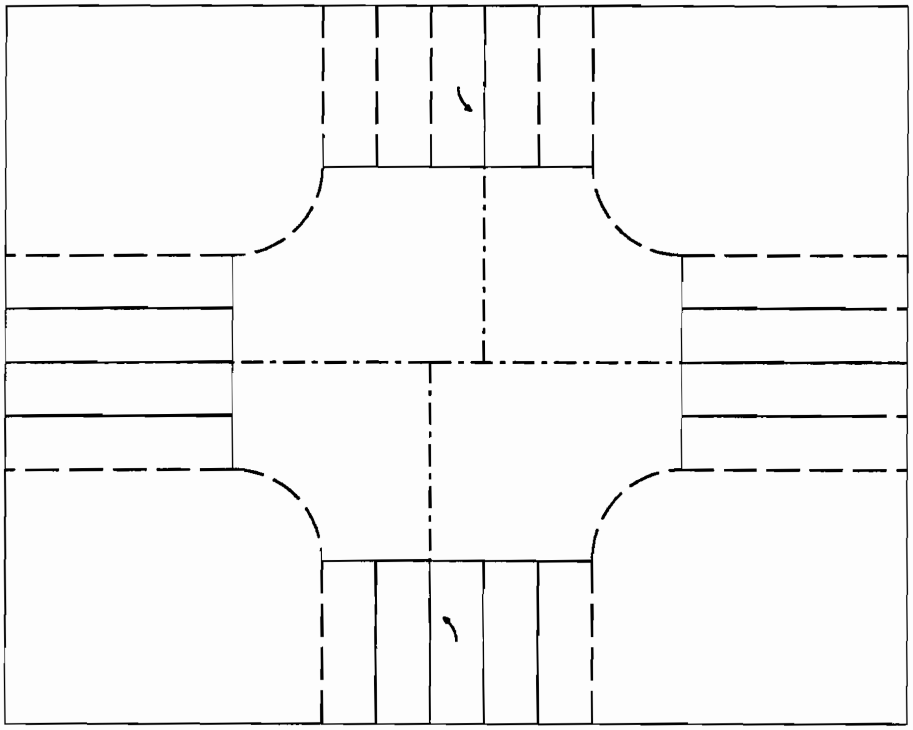


Figure 6.2 Enlarged Standard 5x4 Intersection

controller", use a DataPt to locate and place controller graphic in the north east quadrant near the intersection where the two stop lines would intersect. In response to the prompt "**Keyin: number of Pretimed controller phases**", key in the value "3" and a carriage return. The rectangle with the words "PRETIMED" should be drawn at the entered datapoint.

Step 6 is to place signal faces. Start the command **Tools - Traffic - Signal Face - 3 Lens**. In response to the prompt "**DataPt: identify Inbound Lane**", place a DataPt on the outside Lane of the Southbound Leg (the stop line for a lane is the best object to use to identify a lane), then in response to the prompt "**DataPt/Reset: accept and place 3 Lens Face/reidentify**", place a DataPt on the outside Lane of each of the Northbound, Eastbound, and Westbound legs, and a final acceptance DataPt anywhere not on any IGIDS object. Next, start the command **Tools - Traffic - Signal Face - 3 Lens PL**. In response to the prompt "**DataPt: identify Inbound Lane**", place a DataPt on the inside Lane of the Southbound Leg , then in response to the prompt "**DataPt/Reset: accept and place 3 Lens Prot Left Face/reidentify**", place a DataPt on the inside Lane of the Northbound Leg, and a final acceptance DataPt anywhere not on any IGIDS object. The final intersection example graphics is shown in Figure 6.3.

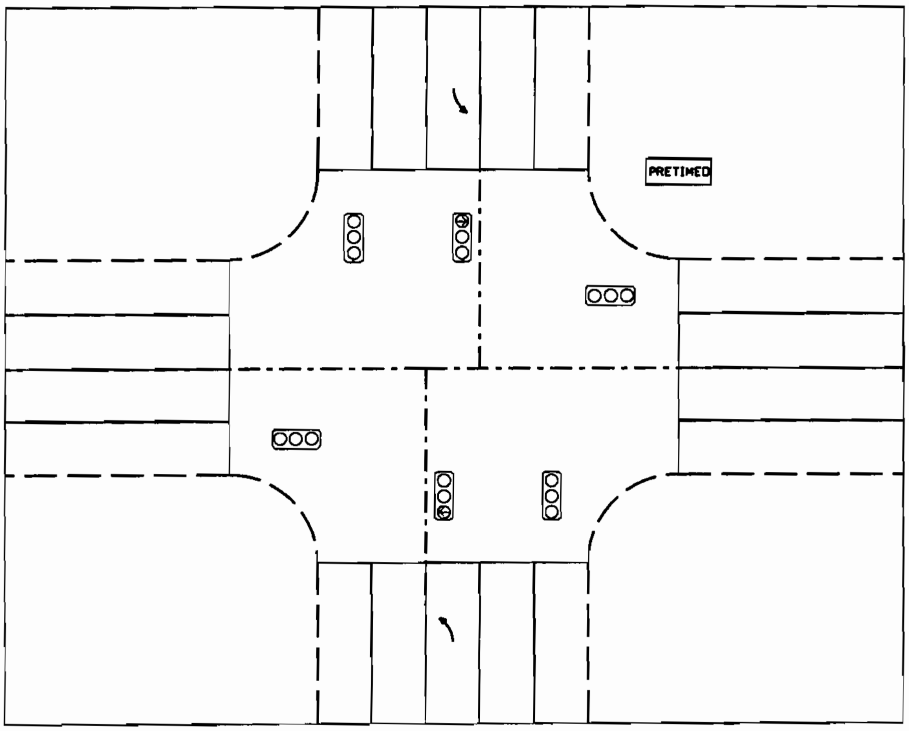


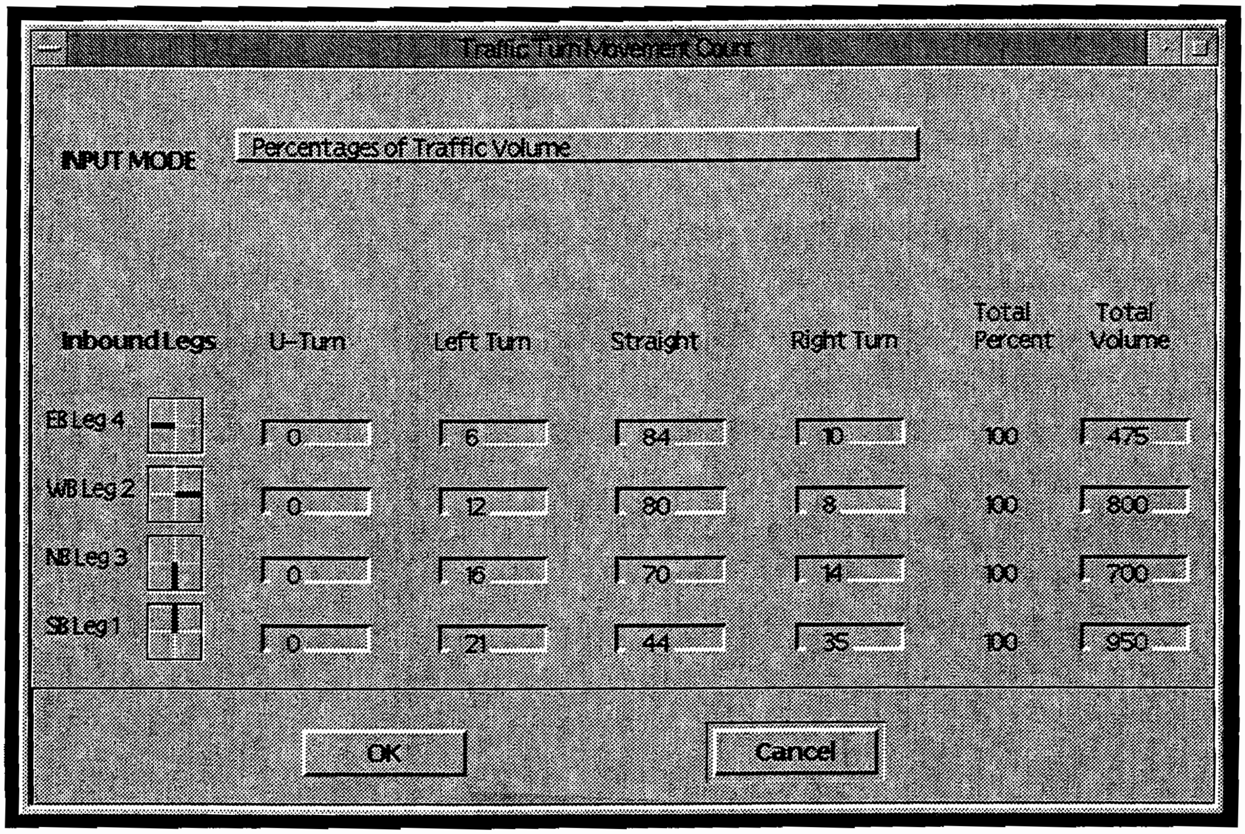
Figure 6.3 Enlarged Example Intersection

Step 7 is to specify signal phasing. Start the **Tools - Traffic - Controller - PHASING** command. In response to the prompt "**Keyin: phase number[1]**", key in "1". In response to the prompt "**DataPt/Reset: id. sig.hd. or chan.sym. to add,remove/new phase**", place a DataPt on the Protected Left Signal Face on the Southbound Leg. The signal face will be hilited. Place a DataPt on the Protected Left Signal Face on the Northbound Leg. This signal face will be hilited and the Southbound Protected Left Signal Face will be shown in green. Place an acceptance DataPt anywhere not on any Signal Face. The Northbound Protected Left Signal Face will be shown in green. The Signal Faces shown in green will be given green indications during phase 1. Press the reset button. In response to the prompt "**Keyin: phase number[1]**", key in "2". In response to the prompt "**DataPt/Reset: id. sig.hd. or chan.sym. to add,remove/new phase**", place a DataPt on the other Signal Face on the Southbound Leg. The signal face will be hilited. Place a DataPt on the other Signal Face on the Northbound Leg. The signal face will be hilited and the Southbound Signal Face will be shown in green. Place an acceptance DataPt anywhere not on any Signal Face. The Northbound Signal Face will be shown in green. The Signal Faces shown in green will be given green indications during phase 2. Press the reset button. In a similar manner, include the East and West leg in phase 3.

To check phasing, in response to the prompt "**Keyin: phase number[1]**", key in "1". The north and south Protected Left Signal Faces will be shown in green. Press the reset button and the north and south Protected Left Signal Faces will be shown in their normal color. Key in "2" and the other north and south Signal Faces will be shown in green. Press the reset button and the other north and south Signal Faces will be shown in normal color. Key in "3" and the east and west Signal Faces will be shown in green. Press the reset button and the east and west Signal Faces will be shown in their normal color. Press the reset button.

Step 8 is to specify signal timing. Start the command **Tools - Traffic - Controller - TIMING.** In response to the prompt "**Keyin/Reset: Ph 1 Green [30.0]/next phase**", key in "11". In response to the prompt "**Keyin/Reset: Ph 1 Yellow Change [3.0]/next phase**", use the Default Command to choose the default value (3.0). In response to the prompt "**Keyin/Reset: Ph 1 All-Red Clearance [1.0]/next phase**", key in "0". The current values are reported in the inform field of the MicroStation™ Command Menu ("Ph 1 green,yellow,red= 11.00,3.00,0.00"). Press the reset button. In a similar manner, enter timing data for phase 2 using 20,3, and 0, and for phase 3 using 30, 3, and 0. To check timing, review the current timing data for each phase in the inform field. Press the reset button to see data for the next phase.

Step 9 is to specify traffic volumes. Start the command **TOOLS - TRAFFIC - VOLUME - Percent+VOL**. For East Bound Leg 3, enter a tab in the U-Turn field, "6" plus a tab in the Left Turn field, "84" plus a tab in the Straight field (notice that the column labeled Total Percent now contains 90), "10" plus a tab in the Right Turn field, and "475" plus a tab in the Total Volume field. In a similar manner, enter the remaining data as shown in Figure 6.4. Notice that after the "950" plus a tab is entered into the South Bound Leg 1 Total Volume field, the **OK** button is enabled. Now press the **OK** button.



Traffic Turn Movement Count

INPUT MODE: Percentages of Traffic Volume

Inbound Legs	U-Turn	Left Turn	Straight	Right Turn	Total Percent	Total Volume
EB Leg 4	0	6	84	10	100	475
WB Leg 2	0	12	80	8	100	800
NB Leg 3	0	16	70	14	100	700
SB Leg 1	0	21	44	35	100	950

Figure 6.4 Traffic Turn Movement Count

Step 10 is to save the intersection to Texas Model for Intersection Traffic files for later use. Start the command **SAVE TO - TX Mdl file**. In response to the "**Keyin/Reset: TEXAS Model GDV file [GDVDATA]/skip**" prompt, enter a carriage return to use the default and in response to the "**Keyin: TEXAS Model SIM file name [SIMDATA]**" prompt, enter a carriage return to use the default.

Step 11 is to save the intersection to a database file by starting the **SAVE TO - Data Base** command. In response to the prompt "**Keyin: IGIDS database file name**", enter "**signal.dbs**".

Step 12 is to end IGIDS by clicking on the "**END IGIDS**" command.

Step 13 is to end MicroStation™by choosing the MicroStation™ **File** then **Exit** command from the MicroStation™ pull down menu or keying in "**exit**" in the keyin field.

Step 14 is to logoff the workstation by entering "**lo**" at the Unix prompt.

REFERENCES

1. A Policy on Geometric Design of Highways and Streets. 1990, American Association of State Highway and Transportation Officials, Suite 225, 444 North Capitol Street, N.W., Washington, D.C., 20001.

2. Highway Capacity Manual, Special Report 209, Transportation Research Board, National Research Council, Washington, D.C., 1985.

3. Lee, Clyde E., Machemehl, Randy B., Rioux, Thomas W., and Inman, Robert F. "The TEXAS Model for Intersection Traffic, Version 3.2," 1993, Center for Transportation Research, The University of Texas at Austin, Austin, Texas, 78705-2650.

4. Courage, K.G., "Signal Operations Analysis Package, Version 84," University of Florida Transportation Research Center, prepared for FHWA Implementation Package, 1985, with McTrans Supplement for SOAP, Version 4.1, June 1991.

5. Texas Truck Off-Tracking Model. Texas Department of Transportation, Austin, Texas.

APPENDIX A

IGIDS Installation Instructions

This appendix contains the IGIDS Installation Instructions for the Unix Operating System for workstations and for the MS-DOS Operating System for PCs.

Unix Installation Instructions

Interactive Graphics Intersection Design System (IGIDS)

1. Login to the system as superuser by entering "root" and the root password.

2. Insert the 3.5" floppy disk labeled IGIDS in the computer.

3. If you want IGIDS to be located somewhere other than "/usr/igids", then create the alternate directory and soft link the directory to "/usr/igids". An example is "mkdir /usr2/igids ; In -s /usr2/igids /usr/igids ; chmod 755 /usr/igids /usr2/igids".

4. Enter "fr_flop I compress -d I cpio -ivmud".

5. Copy or move "igids.ma" to the MicroStation mdlapps directory. An example is "cp /usr/igids/igids.ma /usr/ip32/mstation/mdlapps/igids.ma".

6. Copy or move "ustation_igids" to the "/usr/bin" directory. An example is "cp /usr/igids/ustation_igids /usr/bin/ustation_igids".

7. Logoff the system as superuser by entering "exit".

8. Login to the system as yourself by entering your username and password.

9. Execute MicroStation by entering "ustation_igids" or "ustation_igids <file>".

10. In the "MicroStation Command Window", select the "User" pull-down menu and then select "Environment Variables".

11. Search the environment variable list for the entry "IGIDS_PATH". If the "IGIDS_PATH" environment variable does not exist, then create the "IGIDS_PATH" environment variable by (a) entering "IGIDS_PATH" in the text key-in field above and to the left of the "New" button, (b) entering "/usr/igids" in the text key-in field above the "Delete" button, (c) pressing the 'New" button, (d) and finally pressing the "Save" button. If the "IGIDS_PATH" environment variable does exist and is not "/usr/igids", then set the "IGIDS_PATH" environment variable by (a) selecting the "IGIDS_PATH" environment variable from the list by pressing on it, (b) entering "/usr/igids" in the text key-in field above the "Delete" button, (c) and finally pressing the "Save" button.

12. Search the environment variable list for the entry "MS_FNTLB". If the "MS_FNTLB" environment variable is not "/usr/igids/sys_data/igidsfnt.lib", then set the "MS_FNTLB" environment variable by (a) selecting the "MS_FNTLB" environment variable from the list by pressing on it, (b) entering "/usr/igids/sys_data/igidsfnt.lib" in the text key-in field above the "Delete" button, (c) and finally pressing the "Save" button.

13. Search the environment variable list for the entry "MS_FONTRSC". If the "MS_FONTRSC" environment variable is not "/usr/igids/sys_data", then set the "MS_FONTRSC" environment variable by (a) selecting the "MS_FONTRSC" environment variable from the list by pressing on it, (b) entering "/usr/igids/sys_data" in the text key-in field above the "Delete" button, (c) and finally pressing the "Save" button.

14. Exit the "Environment Variables" menu by double-clicking the "-" icon in the top left corner of the menu.

15. If you set or changed the "MS_FNTLB" or the "MS_FONTRSC" environment variable, enter "exit" in the "MicroStation Command Window" and then execute MicroStation again by entering "ustation_igids" or "ustation_igids <file>".

16. In the "MicroStation Command Window", enter "mdl load igids".

MS-DOS Installation Instructions

Interactive Graphics Intersection Design System (IGIDS)

1. Insert the 3.5" floppy disk labeled IGIDS in the computer.

2. Enter "<disk_fr>install <disk_fr> <disk_to>" where "<disk_fr>" is "a:" or "b:" and "<disk_to>" is "c:" or "d:" or "e:" or "f:" or "g:". An example is "a:install a: c:".

3. Copy or move "igids.ma" to the MicroStation mdlapps directory. An example is "copy c:\igids\igids.ma c:\ustation\mdlapps\igids.ma".

4. Execute MicroStation by entering "ustation" or "ustation <file>".

5. In the "MicroStation Command Window", select the "User" pull-down menu and then select "Environment Variables".

6. Search the environment variable list for the entry "IGIDS_PATH". If the "IGIDS_PATH" environment variable does not exist, then create the "IGIDS_PATH" environment variable by (a) entering "IGIDS_PATH" in the text key-in field above and to the left of the "New" button, (b) entering "C:\IGIDS" in the text key-in field above the "Delete" button, (c) pressing the 'New" button, (d) and finally pressing the "Save" button. If the "IGIDS_PATH" environment variable does exist and is not "C:\IGIDS", then set the "IGIDS_PATH" environment variable by (a) selecting the "IGIDS_PATH" environment variable from the list by pressing on it, (b) entering "C:\IGIDS" in the text key-in field above the "Delete" button, (c) and finally pressing the "Save" button.

7. Search the environment variable list for the entry "MS_FNTLB". If the "MS_FNTLB" environment variable is not "C:\IGIDS\SYS_DATA\IGIDSFNT.LIB", then set the "MS_FNTLB" environment variable by (a) selecting the "MS_FNTLB" environment variable from the list by pressing on it, (b) entering "C:\IGIDS\SYS_DATA\IGIDSFNT.LIB" in the text key-in field above the "Delete" button, (c) and finally pressing the "Save" button.

8. Exit the "Environment Variables" menu by double-clicking the "-" icon in the top left corner of the menu.

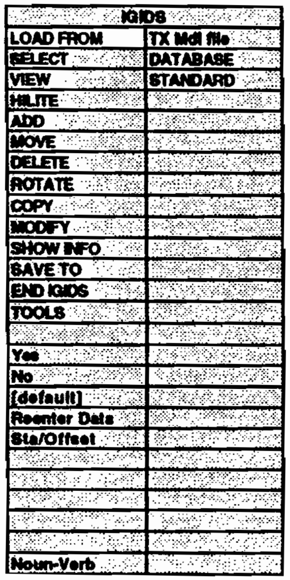
9. If you set or changed the "MS_FNTLB" environment variable, enter "exit" in the "MicroStation Command Window" and then execute MicroStation again by entering "ustation" or "ustation <file>".

10. In the "MicroStation Command Window", enter "mdl load igids".

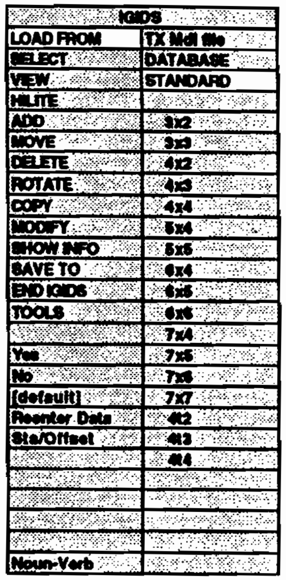
APPENDIX B

IGIDS Command Menus

This appendix contains the IGIDS Command Menus. The IGIDS Command Menus are in command menu order from top to bottom for the commands in the left column and are depicted the same as they appear to the user of IGIDS. All available sub-command options are shown.

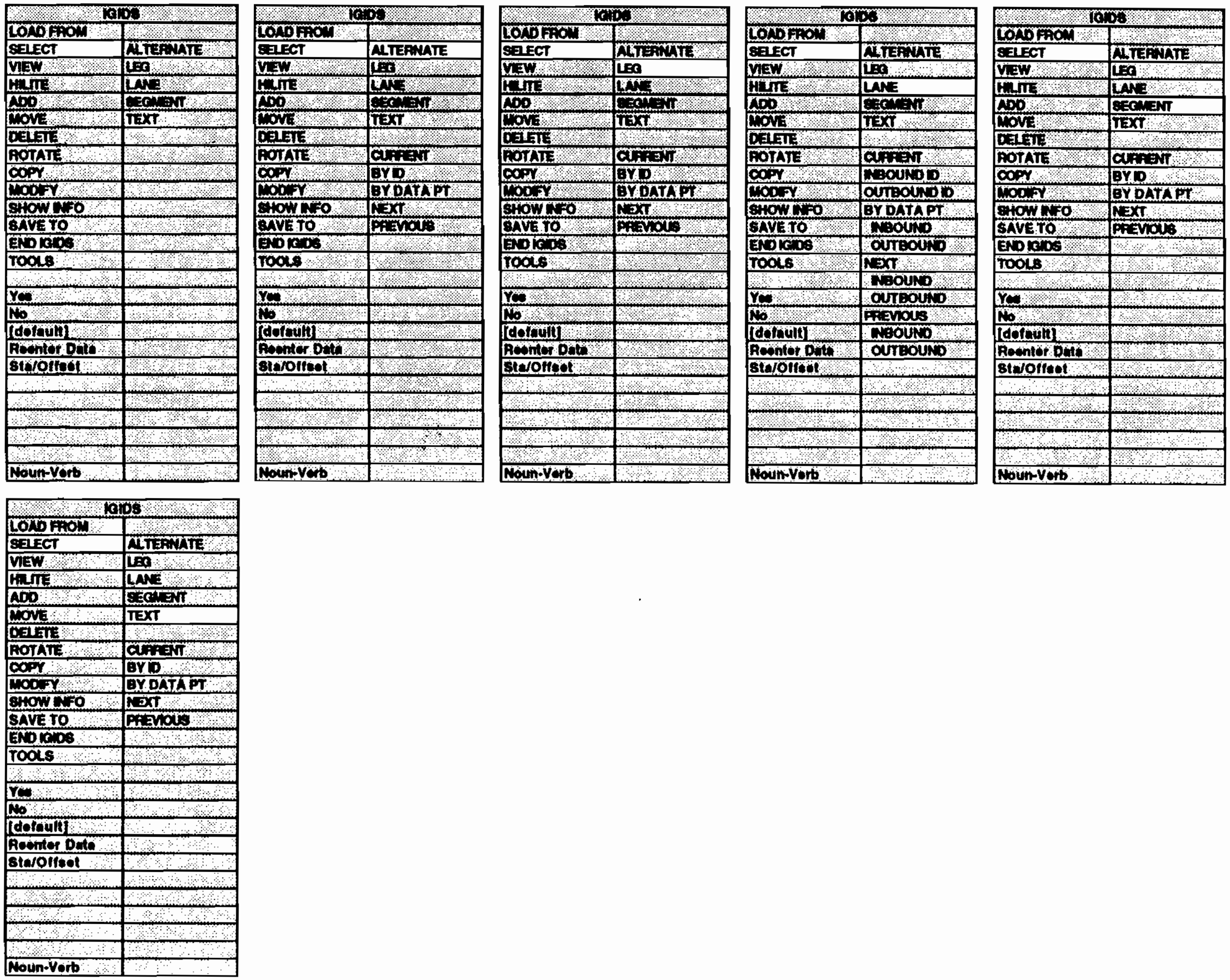


IGIDS	
LOAD FROM	TX Mdl file
SELECT	DATABASE
VIEW	STANDARD
HILITE	
ADD	
MOVE	
DELETE	
ROTATE	
COPY	
MODIFY	
SHOW INFO	
SAVE TO	
END IGIDS	
TOOLS	
Yes	
No	
[default]	
Reenter Data	
Sta/Offset	
Noun-Verb	



IGIDS	
LOAD FROM	TX Mdl file
SELECT	DATABASE
VIEW	STANDARD
HILITE	
ADD	3x2
MOVE	3x3
DELETE	4x2
ROTATE	4x3
COPY	4x4
MODIFY	5x4
SHOW INFO	5x5
SAVE TO	6x4
END IGIDS	6x5
TOOLS	6x6
	7x4
Yes	7x5
No	7x6
[default]	7x7
Reenter Data	4t2
Sta/Offset	4t3
	4t4
Noun-Verb	

Figure B.1 IGIDS Command Menus - LOAD FROM commands



IGIDS	
LOAD FROM	
SELECT	ALTERNATE
VIEW	LEG
HILITE	LANE
ADD	SEGMENT
MOVE	TEXT
DELETE	
ROTATE	
COPY	
MODIFY	
SHOW INFO	
SAVE TO	
END IGIDS	
TOOLS	
Yes	
No	
[default]	
Reenter Data	
Sta/Offset	
Noun-Verb	

IGIDS	
LOAD FROM	
SELECT	ALTERNATE
VIEW	LEG
HILITE	LANE
ADD	SEGMENT
MOVE	TEXT
DELETE	
ROTATE	CURRENT
COPY	BY ID
MODIFY	BY DATA PT
SHOW INFO	NEXT
SAVE TO	PREVIOUS
END IGIDS	
TOOLS	
Yes	
No	
[default]	
Reenter Data	
Sta/Offset	
Noun-Verb	

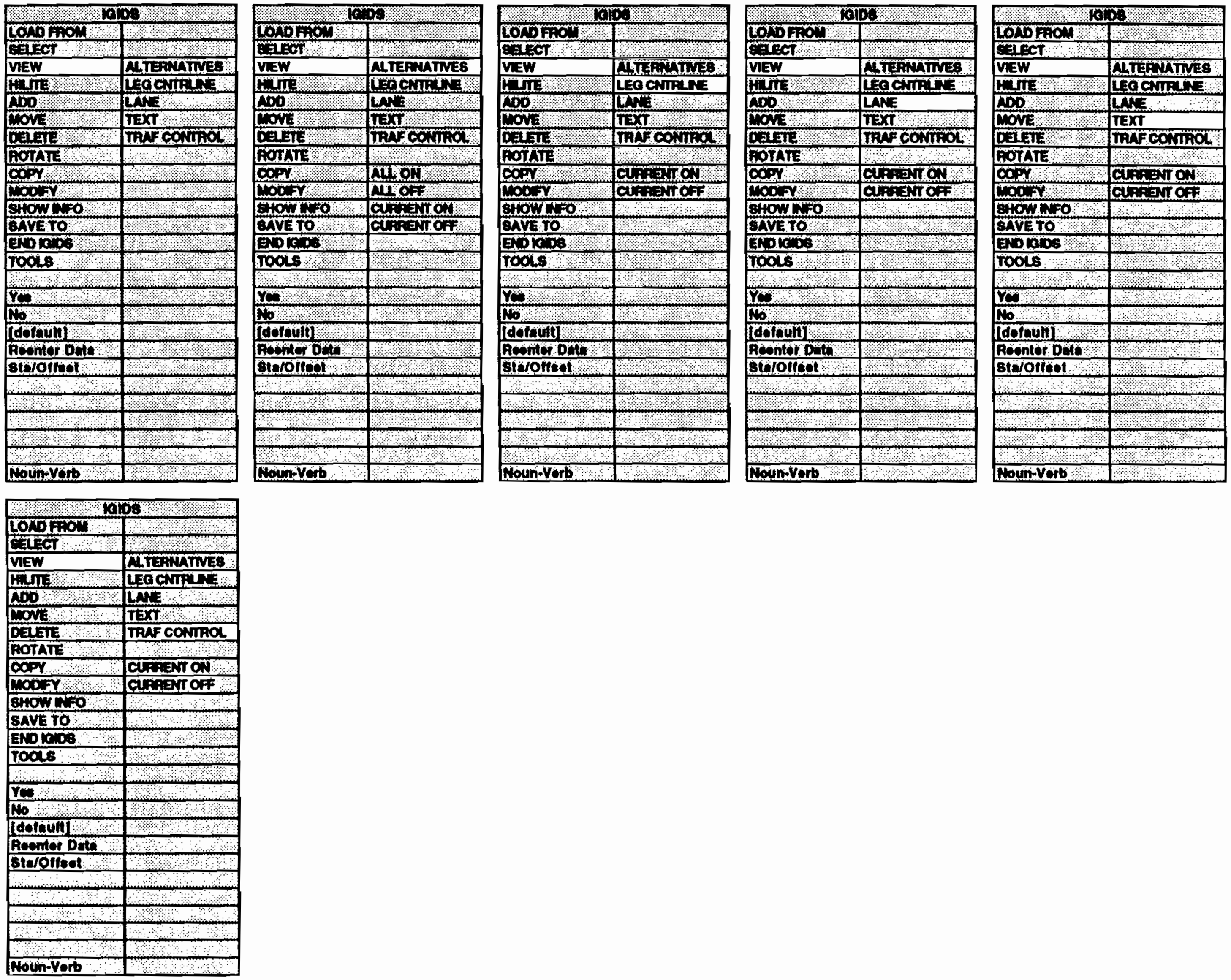
IGIDS	
LOAD FROM	
SELECT	ALTERNATE
VIEW	LEG
HILITE	LANE
ADD	SEGMENT
MOVE	TEXT
DELETE	
ROTATE	CURRENT
COPY	BY ID
MODIFY	BY DATA PT
SHOW INFO	NEXT
SAVE TO	PREVIOUS
END IGIDS	
TOOLS	
Yes	
No	
[default]	
Reenter Data	
Sta/Offset	
Noun-Verb	

IGIDS	
LOAD FROM	
SELECT	ALTERNATE
VIEW	LEG
HILITE	LANE
ADD	SEGMENT
MOVE	TEXT
DELETE	
ROTATE	CURRENT
COPY	INBOUND ID
MODIFY	OUTBOUND ID
SHOW INFO	BY DATA PT
SAVE TO	INBOUND
END IGIDS	OUTBOUND
TOOLS	NEXT
	INBOUND
Yes	OUTBOUND
No	PREVIOUS
[default]	INBOUND
Reenter Data	OUTBOUND
Sta/Offset	
Noun-Verb	

IGIDS	
LOAD FROM	
SELECT	ALTERNATE
VIEW	LEG
HILITE	LANE
ADD	SEGMENT
MOVE	TEXT
DELETE	
ROTATE	CURRENT
COPY	BY ID
MODIFY	BY DATA PT
SHOW INFO	NEXT
SAVE TO	PREVIOUS
END IGIDS	
TOOLS	
Yes	
No	
[default]	
Reenter Data	
Sta/Offset	
Noun-Verb	

IGIDS	
LOAD FROM	
SELECT	ALTERNATE
VIEW	LEG
HILITE	LANE
ADD	SEGMENT
MOVE	TEXT
DELETE	
ROTATE	CURRENT
COPY	BY ID
MODIFY	BY DATA PT
SHOW INFO	NEXT
SAVE TO	PREVIOUS
END IGIDS	
TOOLS	
Yes	
No	
[default]	
Reenter Data	
Sta/Offset	
Noun-Verb	

Figure B.2 IGIDS Command Menus - SELECT commands



IGIDS	
LOAD FROM	
SELECT	
VIEW	ALTERNATIVES
HILITE	LEG CNTRLINE
ADD	LANE
MOVE	TEXT
DELETE	TRAF CONTROL
ROTATE	
COPY	
MODIFY	
SHOW INFO	
SAVE TO	
END IGIDS	
TOOLS	
Yes	
No	
[default]	
Reenter Data	
Sta/Offset	
Noun-Verb	

IGIDS	
LOAD FROM	
SELECT	
VIEW	ALTERNATIVES
HILITE	LEG CNTRLINE
ADD	LANE
MOVE	TEXT
DELETE	TRAF CONTROL
ROTATE	
COPY	ALL ON
MODIFY	ALL OFF
SHOW INFO	CURRENT ON
SAVE TO	CURRENT OFF
END IGIDS	
TOOLS	
Yes	
No	
[default]	
Reenter Data	
Sta/Offset	
Noun-Verb	

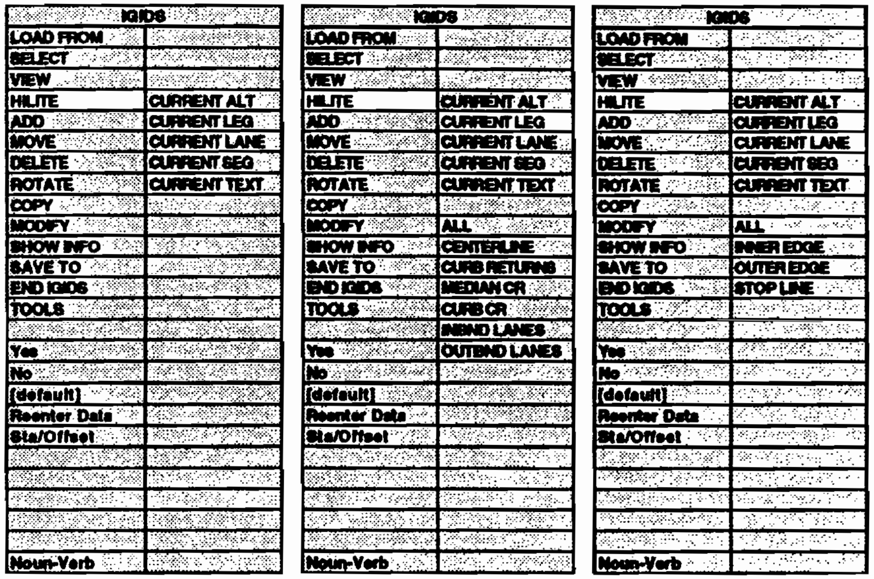
IGIDS	
LOAD FROM	
SELECT	
VIEW	ALTERNATIVES
HILITE	LEG CNTRLINE
ADD	LANE
MOVE	TEXT
DELETE	TRAF CONTROL
ROTATE	
COPY	CURRENT ON
MODIFY	CURRENT OFF
SHOW INFO	
SAVE TO	
END IGIDS	
TOOLS	
Yes	
No	
[default]	
Reenter Data	
Sta/Offset	
Noun-Verb	

IGIDS	
LOAD FROM	
SELECT	
VIEW	ALTERNATIVES
HILITE	LEG CNTRLINE
ADD	LANE
MOVE	TEXT
DELETE	TRAF CONTROL
ROTATE	
COPY	CURRENT ON
MODIFY	CURRENT OFF
SHOW INFO	
SAVE TO	
END IGIDS	
TOOLS	
Yes	
No	
[default]	
Reenter Data	
Sta/Offset	
Noun-Verb	

IGIDS	
LOAD FROM	
SELECT	
VIEW	ALTERNATIVES
HILITE	LEG CNTRLINE
ADD	LANE
MOVE	TEXT
DELETE	TRAF CONTROL
ROTATE	
COPY	CURRENT ON
MODIFY	CURRENT OFF
SHOW INFO	
SAVE TO	
END IGIDS	
TOOLS	
Yes	
No	
[default]	
Reenter Data	
Sta/Offset	
Noun-Verb	

IGIDS	
LOAD FROM	
SELECT	
VIEW	ALTERNATIVES
HILITE	LEG CNTRLINE
ADD	LANE
MOVE	TEXT
DELETE	TRAF CONTROL
ROTATE	
COPY	CURRENT ON
MODIFY	CURRENT OFF
SHOW INFO	
SAVE TO	
END IGIDS	
TOOLS	
Yes	
No	
[default]	
Reenter Data	
Sta/Offset	
Noun-Verb	

Figure B.3 IGIDS Command Menus - VIEW commands

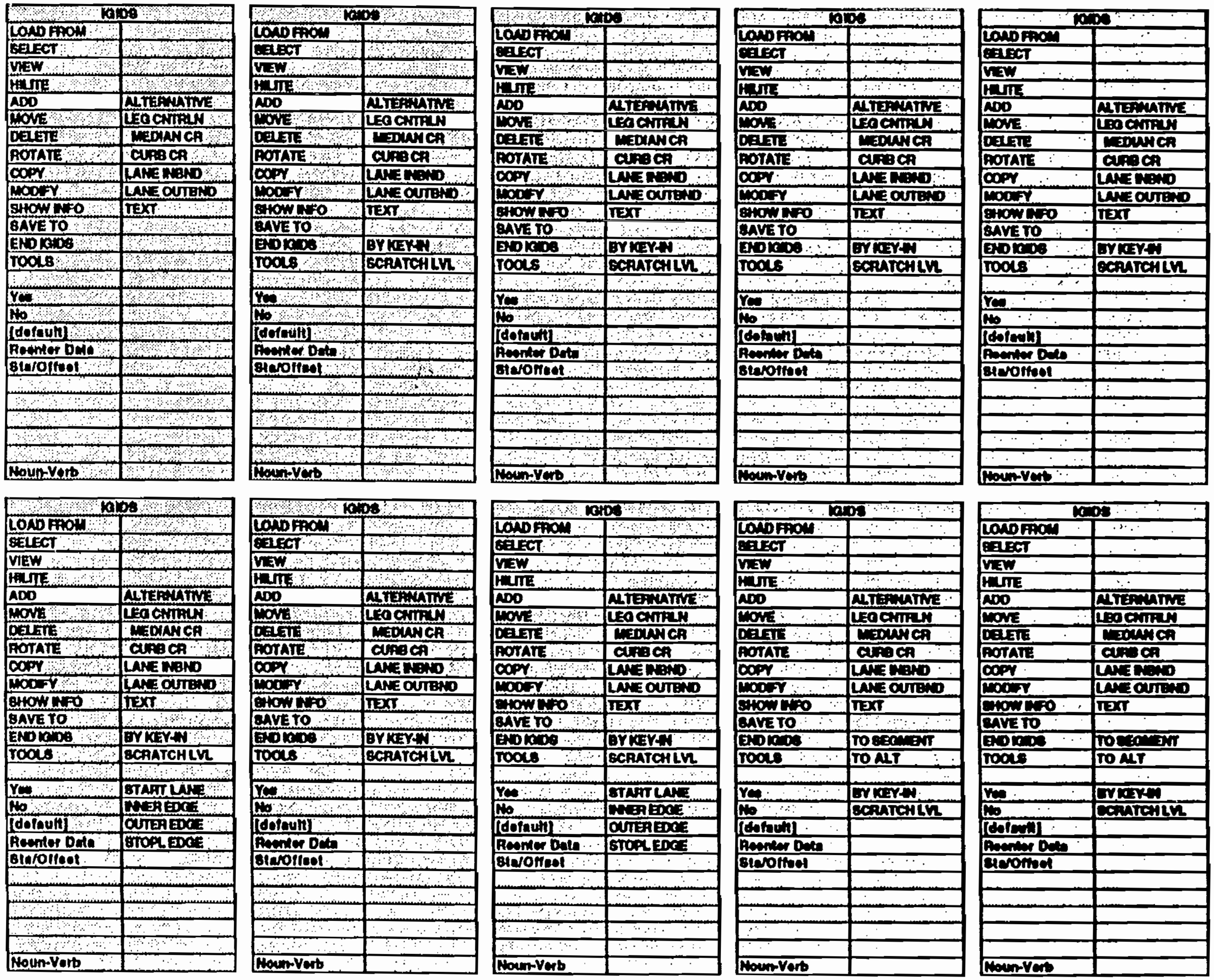


IGIDS	
LOAD FROM	
SELECT	
VIEW	
HILITE	CURRENT ALT
ADD	CURRENT LEG
MOVE	CURRENT LANE
DELETE	CURRENT SEG
ROTATE	CURRENT TEXT
COPY	
MODIFY	
SHOW INFO	
SAVE TO	
END IGIDS	
TOOLS	
Yes	
No	
{default}	
Reenter Data	
Sta/Offset	
Noun-Verb	

IGIDS	
LOAD FROM	
SELECT	
VIEW	
HILITE	CURRENT ALT
ADD	CURRENT LEG
MOVE	CURRENT LANE
DELETE	CURRENT SEG
ROTATE	CURRENT TEXT
COPY	
MODIFY	ALL
SHOW INFO	CENTERLINE
SAVE TO	CURB RETURNS
END IGIDS	MEDIAN CR
TOOLS	CURB CR
	INBND LANES
Yes	OUTBND LANES
No	
{default}	
Reenter Data	
Sta/Offset	
Noun-Verb	

IGIDS	
LOAD FROM	
SELECT	
VIEW	
HILITE	CURRENT ALT
ADD	CURRENT LEG
MOVE	CURRENT LANE
DELETE	CURRENT SEG
ROTATE	CURRENT TEXT
COPY	
MODIFY	ALL
SHOW INFO	INNER EDGE
SAVE TO	OUTER EDGE
END IGIDS	STOP LINE
TOOLS	
Yes	
No	
{default}	
Reenter Data	
Sta/Offset	
Noun-Verb	

Figure B.4 IGIDS Command Menus - HILITE commands



IGIDS	
LOAD FROM	
SELECT	
VIEW	
HILITE	
ADD	ALTERNATIVE
MOVE	LEG CNTRLN
DELETE	MEDIAN CR
ROTATE	CURB CR
COPY	LANE INBND
MODIFY	LANE OUTBND
SHOW INFO	TEXT
SAVE TO	
END IGIDS	
TOOLS	
Yes	
No	
[default]	
Reenter Data	
Sta/Offset	
Noun-Verb	

IGIDS	
LOAD FROM	
SELECT	
VIEW	
HILITE	
ADD	ALTERNATIVE
MOVE	LEG CNTRLN
DELETE	MEDIAN CR
ROTATE	CURB CR
COPY	LANE INBND
MODIFY	LANE OUTBND
SHOW INFO	TEXT
SAVE TO	
END IGIDS	BY KEY-IN
TOOLS	SCRATCH LVL
Yes	
No	
[default]	
Reenter Data	
Sta/Offset	
Noun-Verb	

IGIDS	
LOAD FROM	
SELECT	
VIEW	
HILITE	
ADD	ALTERNATIVE
MOVE	LEG CNTRLN
DELETE	MEDIAN CR
ROTATE	CURB CR
COPY	LANE INBND
MODIFY	LANE OUTBND
SHOW INFO	TEXT
SAVE TO	
END IGIDS	BY KEY-IN
TOOLS	SCRATCH LVL
Yes	
No	
[default]	
Reenter Data	
Sta/Offset	
Noun-Verb	

IGIDS	
LOAD FROM	
SELECT	
VIEW	
HILITE	
ADD	ALTERNATIVE
MOVE	LEG CNTRLN
DELETE	MEDIAN CR
ROTATE	CURB CR
COPY	LANE INBND
MODIFY	LANE OUTBND
SHOW INFO	TEXT
SAVE TO	
END IGIDS	BY KEY-IN
TOOLS	SCRATCH LVL
Yes	
No	
[default]	
Reenter Data	
Sta/Offset	
Noun-Verb	

IGIDS	
LOAD FROM	
SELECT	
VIEW	
HILITE	
ADD	ALTERNATIVE
MOVE	LEG CNTRLN
DELETE	MEDIAN CR
ROTATE	CURB CR
COPY	LANE INBND
MODIFY	LANE OUTBND
SHOW INFO	TEXT
SAVE TO	
END IGIDS	BY KEY-IN
TOOLS	SCRATCH LVL
Yes	
No	
[default]	
Reenter Data	
Sta/Offset	
Noun-Verb	

IGIDS	
LOAD FROM	
SELECT	
VIEW	
HILITE	
ADD	ALTERNATIVE
MOVE	LEG CNTRLN
DELETE	MEDIAN CR
ROTATE	CURB CR
COPY	LANE INBND
MODIFY	LANE OUTBND
SHOW INFO	TEXT
SAVE TO	
END IGIDS	BY KEY-IN
TOOLS	SCRATCH LVL
Yes	START LANE
No	INNER EDGE
[default]	OUTER EDGE
Reenter Data	STOPL EDGE
Sta/Offset	
Noun-Verb	

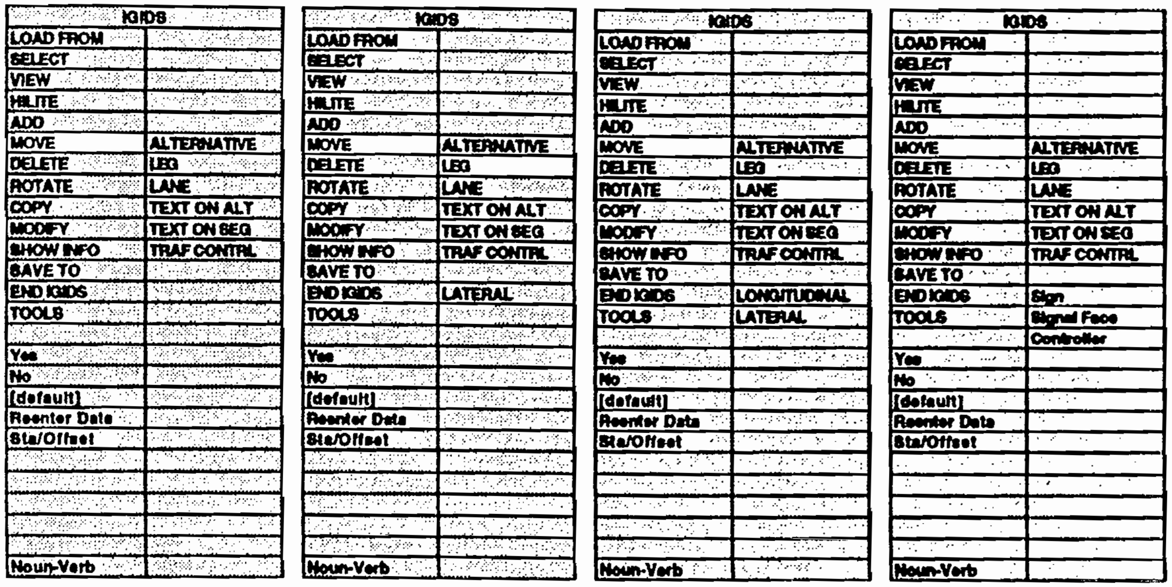
IGIDS	
LOAD FROM	
SELECT	
VIEW	
HILITE	
ADD	ALTERNATIVE
MOVE	LEG CNTRLN
DELETE	MEDIAN CR
ROTATE	CURB CR
COPY	LANE INBND
MODIFY	LANE OUTBND
SHOW INFO	TEXT
SAVE TO	
END IGIDS	BY KEY-IN
TOOLS	SCRATCH LVL
Yes	
No	
[default]	
Reenter Data	
Sta/Offset	
Noun-Verb	

IGIDS	
LOAD FROM	
SELECT	
VIEW	
HILITE	
ADD	ALTERNATIVE
MOVE	LEG CNTRLN
DELETE	MEDIAN CR
ROTATE	CURB CR
COPY	LANE INBND
MODIFY	LANE OUTBND
SHOW INFO	TEXT
SAVE TO	
END IGIDS	BY KEY-IN
TOOLS	SCRATCH LVL
Yes	START LANE
No	INNER EDGE
[default]	OUTER EDGE
Reenter Data	STOPL EDGE
Sta/Offset	
Noun-Verb	

IGIDS	
LOAD FROM	
SELECT	
VIEW	
HILITE	
ADD	ALTERNATIVE
MOVE	LEG CNTRLN
DELETE	MEDIAN CR
ROTATE	CURB CR
COPY	LANE INBND
MODIFY	LANE OUTBND
SHOW INFO	TEXT
SAVE TO	
END IGIDS	TO SEGMENT
TOOLS	TO ALT
Yes	BY KEY-IN
No	SCRATCH LVL
[default]	
Reenter Data	
Sta/Offset	
Noun-Verb	

IGIDS	
LOAD FROM	
SELECT	
VIEW	
HILITE	
ADD	ALTERNATIVE
MOVE	LEG CNTRLN
DELETE	MEDIAN CR
ROTATE	CURB CR
COPY	LANE INBND
MODIFY	LANE OUTBND
SHOW INFO	TEXT
SAVE TO	
END IGIDS	TO SEGMENT
TOOLS	TO ALT
Yes	BY KEY-IN
No	SCRATCH LVL
[default]	
Reenter Data	
Sta/Offset	
Noun-Verb	

Figure A.5 IGIDS Command Menus - ADD commands



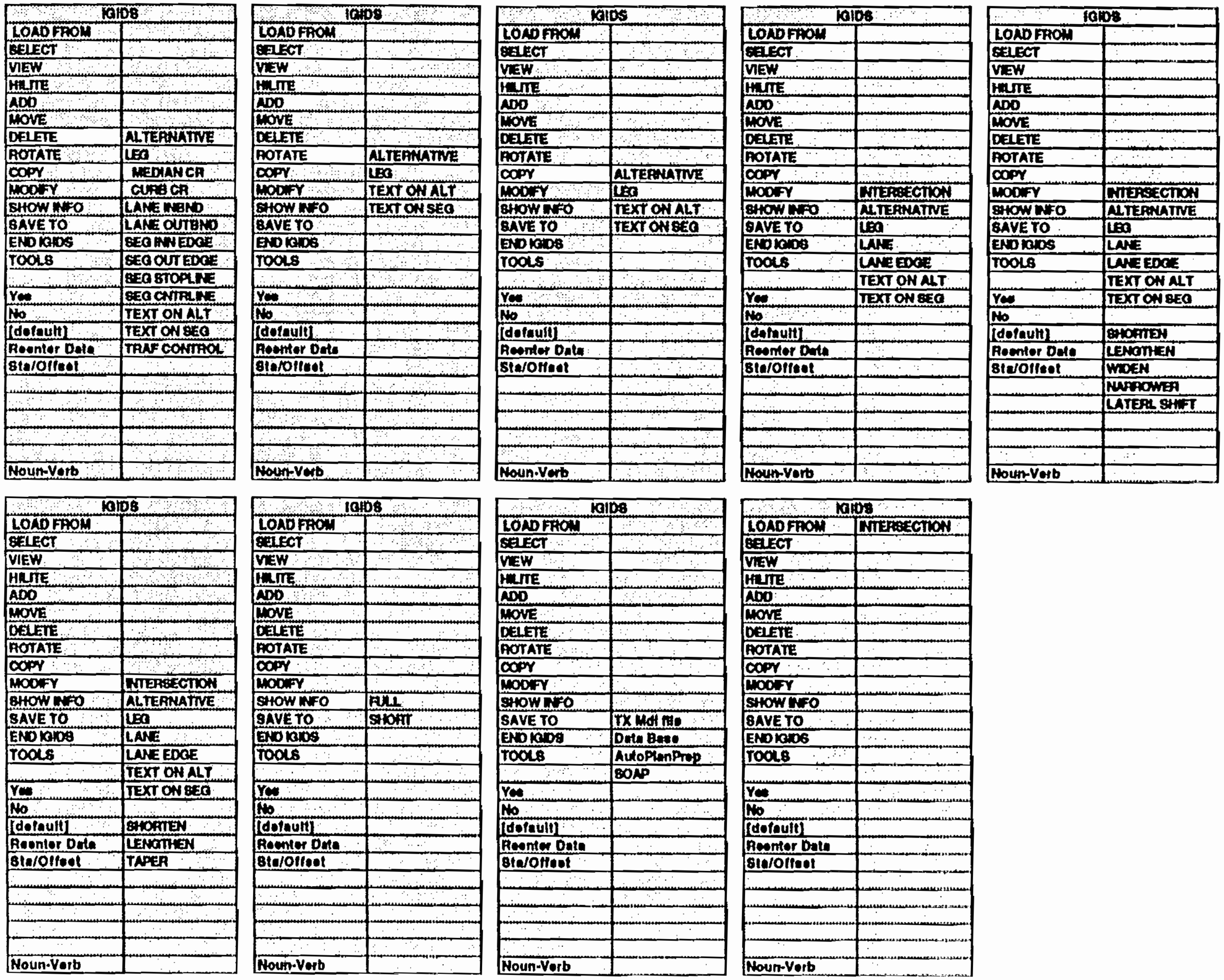
IGIDS	
LOAD FROM	
SELECT	
VIEW	
HILITE	
ADD	
MOVE	ALTERNATIVE
DELETE	LEG
ROTATE	LANE
COPY	TEXT ON ALT
MODIFY	TEXT ON SEG
SHOW INFO	TRAF CONTRL
SAVE TO	
END IGIDS	
TOOLS	
Yes	
No	
[default]	
Reenter Data	
Sta/Offset	
Noun-Verb	

IGIDS	
LOAD FROM	
SELECT	
VIEW	
HILITE	
ADD	
MOVE	ALTERNATIVE
DELETE	LEG
ROTATE	LANE
COPY	TEXT ON ALT
MODIFY	TEXT ON SEG
SHOW INFO	TRAF CONTRL
SAVE TO	
END IGIDS	LATERAL
TOOLS	
Yes	
No	
[default]	
Reenter Data	
Sta/Offset	
Noun-Verb	

IGIDS	
LOAD FROM	
SELECT	
VIEW	
HILITE	
ADD	
MOVE	ALTERNATIVE
DELETE	LEG
ROTATE	LANE
COPY	TEXT ON ALT
MODIFY	TEXT ON SEG
SHOW INFO	TRAF CONTRL
SAVE TO	
END IGIDS	LONGITUDINAL
TOOLS	LATERAL
Yes	
No	
[default]	
Reenter Data	
Sta/Offset	
Noun-Verb	

IGIDS	
LOAD FROM	
SELECT	
VIEW	
HILITE	
ADD	
MOVE	ALTERNATIVE
DELETE	LEG
ROTATE	LANE
COPY	TEXT ON ALT
MODIFY	TEXT ON SEG
SHOW INFO	TRAF CONTRL
SAVE TO	
END IGIDS	Sign
TOOLS	Signal Face
	Controller
Yes	
No	
[default]	
Reenter Data	
Sta/Offset	
Noun-Verb	

Figure B.6 IGIDS Command Menus - MOVE commands



IGIDS	
LOAD FROM	
SELECT	
VIEW	
HILITE	
ADD	
MOVE	
DELETE	ALTERNATIVE
ROTATE	LEG
COPY	MEDIAN CR
MODIFY	CURB CR
SHOW INFO	LANE INBND
SAVE TO	LANE OUTBND
END IGIDS	SEG INN EDGE
TOOLS	SEG OUT EDGE
	SEG STOPLINE
Yes	SEG CNTRLINE
No	TEXT ON ALT
[default]	TEXT ON SEG
Reenter Data	TRAF CONTROL
Sta/Offset	
Noun-Verb	

IGIDS	
LOAD FROM	
SELECT	
VIEW	
HILITE	
ADD	
MOVE	
DELETE	
ROTATE	ALTERNATIVE
COPY	LEG
MODIFY	TEXT ON ALT
SHOW INFO	TEXT ON SEG
SAVE TO	
END IGIDS	
TOOLS	
Yes	
No	
[default]	
Reenter Data	
Sta/Offset	
Noun-Verb	

IGIDS	
LOAD FROM	
SELECT	
VIEW	
HILITE	
ADD	
MOVE	
DELETE	
ROTATE	
COPY	ALTERNATIVE
MODIFY	LEG
SHOW INFO	TEXT ON ALT
SAVE TO	TEXT ON SEG
END IGIDS	
TOOLS	
Yes	
No	
[default]	
Reenter Data	
Sta/Offset	
Noun-Verb	

IGIDS	
LOAD FROM	
SELECT	
VIEW	
HILITE	
ADD	
MOVE	
DELETE	
ROTATE	
COPY	
MODIFY	INTERSECTION
SHOW INFO	ALTERNATIVE
SAVE TO	LEG
END IGIDS	LANE
TOOLS	LANE EDGE
	TEXT ON ALT
Yes	TEXT ON SEG
No	
[default]	
Reenter Data	
Sta/Offset	
Noun-Verb	

IGIDS	
LOAD FROM	
SELECT	
VIEW	
HILITE	
ADD	
MOVE	
DELETE	
ROTATE	
COPY	
MODIFY	INTERSECTION
SHOW INFO	ALTERNATIVE
SAVE TO	LEG
END IGIDS	LANE
TOOLS	LANE EDGE
	TEXT ON ALT
Yes	TEXT ON SEG
No	
[default]	SHORTEN
Reenter Data	LENGTHEN
Sta/Offset	WIDEN
	NARROWER
	LATERL SHIFT
Noun-Verb	

IGIDS	
LOAD FROM	
SELECT	
VIEW	
HILITE	
ADD	
MOVE	
DELETE	
ROTATE	
COPY	
MODIFY	INTERSECTION
SHOW INFO	ALTERNATIVE
SAVE TO	LEG
END IGIDS	LANE
TOOLS	LANE EDGE
	TEXT ON ALT
Yes	TEXT ON SEG
No	
[default]	SHORTEN
Reenter Data	LENGTHEN
Sta/Offset	TAPER
Noun-Verb	

IGIDS	
LOAD FROM	
SELECT	
VIEW	
HILITE	
ADD	
MOVE	
DELETE	
ROTATE	
COPY	
MODIFY	
SHOW INFO	FULL
SAVE TO	SHORT
END IGIDS	
TOOLS	
Yes	
No	
[default]	
Reenter Data	
Sta/Offset	
Noun-Verb	

IGIDS	
LOAD FROM	
SELECT	
VIEW	
HILITE	
ADD	
MOVE	
DELETE	
ROTATE	
COPY	
MODIFY	
SHOW INFO	
SAVE TO	TX Mdl file
END IGIDS	Data Base
TOOLS	AutoPlanPrep
	SOAP
Yes	
No	
[default]	
Reenter Data	
Sta/Offset	
Noun-Verb	

IGIDS	
LOAD FROM	INTERSECTION
SELECT	
VIEW	
HILITE	
ADD	
MOVE	
DELETE	
ROTATE	
COPY	
MODIFY	
SHOW INFO	
SAVE TO	
END IGIDS	
TOOLS	
Yes	
No	
[default]	
Reenter Data	
Sta/Offset	
Noun-Verb	

Figure B.7 IGIDS Command Menus - DELETE through END IGIDS commands

IGIDS	
LOAD FROM	TurnTemplate
SELECT	Sight Dist
VIEW	TEXAS Model
HILITE	HighCapMan
ADD	Traffic
MOVE	
DELETE	P
ROTATE	Bus
COPY	A-Bus
MODIFY	SU
SHOW INFO	WB-40
SAVE TO	WB-50
END IGIDS	WB-60
TOOLS	WB-62
	RMD
Yes	WB-114
No	WB-96
[default]	PT
Reenter Data	PB
Sta/Offset	MH
	Del Graphics
Noun-Verb	

IGIDS	
LOAD FROM	TurnTemplate
SELECT	Sight Dist
VIEW	TEXAS Model
HILITE	HighCapMan
ADD	Traffic
MOVE	
DELETE	Stopped
ROTATE	Yield
COPY	No Control
MODIFY	Del Graphics
SHOW INFO	
SAVE TO	
END IGIDS	
TOOLS	
Yes	
No	
[default]	
Reenter Data	
Sta/Offset	
Noun-Verb	

IGIDS	
LOAD FROM	TurnTemplate
SELECT	Sight Dist
VIEW	TEXAS Model
HILITE	HighCapMan
ADD	Traffic
MOVE	
DELETE	Animation
ROTATE	Load SIMSTA
COPY	Graph TD
MODIFY	Graph ATD
SHOW INFO	Graph OATD
SAVE TO	Graph QD
END IGIDS	Graph AQD
TOOLS	Graph OAQD
	Graph SD
Yes	Graph ASD
No	Graph OASD
[default]	Graph DMPH
Reenter Data	Graph ADMPH
Sta/Offset	Graph OADMPH
	Graph Volume
	Graph Turn %
	Graph Queues
	Graph Probe
	Del Graphics
Noun-Verb	

IGIDS	
LOAD FROM	TurnTemplate
SELECT	Sight Dist
VIEW	TEXAS Model
HILITE	HighCapMan
ADD	Traffic
MOVE	
DELETE	Chapter 9
ROTATE	Del Graphics
COPY	
MODIFY	
SHOW INFO	
SAVE TO	
END IGIDS	
TOOLS	
Yes	
No	
[default]	
Reenter Data	
Sta/Offset	
Noun-Verb	

IGIDS	
LOAD FROM	TurnTemplate
SELECT	Sight Dist
VIEW	TEXAS Model
HILITE	HighCapMan
ADD	Traffic
MOVE	
DELETE	Sign
ROTATE	Signal Face
COPY	Controller
MODIFY	Channelize
SHOW INFO	Volume
SAVE TO	
END IGIDS	Yield
TOOLS	Stop
	DELETE
Yes	MOVE
No	
[default]	
Reenter Data	
Sta/Offset	
Noun-Verb	

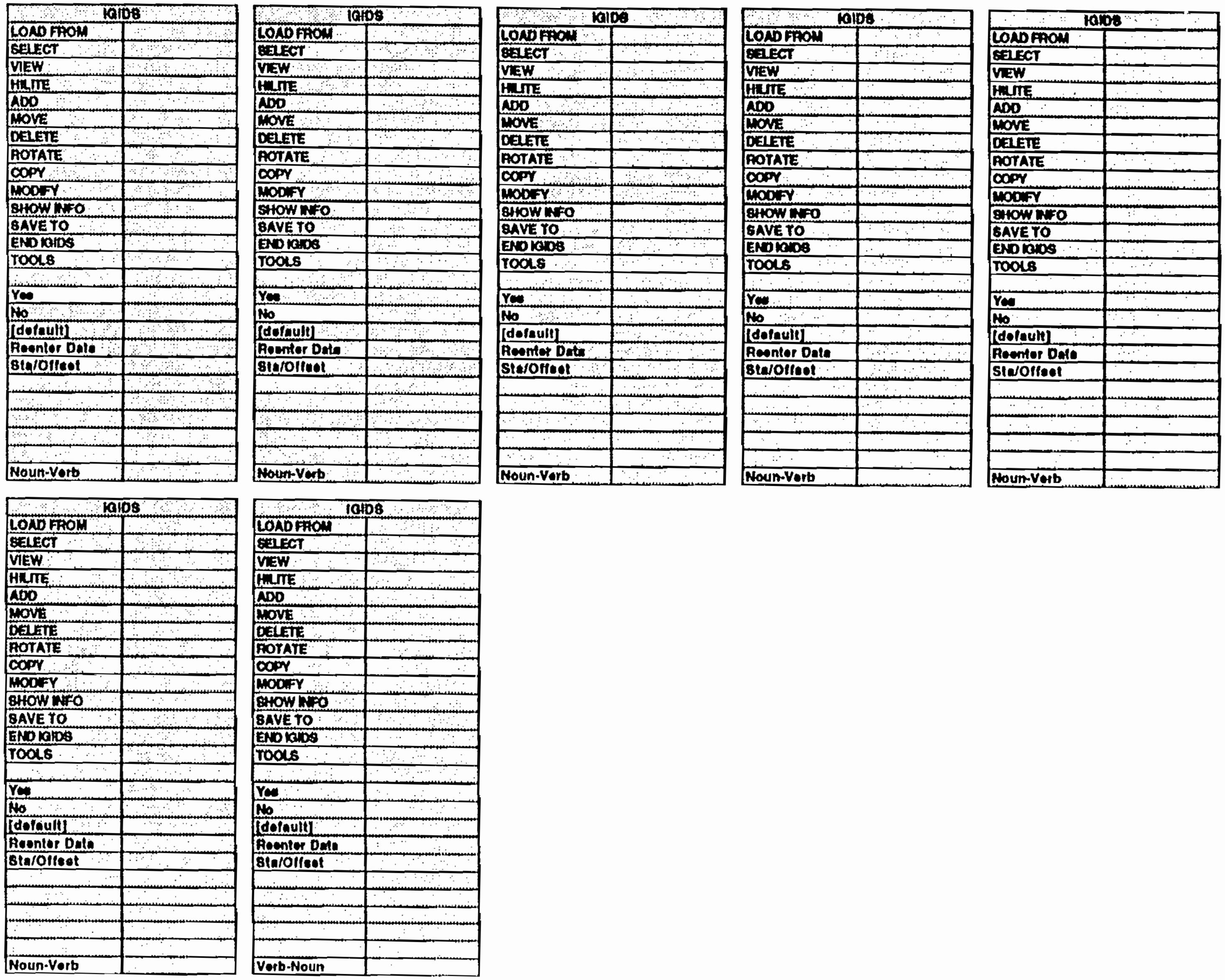
IGIDS	
LOAD FROM	TurnTemplate
SELECT	Sight Dist
VIEW	TEXAS Model
HILITE	HighCapMan
ADD	Traffic
MOVE	
DELETE	Sign
ROTATE	Signal Face
COPY	Controller
MODIFY	Channelize
SHOW INFO	Volume
SAVE TO	
END IGIDS	3 Lens
TOOLS	3 Lens PL
	4 Lens
Yes	DELETE
No	MOVE
[default]	
Reenter Data	
Sta/Offset	
Noun-Verb	

IGIDS	
LOAD FROM	TurnTemplate
SELECT	Sight Dist
VIEW	TEXAS Model
HILITE	HighCapMan
ADD	Traffic
MOVE	
DELETE	Sign
ROTATE	Signal Face
COPY	Controller
MODIFY	Channelize
SHOW INFO	Volume
SAVE TO	
END IGIDS	Pretimed
TOOLS	NEMA
	DELETE
Yes	MOVE
No	PHASING
[default]	TIMING
Reenter Data	
Sta/Offset	
Noun-Verb	

IGIDS	
LOAD FROM	TurnTemplate
SELECT	Sight Dist
VIEW	TEXAS Model
HILITE	HighCapMan
ADD	Traffic
MOVE	
DELETE	Sign
ROTATE	Signal Face
COPY	Controller
MODIFY	Channelize
SHOW INFO	Volume
SAVE TO	
END IGIDS	Left
TOOLS	Straight
	Right
Yes	U-Turn
No	DELETE
[default]	
Reenter Data	
Sta/Offset	
Noun-Verb	

IGIDS	
LOAD FROM	TurnTemplate
SELECT	Sight Dist
VIEW	TEXAS Model
HILITE	HighCapMan
ADD	Traffic
MOVE	
DELETE	Sign
ROTATE	Signal Face
COPY	Controller
MODIFY	Channelize
SHOW INFO	Volume
SAVE TO	
END IGIDS	Percent+VOL
TOOLS	TMC Volume
Yes	
No	
[default]	
Reenter Data	
Sta/Offset	
Noun-Verb	

Figure B.8 IGIDS Command Menus - TOOLS commands



IGIDS	
LOAD FROM	
SELECT	
VIEW	
HILITE	
ADD	
MOVE	
DELETE	
ROTATE	
COPY	
MODIFY	
SHOW INFO	
SAVE TO	
END IGIDS	
TOOLS	
Yes	
No	
[default]	
Reenter Data	
Sta/Offset	
Noun-Verb	

IGIDS	
LOAD FROM	
SELECT	
VIEW	
HILITE	
ADD	
MOVE	
DELETE	
ROTATE	
COPY	
MODIFY	
SHOW INFO	
SAVE TO	
END IGIDS	
TOOLS	
Yes	
No	
[default]	
Reenter Data	
Sta/Offset	
Noun-Verb	

IGIDS	
LOAD FROM	
SELECT	
VIEW	
HILITE	
ADD	
MOVE	
DELETE	
ROTATE	
COPY	
MODIFY	
SHOW INFO	
SAVE TO	
END IGIDS	
TOOLS	
Yes	
No	
[default]	
Reenter Data	
Sta/Offset	
Noun-Verb	

IGIDS	
LOAD FROM	
SELECT	
VIEW	
HILITE	
ADD	
MOVE	
DELETE	
ROTATE	
COPY	
MODIFY	
SHOW INFO	
SAVE TO	
END IGIDS	
TOOLS	
Yes	
No	
[default]	
Reenter Data	
Sta/Offset	
Noun-Verb	

IGIDS	
LOAD FROM	
SELECT	
VIEW	
HILITE	
ADD	
MOVE	
DELETE	
ROTATE	
COPY	
MODIFY	
SHOW INFO	
SAVE TO	
END IGIDS	
TOOLS	
Yes	
No	
[default]	
Reenter Data	
Sta/Offset	
Noun-Verb	

IGIDS	
LOAD FROM	
SELECT	
VIEW	
HILITE	
ADD	
MOVE	
DELETE	
ROTATE	
COPY	
MODIFY	
SHOW INFO	
SAVE TO	
END IGIDS	
TOOLS	
Yes	
No	
[default]	
Reenter Data	
Sta/Offset	
Noun-Verb	

IGIDS	
LOAD FROM	
SELECT	
VIEW	
HILITE	
ADD	
MOVE	
DELETE	
ROTATE	
COPY	
MODIFY	
SHOW INFO	
SAVE TO	
END IGIDS	
TOOLS	
Yes	
No	
[default]	
Reenter Data	
Sta/Offset	
Verb-Noun	

Figure B.9 IGIDS Command Menus - YES through Noun-Verb commands

APPENDIX C

IGIDS Command Descriptions

This appendix contains the IGIDS Diagram Notes, the IGIDS Command Overview, and the IGIDS Command Descriptions. The IGIDS Command Descriptions are in alphabetical order.

IGIDS Diagram Notes

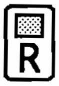
Square cornered boxes enclose a prompt to be presented to the user. Each prompt requires a user input for processing to continue.

Round cornered boxes are for reporting actions that are taken by IGIDS.

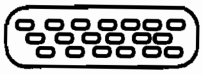
Diamond shaped boxes enclose a question. Some of the corners are labeled with possible answers to the question. The answer determines the path to be taken from this box. Most of these questions have a "yes" or "no' answer.

There are 4 types of user input:

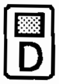
(1) Reset - Executed by pressing the mouse reset button. Represented by this symbol:



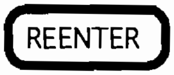
(2) KeyIn - A sequence of key presses that are ended by a pressing the return key. Represented by this symbol:



(3) DataPt - Executed by using the mouse to position the cursor at the desired coordinates and then pressing the mouse data button. Represented by this symbol:



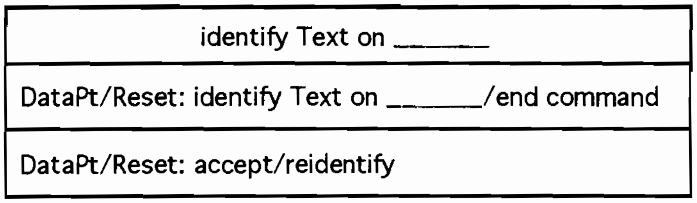
(4) Reenter - An IGIDS command to request that IGIDS re-prompt for the most recently keyed-in data. Represented by this symbol:



Each line leaving a square cornered box is labeled with the authorizing event. These labels are placed upon the lines.

Each line entering an Identify Object box is labeled to indicate the point of entry into the Identify Object operation. These labels are adjacent to the lines.

Diagrams that are for more than one command have underscores (______) substituted for words that are specific to a command. An example is the **MOVE - TEXT** diagram. This is used for both **Primitive Command: MOVE - TEXT ON SEG** and **Primitive Command: MOVE - TEXT ON ALT**. In the Identify Object block for this diagram, the underscore may be replaced by either the word Alternative or the word Seg, as appropriate.


identify Text on ______
DataPt/Reset: identify Text on ______/end command
DataPt/Reset: accept/reidentify

Underscores are also used to replace parts of prompts that may vary, such as the Leg number when prompting for the volume on a Leg.

IGIDS Command Overview

IGIDS Commands

An IGIDS command is a request from the user for IGIDS to take some action. IGIDS commands are initiated by a Click on an IGIDS menu item. There are three types of IGIDS commands as documented in this Appendix:

(1) **Primitive Command:** an IGIDS or Microstation™ command that initiates an action that requires user interaction. When issued during the execution of another IGIDS or Microstation™ command, it cancels any active IGIDS or Microstation™ command.

(2) **Temporary Command:** an IGIDS or Microstation™ command that initiates an action that requires user interaction. When issued during the execution of another IGIDS or Microstation™ command, it temporarily suspends the command in progress. When the temporary command is ended, the suspended command continues from the point where it was suspended.

(3) **Transient Command:** an IGIDS or Microstation™ command that initiates an action that requires no user interaction. When issued during the execution of another IGIDS or Microstation™ command, it does not end the command in progress.

Identify Object subcommand

An operation that is common to many commands is the identification of an existing IGIDS object for processing. When in the **Verb-Noun** mode, the user is prompted to identify the Object of choice by placing a data point (DataPt) near the Object. An Object near the DataPt is then hilited. When in the **Noun-Verb** mode, the selected Object of the appropriate type is hilited automatically. Next, in either mode, the user is prompted to confirm with a DataPt or deny with a Reset that the hilited Object is the correct one. If the hilited Object is not the Object of choice, the user will be prompted to identify another Object with a new DataPt.

If there is only one object of the specified type, it will be assumed to be the object of choice and will automatically become the identified object. The diagram of this operation is shown below in Figure 1(a). When a part of the diagram for an IGIDS command, it will be represented in the simplified form as shown in Figure 1(b).

As Figure 1(b) shows, there are 3 points of entry into and 3 points of exit from the identification diagram. All of these points may not be appropriate for some commands. Only the entry and exit points that are needed will be used in diagrams for specific commands.

The identify Object block in Figure 1(c) is used in **Primitive Command: ROTATE - LEG**. The exit by Reset and entry by DataPt are not needed for this command so are not shown in the diagram. As is typical, the placement of the entry and exit points has been adjusted to suit the particular diagram.

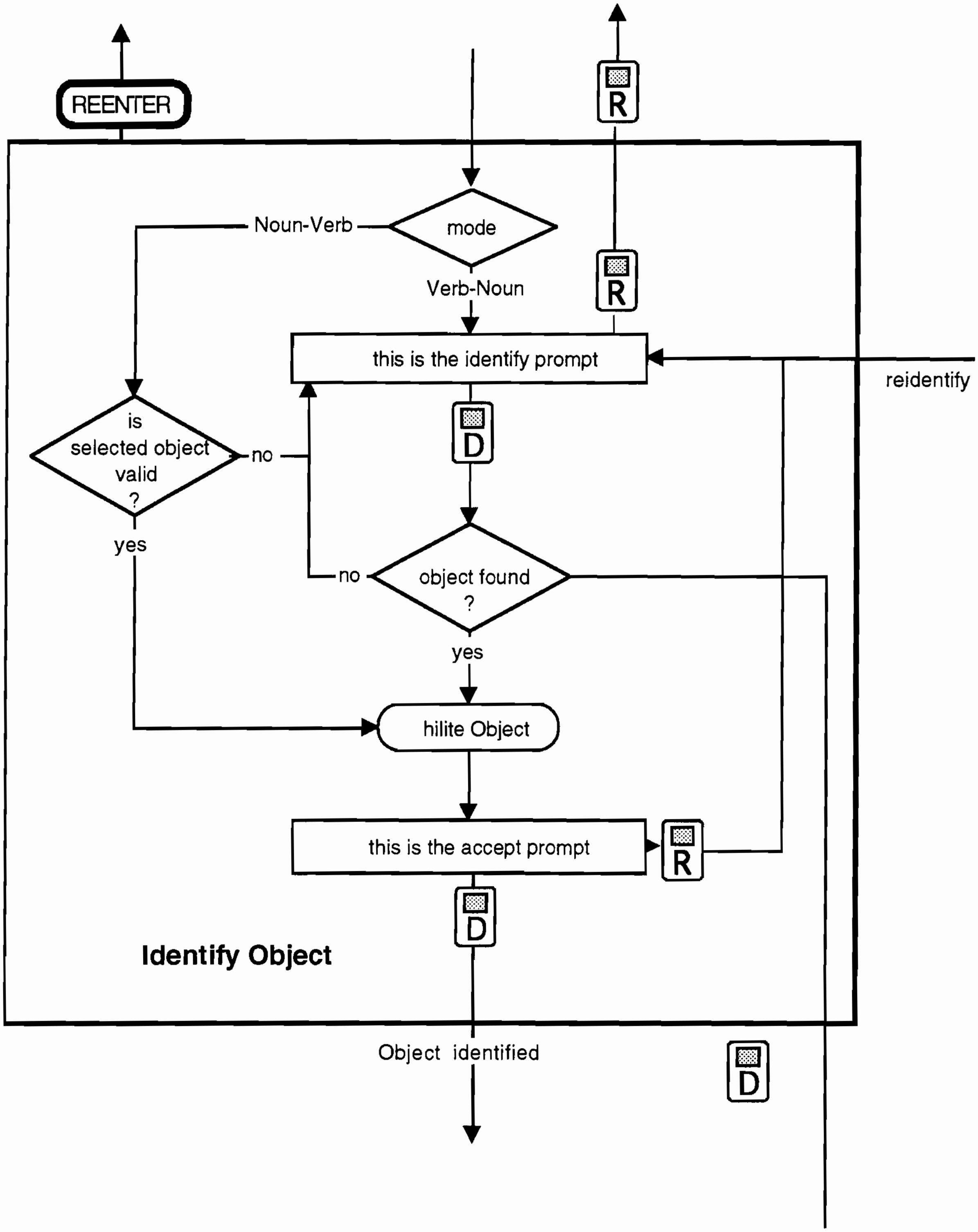


Figure 1(a) Identify object.

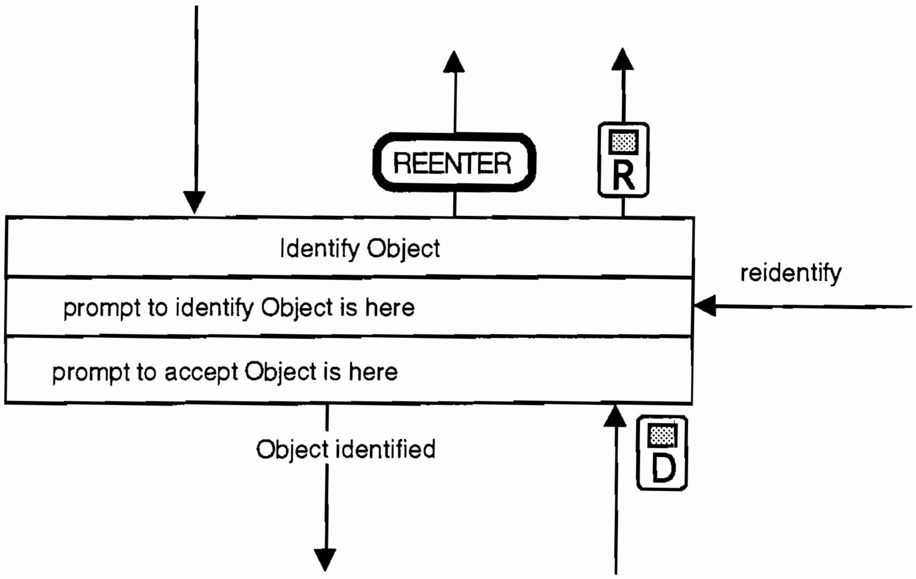

REENTER
R
Identify Object
prompt to identify Object is here
prompt to accept Object is here
reidentify
Object identified
D

Figure 1(b) Identify object as shown in diagrams.

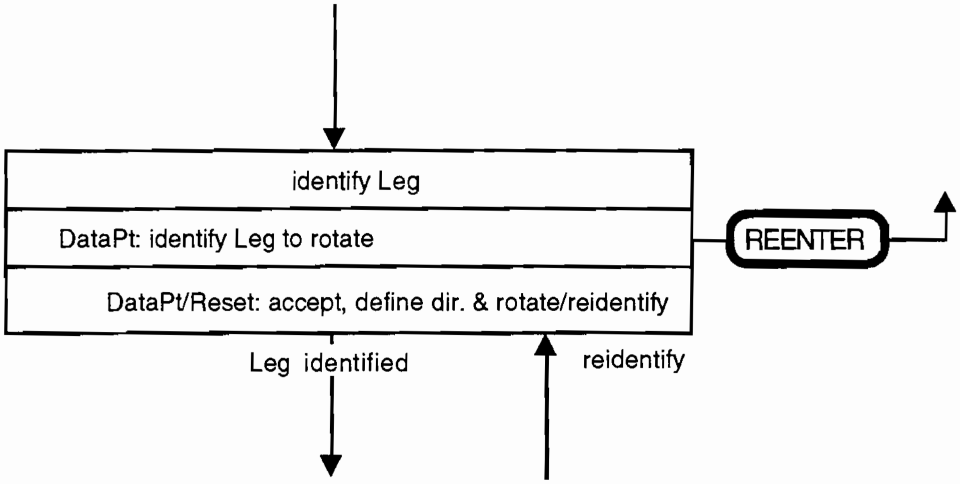

identify Leg
DataPt: identify Leg to rotate
DataPt/Reset: accept, define dir. & rotate/reidentify
REENTER
Leg identified
reidentify

Figure 1(c) Identify object as in the **ROTATE-LEG** diagram.

IGIDS Command Descriptions

The IGIDS Command Descriptions are in alphabetical order. The major groupings are:

(1) **ADD**,

(2) **COPY**,

(3) **DELETE**,

(4) **END IGIDS**,

(5) **HILITE**,

(6) **LOAD FROM**,

(7) **MODIFY**,

(8) **MOVE**,

(9) **No**,

(10) **Noun-Verb**,

(11) **Reenter Data**,

(12) **ROTATE**,

(13) **SAVE TO**

(14) **SELECT**,

(15) **SHOW INFO**,

(16) **STA/OFFSET**,

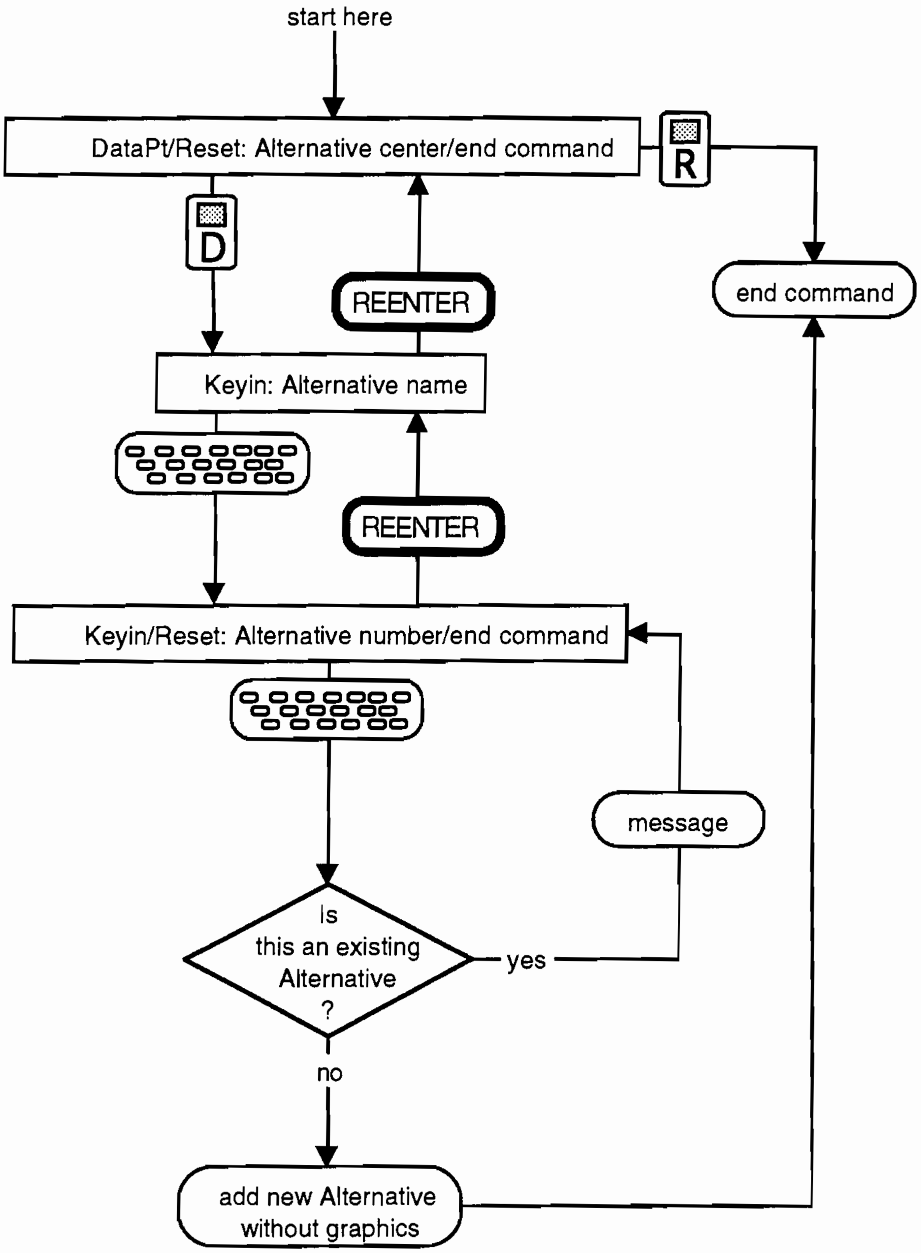
(17) **TOOLS**,

(18) **VIEW**,

(19) **Yes**, and

(20) **[Default]**.

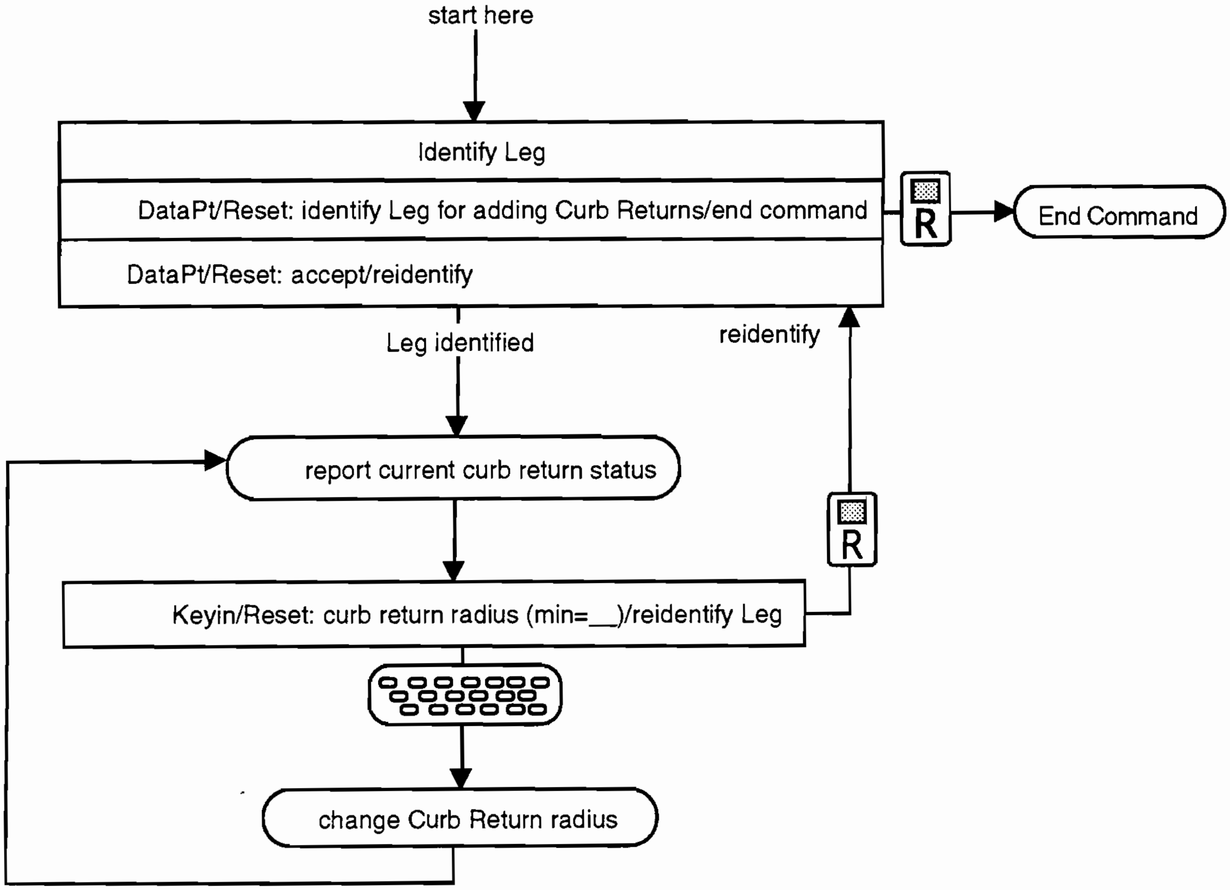
Primitive Command: ADD - ALTERNATIVE



start here
DataPt/Reset: Alternative center/end command
R
D
REENTER
end command
Keyin: Alternative name
REENTER
Keyin/Reset: Alternative number/end command
message
Is this an existing Alternative ?
yes
no
add new Alternative without graphics

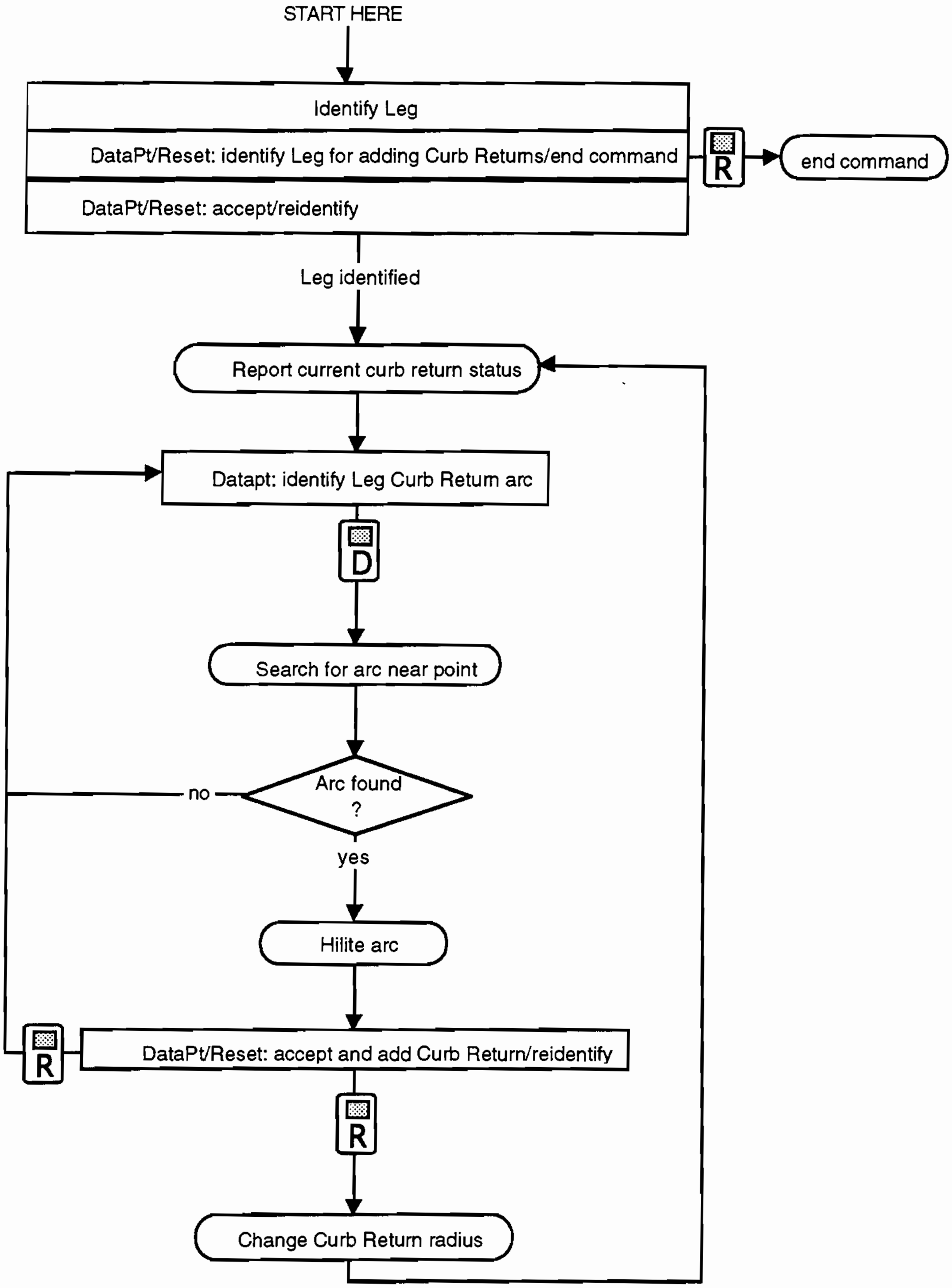
Alternative center coordinates, name and ID number are specified. There is no graphical evidence that the Alternative was added.

Primitive Command: ADD - CURB CR - BY KEYIN


start here
Identify Leg
DataPt/Reset: identify Leg for adding Curb Returns/end command
DataPt/Reset: accept/reidentify
R
End Command
Leg identified
reidentify
report current curb return status
R
Keyin/Reset: curb return radius (min=__)/reidentify Leg
change Curb Return radius

Add or revise the Curb Return. A keyed in radius is used for the Curb Return radius. This is the Curb Return between the Inbound Lanes of the identified Leg and the Outbound Lanes of the adjacent Leg.

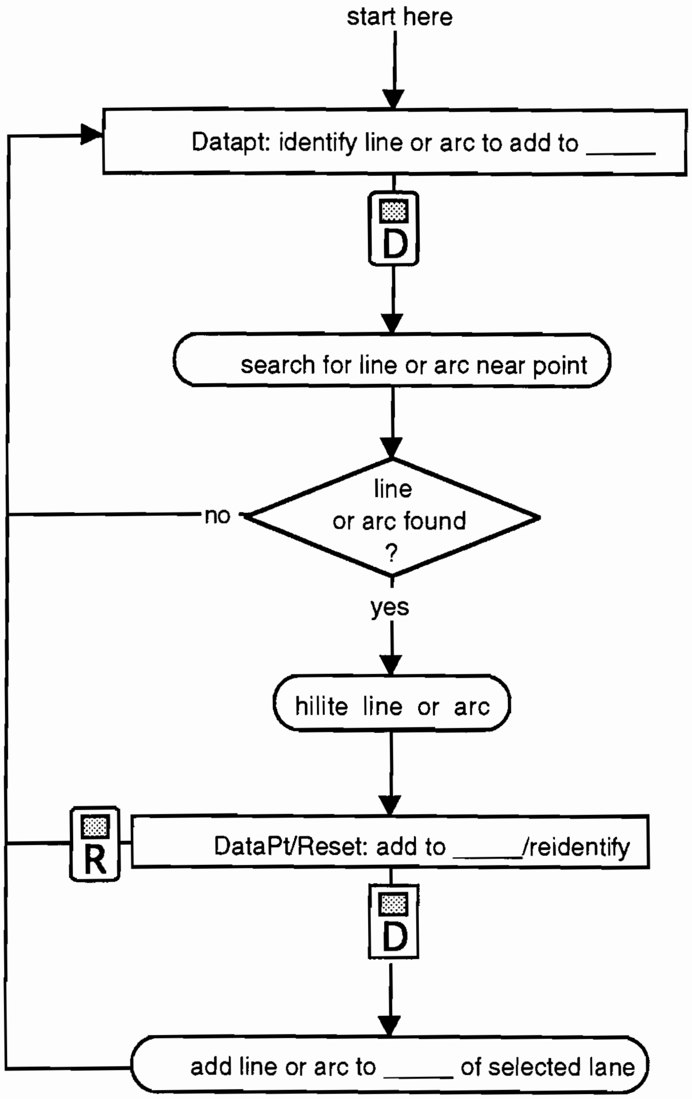
Primitive Command: ADD - CURB CR - SCRATCH LVL



START HERE
Identify Leg
DataPt/Reset: identify Leg for adding Curb Returns/end command
R
end command
DataPt/Reset: accept/reidentify
Leg identified
Report current curb return status
Datapt: identify Leg Curb Return arc
D
Search for arc near point
Arc found ?
no
yes
Hilite arc
R
DataPt/Reset: accept and add Curb Return/reidentify
R
Change Curb Return radius

Add or revise the Curb Return. The radius of an identified arc will be used for the Curb Return radius. The arc may not be an IGIDS arc, but must be on the scratch level or in an attached reference file. This is the Curb Return between the Inbound Lanes of the identified Leg and the Outbound Lanes of the adjacent Leg.

Primitive Command: ADD - LANE INBND - SCRATCH LVL


start here
Datapt: identify line or arc to add to _____
D
search for line or arc near point
line or arc found ?
no
yes
hilite line or arc
R
DataPt/Reset: add to _____/reidentify
D
add line or arc to _____ of selected lane

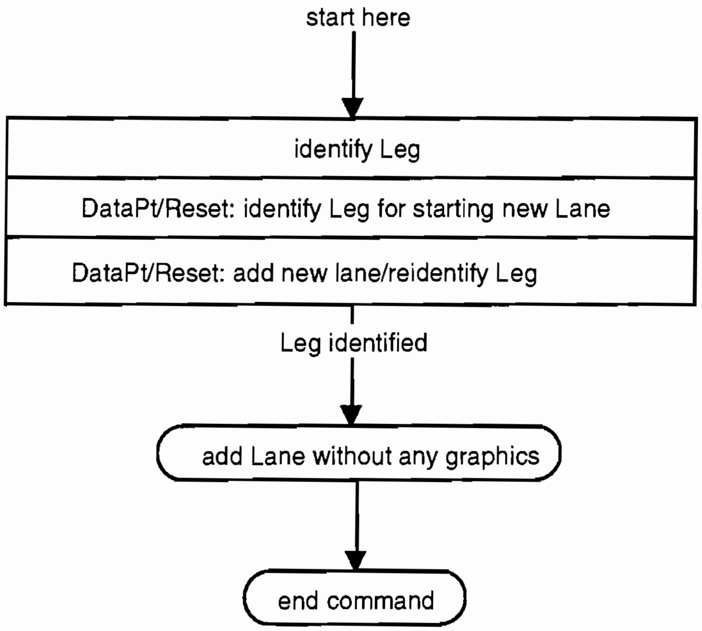
Use this diagram for the three commands that follow:

Primitive Command: ADD - LANE INBND - SCRATCH LVL - INNER EDGE
Add a duplicate of an existing line or arc to the inner edge of the selected Lane. The existing line or arc cannot be an IGIDS Object, but must be on the scratch level or in an attached reference file.

Primitive Command: ADD - LANE INBND - SCRATCH LVL - OUTER EDGE
Add a duplicate of an existing line or arc to the outer edge of the selected Lane. The existing line or arc cannot be an IGIDS Object, but must be on the scratch level or in an attached reference file.

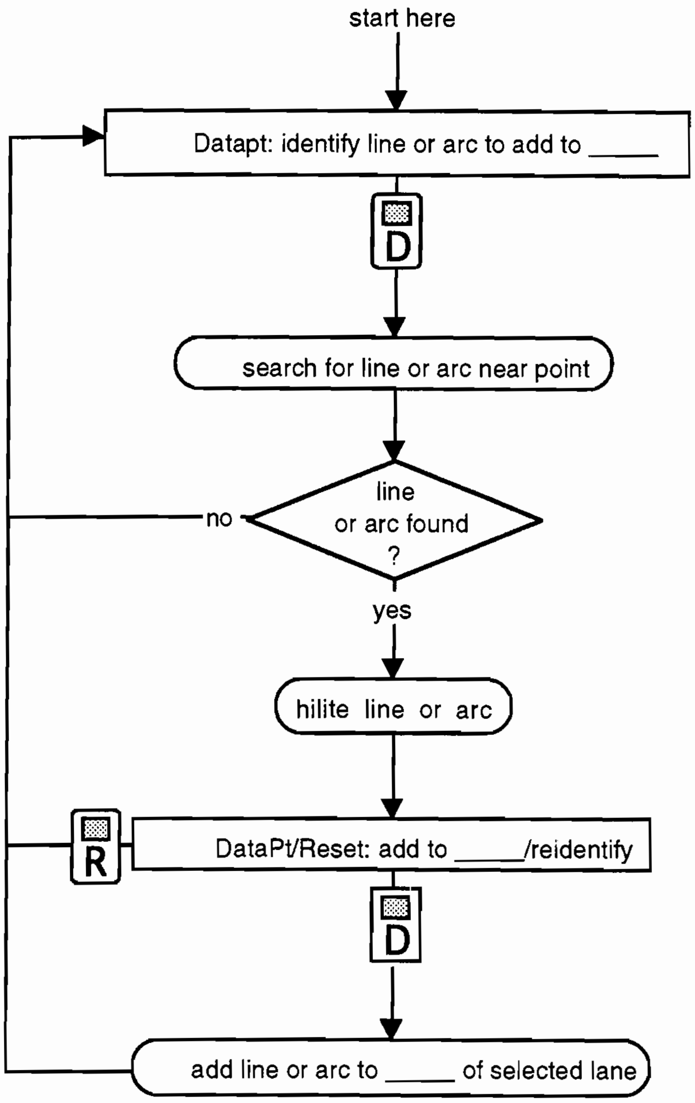
Primitive Command: ADD - LANE INBND - SCRATCH LVL - STOPL EDGE
Add a duplicate of an existing line or arc to the stopline of the selected Lane. The existing line or arc cannot be an IGIDS Object, but must be on the scratch level or in an attached reference file.

Primitive Command: ADD - LANE INBND - SCRATCH LVL - START LANE


start here
identify Leg
DataPt/Reset: identify Leg for starting new Lane
DataPt/Reset: add new lane/reidentify Leg
Leg identified
add Lane without any graphics
end command

Add a new Inbound Lane to the selected Leg. The new Lane is made the selected Lane. There is no graphical evidence that the Lane has been added.

Primitive Command: ADD - LANE OUTBND - SCRATCH LVL


start here
Datapt: identify line or arc to add to _____
D
search for line or arc near point
line or arc found ?
no
yes
hilite line or arc
R
DataPt/Reset: add to _____/reidentify
D
add line or arc to _____ of selected lane

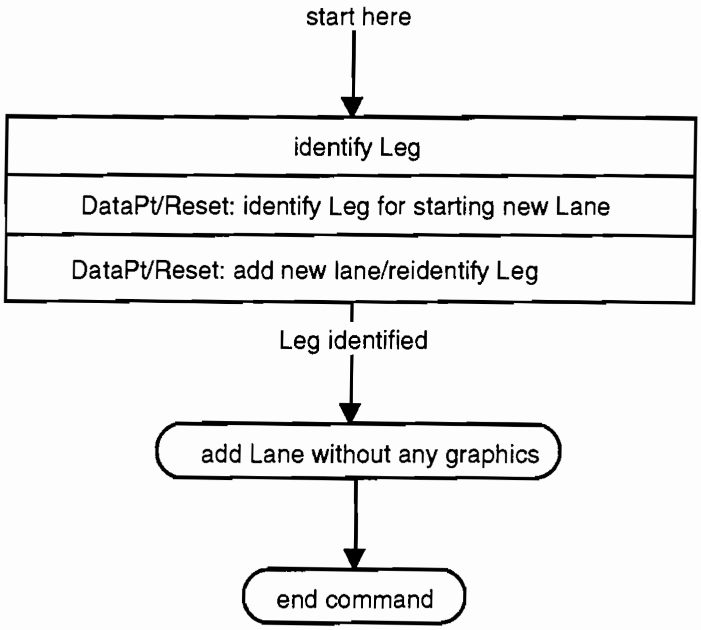
Use this diagram for the three commands that follow:

Primitive Command: ADD - LANE OUTBND - SCRATCH LVL - INNER EDGE
Add a duplicate of an existing line or arc to the inner edge of the selected Lane. The existing line or arc cannot be an IGIDS Object, but must be on the scratch level or in an attached reference file.

Primitive Command: ADD - LANE OUTBND - SCRATCH LVL - OUTER EDGE
Add a duplicate of an existing line or arc to the outer edge of the selected Lane. The existing line or arc cannot be an IGIDS Object, but must be on the scratch level or in an attached reference file.

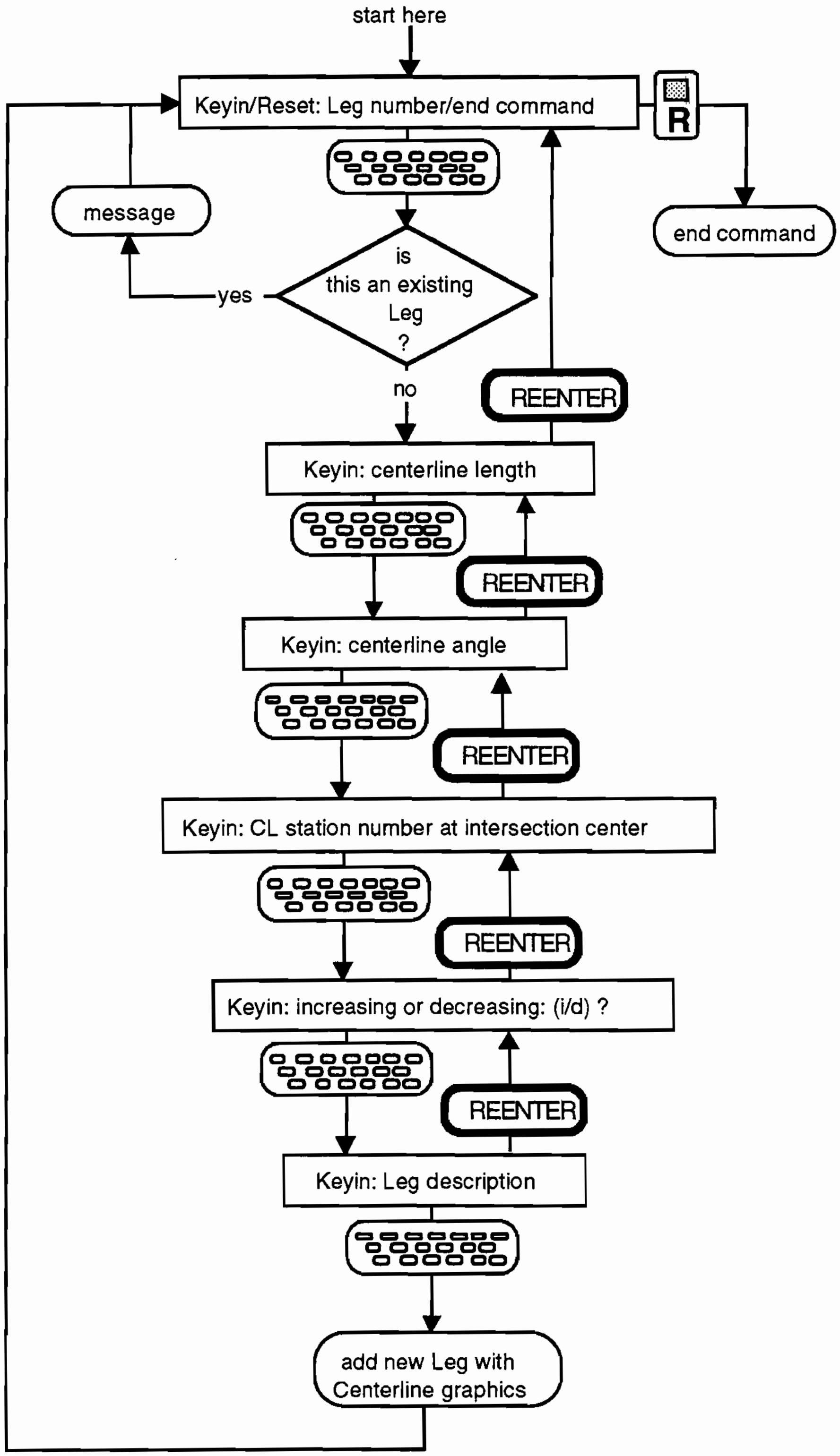
Primitive Command: ADD - LANE OUTBND - SCRATCH LVL - STOPL EDGE
Add a duplicate of an existing line or arc to the stopline of the selected Lane. The existing line or arc cannot be an IGIDS Object, but must be on the scratch level or in an attached reference file.

Primitive Command: ADD - LANE OUTBND - SCRATCH LVL - START LANE


start here
identify Leg
DataPt/Reset: identify Leg for starting new Lane
DataPt/Reset: add new lane/reidentify Leg
Leg identified
add Lane without any graphics
end command

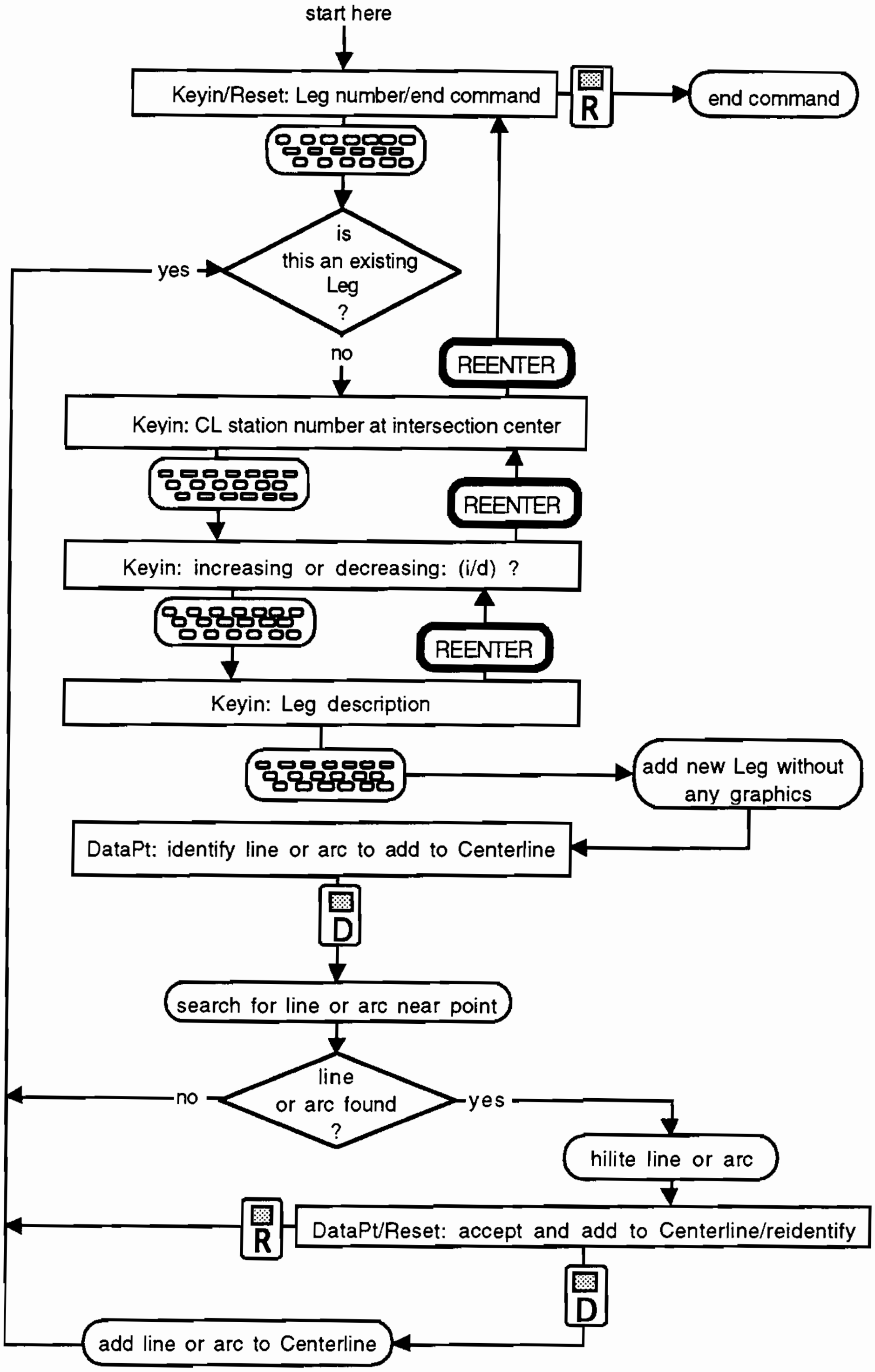
Add a new Outbound Lane to the selected Leg. The new Lane is made the selected Lane. There is no graphical evidence that the Lane has been added.

Primitive Command: ADD - LEG CNTRLN - BY KEY-IN


start here
Keyin/Reset: Leg number/end command
R
end command
message
is this an existing Leg ?
yes
no
REENTER
Keyin: centerline length
REENTER
Keyin: centerline angle
REENTER
Keyin: CL station number at intersection center
REENTER
Keyin: increasing or decreasing: (i/d) ?
REENTER
Keyin: Leg description
add new Leg with Centerline graphics

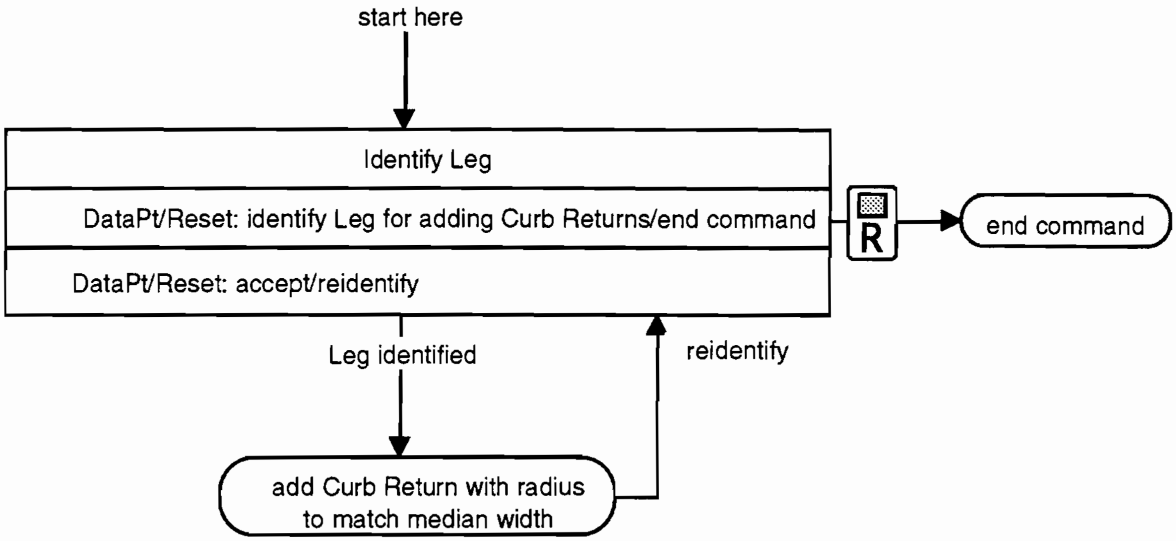
Add a new Leg to the selected Alternative. Leg ID number, centerline length, centerline angle, station number at center of intersection and direction of stationing and Leg description must be specified. The centerline will be one straight Seg and will start at the center of the Alternative.

Primitive Command: ADD - LEG CNTRLN - SCRATCH LVL


start here
Keyin/Reset: Leg number/end command
R
end command
is this an existing Leg ?
yes
no
REENTER
Keyin: CL station number at intersection center
REENTER
Keyin: increasing or decreasing: (i/d) ?
REENTER
Keyin: Leg description
add new Leg without any graphics
DataPt: identify line or arc to add to Centerline
D
search for line or arc near point
line or arc found ?
no
yes
hilite line or arc
R
DataPt/Reset: accept and add to Centerline/reidentify
D
add line or arc to Centerline

For the selected Alternative, add a new Leg and/or Centerline Segs. Leg ID number is specified. If the leg doesn't exist, add a new Leg. For the new Leg, station number at center of intersection and direction of stationing and Leg description must be specified. Duplicates of existing lines or arcs may be added to the new or an existing Leg Centerline. The existing lines or arcs cannot be IGIDS Objects, but must be on the scratch level or in an attached reference file.

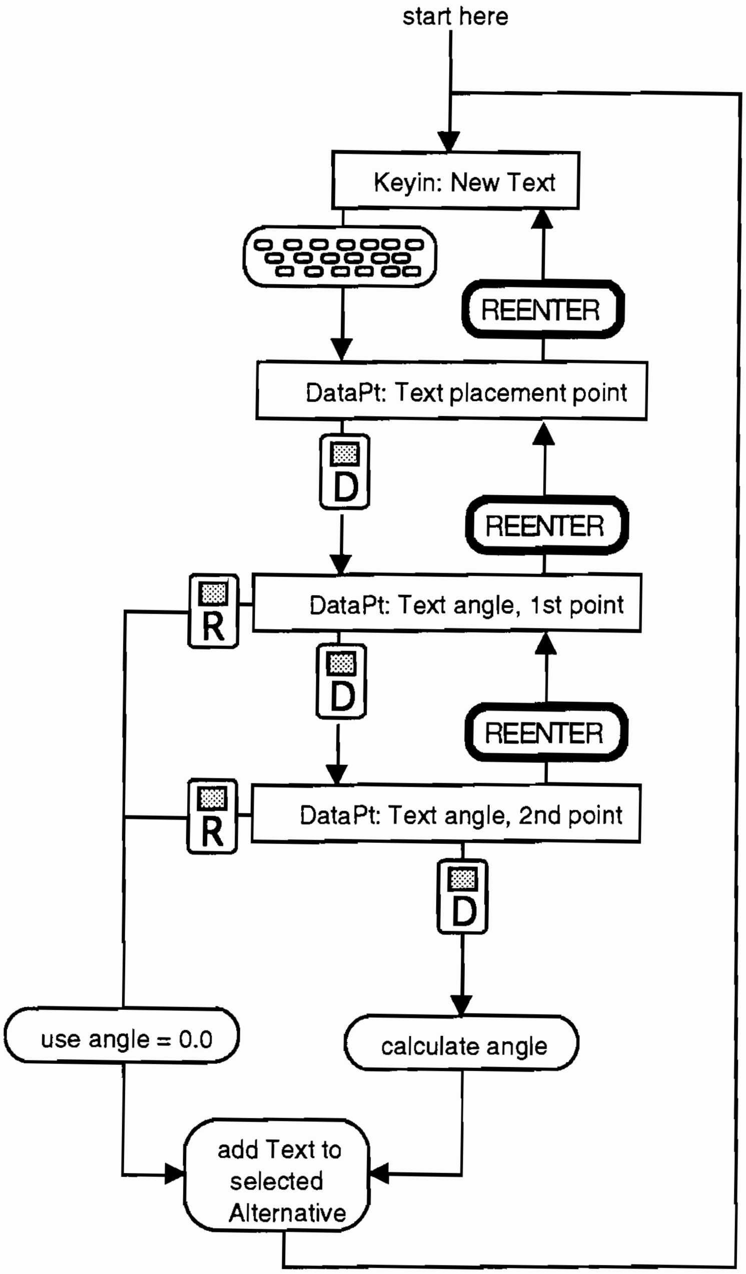
Primitive Command: ADD - MEDIAN CR - BY KEYIN


start here
Identify Leg
DataPt/Reset: identify Leg for adding Curb Returns/end command
R
end command
DataPt/Reset: accept/reidentify
Leg identified
reidentify
add Curb Return with radius to match median width

Add the median Curb Return. The Curb Return radius is automatically set to span the median. This is the Curb Return that closes the median between the Inbound Lanes and Outbound Lanes of the identified Leg.

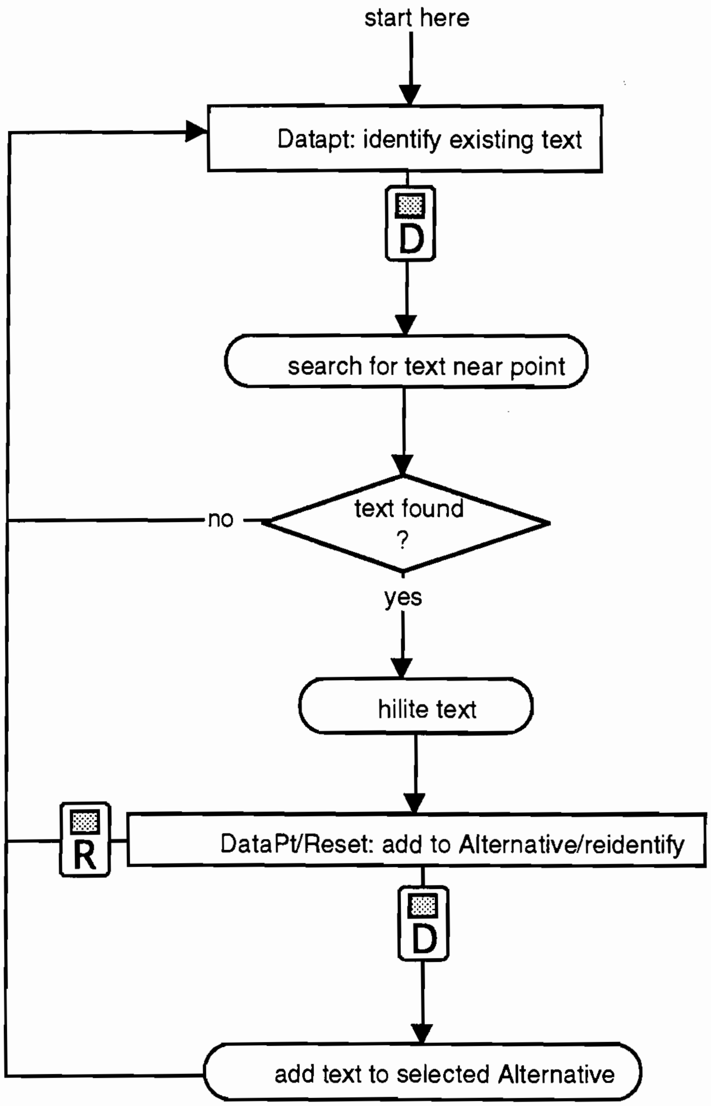
Primitive Command: ADD - MEDIAN CR - SCRATCH LVL
Add the median Curb Return. Not programmed yet.

Primitive Command: ADD - TEXT - TO ALT - BY KEY-IN


start here
Keyin: New Text
REENTER
DataPt: Text placement point
D
REENTER
R
DataPt: Text angle, 1st point
D
REENTER
R
DataPt: Text angle, 2nd point
D
use angle = 0.0
calculate angle
add Text to selected Alternative

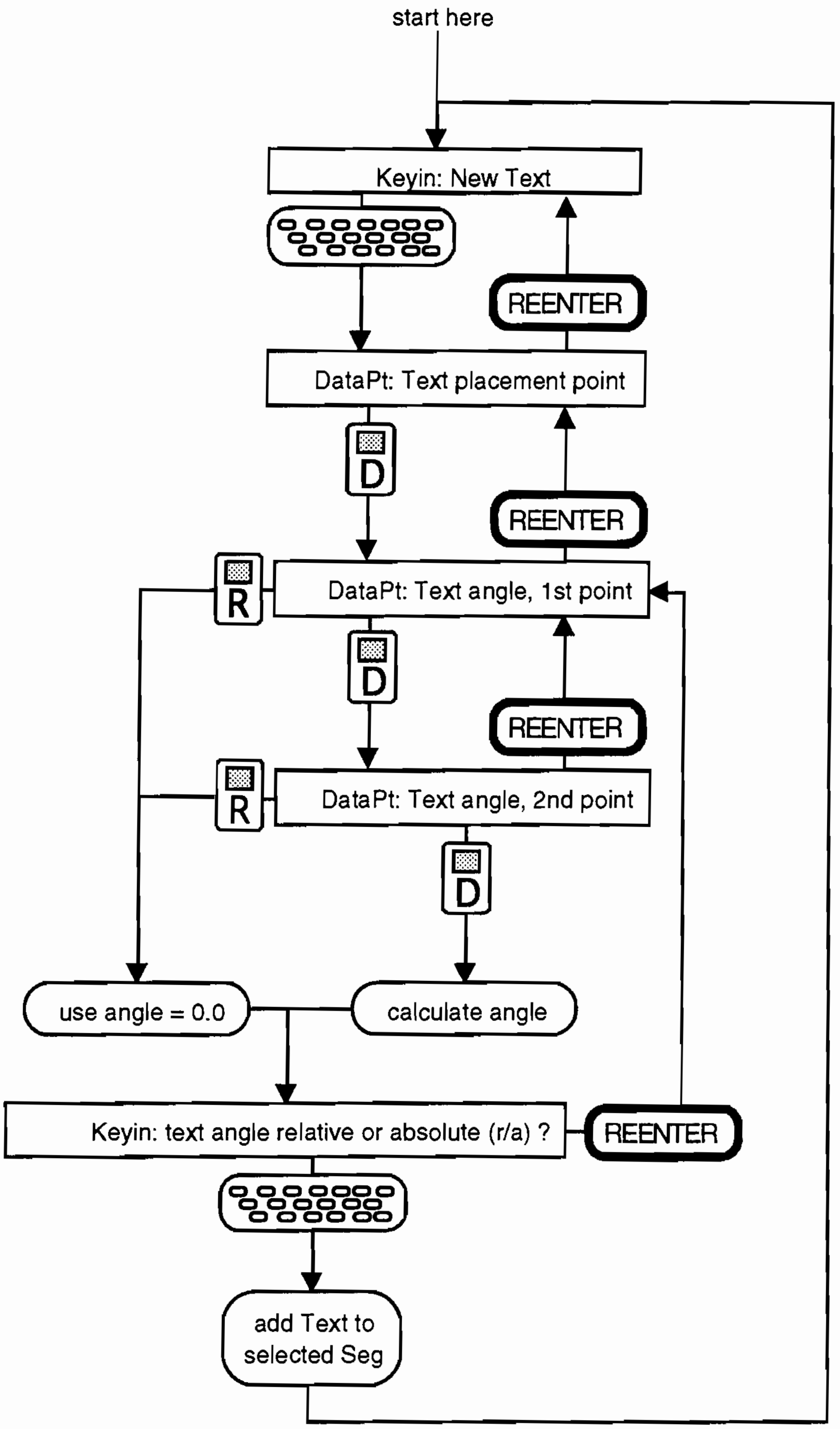
Attach user specified Text to the selected Alternative. The location and absolute angle must be specified by the user. The Graphics Engine's current text size will be used.

Primitive Command: ADD - TEXT - TO ALT - SCRATCH LVL


start here
Datapt: identify existing text
D
search for text near point
text found
?
no
yes
hilite text
R
DataPt/Reset: add to Alternative/reidentify
D
add text to selected Alternative

Attach a copy of existing text to the selected Alternative. The existing text cannot be IGIDS Text, but must be on the scratch level or in an attached reference file. The characteristics of the existing text will be used.

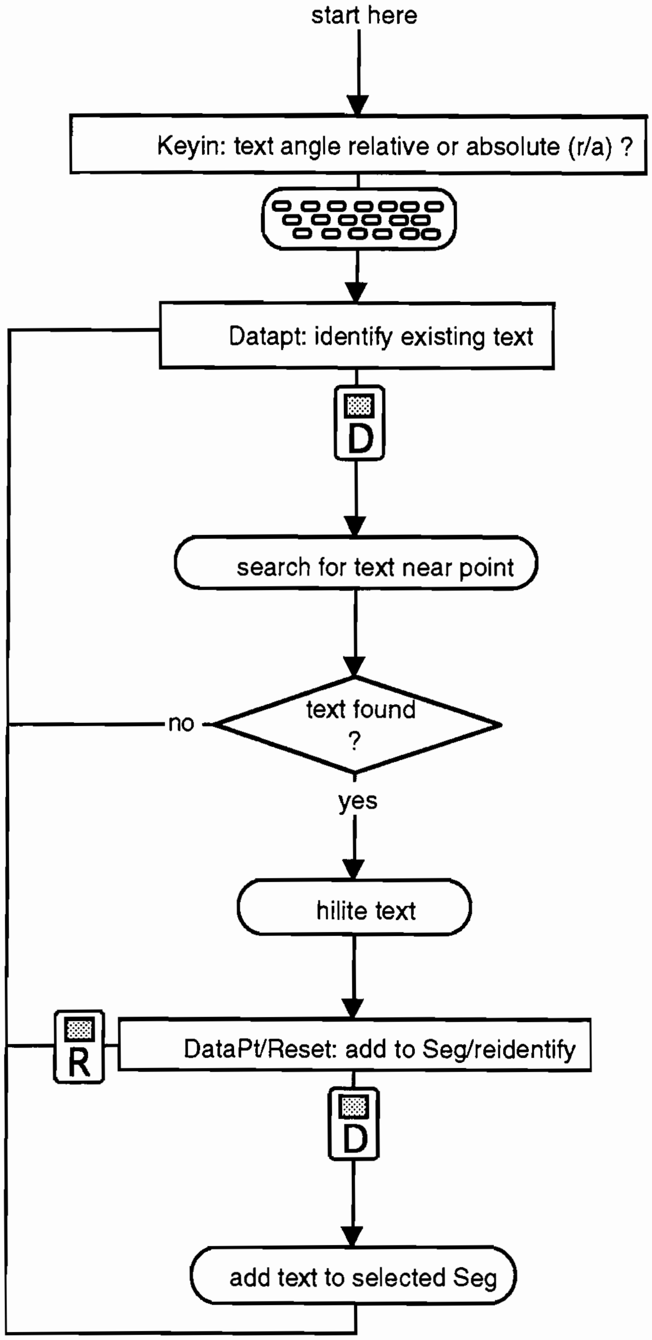
Primitive Command: ADD - TEXT - TO SEG - BY KEY-IN



start here
Keyin: New Text
REENTER
DataPt: Text placement point
D
REENTER
R
DataPt: Text angle, 1st point
D
REENTER
R
DataPt: Text angle, 2nd point
D
use angle = 0.0
calculate angle
Keyin: text angle relative or absolute (r/a) ?
REENTER
add Text to selected Seg

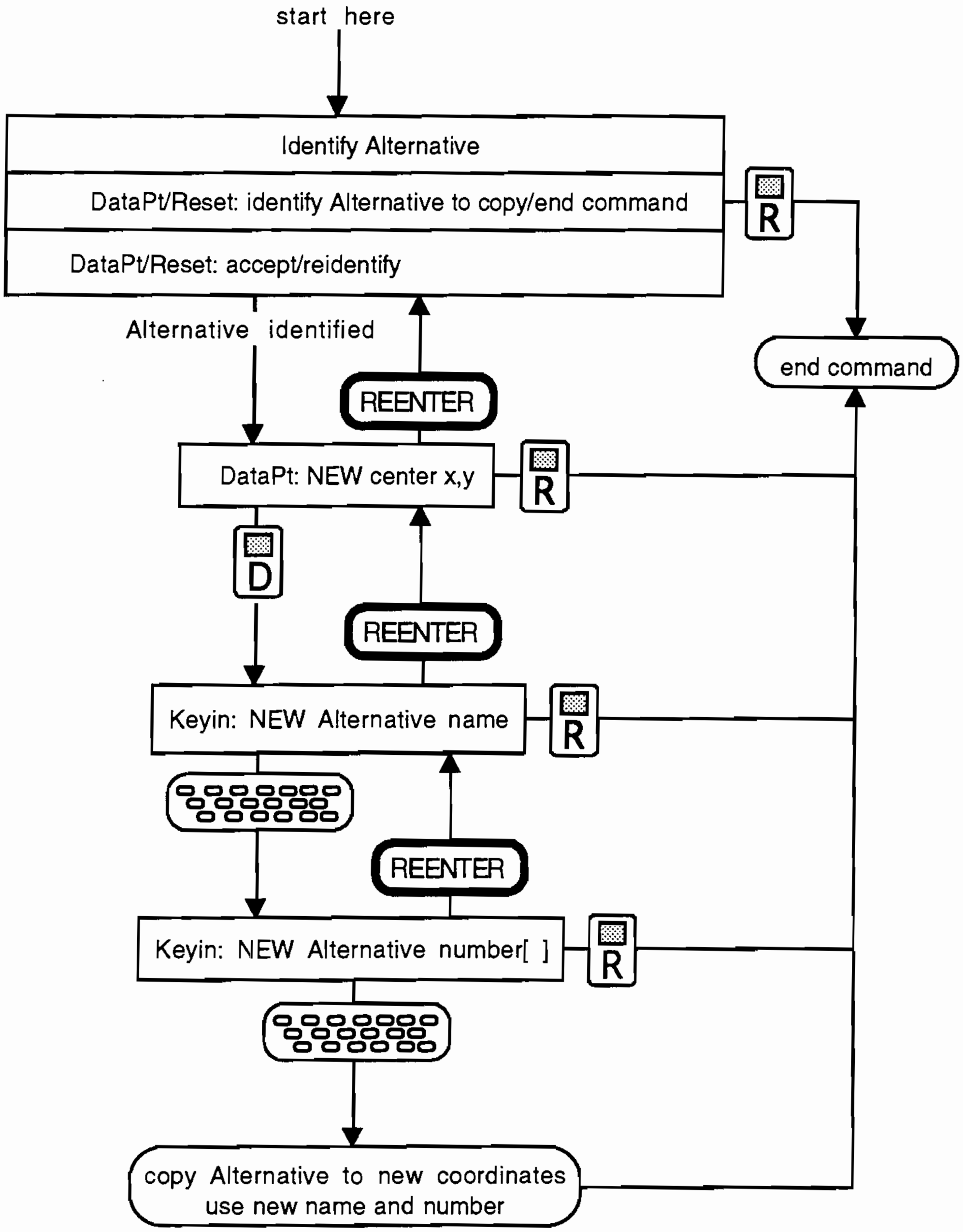
Attach user specified Text to the selected Seg. The location and rotation angle must be specified. The angle may be either absolute or relative. The Graphics Engine's current text size will be used.

Primitive Command: ADD - TEXT - TO SEG - SCRATCH LVL


start here
Keyin: text angle relative or absolute (r/a) ?
Datapt: identify existing text
D
search for text near point
text found
?
no
yes
hilite text
R
DataPt/Reset: add to Seg/reidentify
D
add text to selected Seg

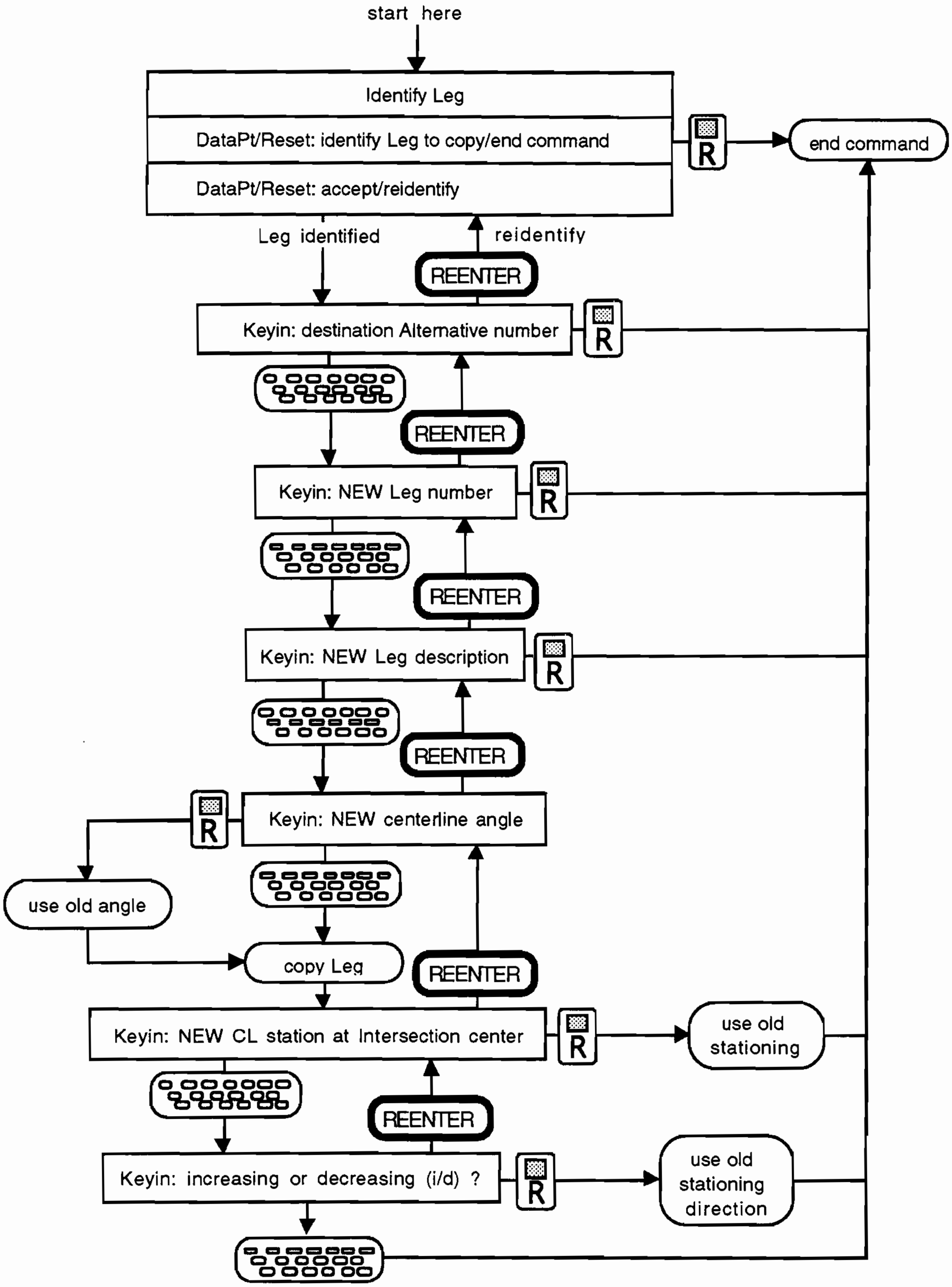
Attach a copy of existing text to the selected Seg. The existing text cannot be IGIDS Text, but must be on the scratch level or in an attached reference file. The location and rotation angle must be specified. The angle may be either absolute or relative.

Primitive Command: COPY- ALTERNATIVE


start here
Identify Alternative
DataPt/Reset: identify Alternative to copy/end command
R
DataPt/Reset: accept/reidentify
Alternative identified
end command
REENTER
DataPt: NEW center x,y
R
D
REENTER
Keyin: NEW Alternative name
R
REENTER
Keyin: NEW Alternative number[ ]
R
copy Alternative to new coordinates
use new name and number

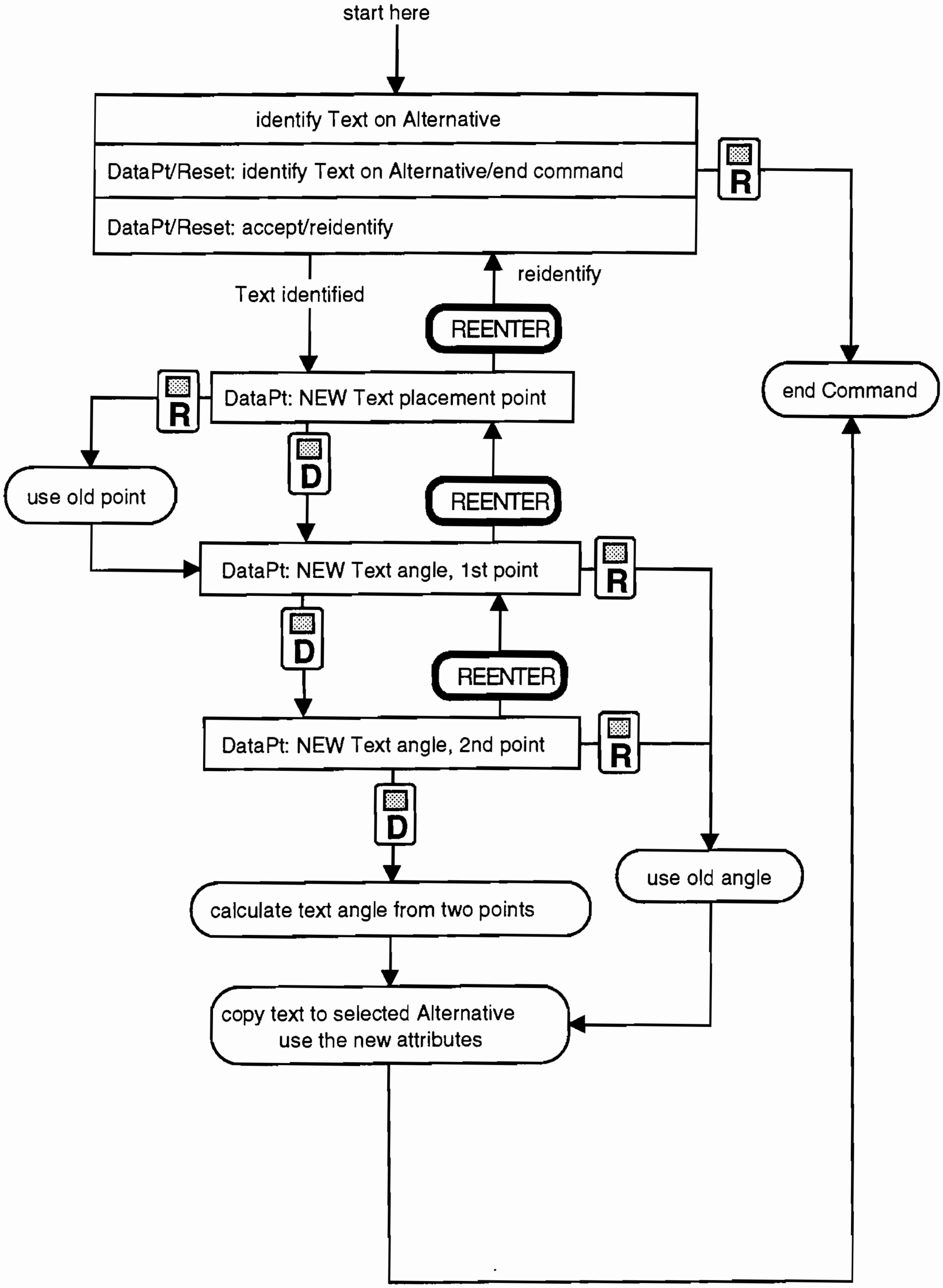
Make a copy of an existing Alternative. New center point coordinates, name and ID number may be assigned to the new Alternative.

Primitive Command: COPY- LEG


start here
Identify Leg
DataPt/Reset: identify Leg to copy/end command
DataPt/Reset: accept/reidentify
R
end command
Leg identified
reidentify
REENTER
Keyin: destination Alternative number
R
REENTER
Keyin: NEW Leg number
R
REENTER
Keyin: NEW Leg description
R
REENTER
R
Keyin: NEW centerline angle
use old angle
copy Leg
REENTER
Keyin: NEW CL station at Intersection center
R
use old stationing
REENTER
Keyin: increasing or decreasing (i/d) ?
R
use old stationing direction

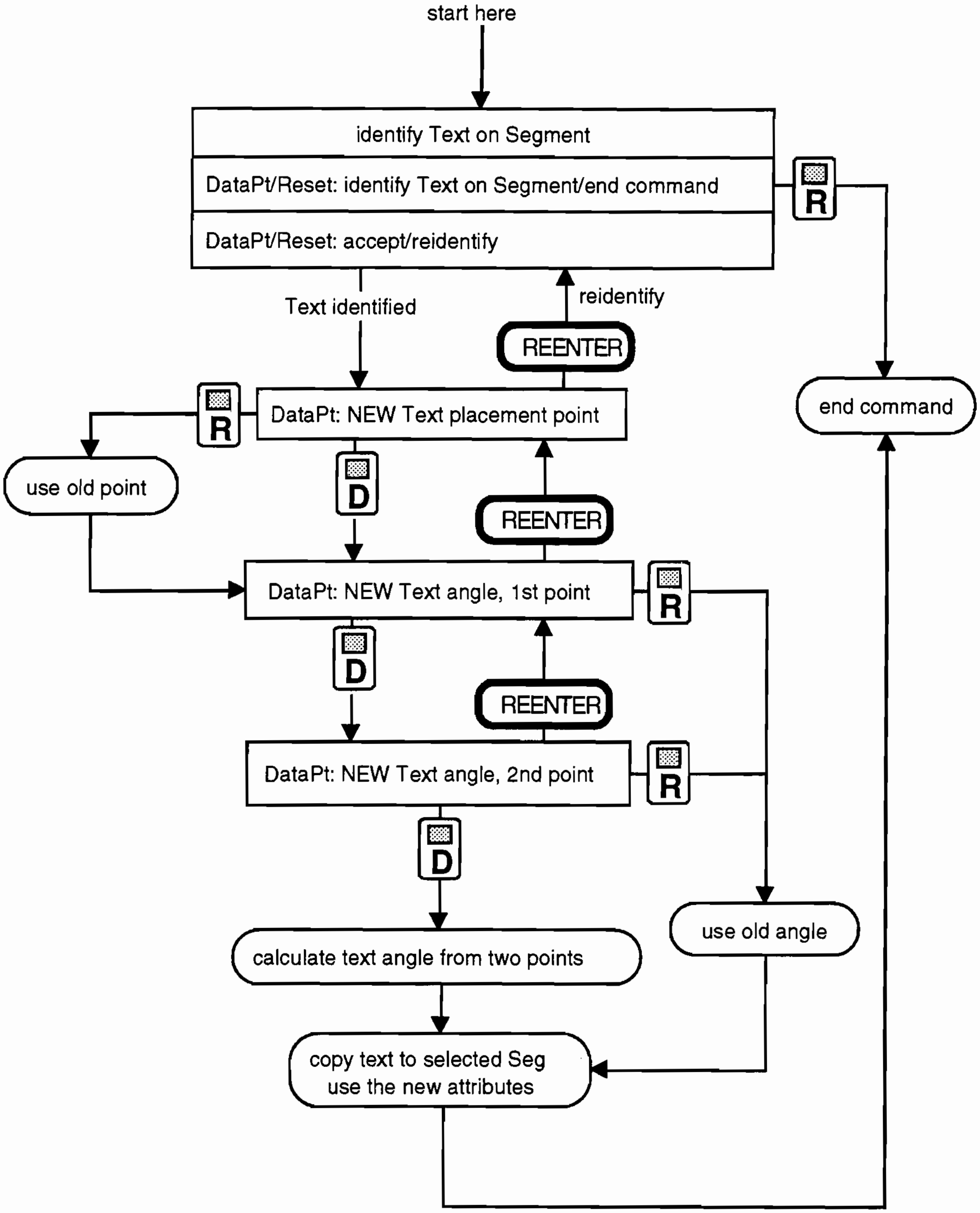
Make a copy of an existing Leg. The Leg may be copied to any Alternative. A new ID number, description, centerline angle, station number at center of intersection and direction of stationing may be assigned to the new Leg.

Primitive Command: COPY - TEXT ON ALT


start here
identify Text on Alternative
DataPt/Reset: identify Text on Alternative/end command
DataPt/Reset: accept/reidentify
R
Text identified
reidentify
REENTER
end Command
R
DataPt: NEW Text placement point
D
use old point
REENTER
DataPt: NEW Text angle, 1st point
R
D
REENTER
DataPt: NEW Text angle, 2nd point
R
D
use old angle
calculate text angle from two points
copy text to selected Alternative
use the new attributes

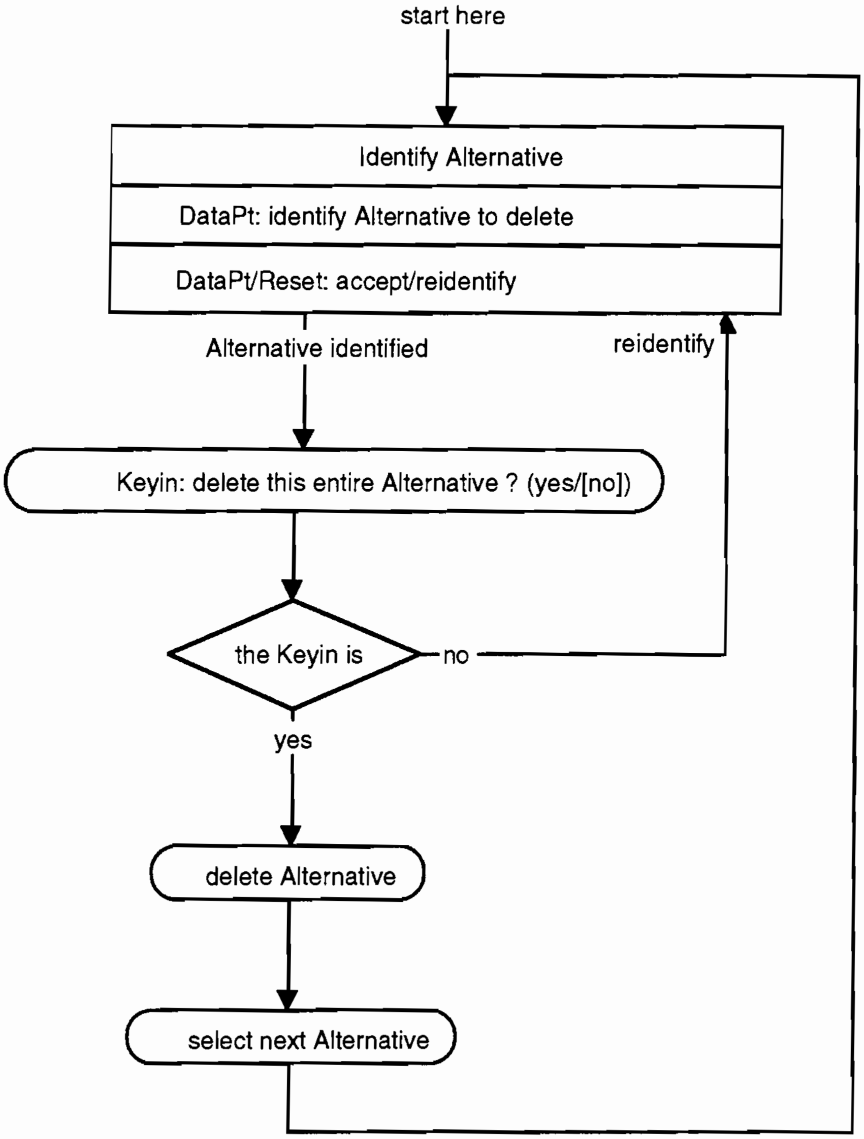
Make a copy of an existing Text on an Alternative and attach it to the selected Alternative. The new Text may have a new location and angle.

Primitive Command: COPY - TEXT ON SEG


start here
identify Text on Segment
DataPt/Reset: identify Text on Segment/end command
DataPt/Reset: accept/reidentify
R
Text identified
reidentify
REENTER
end command
R
DataPt: NEW Text placement point
use old point
D
REENTER
DataPt: NEW Text angle, 1st point
R
D
REENTER
DataPt: NEW Text angle, 2nd point
R
D
use old angle
calculate text angle from two points
copy text to selected Seg
use the new attributes

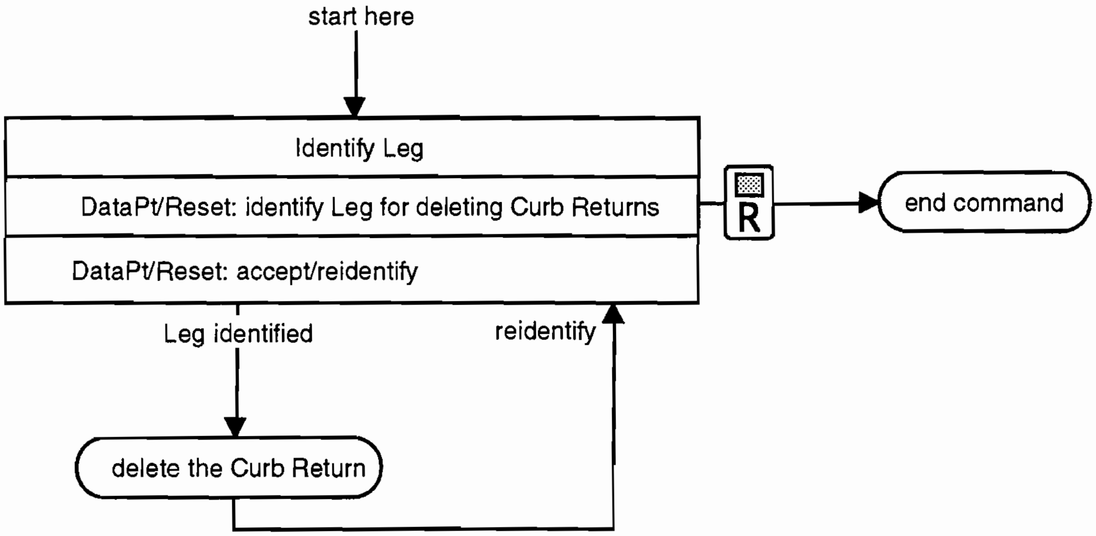
Make a copy of an existing Text on a Seg and attach it to the selected Seg. The new Text may have a new location and angle.

Primitive Command: DELETE - ALTERNATIVE



Delete an existing Alternative. Before each deletion, the user must confirm that the Alternative is to be deleted.

Primitive Command: DELETE - Curb Return

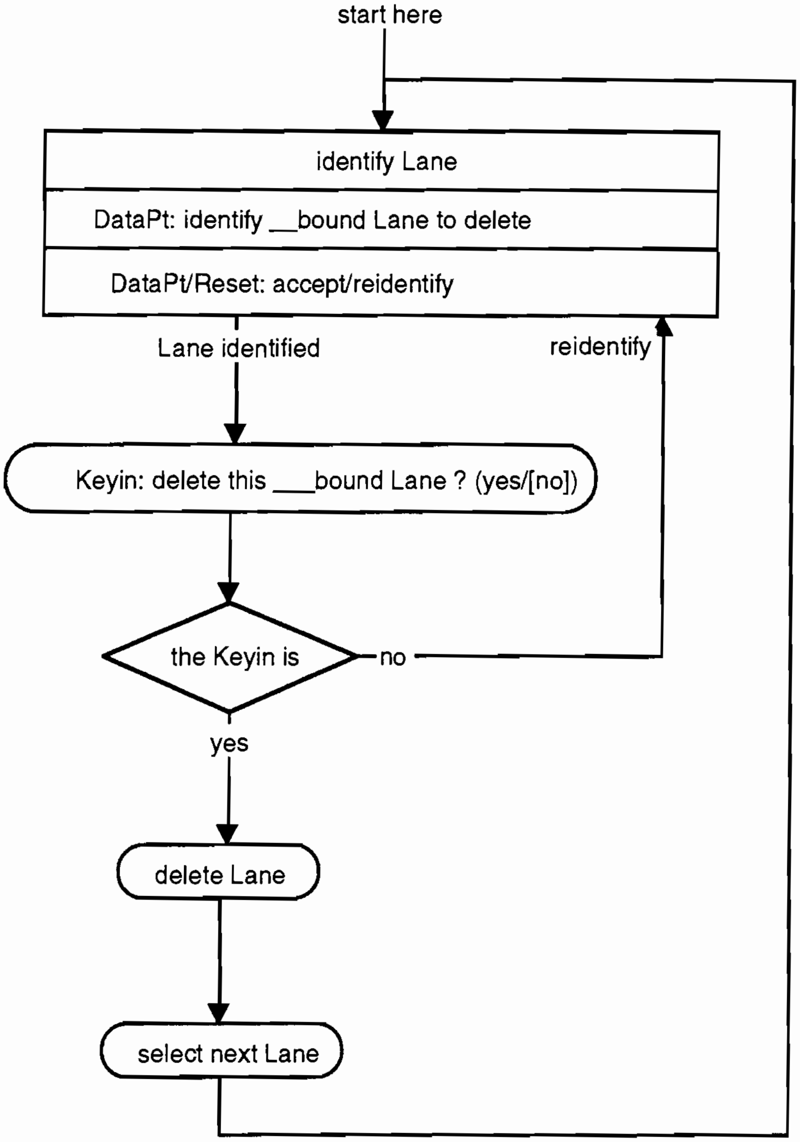

start here
Identify Leg
DataPt/Reset: identify Leg for deleting Curb Returns
DataPt/Reset: accept/reidentify
R
end command
Leg identified
reidentify
delete the Curb Return

Use this diagram for the 2 **DELETE - Curb Return** commands that follow.

Primitive Command: DELETE - CURB CR
Delete an existing curb Lane Curb Return. This is the Curb Return between the Inbound Lanes of the identified Leg and the Outbound Lanes of the adjacent Leg.

Primitive Command: DELETE - MEDIAN CR
Delete an existing median Lane Curb Return. This is the Curb Return that closes the median between the Inbound Lanes and Outbound Lanes of the identified Leg.

Primitive Command: DELETE - LANE

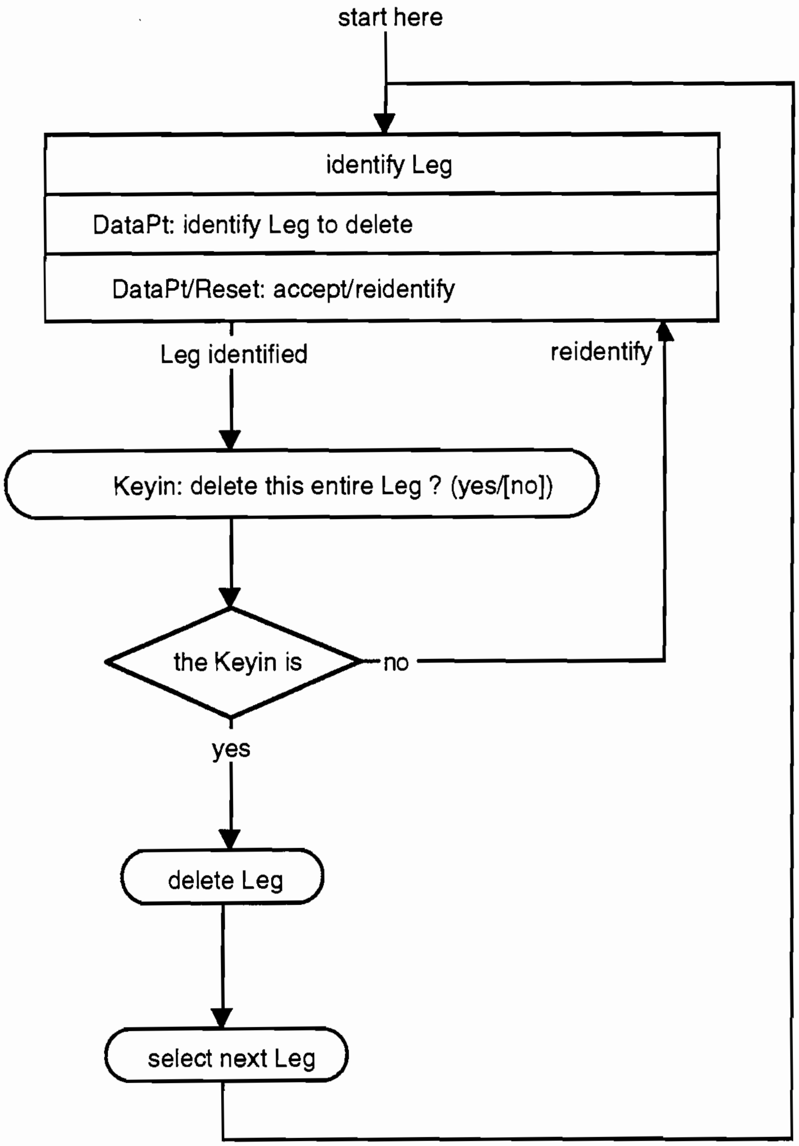

start here
identify Lane
DataPt: identify __bound Lane to delete
DataPt/Reset: accept/reidentify
Lane identified
reidentify
Keyin: delete this ___bound Lane ? (yes/[no])
the Keyin is
no
yes
delete Lane
select next Lane

Use this diagram for the 2 **DELETE - LANE** commands that follow.

Primitive Command: DELETE - LANE INBND
Delete an existing Inbound Lane. Before each deletion, the user must confirm that the Lane is to be deleted.

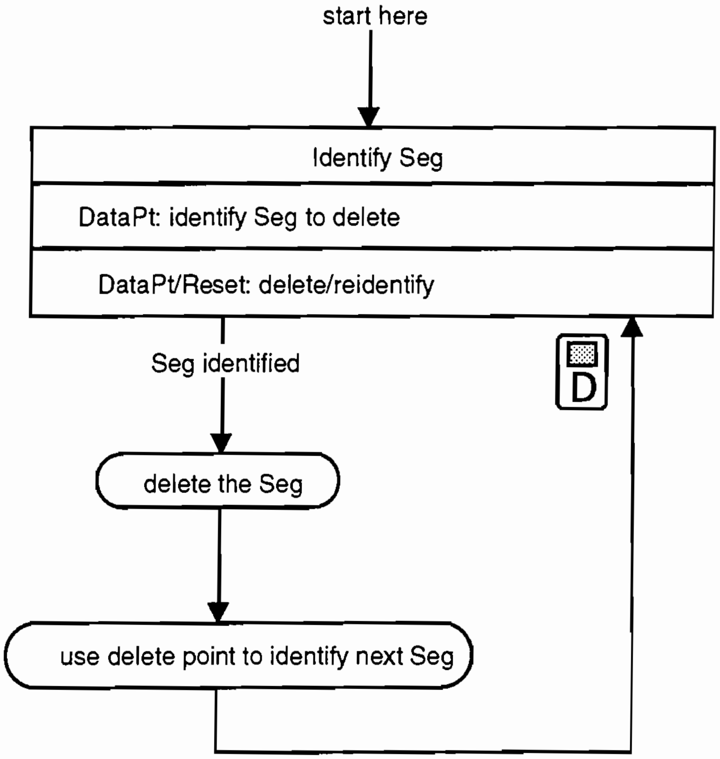
Primitive Command: DELETE - LANE OUTBND
Delete an existing Outbound Lane. Before each deletion, the user must confirm that the Lane is to be deleted.

Primitive Command: DELETE - LEG


start here
identify Leg
DataPt: identify Leg to delete
DataPt/Reset: accept/reidentify
Leg identified
reidentify
Keyin: delete this entire Leg ? (yes/[no])
the Keyin is
no
yes
delete Leg
select next Leg

Delete an existing Leg. Before each deletion, the user must confirm that the Leg is to be deleted.

Primitive Command: DELETE - SEG


start here
Identify Seg
DataPt: identify Seg to delete
DataPt/Reset: delete/reidentify
Seg identified
D
delete the Seg
use delete point to identify next Seg

Use this diagram for the 4 **DELETE - SEG** commands that follow. Identify Object will only find Objects of the specific type that the command is designed to delete. For example, it is impossible to identify an Inner Edge Seg when trying to delete a Stopline Seg.

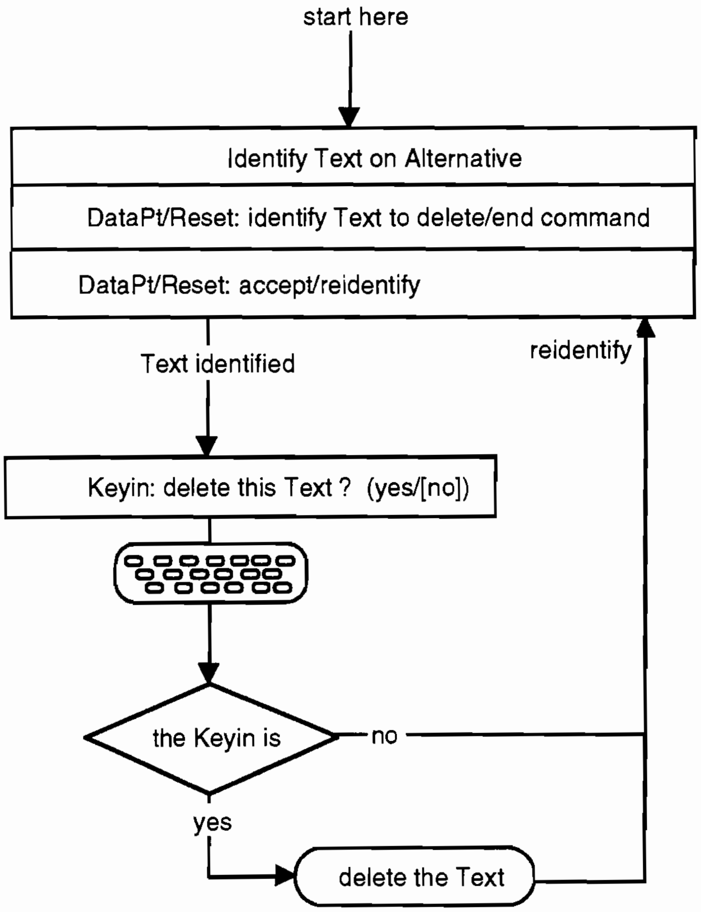
Primitive Command: DELETE - SEG CNTRLINE
Not programmed yet.

Primitive Command: DELETE - SEG INN EDGE
Delete an existing Inner Edge Seg.

Primitive Command: DELETE - SEG OUT EDGE
Delete an existing Outer Edge Seg.

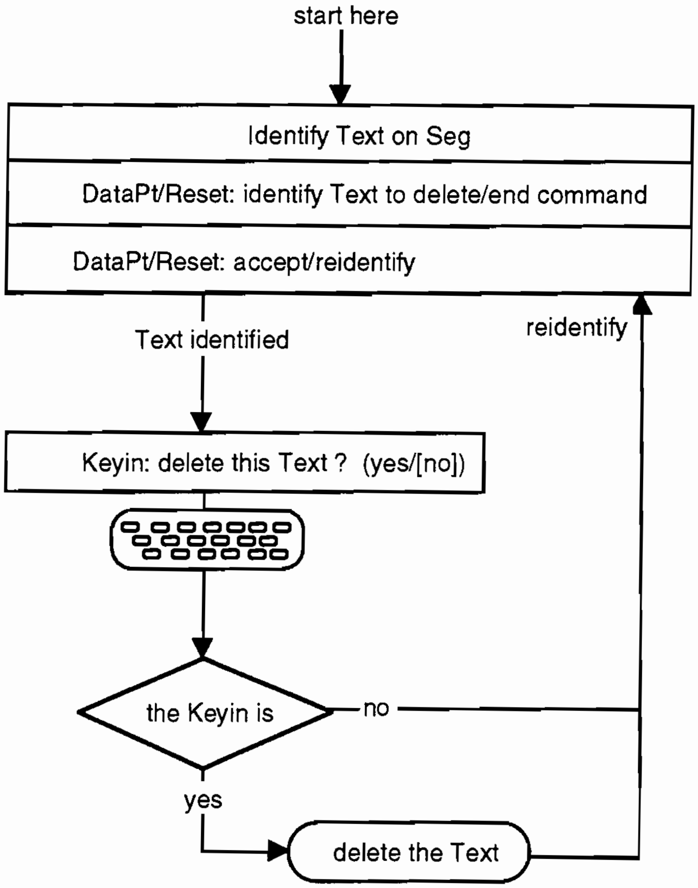
Primitive Command: DELETE - SEG STOPLINE
Delete an existing Stopline Seg.

Primitive Command: DELETE - TEXT ON ALT


start here
Identify Text on Alternative
DataPt/Reset: identify Text to delete/end command
DataPt/Reset: accept/reidentify
Text identified
reidentify
Keyin: delete this Text ? (yes/[no])
the Keyin is
no
yes
delete the Text

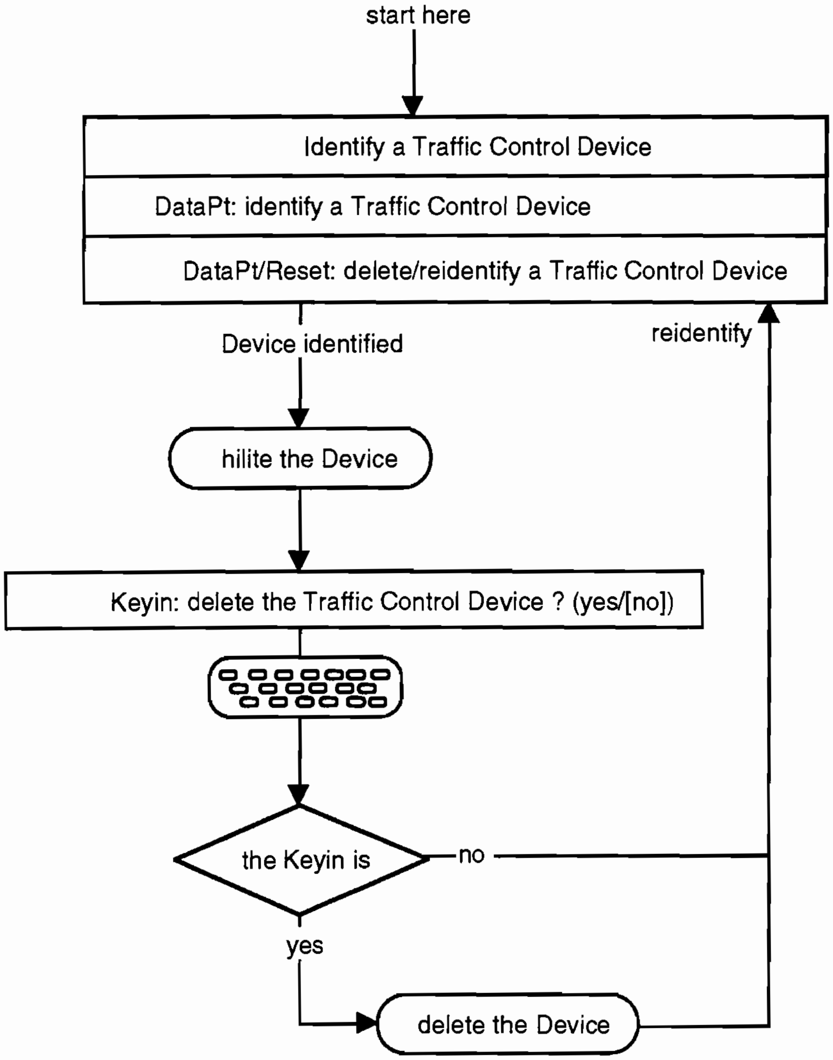
Delete an existing Text on an Alternative. Before each deletion, the user must confirm that the Text is to be deleted.

Primitive Command: DELETE - TEXT ON SEG


start here
Identify Text on Seg
DataPt/Reset: identify Text to delete/end command
DataPt/Reset: accept/reidentify
Text identified
reidentify
Keyin: delete this Text ? (yes/[no])
the Keyin is
no
yes
delete the Text

Delete an existing Text on a Seg. Before each deletion, the user must confirm that the Text is to be deleted.

Primitive Command: DELETE - TRAF CONTRL


start here
Identify a Traffic Control Device
DataPt: identify a Traffic Control Device
DataPt/Reset: delete/reidentify a Traffic Control Device
Device identified
reidentify
hilite the Device
Keyin: delete the Traffic Control Device ? (yes/[no])
the Keyin is
no
yes
delete the Device

Delete an existing traffic control device. Before each deletion, the user must confirm that the device is to be deleted.

Primitive Command: END IGIDS
Stop IGIDS. IGIDS graphics will remain in the Graphics Engine's database, but the IGIDS data cannot be recreated from this. If IGIDS data has not been saved in some form, it will be lost.

Transient Command: HILITE - CURRENT ALT
Hilite all of the Legs and Text on an Alternative of the selected Alternative.

Transient Command: HILITE - CURRENT LANE - ALL
Hilite the Inner Edge Segs, Outer Edge Segs and Stopline Segs of the selected Lane.

Transient Command: HILITE - CURRENT LANE - INNER EDGE
Hilite the Inner Edge Segs of the selected Lane.

Transient Command: HILITE - CURRENT LANE - OUTER EDGE
Hilite the Outer Edge Segs of the selected Lane.

Transient Command: HILITE - CURRENT LANE - STOP LINE
Hilite the Stopline Segs of the selected Lane.

Transient Command: HILITE - CURRENT LEG - ALL
Hilite the Inbound Lanes, Outbound Lanes, Centerline Segs, Median Lane Curb Return Segs, and Curb Lane Curb Return Segs of the selected Leg.

Transient Command: HILITE - CURRENT LEG - CENTERLINE
Hilite the Centerline Segs of the selected Leg.

Transient Command: HILITE - CURRENT LEG - CURB CR
Hilite the Curb Lane Curb Return Segs of the selected Leg.

Transient Command: HILITE - CURRENT LEG - CURB RETURNS
Hilite the Median Lane Curb Return Segs and Curb Lane Curb Return Segs of the selected Leg.

Transient Command: HILITE - CURRENT LEG - INBND LANES
Hilite the Inbound Lanes of the selected Leg.

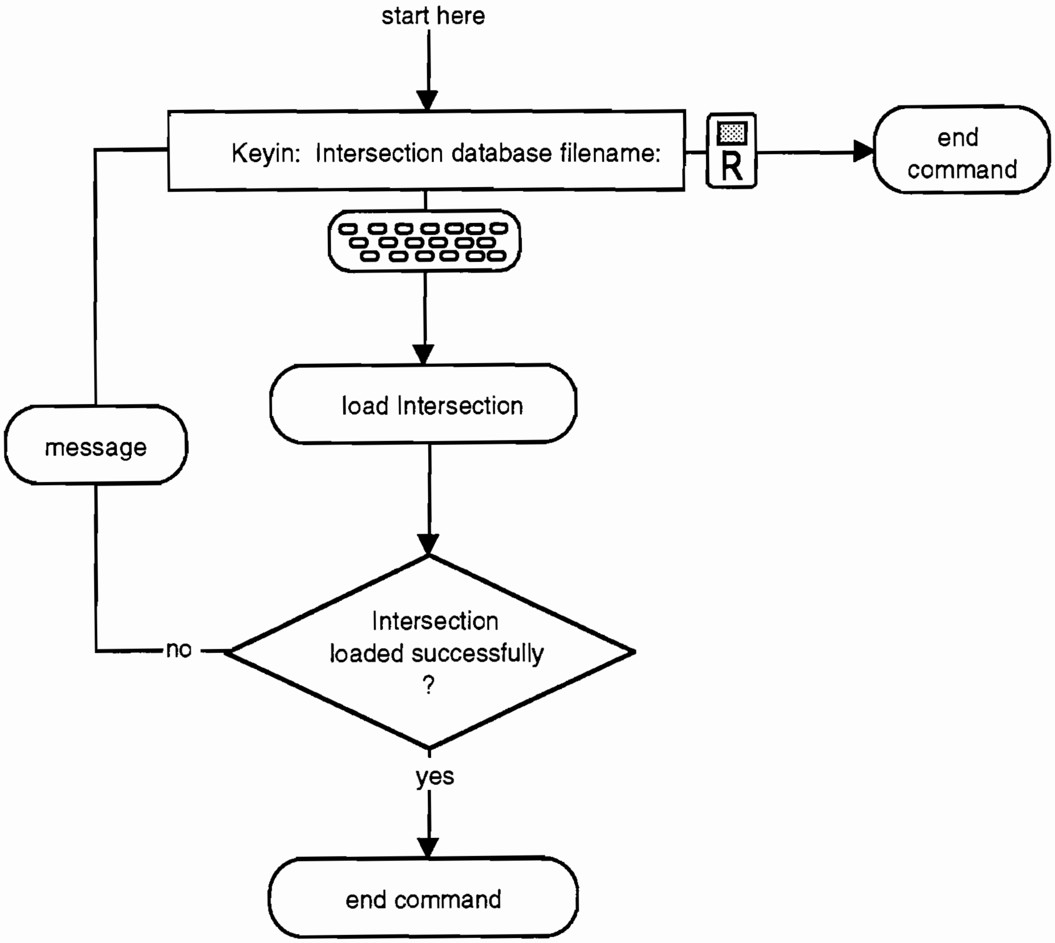
Transient Command: HILITE - CURRENT LEG - MEDIAN CR
Hilite the Median Lane Curb Return Segs of the selected Leg.

Transient Command: HILITE - CURRENT LEG - OUTBND LANES
Hilite the Outbound Lanes of the selected Leg.

Transient Command: HILITE - CURRENT SEG
Hilite the selected Seg.

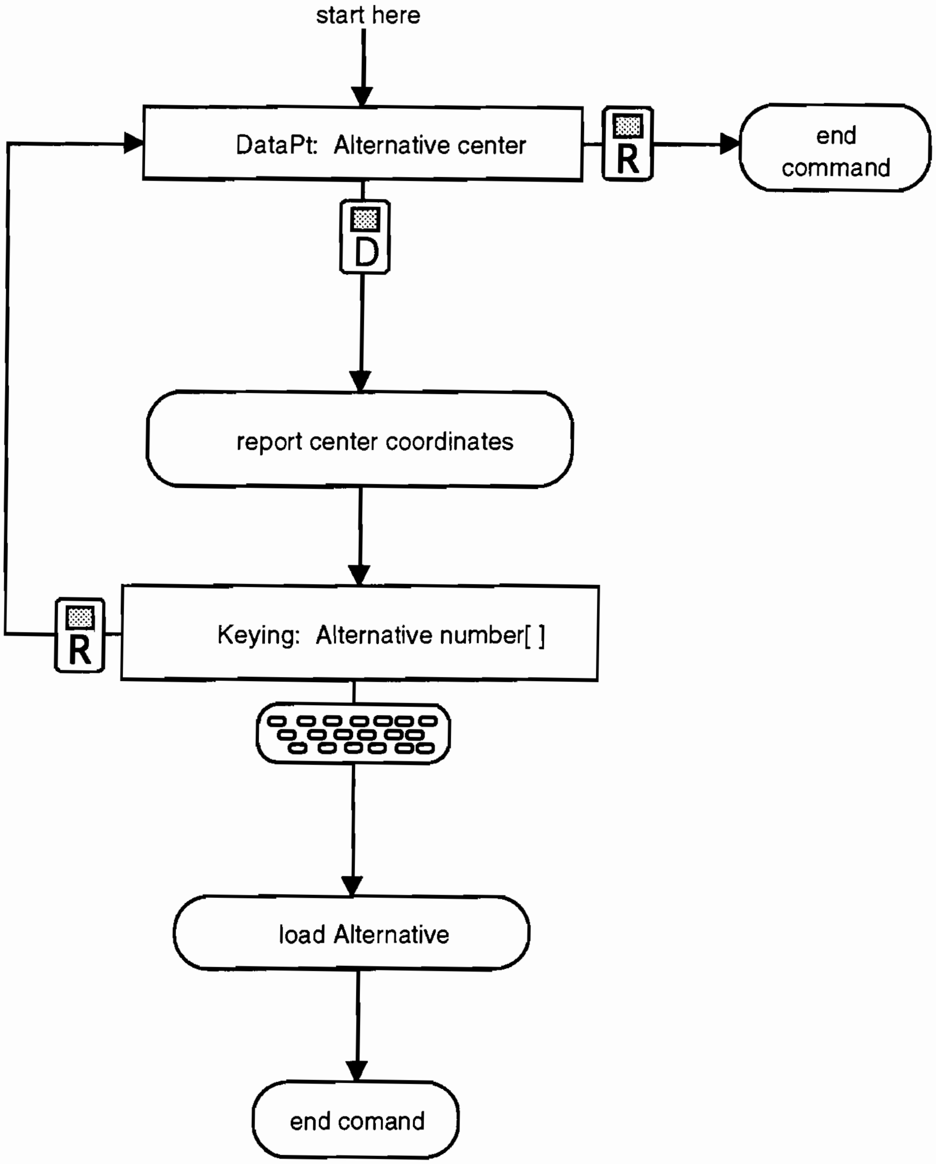
Transient Command: HILITE - CURRENT TEXT
Hilite the selected Text.

Primitive Command: LOAD FROM - DATABASE


start here
Keyin: Intersection database filename:
R
end command
load Intersection
message
Intersection loaded successfully ?
no
yes
end command

Load Intersection data from a file that was written by **Primitive Command: SAVE TO - Data Base**. All data from the current IGIDS session will be lost.

Primitive Command: LOAD FROM - STANDARD


start here
DataPt: Alternative center
R
end command
D
report center coordinates
Keying: Alternative number[ ]
R
load Alternative
end comand

Use this diagram for the 17 **LOAD FROM - STANDARD** commands that follow.

Primitive Command: LOAD FROM - STANDARD - 3x2
Load a standard 4 leg Alternative that has 1 through Inbound Lane, 1 exclusive left turn Inbound Lane, and 1 Outbound Lane on the north-south street and 1 through Inbound Lane and 1 Outbound Lane on the east-west street. Coordinates of the center point and the ID number must be specified.

Primitive Command: LOAD FROM - STANDARD - 3x3
Load a standard 4 leg Alternative that has 1 through Inbound Lane, 1 exclusive left turn Inbound Lane, and 1 Outbound Lane in each direction. Coordinates of the center point and the ID number must be specified.

Primitive Command: LOAD FROM - STANDARD - 4t2

Load a standard 3 leg "T" Alternative. The northbound approach T's into the east-west street. The northbound approach has 1 Inbound Lane and 1 Outbound Lane. The east-west street has 2 through Inbound Lanes and 2 Outbound Lanes. Coordinates of the center point and the ID number must be specified.

Primitive Command: LOAD FROM - STANDARD - 4t3
Load a standard 3 leg "T" Alternative. The northbound approach T's into the east-west street. The northbound approach has 2 Inbound Lanes and 1 Outbound Lane. The east-west street has 2 Inbound Lanes and 2 Outbound Lanes. Coordinates of the center point and the ID number must be specified.

Primitive Command: LOAD FROM - STANDARD - 4t4
Load a standard 3 leg "T" Alternative. The northbound approach T's into the east-west street. Each leg has 2 Inbound Lanes and 2 Outbound Lanes. Coordinates of the center point and the ID number must be specified.

Primitive Command: LOAD FROM - STANDARD - 4x2
Load a standard 4 leg Alternative that has 2 through Inbound Lanes and 2 Outbound Lanes on the north-south street and 1 through Inbound Lane and 1 Outbound Lane on the east-west street. Coordinates of the center point and the ID number must be specified.

Primitive Command: LOAD FROM - STANDARD - 4x3
Load a standard 4 leg Alternative that has 2 through Inbound Lanes and 2 Outbound Lanes on the north-south street and 1 through Inbound Lane, 1 exclusive left turn Inbound Lane, and 1 Outbound Lane on the east-west street. Coordinates of the center point and the ID number must be specified.

Primitive Command: LOAD FROM - STANDARD - 4x4
Load a standard 4 leg Alternative that has 2 through Inbound Lanes and 2 Outbound Lanes in each direction. Coordinates of the center point and the ID number must be specified.

Primitive Command: LOAD FROM - STANDARD - 5x4
Load a standard 4 leg Alternative that has 2 through Inbound Lanes, 1 exclusive left turn Inbound Lane, and 2 Outbound Lanes on the north-south street and 2 through Inbound Lanes and 2 Outbound Lanes on the east-west street. Coordinates of the center point and the ID number must be specified.

Primitive Command: LOAD FROM - STANDARD - 5x5
Load a standard 4 leg Alternative that has 2 through Inbound Lanes, 1 exclusive left turn Inbound Lane, and 2 Outbound Lanes in each direction. Coordinates of the center point and the ID number must be specified.

Primitive Command: LOAD FROM - STANDARD - 6x4
Load a standard 4 leg Alternative that has 3 through Inbound Lanes and 3 Outbound Lanes on the north-south street and 2 through Inbound Lanes and 2 Outbound Lanes on the east-west street. Coordinates of the center point and the ID number must be specified.

Primitive Command: LOAD FROM - STANDARD - 6x5
Load a standard 4 leg Alternative that has 3 through Inbound Lanes and 3 Outbound Lanes on the north-south street and 2 through Inbound Lanes, 1 exclusive left turn Inbound Lane, and 3 Outbound Lanes on the east-west street. Coordinates of the center point and the ID number must be specified.

Primitive Command: LOAD FROM - STANDARD - 6x6
Load a standard 4 leg Alternative that has 3 through Inbound Lanes and 3 Outbound Lanes in each direction. Coordinates of the center point and the ID number must be specified.

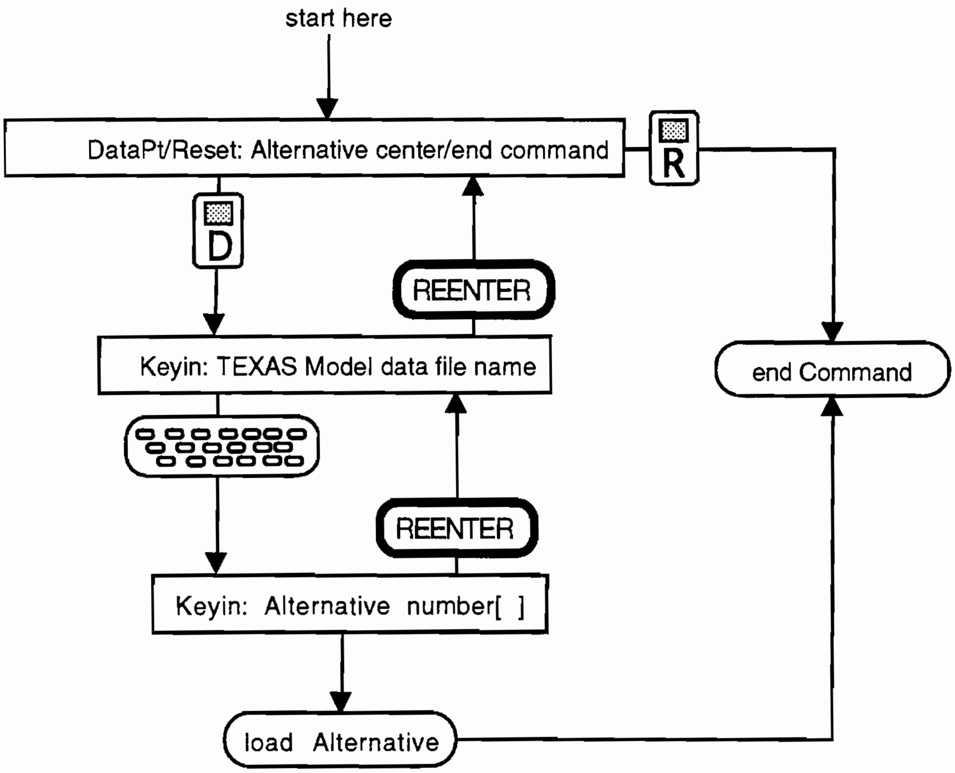
Primitive Command: LOAD FROM - STANDARD - 7x4
Load a standard 4 leg Alternative that has 3 through Inbound Lanes, 1 exclusive left turn Inbound Lane, and 3 Outbound Lanes on the north-south street and 2 through Inbound Lanes and 2 Outbound Lanes on the east-west street. Coordinates of the center point and the ID number must be specified.

Primitive Command: LOAD FROM - STANDARD - 7x5
Load a standard 4 leg Alternative that has 3 through Inbound Lanes, 1 exclusive left turn Inbound Lane, and 3 Outbound Lanes on the north-south street and 2 through Inbound Lanes, 1 exclusive left turn Inbound Lane, and 2 Outbound Lanes on the east-west street. Coordinates of the center point and the ID number must be specified.

Primitive Command: LOAD FROM - STANDARD - 7x6
Load a standard 4 leg Alternative that has 3 through Inbound Lanes, 1 exclusive left turn Inbound Lane, and 3 Outbound Lanes on the north-south street and 3 through Inbound Lanes and 3 Outbound Lanes on the east-west street. Coordinates of the center point and the ID number must be specified.

Primitive Command: LOAD FROM - STANDARD - 7x7
Load a standard 4 leg Alternative that has 3 through Inbound Lanes, 1 exclusive left turn Inbound Lane, and 3 Outbound Lanes in each direction. Coordinates of the center point and the ID number must be specified.

Primitive Command: LOAD FROM - TX Mdl file

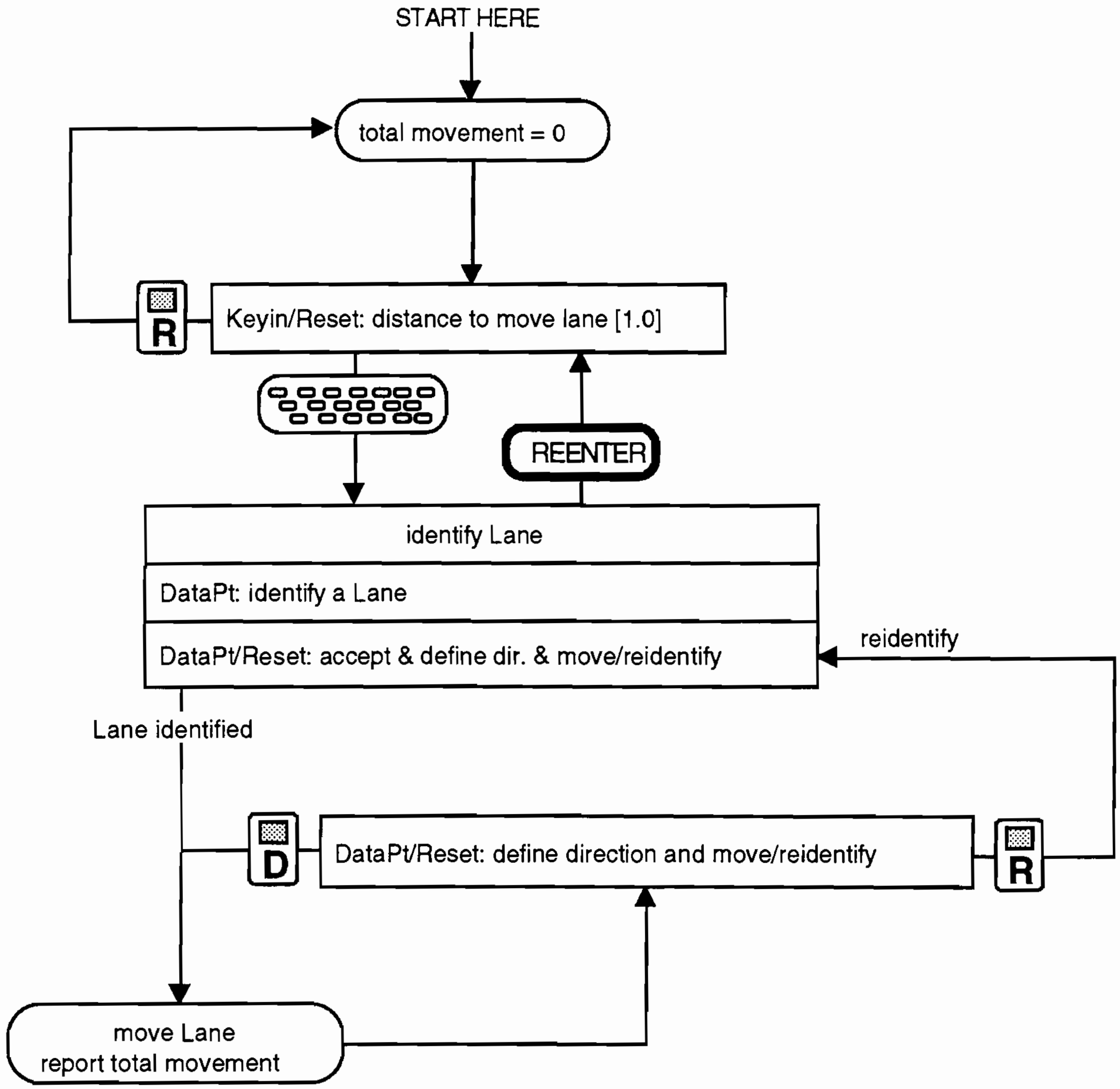

start here
DataPt/Reset: Alternative center/end command
R
D
REENTER
Keyin: TEXAS Model data file name
end Command
REENTER
Keyin: Alternative number[ ]
load Alternative

Load data for an Alternative from a file that was written by the TEXAS Model processor called GDVDATA. Specify the coordinates of the center point, data file name and ID number.

Primitive Command: MODIFY - ALTERNATIVE
Not programmed yet.

Primitive Command: MODIFY - INTERSECTION
Not programmed yet.

Primitive Command: MODIFY - LANE - LATERAL SHIFT

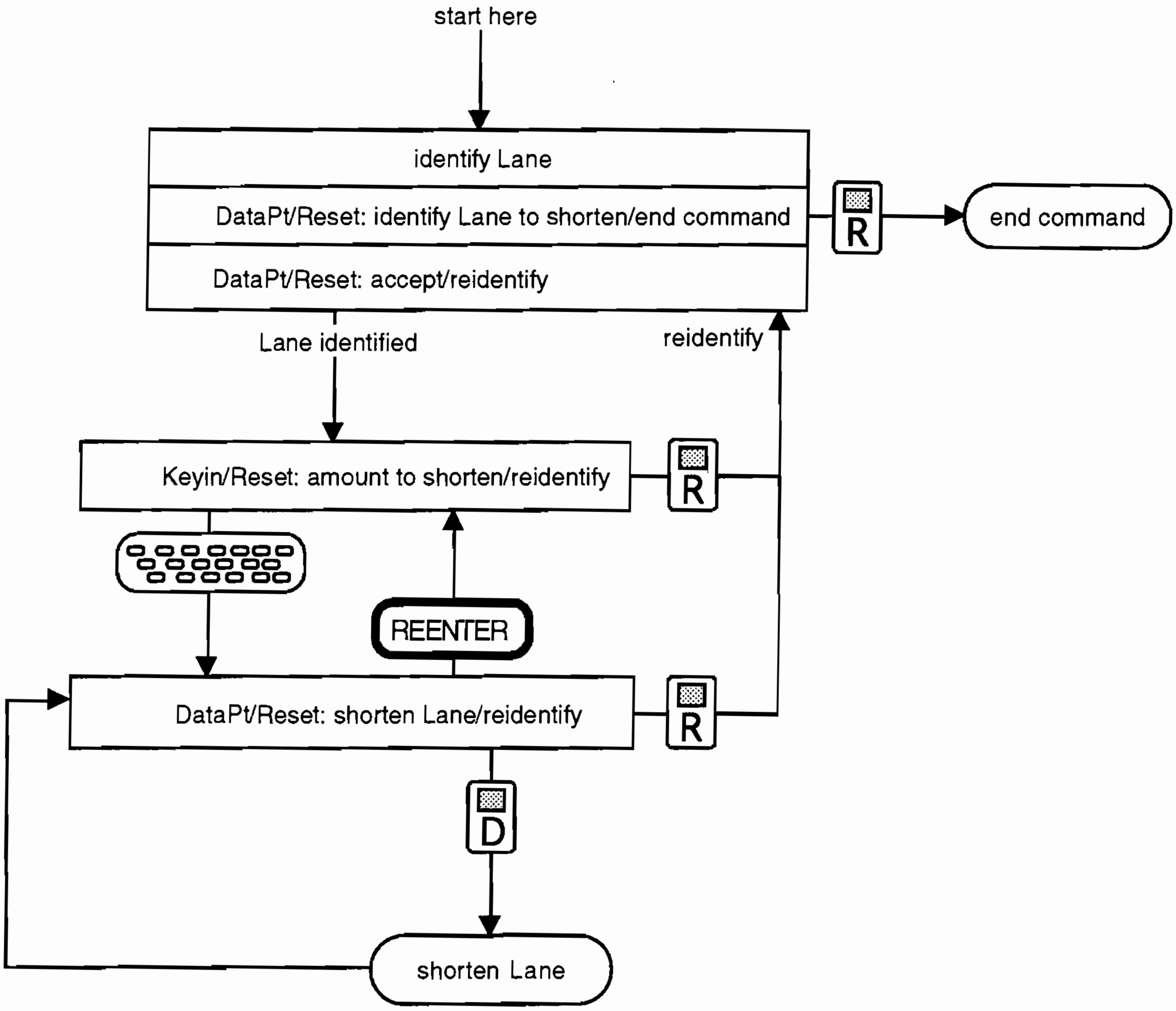

START HERE
total movement = 0
R
Keyin/Reset: distance to move lane [1.0]
REENTER
identify Lane
DataPt: identify a Lane
DataPt/Reset: accept & define dir. & move/reidentify
reidentify
Lane identified
D
DataPt/Reset: define direction and move/reidentify
R
move Lane
report total movement

Shift a Lane laterally by a specified distance. The direction of movement is determined by the location of the move DataPt with respect to the Leg centerline. The radii of any arcs on the Lane edges will be adjusted by the amount of the shift. This is the same as **Primitive Command: MOVE - LANE - LATERAL**.

Primitive Command: MODIFY - LANE - LENGTHEN
Not programmed yet.

Primitive Command: MODIFY - LANE - NARROWER
Not programmed yet.

Primitive Command: MODIFY - LANE - SHORTEN


start here
identify Lane
DataPt/Reset: identify Lane to shorten/end command
DataPt/Reset: accept/reidentify
R
end command
Lane identified
reidentify
Keyin/Reset: amount to shorten/reidentify
R
REENTER
DataPt/Reset: shorten Lane/reidentify
R
D
shorten Lane

Shorten the end of a Lane that is nearest the intersection center.

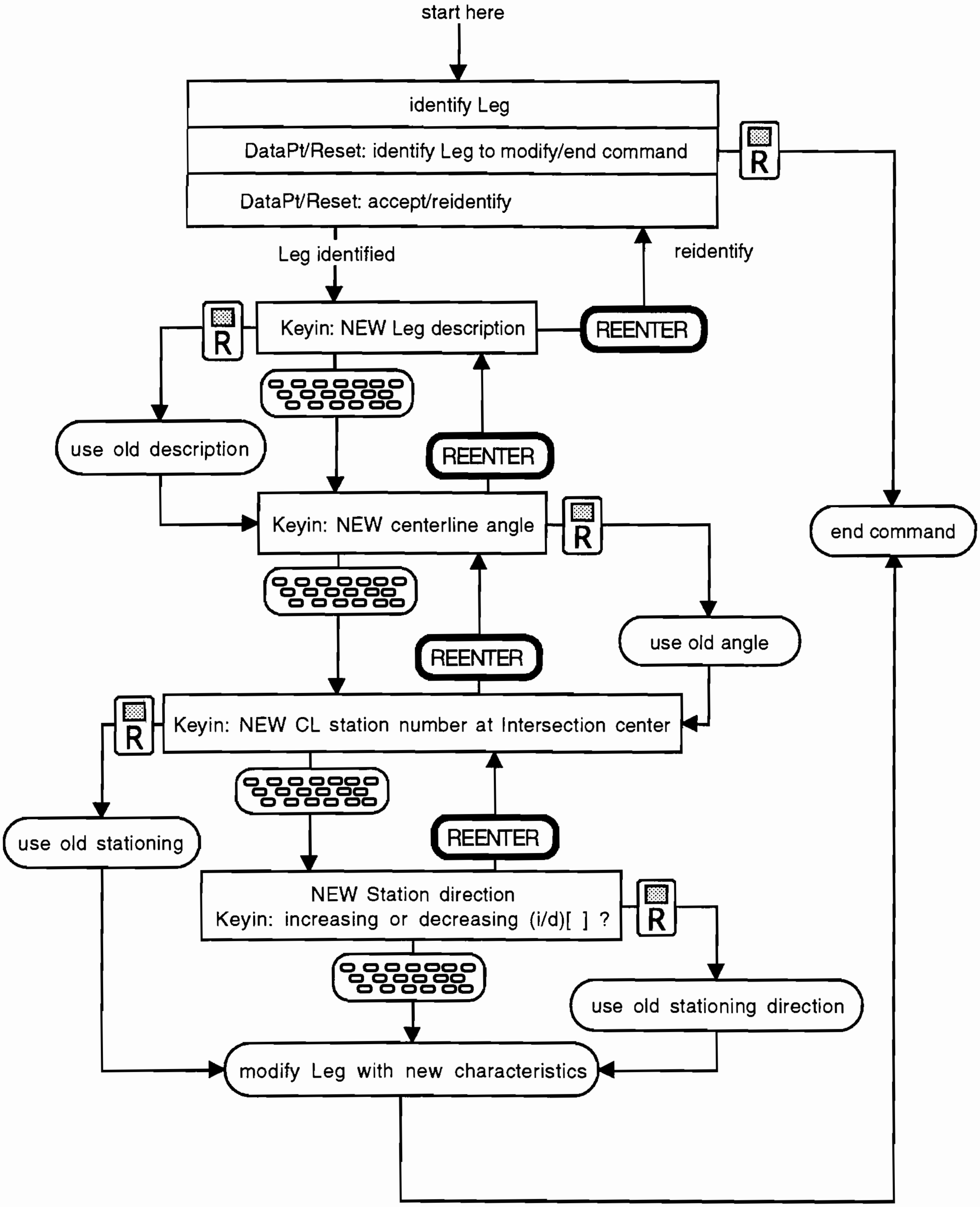
Primitive Command: MODIFY - LANE - WIDEN
Not programmed yet.

Primitive Command: MODIFY - LANE EDGE - LENGTHEN
Not programmed yet.

Primitive Command: MODIFY - LANE EDGE - SHORTEN
Not programmed yet.

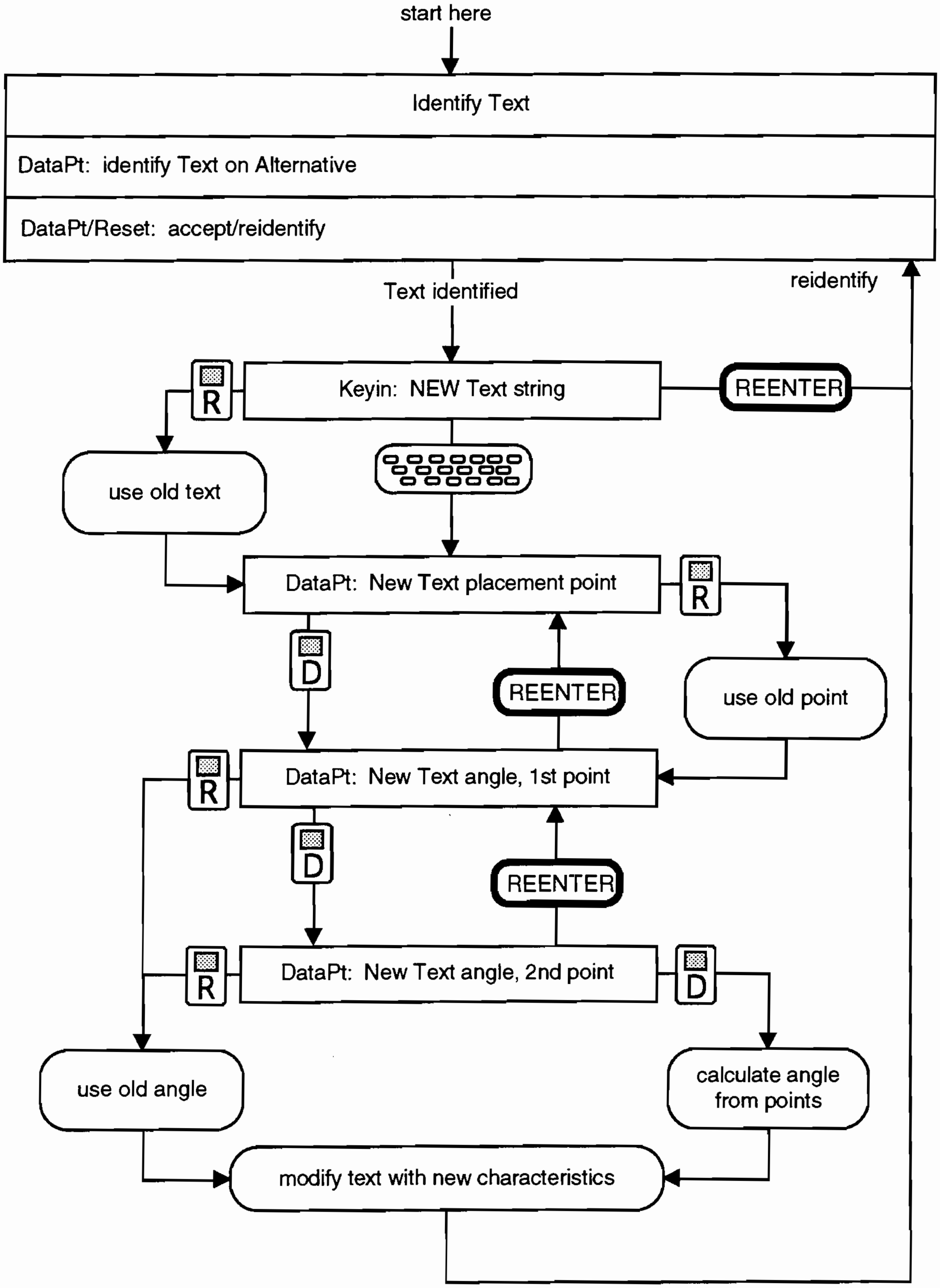
Primitive Command: MODIFY - LANE EDGE - TAPER
Not programmed yet.

Primitive Command: MODIFY - LEG


start here
identify Leg
DataPt/Reset: identify Leg to modify/end command
DataPt/Reset: accept/reidentify
R
Leg identified
reidentify
Keyin: NEW Leg description
R
REENTER
use old description
REENTER
Keyin: NEW centerline angle
R
end command
use old angle
REENTER
Keyin: NEW CL station number at Intersection center
R
use old stationing
REENTER
NEW Station direction
Keyin: increasing or decreasing (i/d)[ ] ?
R
use old stationing direction
modify Leg with new characteristics

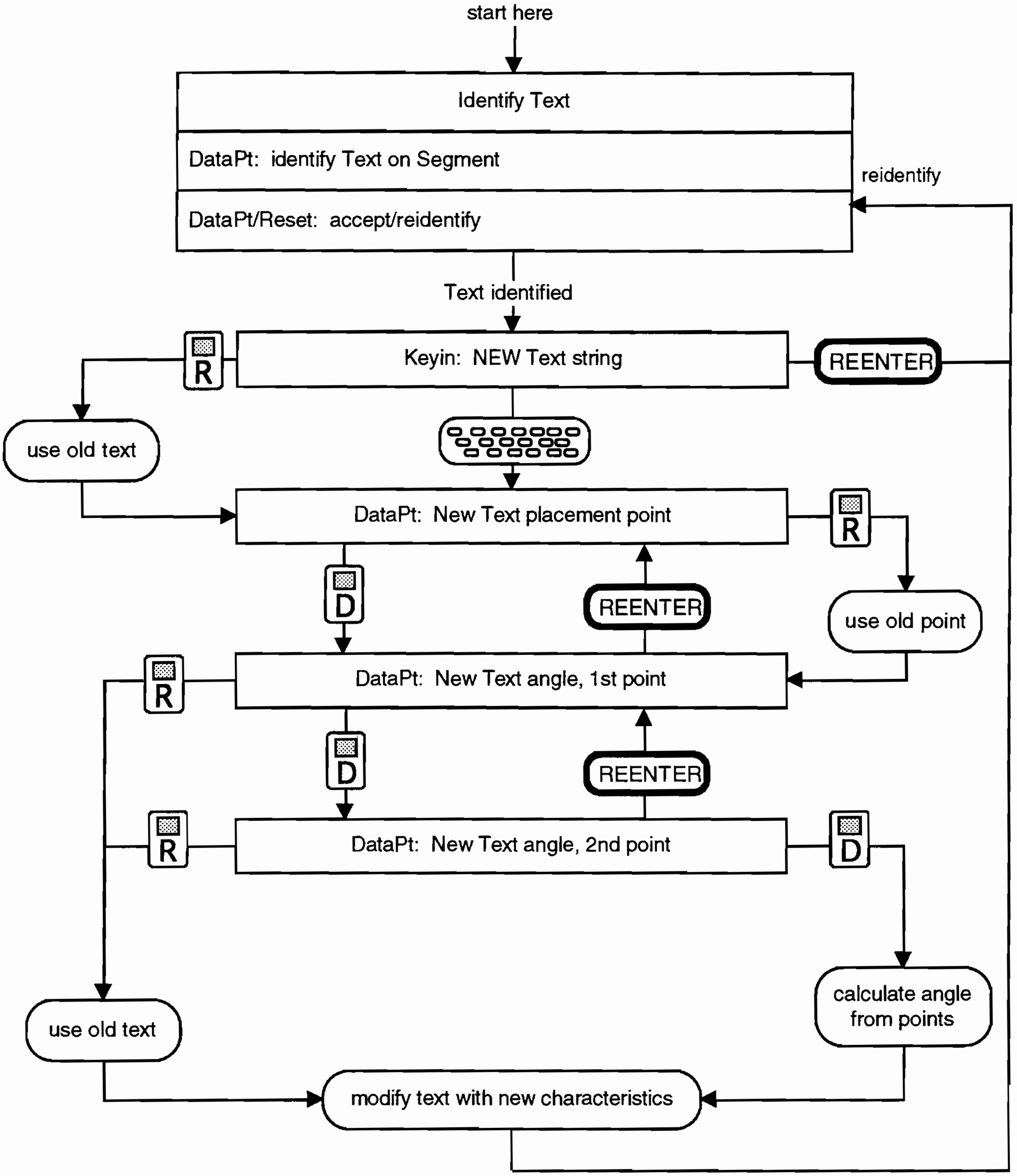
Change the Leg description, centerline angle, station number at center of intersection and direction of stationing.

Primitive Command: MODIFY - TEXT ON ALT


start here
Identify Text
DataPt: identify Text on Alternative
DataPt/Reset: accept/reidentify
Text identified
reidentify
Keyin: NEW Text string
R
REENTER
use old text
DataPt: New Text placement point
R
D
REENTER
use old point
DataPt: New Text angle, 1st point
R
D
REENTER
DataPt: New Text angle, 2nd point
R
D
use old angle
calculate angle from points
modify text with new characteristics

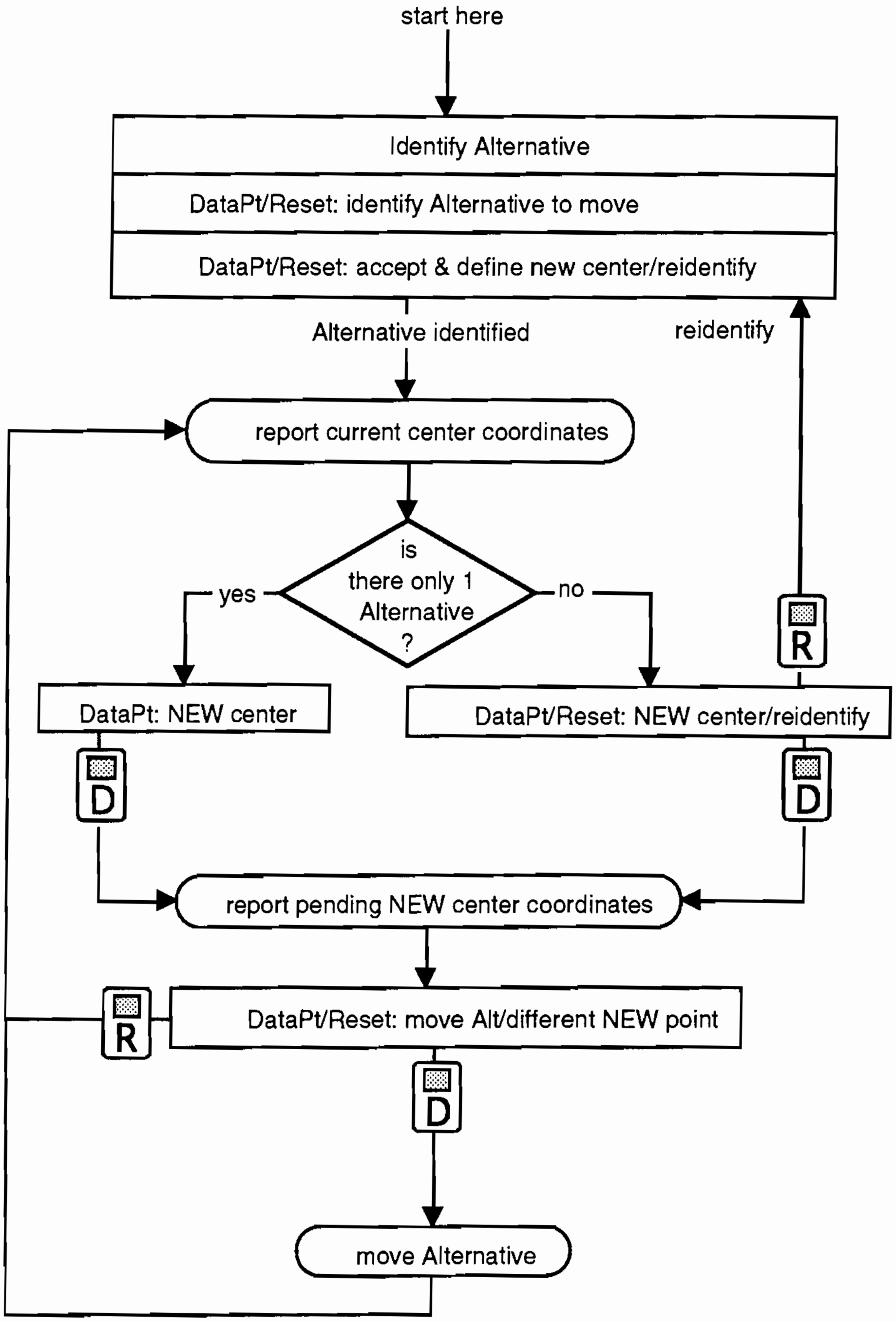
Change the text, location and angle of an existing Text on an Alternative.

Primitive Command: MODIFY - TEXT ON SEG


start here
Identify Text
DataPt: identify Text on Segment
DataPt/Reset: accept/reidentify
reidentify
Text identified
R
Keyin: NEW Text string
REENTER
use old text
DataPt: New Text placement point
R
D
REENTER
use old point
R
DataPt: New Text angle, 1st point
D
REENTER
R
DataPt: New Text angle, 2nd point
D
use old text
calculate angle from points
modify text with new characteristics

Change the text, location and angle of an existing Text on a Seg.

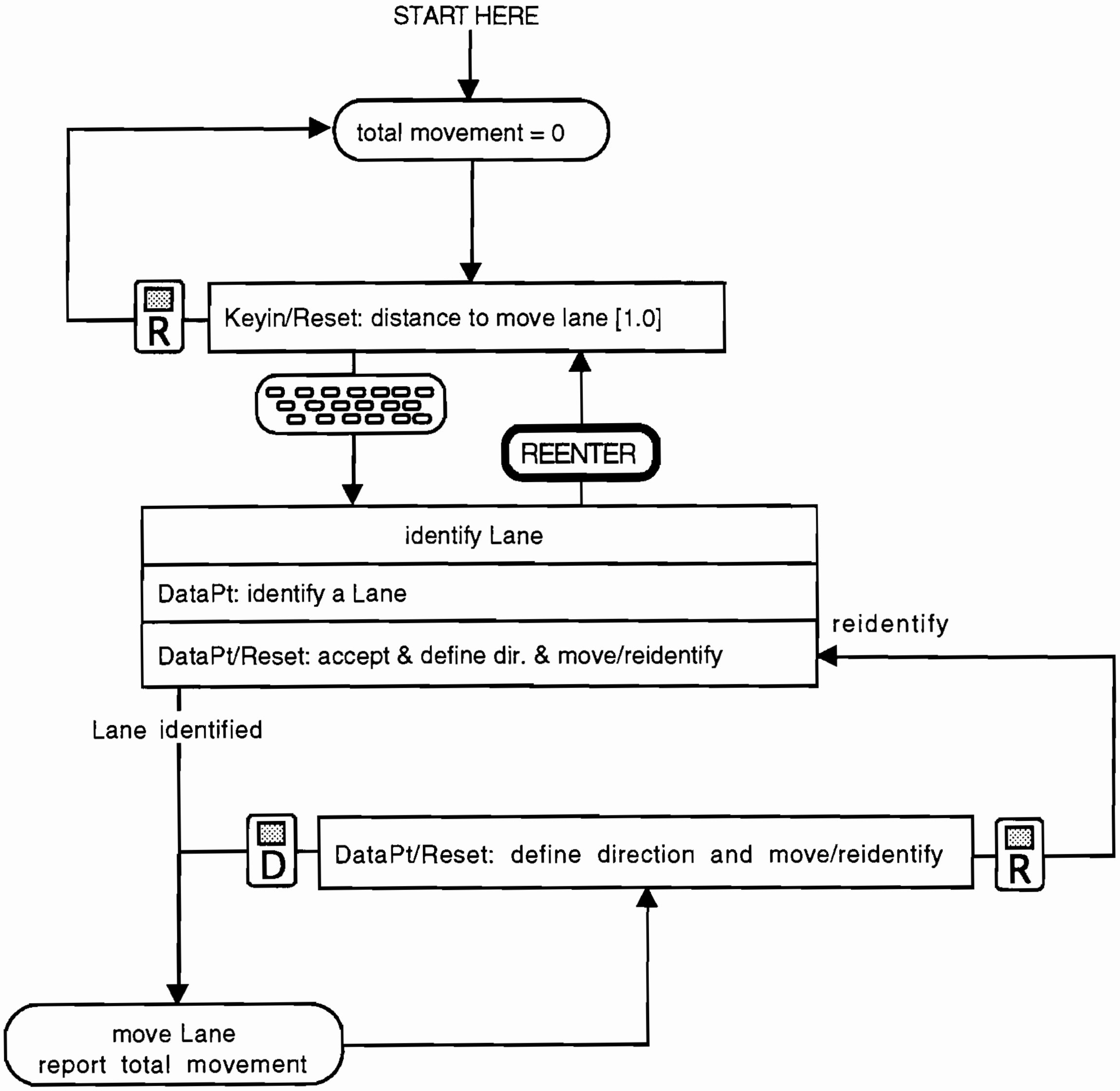
Primitive Command: MOVE - ALTERNATIVE


start here
Identify Alternative
DataPt/Reset: identify Alternative to move
DataPt/Reset: accept & define new center/reidentify
Alternative identified
reidentify
report current center coordinates
is there only 1 Alternative ?
yes
no
R
DataPt: NEW center
DataPt/Reset: NEW center/reidentify
D
D
report pending NEW center coordinates
R
DataPt/Reset: move Alt/different NEW point
D
move Alternative

Move an Alternative by specifying a new center point.

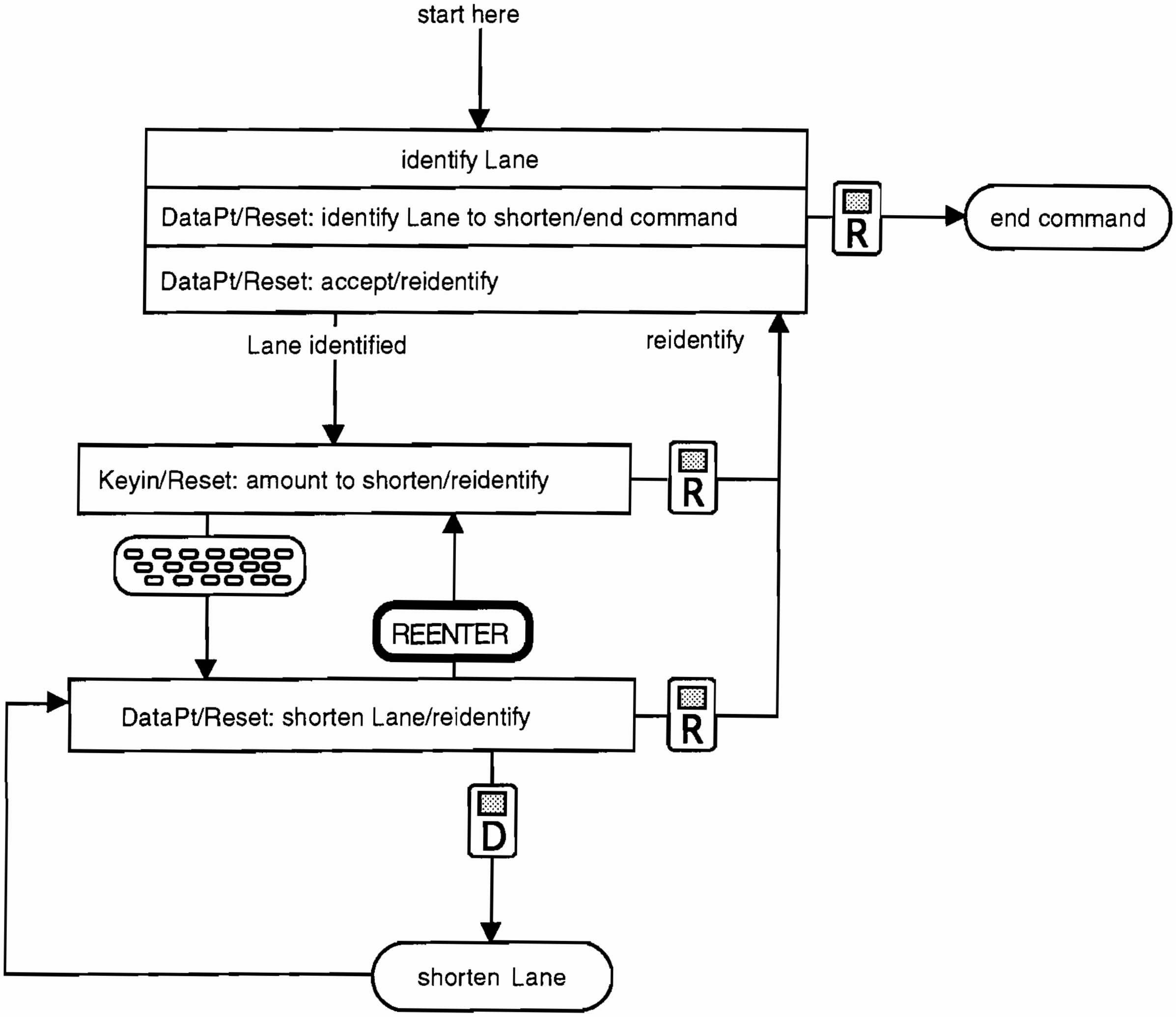
Primitive Command: MOVE - CONTROLLER
Not programmed yet.

Primitive Command: MOVE - LANE - LATERAL


START HERE
total movement = 0
R
Keyin/Reset: distance to move lane [1.0]
REENTER
identify Lane
DataPt: identify a Lane
DataPt/Reset: accept & define dir. & move/reidentify
reidentify
Lane identified
D
DataPt/Reset: define direction and move/reidentify
R
move Lane
report total movement

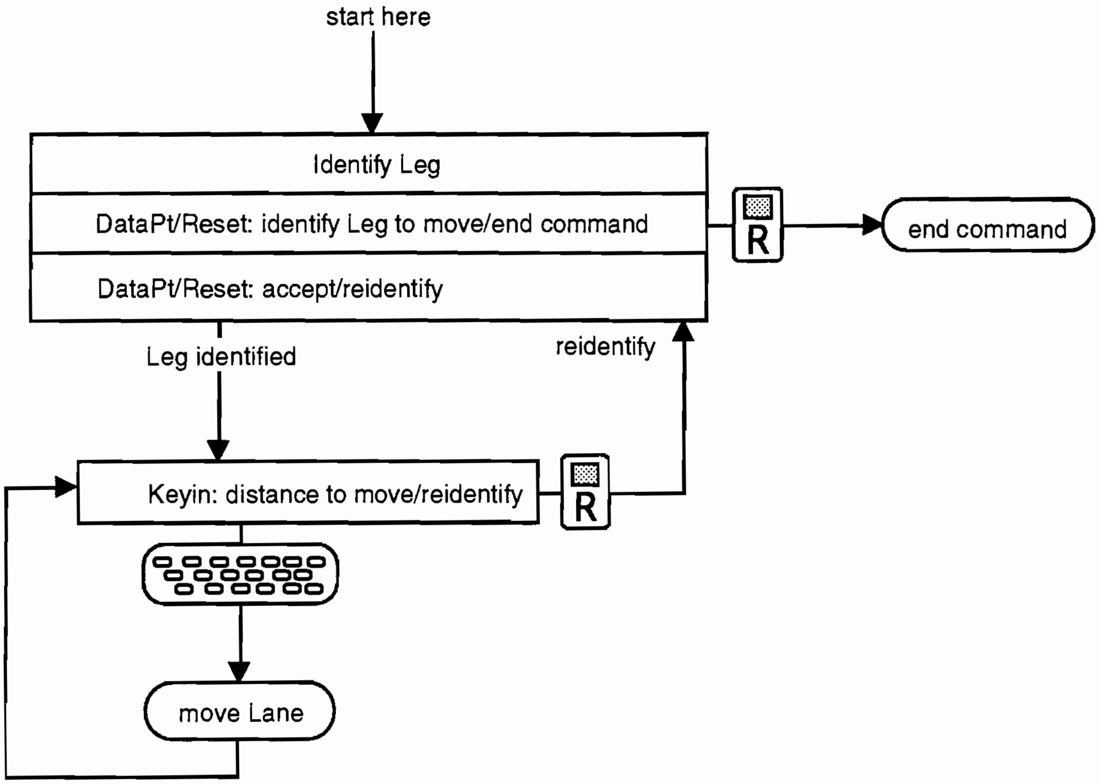
Shift a Lane laterally by a specified distance. The direction of movement is determined by the location of the move DataPt with respect to the Leg centerline. The radii of any arcs on the Lane edges will be adjusted by the amount of the shift. This is the same as **Primitive Command: MODIFY - LANE - LATERAL SHIFT**.

Primitive Command: MOVE - LANE - LONGITUDINAL


start here
identify Lane
DataPt/Reset: identify Lane to shorten/end command
DataPt/Reset: accept/reidentify
R
end command
Lane identified
reidentify
Keyin/Reset: amount to shorten/reidentify
R
REENTER
DataPt/Reset: shorten Lane/reidentify
R
D
shorten Lane

Shorten the end of a Lane that is nearest the intersection center. This is the same as **Primitive Command: MODIFY - LANE - SHORTEN**

Primitive Command: MOVE - LEG - LATERAL

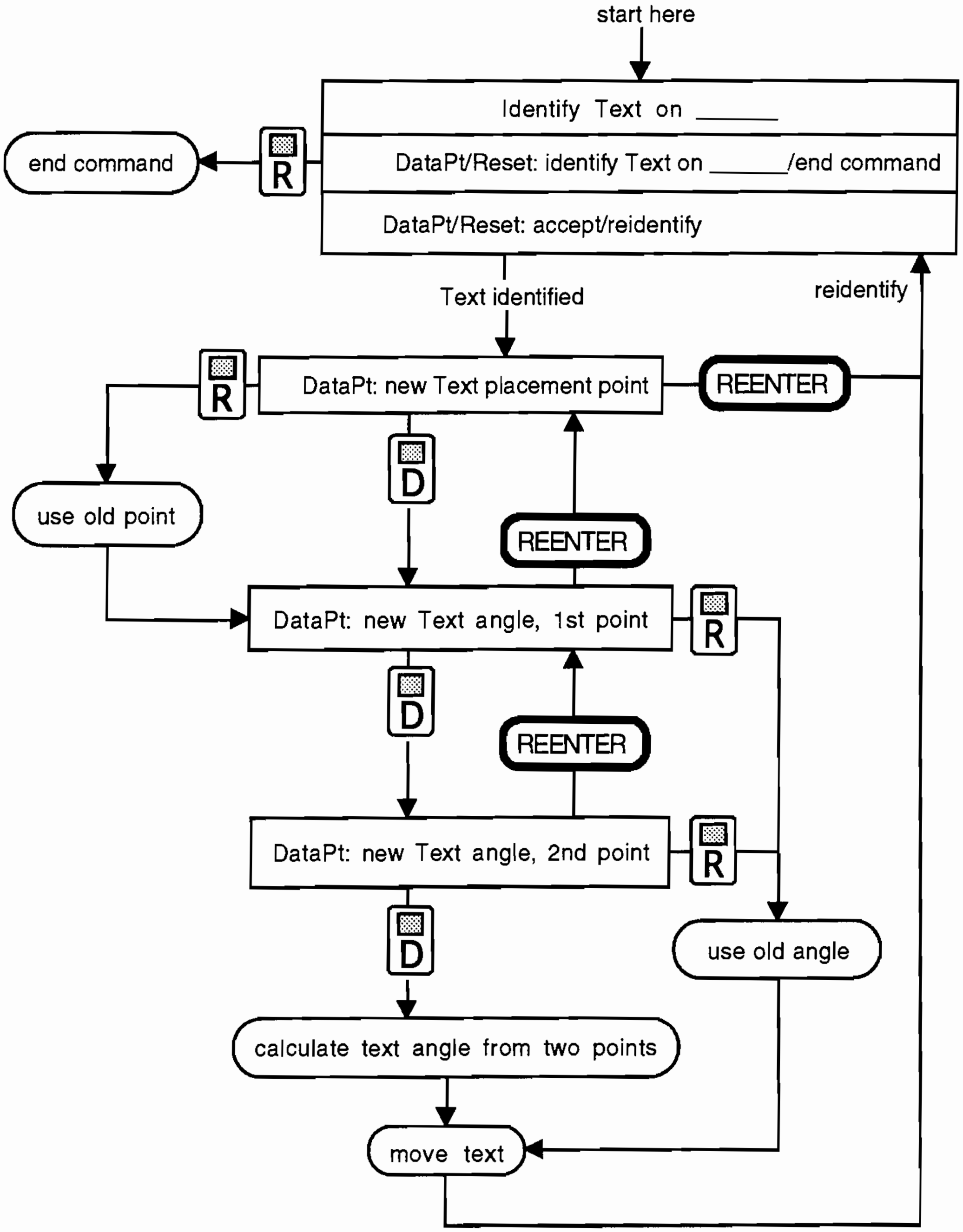

start here
Identify Leg
DataPt/Reset: identify Leg to move/end command
DataPt/Reset: accept/reidentify
R
end command
Leg identified
reidentify
Keyin: distance to move/reidentify
R
move Lane

Move a Leg laterally by a specified distance. The direction of movement is determined by the arithmetic sign of the distance. Plus will move to the right when facing in the direction of inbound traffic.

Primitive Command: MOVE - SIGN
Not programmed yet.

Primitive Command: MOVE - SIGNAL FACE
Not programmed yet.

Primitive Command: MOVE - TEXT


start here
Identify Text on ______
DataPt/Reset: identify Text on ______/end command
DataPt/Reset: accept/reidentify
end command
R
Text identified
reidentify
DataPt: new Text placement point
REENTER
R
use old point
D
REENTER
DataPt: new Text angle, 1st point
R
D
REENTER
DataPt: new Text angle, 2nd point
R
D
use old angle
calculate text angle from two points
move text

Use this diagram for the 2 **MOVE - TEXT** commands that follow.

Primitive Command: MOVE - TEXT ON ALT
Move an existing Text on an Alternative. The rotation angle may also be changed.

Primitive Command: MOVE - TEXT ON SEG
Move an existing Text on a Seg. The rotation angle may also be changed.

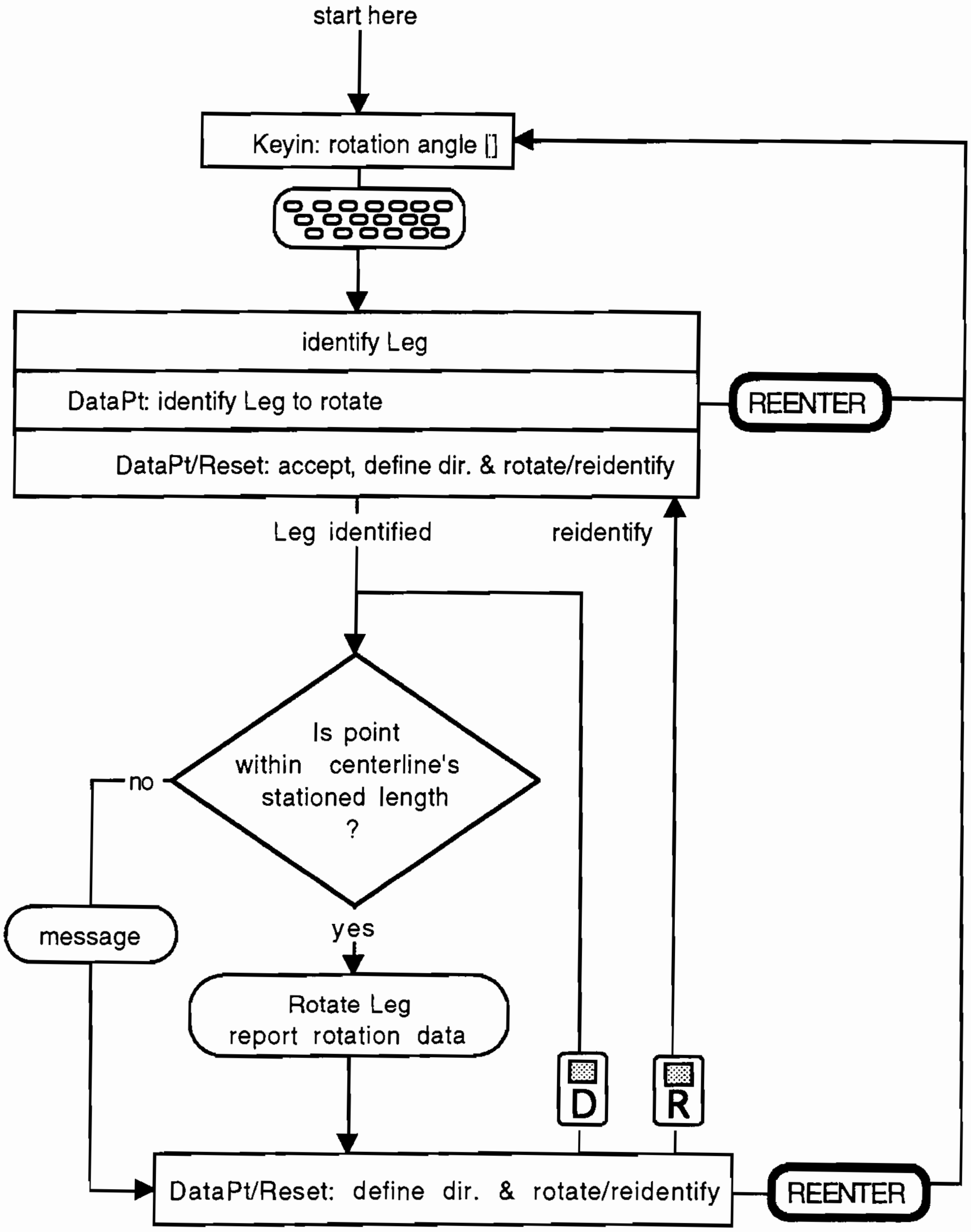
Transient Command: No
In reply to a prompt requesting a "yes" or "no" response, send "no" to IGIDS. This is the same as entering "no" through the keyboard.

Transient Command: Noun-Verb or Verb-Noun
This command toggles between the two methods of command processing. The **Noun-Verb** method always uses the selected IGIDS Object as the default choice when identifying an Object for processing. The **Verb-Noun** method always prompts the user to identify an Object for processing.

Transient Command: Reenter Data
In an IGIDS command, move backward in the processing sequence to where IGIDS most recently prompted for a Keyin of data and reprompt for the data.

Primitive Command: ROTATE - ALTERNATIVE
Not programmed yet.

Primitive Command: ROTATE - LEG

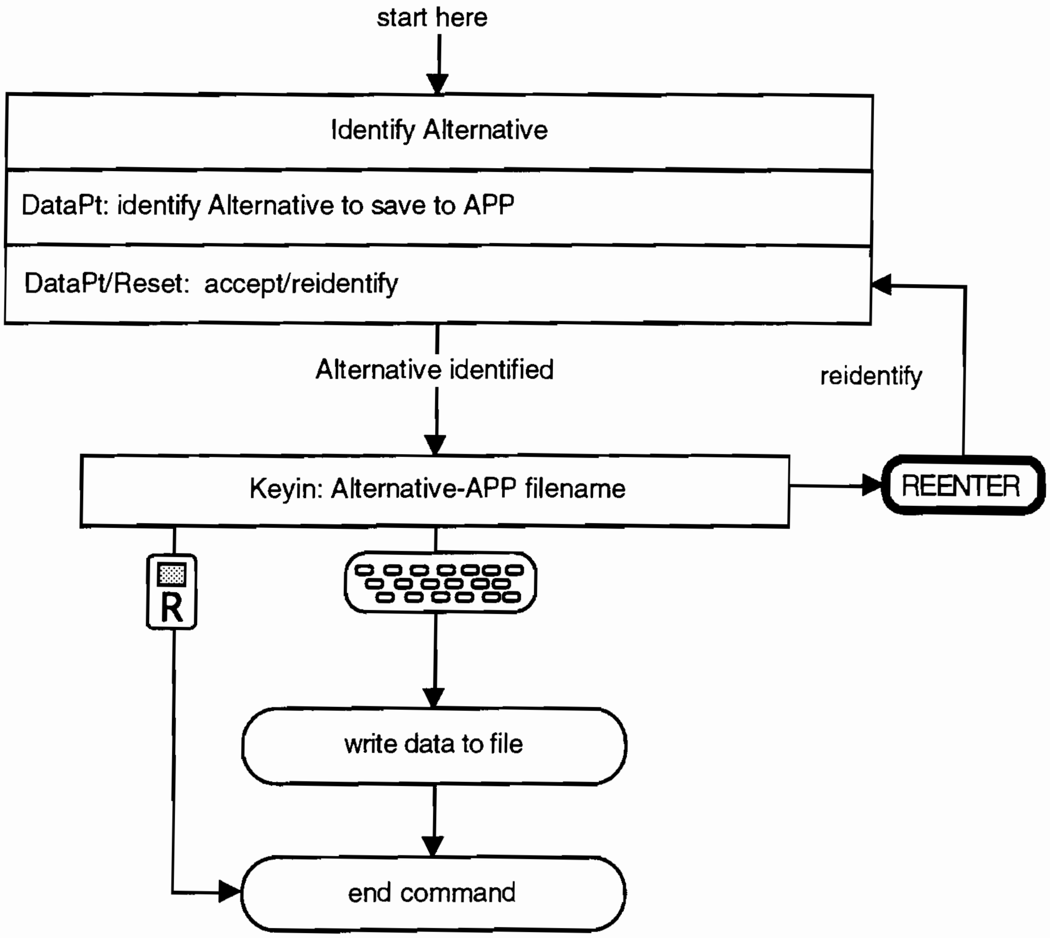

start here
Keyin: rotation angle []
identify Leg
DataPt: identify Leg to rotate
DataPt/Reset: accept, define dir. & rotate/reidentify
REENTER
Leg identified
reidentify
Is point within centerline's stationed length ?
no
yes
message
Rotate Leg report rotation data
D
R
DataPt/Reset: define dir. & rotate/reidentify
REENTER

Rotate a Leg through a specified angle. The direction of rotation is determined by the direction of the rotate DataPt with respect to the Leg centerline.

Primitive Command: **ROTATE - TEXT ON ALT**
Not programmed yet.

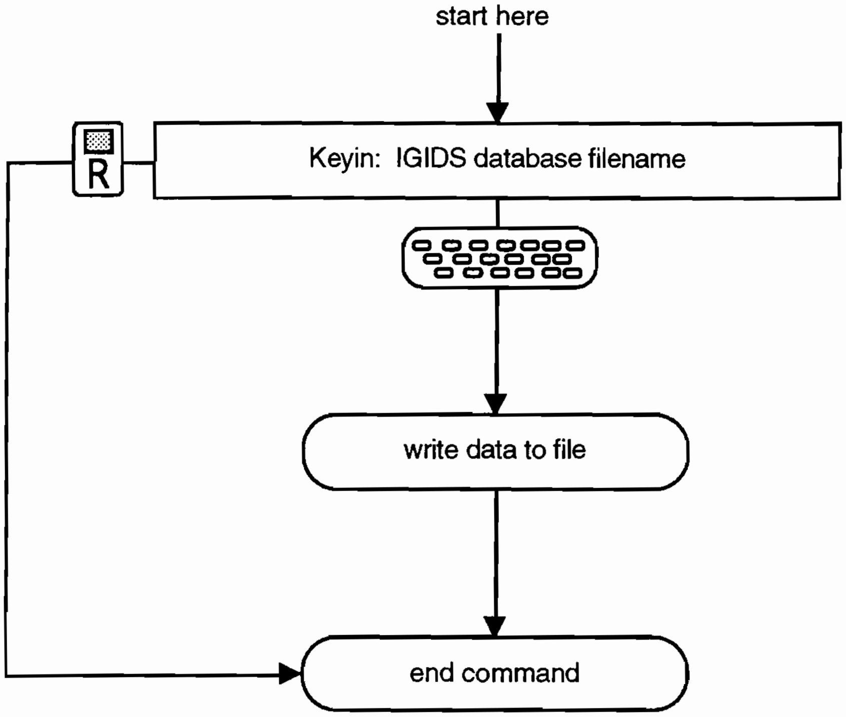
Primitive Command: **ROTATE - TEXT ON SEG**
Not programmed yet.

Primitive Command: SAVE TO - AutoPlanPrep


start here
Identify Alternative
DataPt: identify Alternative to save to APP
DataPt/Reset: accept/reidentify
Alternative identified
reidentify
Keyin: Alternative-APP filename
REENTER
R
write data to file
end command

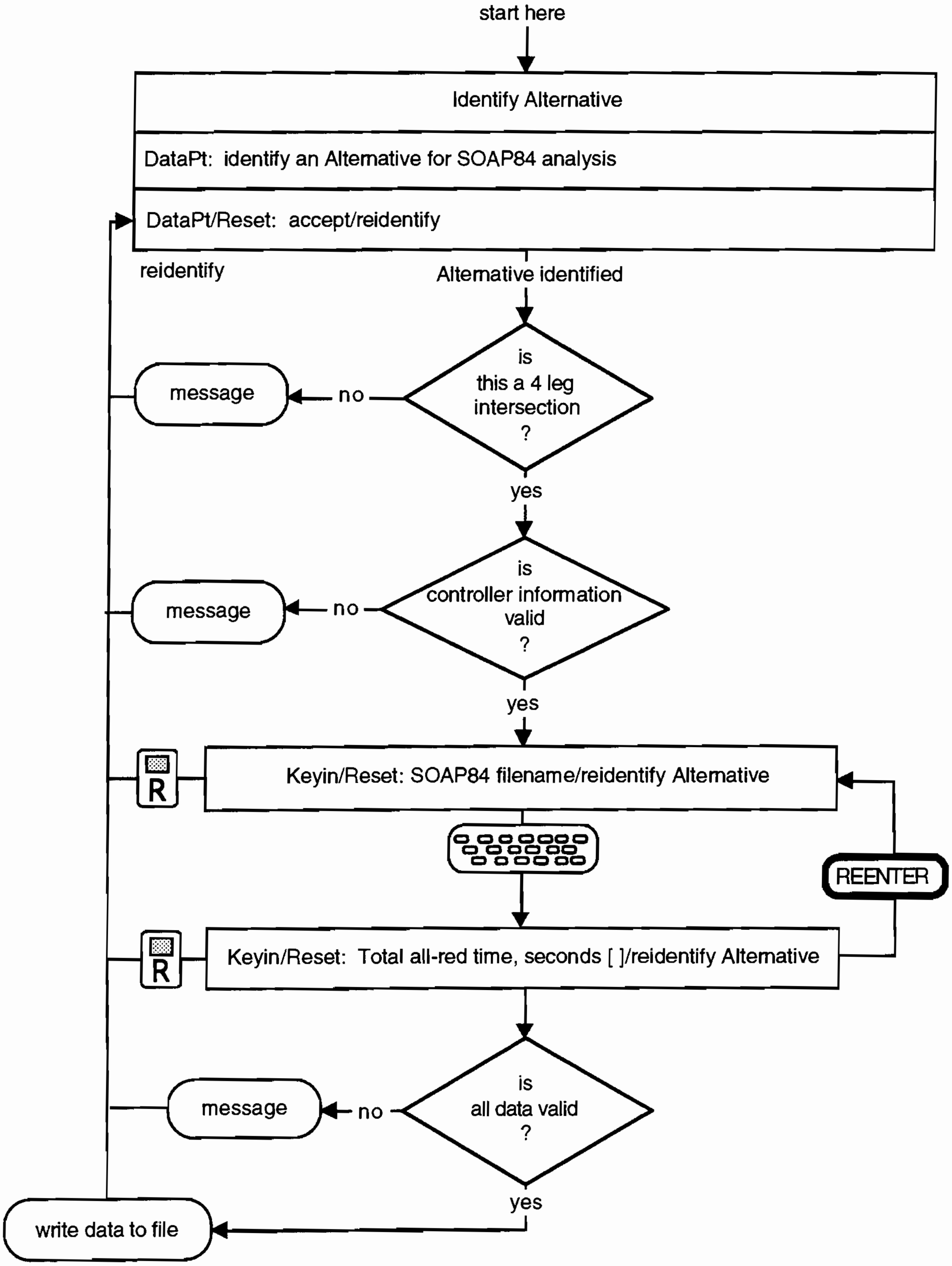
Save data from one Alternative into a file that can be read by TX-DOT's Automatic Plan Preparation software.

Primitive Command: SAVE TO - Data Base



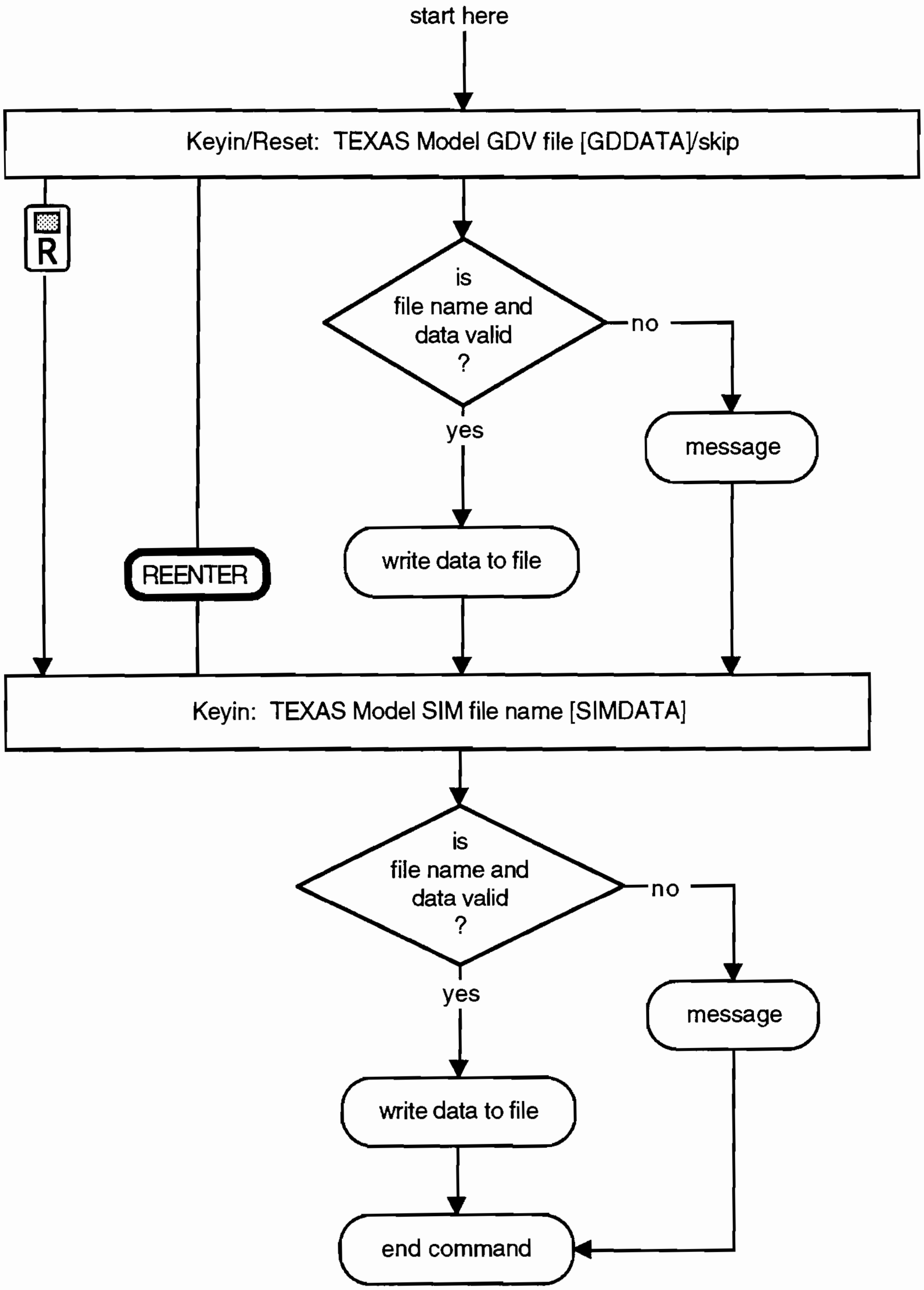
Save data from all Alternatives into a file that can be read by IGIDS. This will save all IGIDS data from the current session. Data saved by this command can be read by **Primitive Command: LOAD FROM - DATABASE.**

Primitive Command: SAVE TO - SOAP



Save data from one Alternative into a file that can be read by the Signal Operations Analysis Package software.

Primitive Command: SAVE TO - TX Mdl file

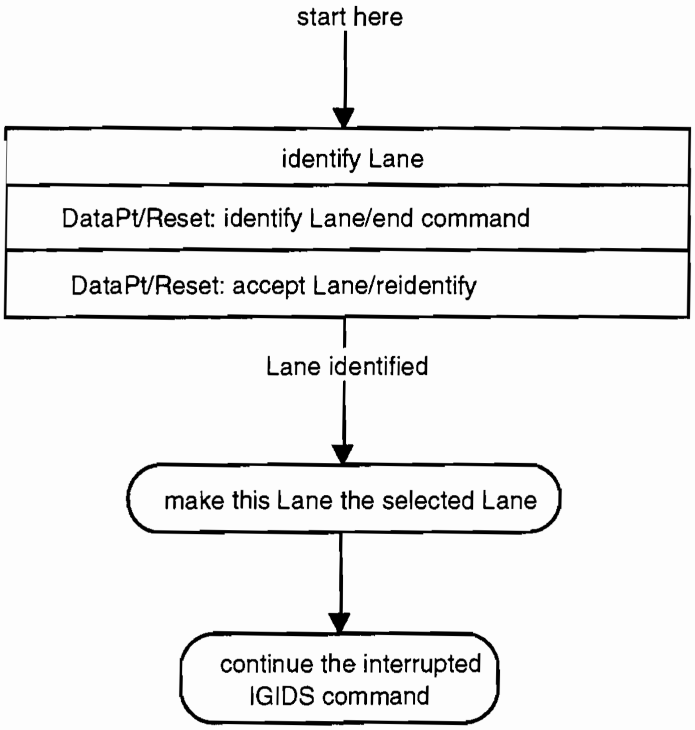


Save data from the currently selected Alternative into files that can be read by the TEXAS Model for Intersection Traffic. The first file will be readable by the TEXAS Model processor called GDVDATA. The second file will be readable by the TEXAS Model processor called SIMDATA.

Transient Command: SELECT - ALTERNATIVE - NEXT
Make the next Alternative in the selection list the selected Seg

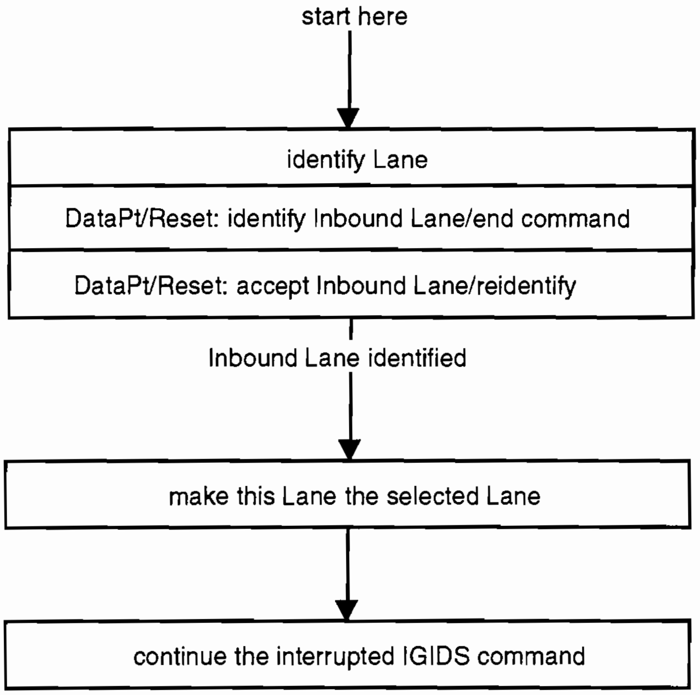
Transient Command: SELECT - ALTERNATIVE - PREVIOUS
Make the previous Alternative in the selection list the selected Seg

Temporary Command: SELECT - LANE - BY DATA PT


start here
identify Lane
DataPt/Reset: identify Lane/end command
DataPt/Reset: accept Lane/reidentify
Lane identified
make this Lane the selected Lane
continue the interrupted IGIDS command

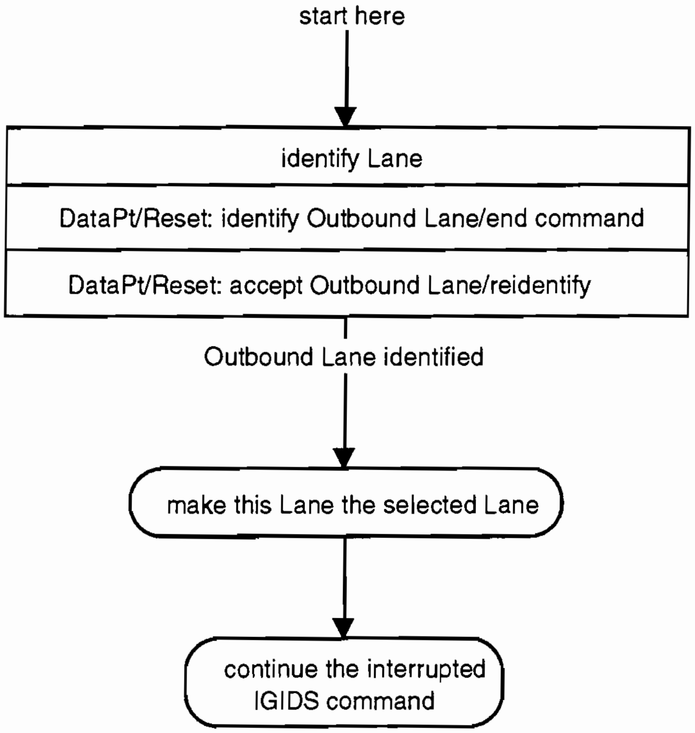
Identify a Lane to be the selected Lane.

Temporary Command: **SELECT - LANE - (BY DATA PT) INBOUND**


start here
identify Lane
DataPt/Reset: identify Inbound Lane/end command
DataPt/Reset: accept Inbound Lane/reidentify
Inbound Lane identified
make this Lane the selected Lane
continue the interrupted IGIDS command

Identify an Inbound Lane to be the selected Lane.

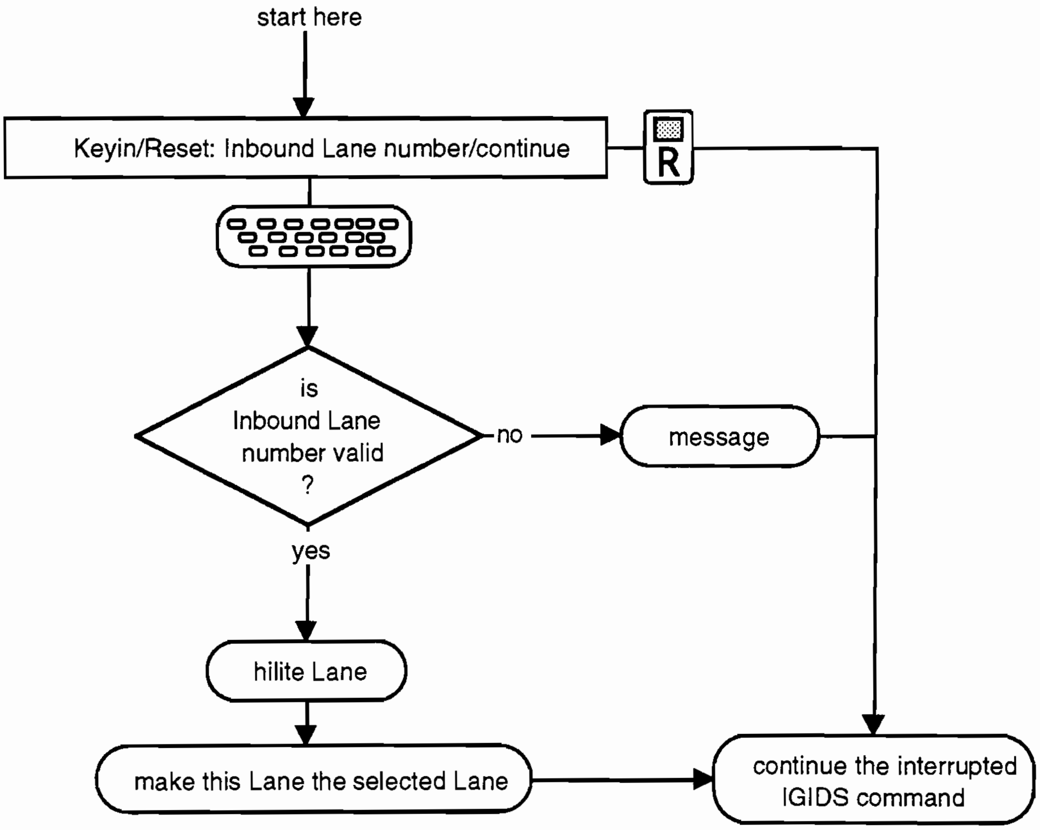
Temporary Command: SELECT - LANE - (BY DATA PT) OUTBOUND


start here
identify Lane
DataPt/Reset: identify Outbound Lane/end command
DataPt/Reset: accept Outbound Lane/reidentify
Outbound Lane identified
make this Lane the selected Lane
continue the interrupted IGIDS command

Identify an Outbound Lane to be the selected Lane.

Transient Command: SELECT - LANE - CURRENT
Hilite the selected Lane.

Temporary Command: **SELECT - LANE - INBOUND ID**


start here
Keyin/Reset: Inbound Lane number/continue
R
is Inbound Lane number valid ?
no
message
yes
hilite Lane
make this Lane the selected Lane
continue the interrupted IGIDS command

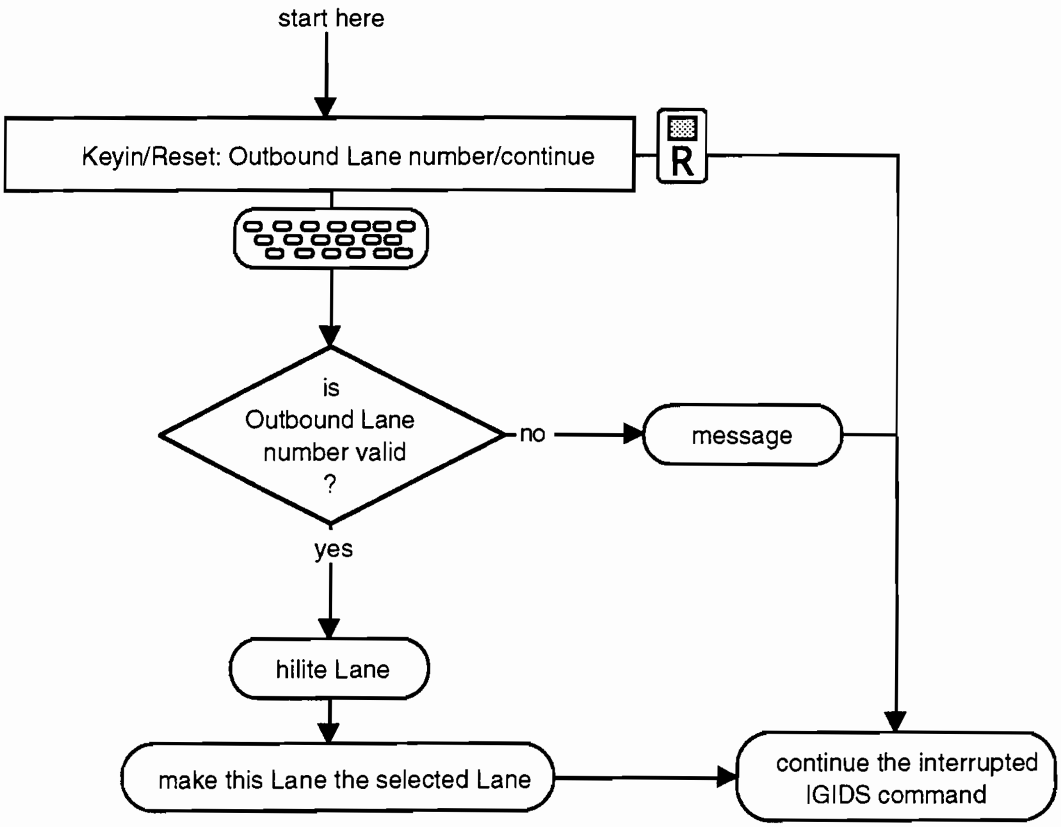
Specify the ID number of an Inbound Lane on the selected Leg to be the selected Lane.

Transient Command: SELECT - LANE - NEXT
Make the Lane with the next higher ID number the selected Lane. If the currently selected lane has the highest ID number, make the opposite direction Lane with the ID number 1 the selected Lane.

Transient Command: SELECT - LANE - (NEXT) INBOUND
Make the Inbound Lane with the next higher ID number the selected Lane

Transient Command: SELECT - LANE - (NEXT) OUTBOUND
Make the Outbound Lane with the next higher ID number the selected Lane

Temporary Command: **SELECT - LANE - OUTBOUND ID**


start here
Keyin/Reset: Outbound Lane number/continue
R
is Outbound Lane number valid ?
no
message
yes
hilite Lane
make this Lane the selected Lane
continue the interrupted IGIDS command

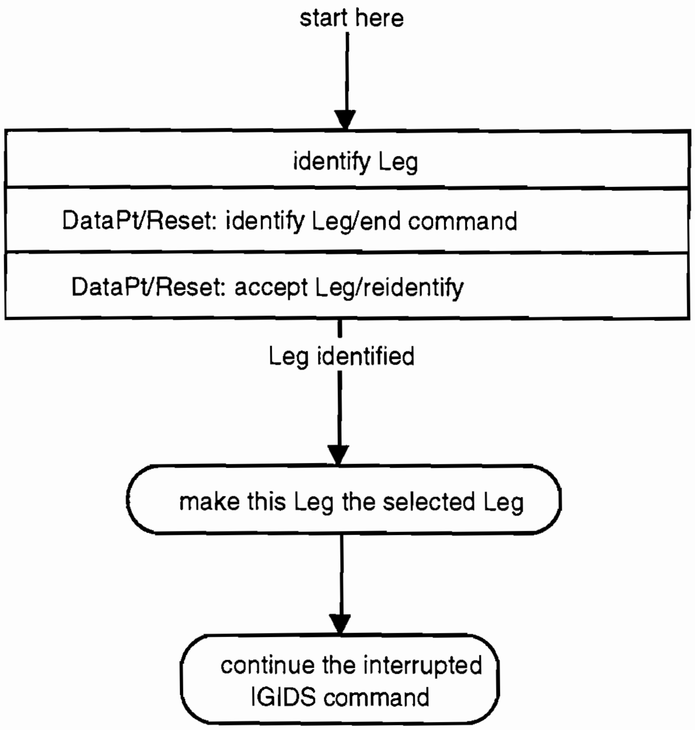
Specify the ID number of an Outbound Lane on the selected Leg to be the selected Lane.

Transient Command: SELECT - LANE - PREVIOUS
Make the Lane with the next lower ID number the selected Lane. If the currently selected lane has ID number 1, make the opposite direction Lane with the highest ID number the selected Lane.

Transient Command: SELECT - LANE - (PREVIOUS) INBOUND
Make the Inbound Lane with the next lower ID number the selected Lane

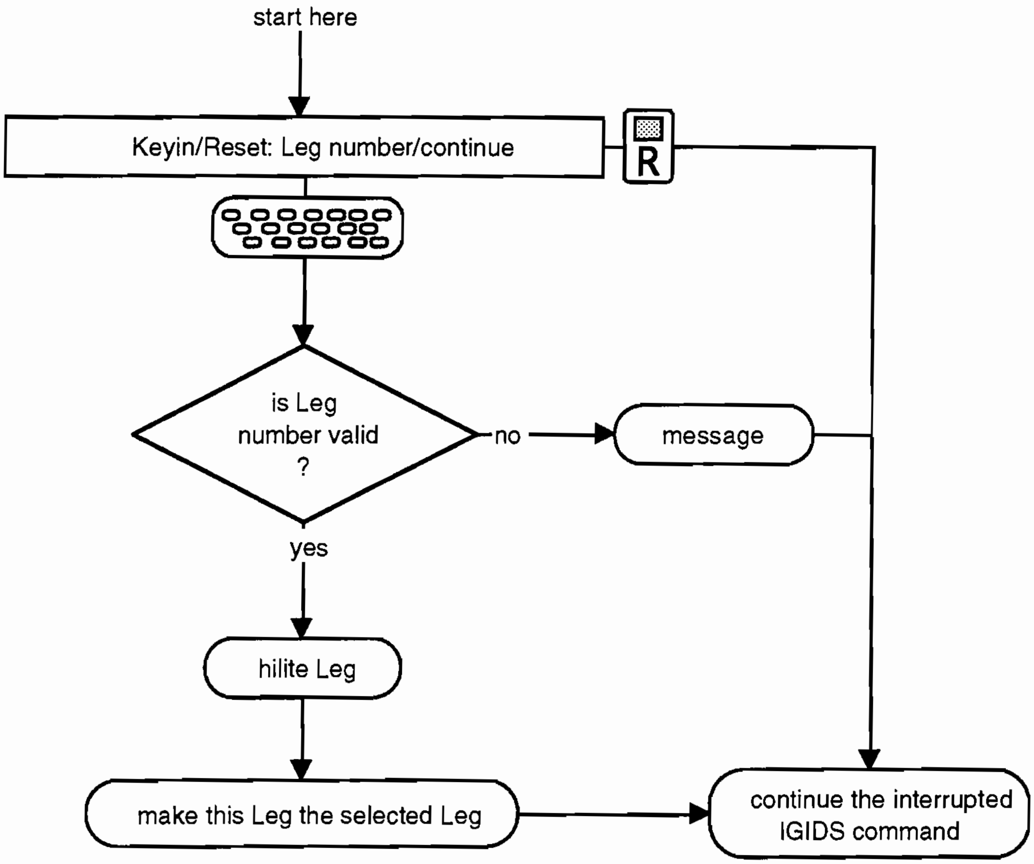
Transient Command: SELECT - LANE - (PREVIOUS) OUTBOUND
Make the Outbound Lane with the next lower ID number the selected Lane

Temporary Command: **SELECT - LEG - BY DATA PT**


start here
identify Leg
DataPt/Reset: identify Leg/end command
DataPt/Reset: accept Leg/reidentify
Leg identified
make this Leg the selected Leg
continue the interrupted IGIDS command

Identify a Leg to be the selected Leg.

Temporary Command: **SELECT - LEG - BY ID**


start here
Keyin/Reset: Leg number/continue
R
is Leg number valid ?
no
message
yes
hilite Leg
make this Leg the selected Leg
continue the interrupted IGIDS command

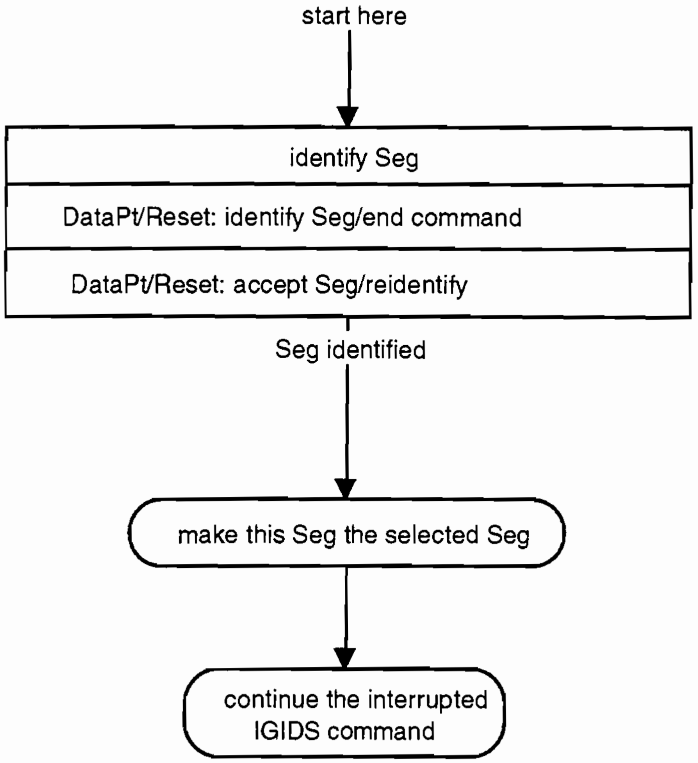
Specify the ID number of a Leg in the selected Alternative to be the selected Leg.

Transient Command: **SELECT - LEG- CURRENT**
Hilite the selected Leg.

Transient Command: **SELECT - LEG - NEXT**
Make the Leg that is nearest to the selected Leg in a clockwise direction the selected Leg.

Transient Command: **SELECT - LEG - PREVIOUS**
Make the Leg that is nearest to the selected Leg in a counterclockwise direction the selected Leg.

Temporary Command: **SELECT - SEG - BY DATA PT**


start here
identify Seg
DataPt/Reset: identify Seg/end command
DataPt/Reset: accept Seg/reidentify
Seg identified
make this Seg the selected Seg
continue the interrupted IGIDS command

Identify the Seg to become the selected Seg

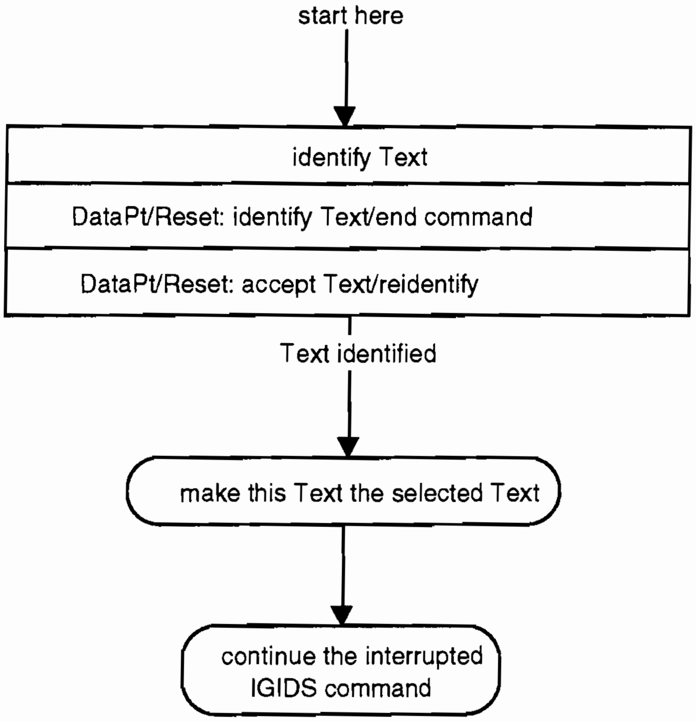
Temporary Command: **SELECT - SEG - BY ID**
Specify the ID number of the Seg to become the selected Seg

Transient Command: **SELECT - SEG - CURRENT**
Hilite the selected Seg

Transient Command: **SELECT - SEG - NEXT**
Make the next Seg in the selection list the selected Seg

Transient Command: **SELECT - SEG - PREVIOUS**
Make the previous Seg in the selection list the selected Seg

Temporary Command: SELECT - TEXT - BY DATA PT


start here
identify Text
DataPt/Reset: identify Text/end command
DataPt/Reset: accept Text/reidentify
Text identified
make this Text the selected Text
continue the interrupted IGIDS command

Identify the Text to become the selected Text

Temporary Command: SELECT - TEXT - BY ID
Specify the ID number of the Text to become the selected Text

Transient Command: SELECT - TEXT - CURRENT
Hilite the selected Text

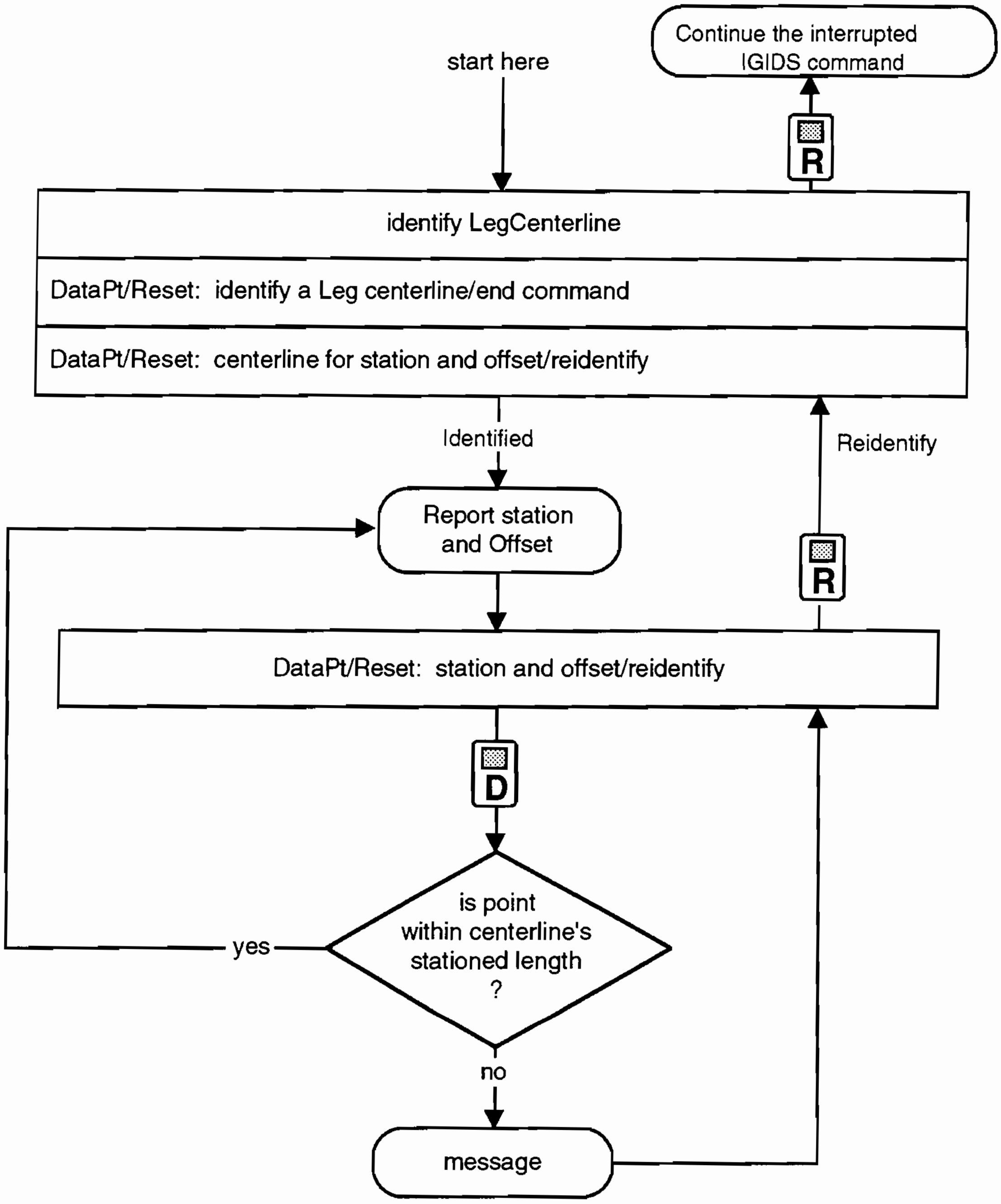
Transient Command: SELECT - TEXT - NEXT
Make the next Text in the selection list the selected Text

Transient Command: SELECT - TEXT - PREVIOUS
Make the previous Text in the selection list the selected Text

Transient Command: SHOW INFO - FULL
Show a more complete description of the currently selected objects in a window.

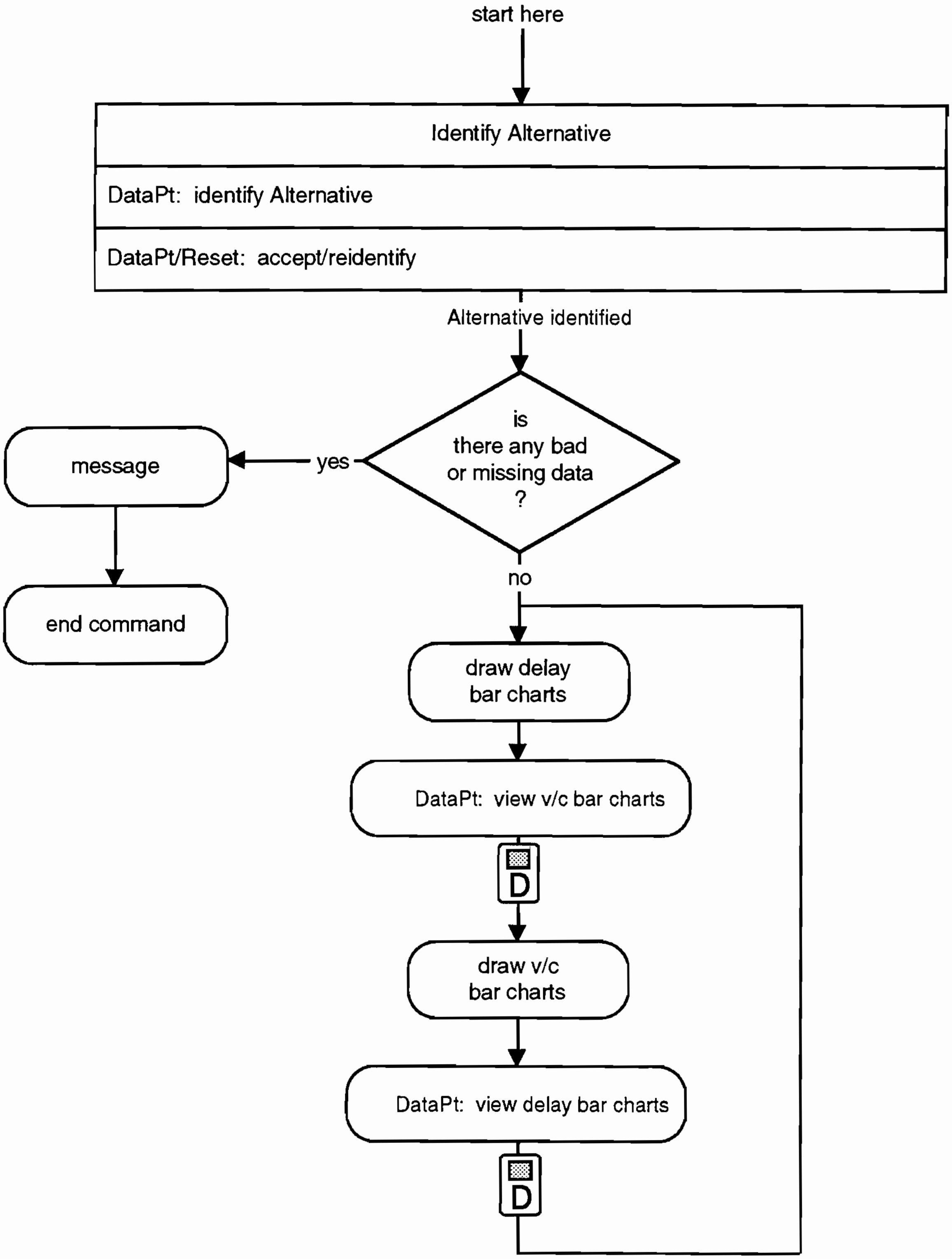
Transient Command: SHOW INFO - SHORT
Provide a single line of information about the currently selected Objects. This line will list the current ID number of the selected Alternative, Leg and Lane.

Temporary Command: Sta/Offset


start here
Continue the interrupted IGIDS command
R
identify LegCenterline
DataPt/Reset: identify a Leg centerline/end command
DataPt/Reset: centerline for station and offset/reidentify
Identified
Reidentify
Report station and Offset
R
DataPt/Reset: station and offset/reidentify
D
is point within centerline's stationed length ?
yes
no
message

Report the station and offset of a point, based on the stationing of an identified Leg centerline.

Primitive Command: TOOLS - HighCapMan - Chapter 9

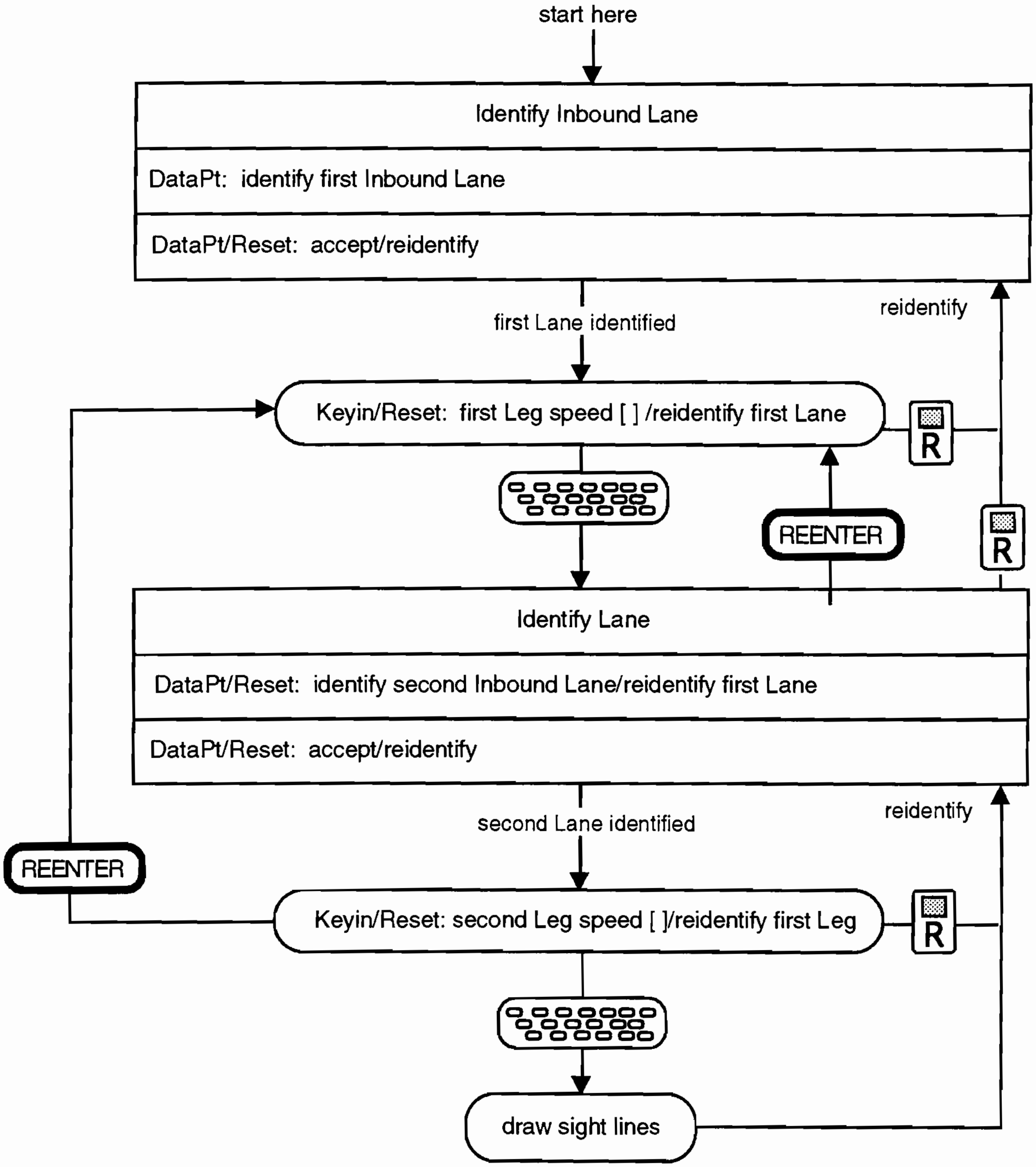

start here
Identify Alternative
DataPt: identify Alternative
DataPt/Reset: accept/reidentify
Alternative identified
is there any bad or missing data ?
yes
message
end command
no
draw delay bar charts
DataPt: view v/c bar charts
D
draw v/c bar charts
DataPt: view delay bar charts
D

A partial implementation of the procedures described in Chapter 9 of the Highway Capacity Manual. Determination of the critical lane groups for the intersection is not implemented

Transient Command: TOOLS - HighCapMan - Del Graphics
Delete all of the Highway Capacity Manual v/c and delay bar charts.

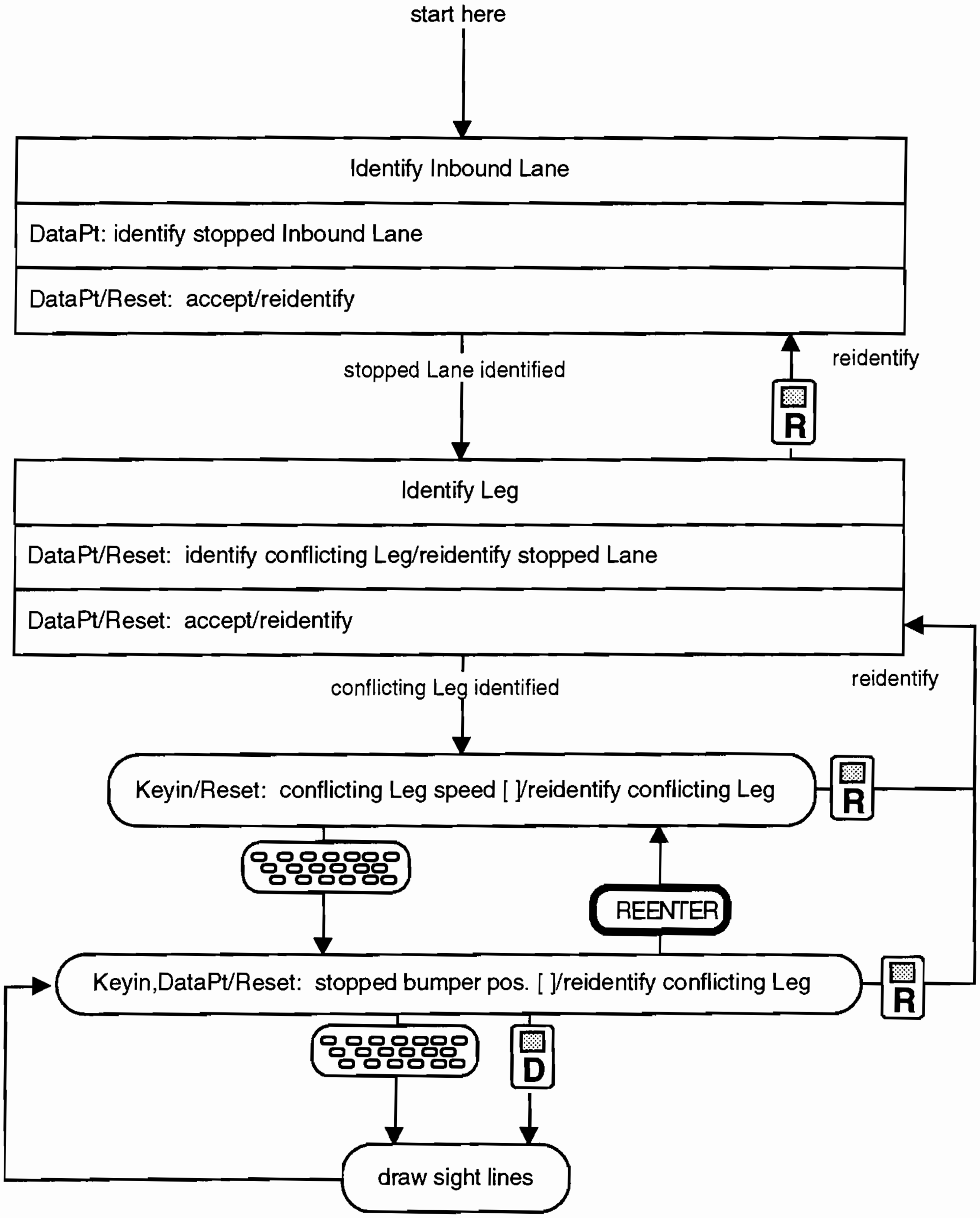
Transient Command: TOOLS - Sight Dist - Del Graphics
Delete all of the Sight Distance graphics.

Primitive Command: TOOLS - Sight Dist - No Control


start here
Identify Inbound Lane
DataPt: identify first Inbound Lane
DataPt/Reset: accept/reidentify
first Lane identified
reidentify
Keyin/Reset: first Leg speed [ ] /reidentify first Lane
R
REENTER
R
Identify Lane
DataPt/Reset: identify second Inbound Lane/reidentify first Lane
DataPt/Reset: accept/reidentify
second Lane identified
reidentify
REENTER
Keyin/Reset: second Leg speed [ ]/reidentify first Leg
R
draw sight lines

Draw sight line graphics for two approaches of an uncontrolled intersection.

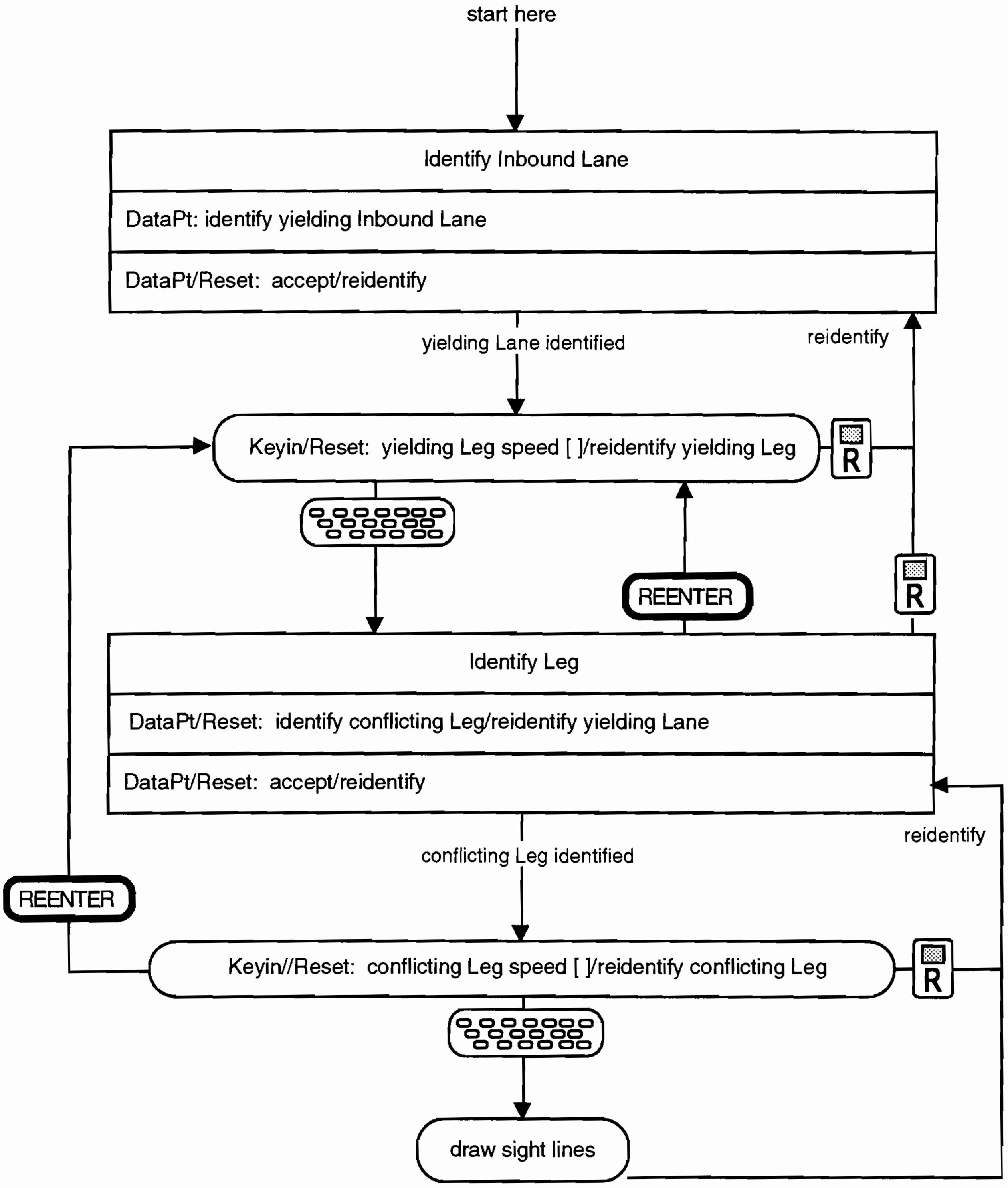
Primitive Command: TOOLS - Sight Dist - Stopped



start here
Identify Inbound Lane
DataPt: identify stopped Inbound Lane
DataPt/Reset: accept/reidentify
stopped Lane identified
reidentify
R
Identify Leg
DataPt/Reset: identify conflicting Leg/reidentify stopped Lane
DataPt/Reset: accept/reidentify
conflicting Leg identified
reidentify
Keyin/Reset: conflicting Leg speed [ ]/reidentify conflicting Leg
R
REENTER
Keyin,DataPt/Reset: stopped bumper pos. [ ]/reidentify conflicting Leg
R
D
draw sight lines

Draw sight line graphics for the intersection of a stop sign controlled approach and an uncontrolled approach.

Primitive Command: TOOLS - Sight Dist - Yield

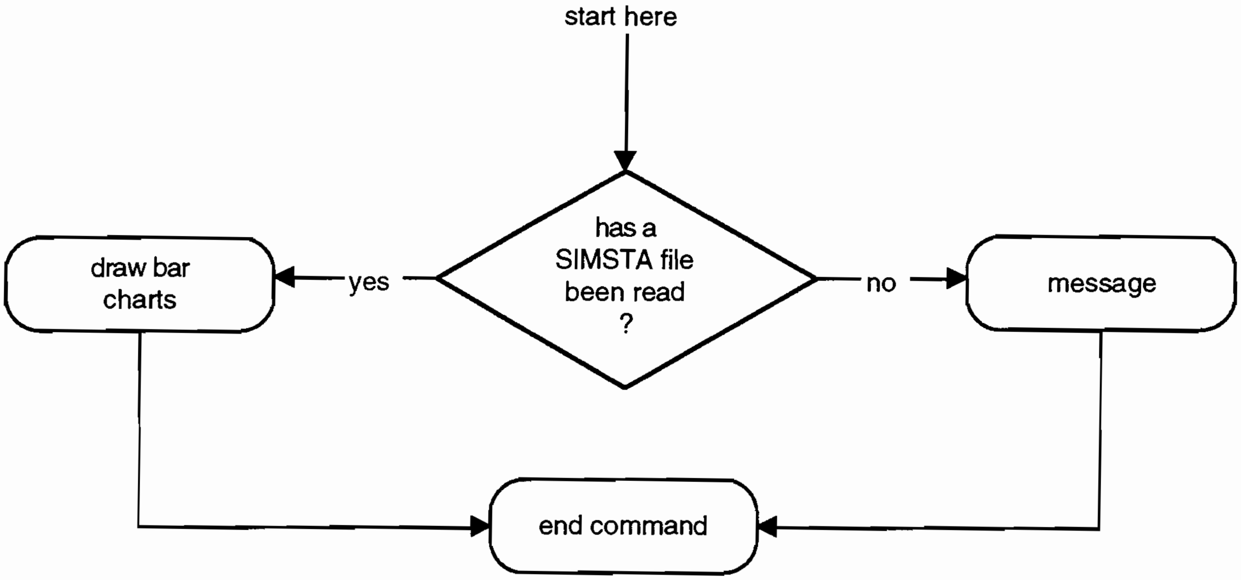

start here
Identify Inbound Lane
DataPt: identify yielding Inbound Lane
DataPt/Reset: accept/reidentify
yielding Lane identified
reidentify
Keyin/Reset: yielding Leg speed [ ]/reidentify yielding Leg
R
REENTER
R
Identify Leg
DataPt/Reset: identify conflicting Leg/reidentify yielding Lane
DataPt/Reset: accept/reidentify
conflicting Leg identified
reidentify
REENTER
Keyin//Reset: conflicting Leg speed [ ]/reidentify conflicting Leg
R
draw sight lines

Draw sight line graphics for the intersection of a yield sign controlled approach and an uncontrolled approach.

Primitive Command: TOOLS - TEXAS Model - Animation
Not programmed yet.

Transient Command: TOOLS - TEXAS Model - Del Graphics
Delete all of the TEXAS Model statistics bar charts.

Primitive Command: TOOLS - TEXAS Model - Graph



Use this diagram for the 14 **TOOLS - TEXAS Model - Graph** commands that follow.

Primitive Command: TOOLS - TEXAS Model - Graph ADMPH
Draw bar charts showing the Average Delay Below 10 MPH statistics from a TEXAS Model analysis.

Primitive Command: TOOLS - TEXAS Model - Graph AQD
Draw bar charts showing the Average Queue Delay statistics from a TEXAS Model analysis.

Primitive Command: TOOLS - TEXAS Model - Graph ASD
Draw bar charts showing the Average Stopped Delay statistics from a TEXAS Model analysis.

Primitive Command: TOOLS - TEXAS Model - Graph ATD
Draw bar charts showing the Average Total Delay statistics from a TEXAS Model analysis.

Primitive Command: TOOLS - TEXAS Model - Graph DMPH
Draw bar charts showing the Delay Below XX MPH statistics from a TEXAS Model analysis.

Primitive Command: TOOLS - TEXAS Model - Graph OADMPH
Draw bar charts showing the Overall Average Delay Below 10 MPH statistics from a TEXAS Model analysis.

Primitive Command: TOOLS - TEXAS Model - Graph OAQD
Draw bar charts showing the Overall Average Queue Delay statistics from a TEXAS Model analysis.

Primitive Command: TOOLS - TEXAS Model - Graph OASD
Draw bar charts showing the Overall Average Stopped Delay statistics from a TEXAS Model analysis.

Primitive Command: TOOLS - TEXAS Model - Graph OATD
Draw bar charts showing the Overall Average Total Delay statistics from a TEXAS Model analysis.

Primitive Command: TOOLS - TEXAS Model - Graph Probs
Draw bar chart showing the 95 percent confidence interval statistics from a TEXAS Model analysis.

Primitive Command: TOOLS - TEXAS Model - Graph QD
Draw bar charts showing the Queue Delay statistics from a TEXAS Model analysis.

Primitive Command: TOOLS - TEXAS Model - Graph Queues
Draw bar charts showing the Maximum and Average Queue Length statistics from a TEXAS Model analysis.

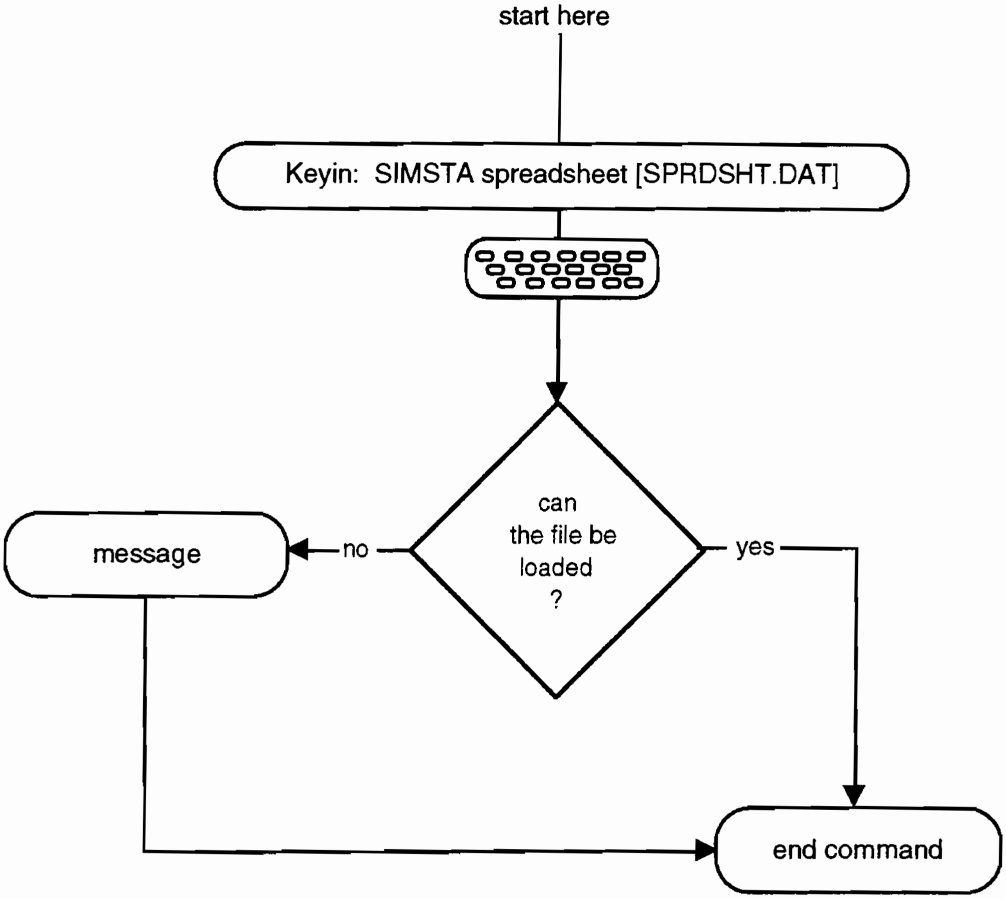
Primitive Command: TOOLS - TEXAS Model - Graph SD
Draw bar charts showing the Stopped Delay statistics from a TEXAS Model analysis.

Primitive Command: TOOLS - TEXAS Model - Graph TD
Draw bar charts showing the Total Delay statistics from a TEXAS Model analysis.

Primitive Command: TOOLS - TEXAS Model - Graph Turn %
Draw bar charts showing the Percent of Vehicles Making Turning Movements statistics from a TEXAS Model analysis.

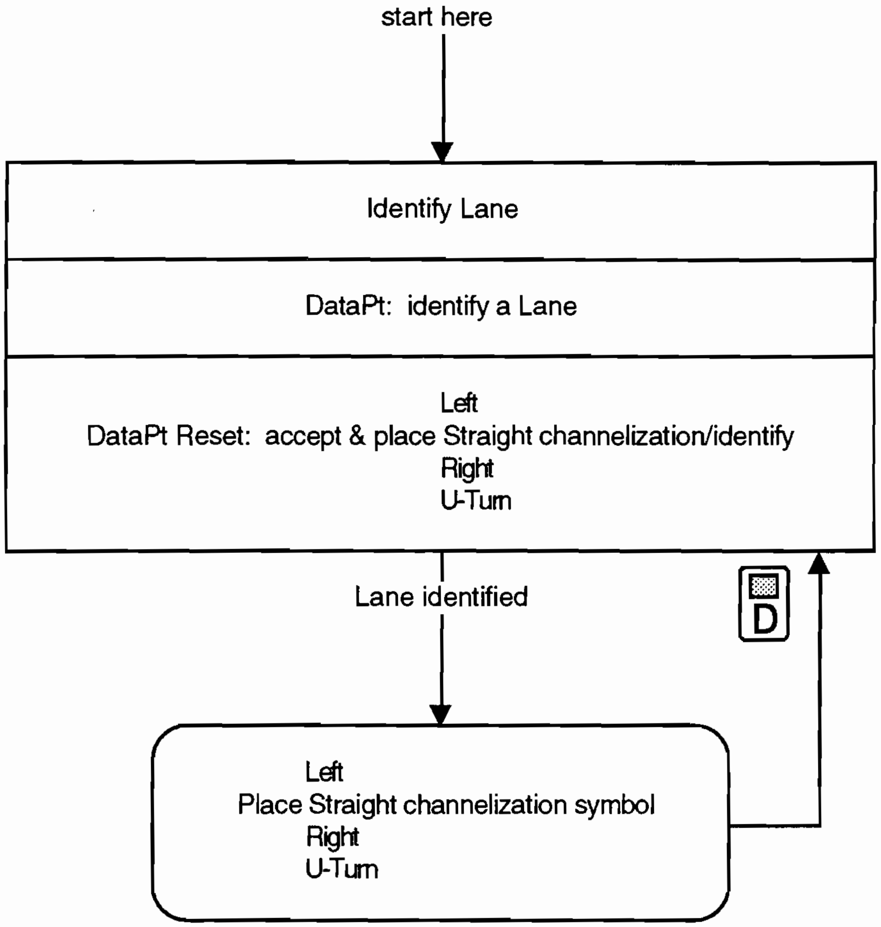
Primitive Command: TOOLS - TEXAS Model - Graph Volume
Draw bar charts showing the Volume Processed statistics from a TEXAS Model analysis.

Primitive Command: TOOLS - TEXAS Model - Load SIMSTA



Load statistical data from file created by the TEXAS Model. This load must be done before displaying statistical data from the TEXAS Model. It is the responsibility of the user to be sure that the statistics file matches the selected Alternative.

Primitive Command: TOOLS - Traffic - Channelize



start here
Identify Lane
DataPt: identify a Lane
Left
DataPt Reset: accept & place Straight channelization/identify
Right
U-Turn
Lane identified
D
Left
Place Straight channelization symbol
Right
U-Turn

Use this diagram for the 4 **TOOLS - Traffic - Channelize** commands that follow.

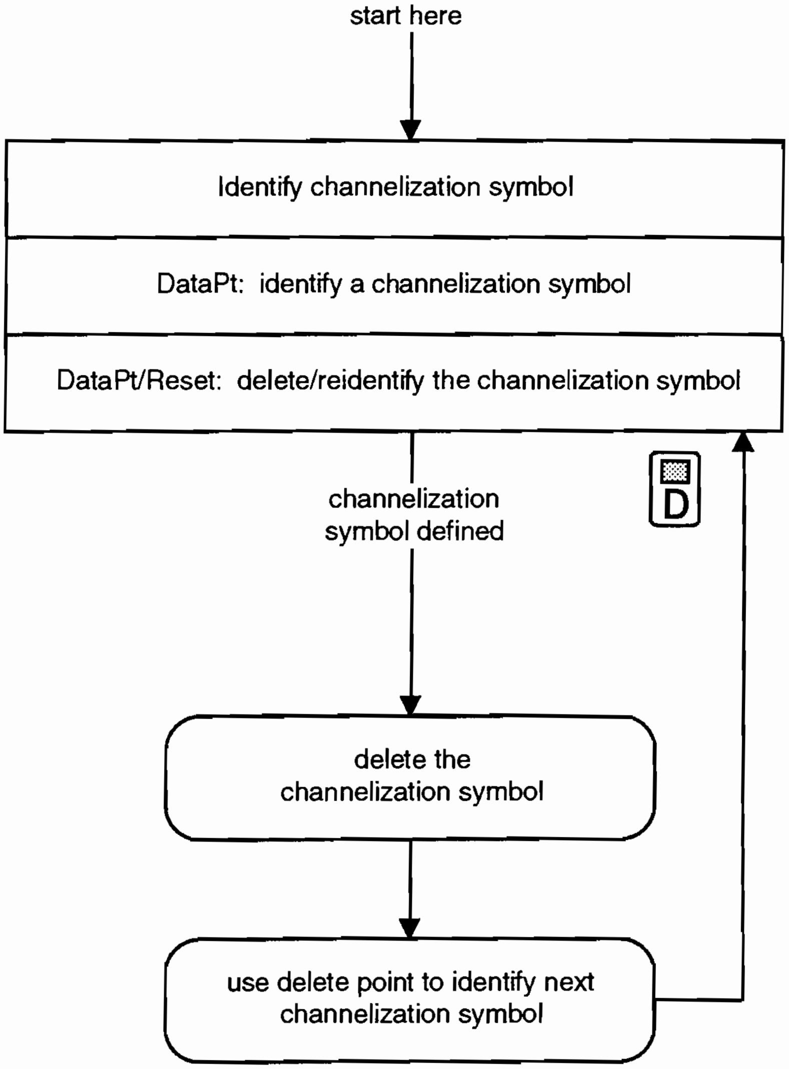
Primitive Command: TOOLS - Traffic - Channelize - Left
Add a left turn channelization symbol to a Lane.

Primitive Command: TOOLS - Traffic - Channelize - Right
Add a right turn channelization symbol to a Lane.

Primitive Command: TOOLS - Traffic - Channelize - Straight
Add a straight through channelization symbol to a Lane.

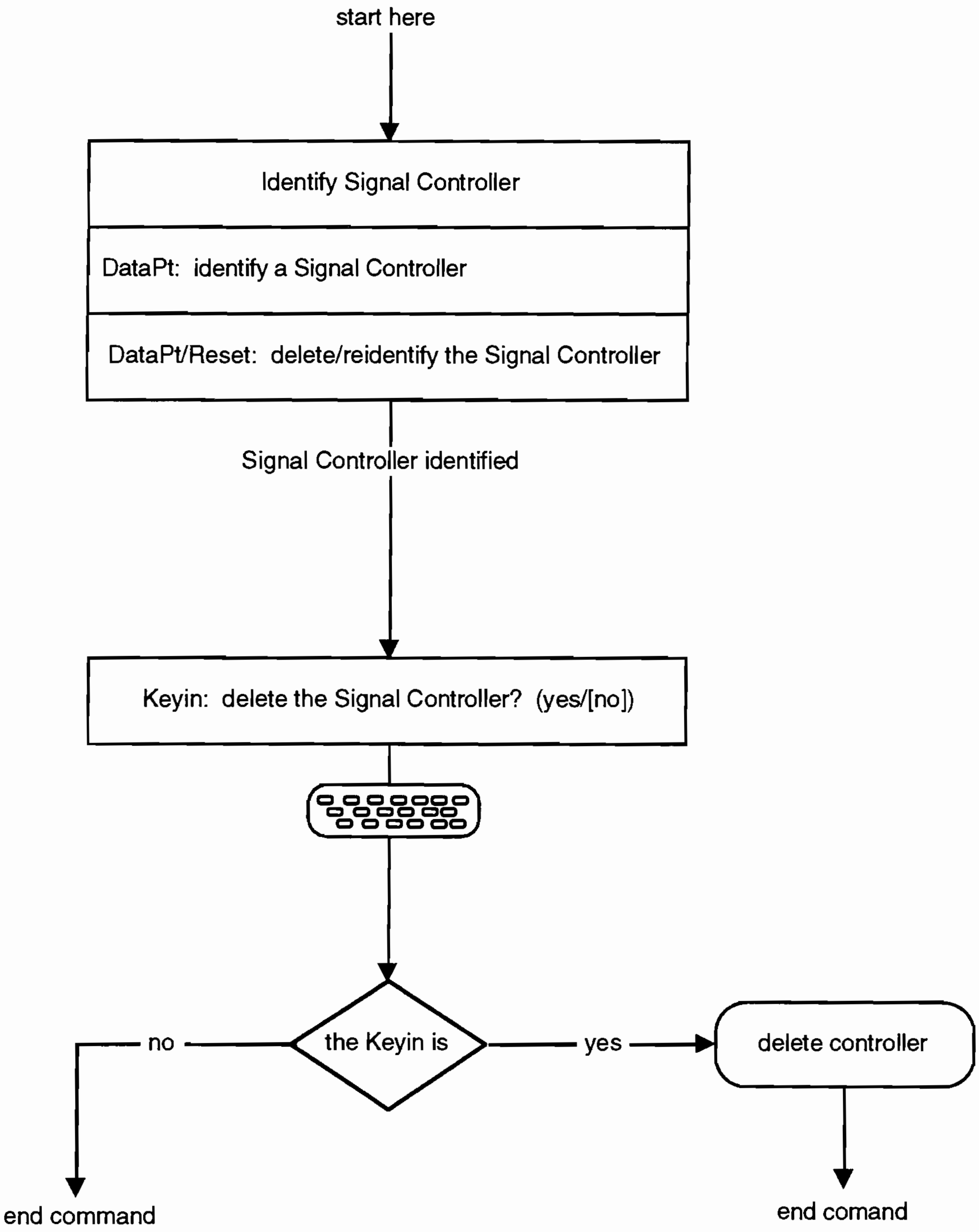
Primitive Command: TOOLS - Traffic - Channelize - U-Turn
Add a u-turn channelization symbol to a Lane.

Primitive Command: TOOLS - Traffic - Channelize - DELETE


start here
Identify channelization symbol
DataPt: identify a channelization symbol
DataPt/Reset: delete/reidentify the channelization symbol
channelization
symbol defined
D
delete the
channelization symbol
use delete point to identify next
channelization symbol

Delete an existing channelization symbol.

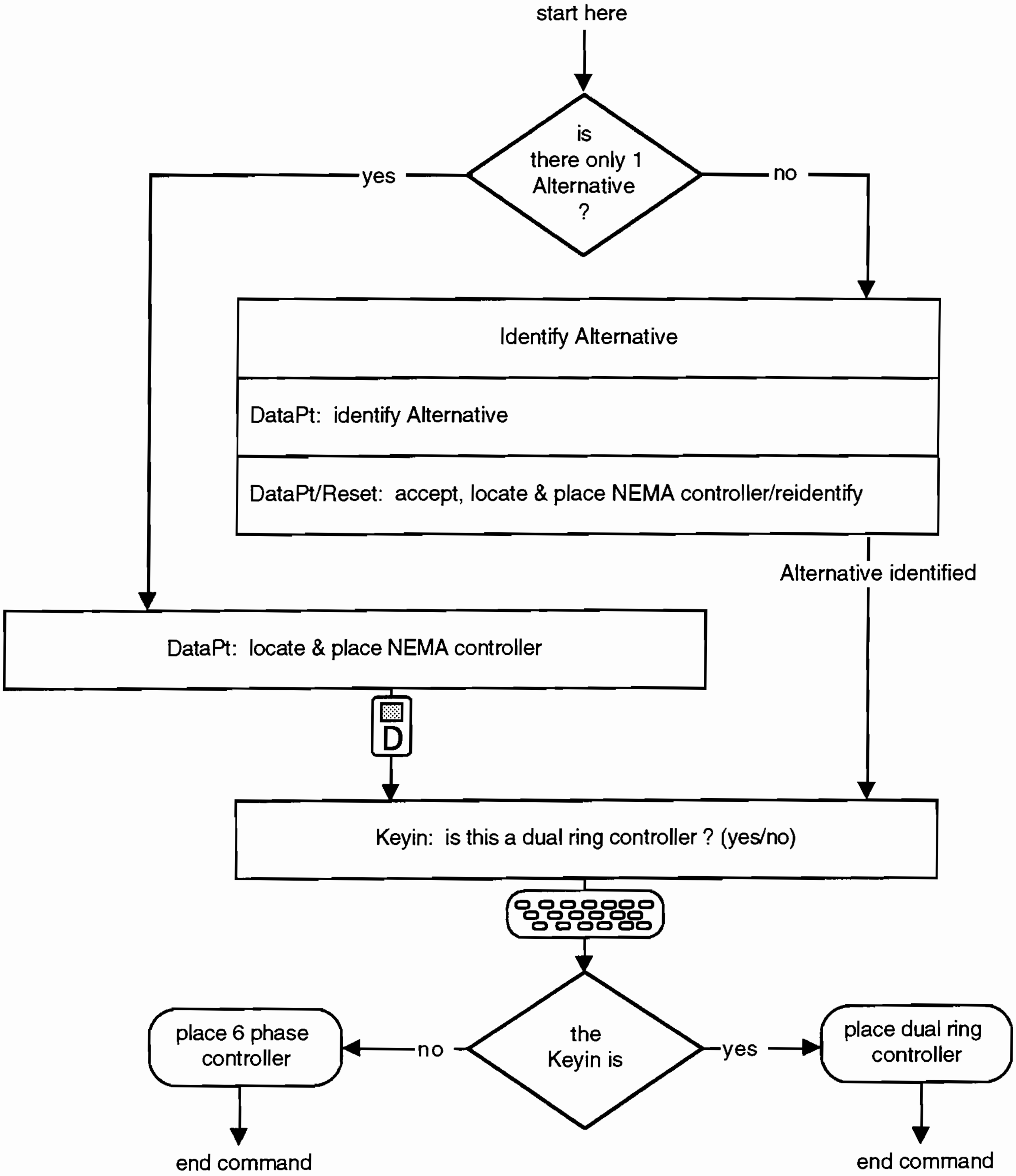
Primitive Command: TOOLS - Traffic - Controller - DELETE


start here
Identify Signal Controller
DataPt: identify a Signal Controller
DataPt/Reset: delete/reidentify the Signal Controller
Signal Controller identified
Keyin: delete the Signal Controller? (yes/[no])
no
the Keyin is
yes
delete controller
end command
end comand

Delete an existing traffic signal controller.

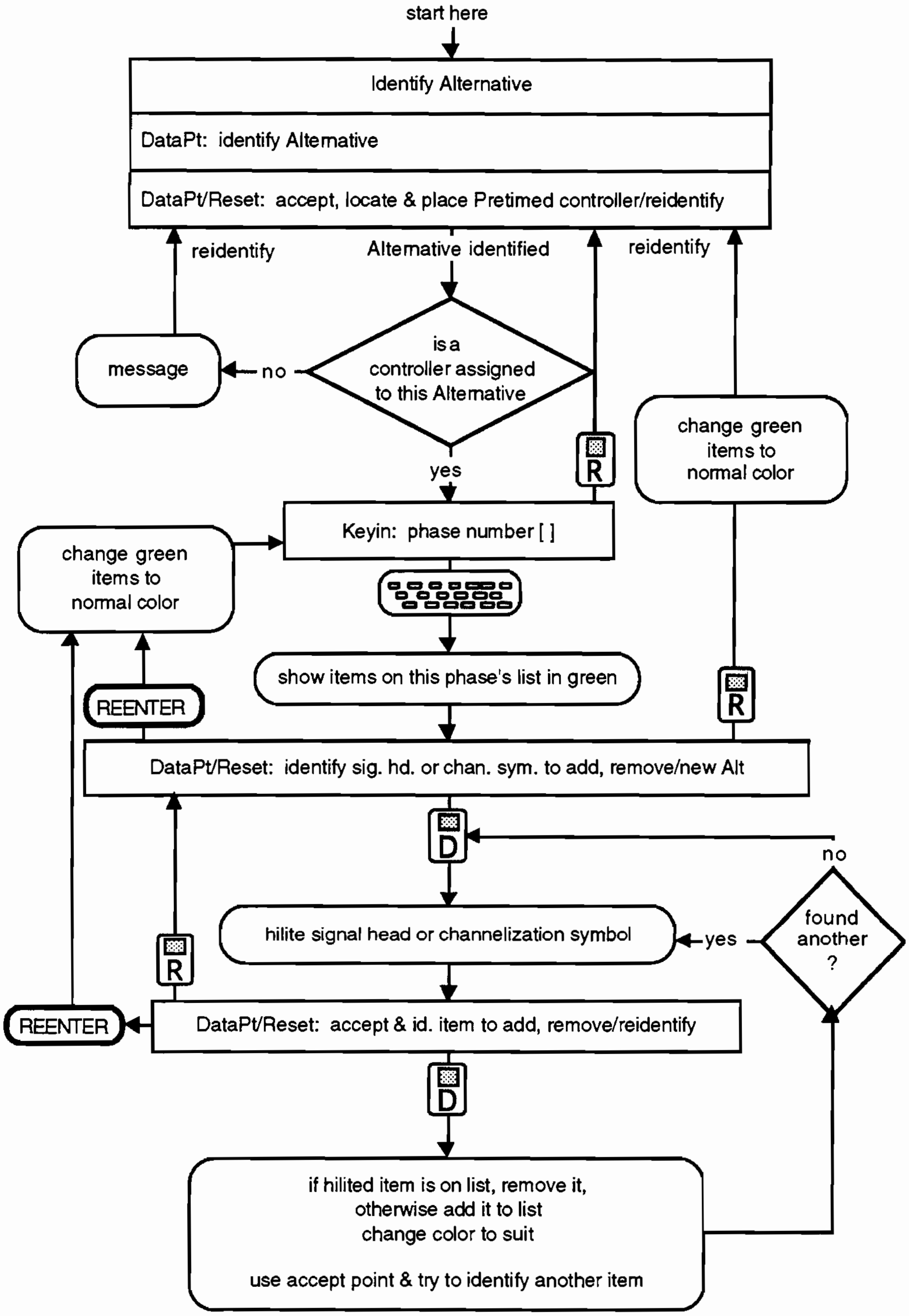
Primitive Command: TOOLS - Traffic - Controller - MOVE
Not programmed yet.

Primitive Command: TOOLS - Traffic - Controller - NEMA


start here
is there only 1 Alternative ?
yes
no
Identify Alternative
DataPt: identify Alternative
DataPt/Reset: accept, locate & place NEMA controller/reidentify
Alternative identified
DataPt: locate & place NEMA controller
D
Keyin: is this a dual ring controller ? (yes/no)
the Keyin is
no
yes
place 6 phase controller
place dual ring controller
end command
end command

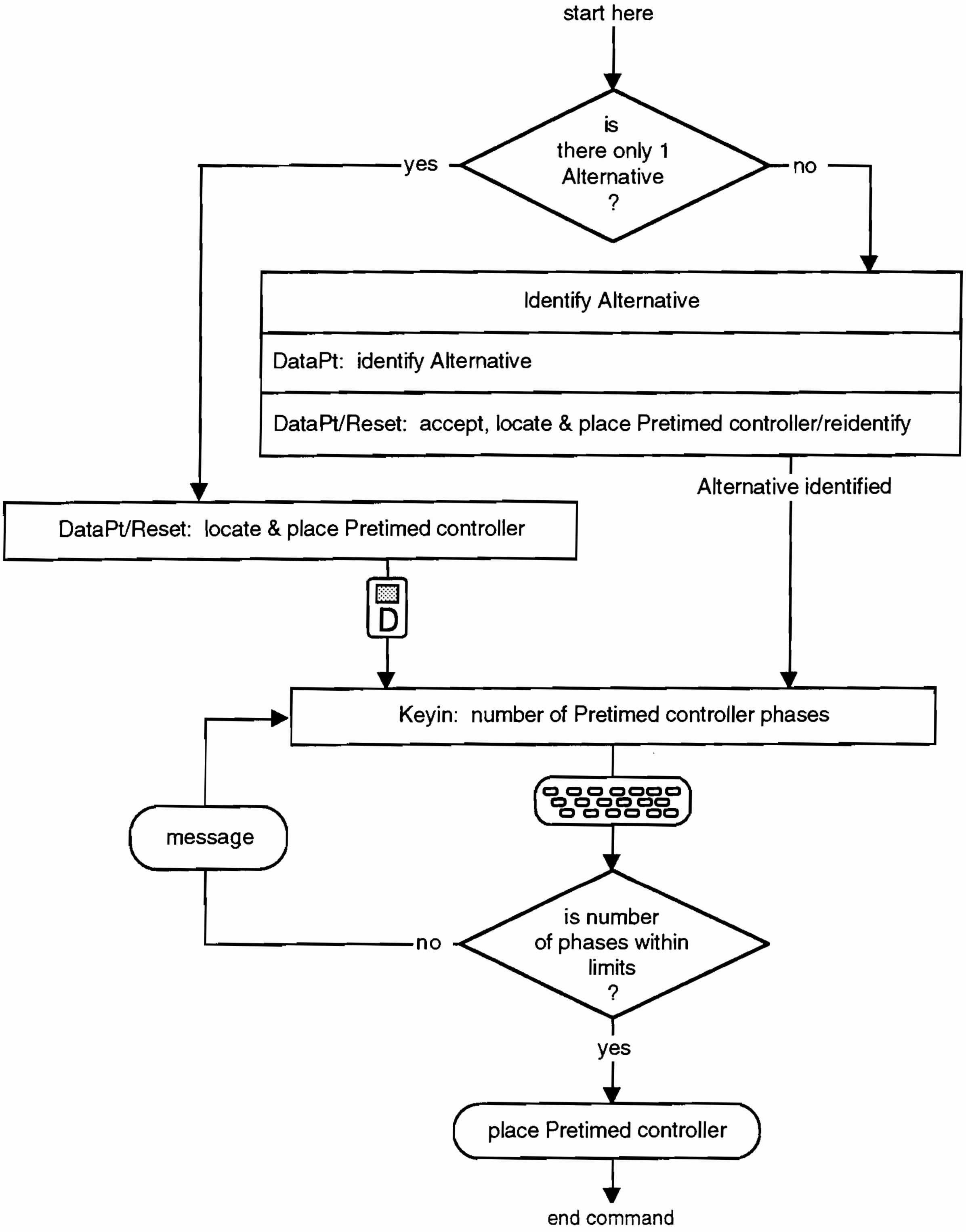
Add a standard NEMA traffic signal controller to an Alternative. The user is asked if the controller is to be a dual ring controller. If the response is "NO", a 6 phase single ring controller is added.

Primitive Command: TOOLS - Traffic - Controller - PHASING


start here
Identify Alternative
DataPt: identify Alternative
DataPt/Reset: accept, locate & place Pretimed controller/reidentify
reidentify
Alternative identified
reidentify
message
no
is a controller assigned to this Alternative
yes
R
change green items to normal color
Keyin: phase number [ ]
change green items to normal color
show items on this phase's list in green
REENTER
R
DataPt/Reset: identify sig. hd. or chan. sym. to add, remove/new Alt
D
no
found another ?
yes
hilite signal head or channelization symbol
R
REENTER
DataPt/Reset: accept & id. item to add, remove/reidentify
D
if hilited item is on list, remove it, otherwise add it to list change color to suit
use accept point & try to identify another item

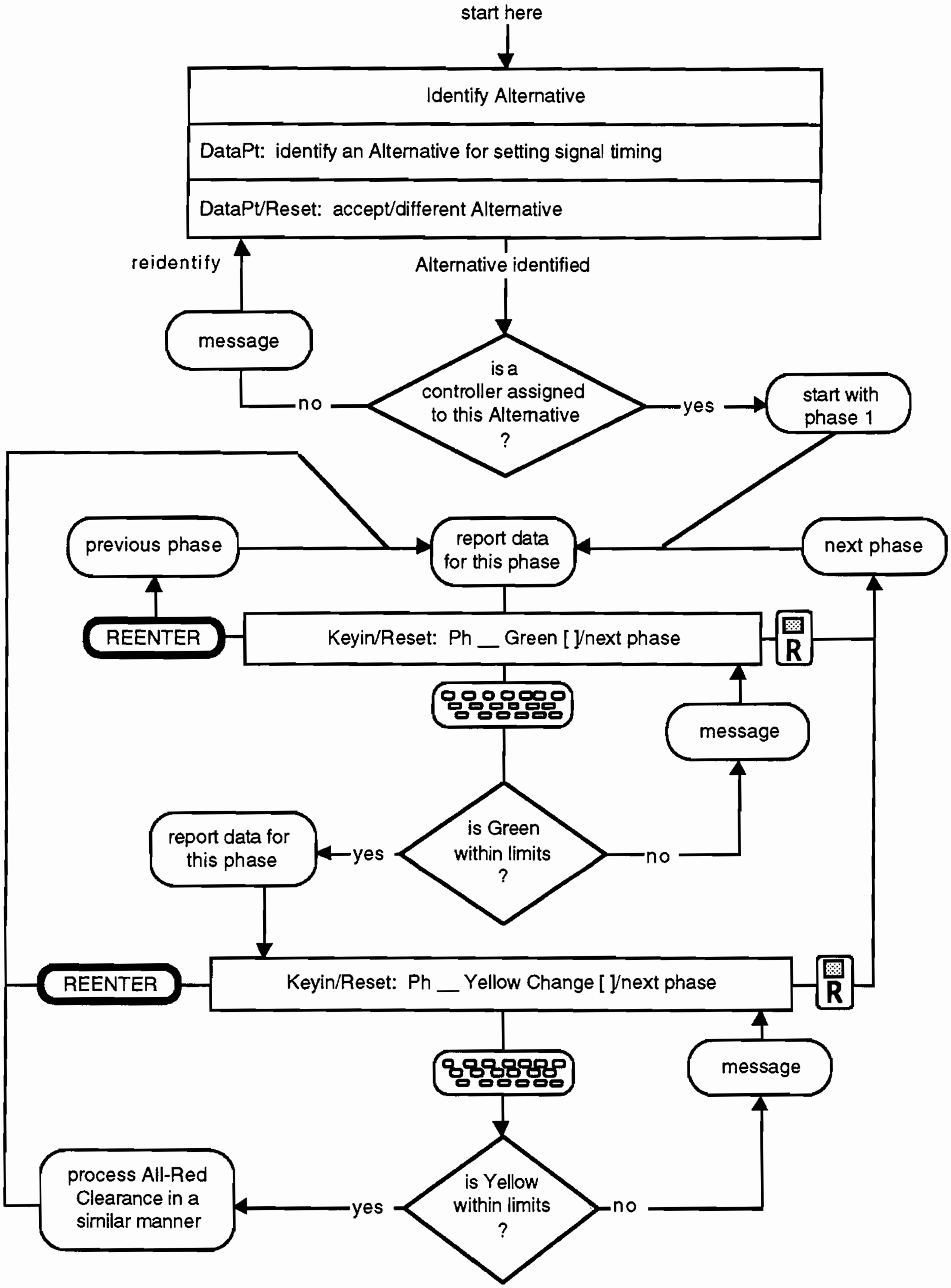
Specify the traffic phases to be associated with each controller phase. The user is prompted for a controller phase. The user is then prompted to identify signal faces and/or channelization symbols to be added or removed from the list for the controller phase. A circular green will permit all movements, except an exclusive left that does not move in another phase, to move. A protected left will permit exclusive lefts to move. Identifying selected channelization symbols in addition to signal faces will modify the above. Items on the list are shown in green.

Primitive Command: TOOLS - Traffic - Controller - Pretimed


start here
is there only 1 Alternative ?
yes
no
Identify Alternative
DataPt: identify Alternative
DataPt/Reset: accept, locate & place Pretimed controller/reidentify
Alternative identified
DataPt/Reset: locate & place Pretimed controller
D
Keyin: number of Pretimed controller phases
message
is number of phases within limits ?
no
yes
place Pretimed controller
end command

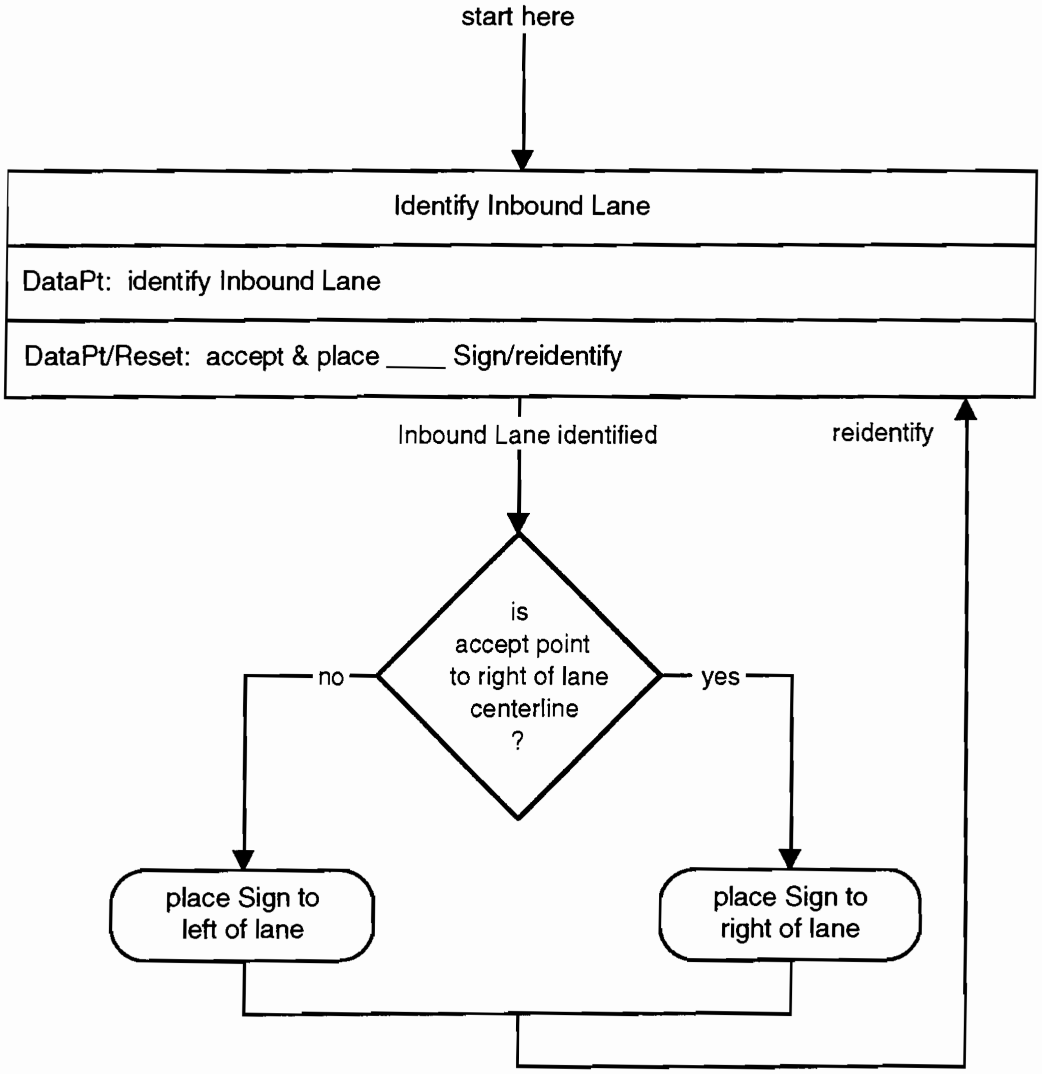
Add a pretimed traffic signal controller to an Alternative. The user is prompted for the number of controller phases. Must be 2 through 8.

Primitive Command: TOOLS - Traffic - Controller - TIMING


start here
Identify Alternative
DataPt: identify an Alternative for setting signal timing
DataPt/Reset: accept/different Alternative
reidentify
Alternative identified
message
no
is a controller assigned to this Alternative ?
yes
start with phase 1
previous phase
report data for this phase
next phase
REENTER
Keyin/Reset: Ph __ Green [ ]/next phase
R
message
report data for this phase
yes
is Green within limits ?
no
REENTER
Keyin/Reset: Ph __ Yellow Change [ ]/next phase
R
message
process All-Red Clearance in a similar manner
yes
is Yellow within limits ?
no

Specify the phase timing for a pretimed controller. The user is prompted (one interval at a time) for the green interval, yellow change interval and all-red clearance interval for a phase. Use Reset to go the next phase. This command is not yet programmed for a NEMA controller.

Primitive Command: TOOLS - Traffic - Sign

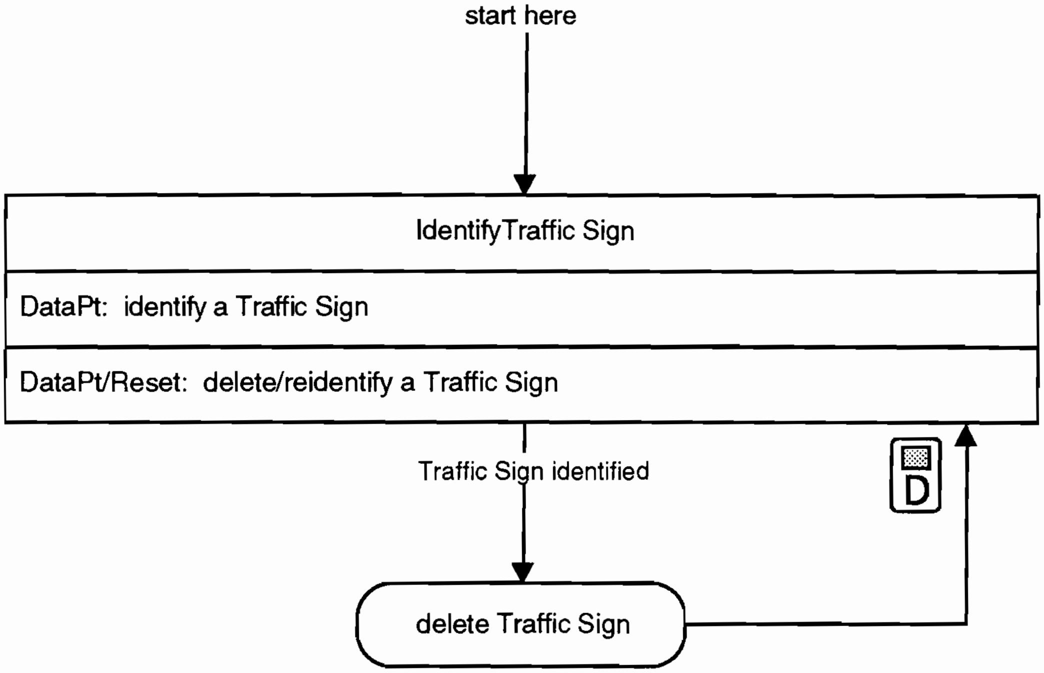

start here
Identify Inbound Lane
DataPt: identify Inbound Lane
DataPt/Reset: accept & place ____ Sign/reidentify
Inbound Lane identified
reidentify
is accept point to right of lane centerline ?
no
yes
place Sign to left of lane
place Sign to right of lane

Use this diagram for the 2 **TOOLS - Traffic - Sign** commands that follow.

Primitive Command: TOOLS - Traffic - Sign - Stop
Add a stop sign to an inbound Lane. The location of the acceptance data point with respect to the center of the lane will determine if the sign is located to the left or right of the lane.

Primitive Command: TOOLS - Traffic - Sign - Yield
Add a yield sign to an inbound Lane. The location of the acceptance data point with respect to the center of the lane will determine if the sign is located to the left or right of the lane.

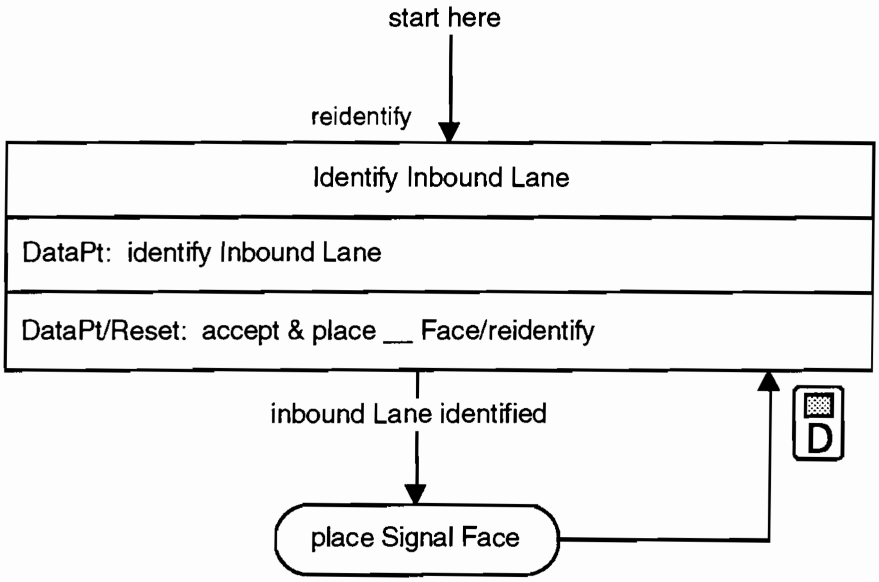
Primitive Command: TOOLS - Traffic - Sign - DELETE


start here
IdentifyTraffic Sign
DataPt: identify a Traffic Sign
DataPt/Reset: delete/reidentify a Traffic Sign
Traffic Sign identified
D
delete Traffic Sign

Delete an existing stop or yield sign

Primitive Command: TOOLS - Traffic - Sign - MOVE
Not programmed yet.

Primitive Command: TOOLS - Traffic - Signal Face



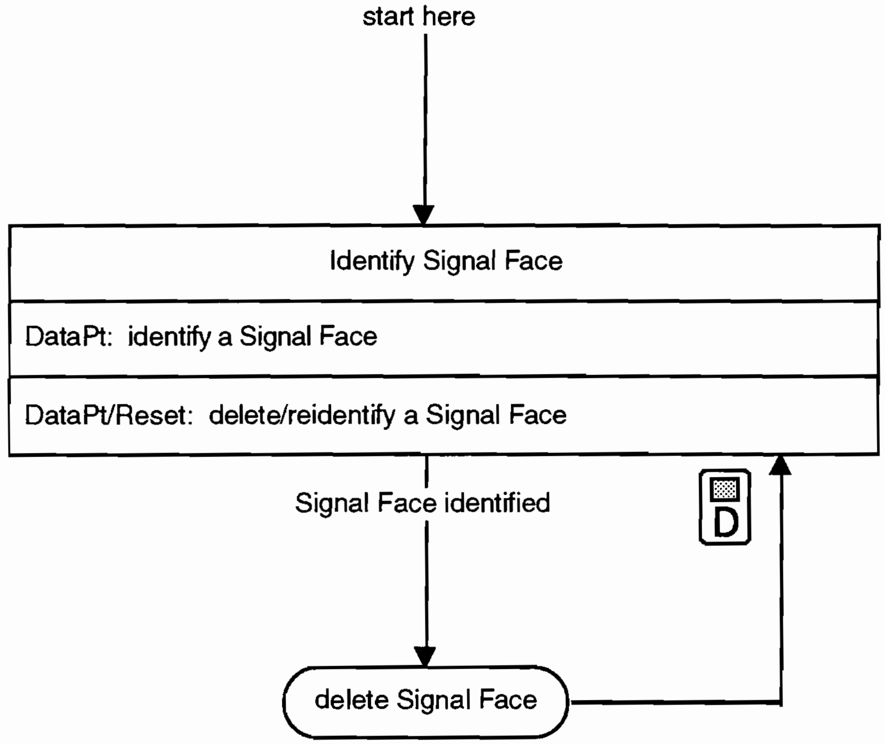
Use this diagram for the 3 **TOOLS - Traffic - Signal Face** commands that follow.

Primitive Command: TOOLS - Traffic - Signal Face - 3 Lens
Add a 3 lens signal face to a lane. This face presents a circular green to the appropriate movements on the Leg.

Primitive Command: TOOLS - Traffic - Signal Face - 3 Lens PL
Add a 3 lens signal face to a lane. This face presents a left green arrow green to the appropriate movements on the Leg.

Primitive Command: TOOLS - Traffic - Signal Face - 4 Lens
Add a 4 lens signal face to a lane. Don't use this command. Place a 3 lens signal face, instead.

Primitive Command: TOOLS - Traffic - Signal Face - DELETE


start here
Identify Signal Face
DataPt: identify a Signal Face
DataPt/Reset: delete/reidentify a Signal Face
Signal Face identified
D
delete Signal Face

Delete an existing signal face.

Primitive Command: TOOLS - Traffic - Signal Face - MOVE
Not programmed yet.

Primitive Command: TOOLS - Traffic - Volume

A different dialog box is presented based upon the number of legs (3, 4, 5, or 6) for the Alternative. All Traffic Volume dialog boxes have the same general layout and functionality. The dialog boxes for the 4 leg Alternative is somewhat different from the others. The input mode of each dialog box may be changed between **Percent+VOL** (Percentages of Traffic Volume) and **TMC Volume** (Traffic Volumes: Vehicles per Hour) when the data will allow the transfer using the option button labeled "INPUT MODE" in the top center of the dialog box. In all dialog boxes, each inbound leg has a row of input data boxes. To the left of the input data boxes is the leg number and a small diagram of the Alternative with the leg hilited. The input data boxes may be traversed from left to right by using the tab key on the keyboard. A tab will move the cursor from the rightmost input data box on one row to the leftmost input data box on the next row. Standard MicroStation dialog box editing functions may be used to traverse and edit the data. The push button labeled "Cancel" may be pressed at anytime and the input will be discarded and the dialog box closed. When all input data has been entered and is correct, a push button labeled "OK" will appear in the lower left of the dialog box. Pressing the push button labeled "OK" will save the input data to the Alternative and close the dialog box. When the input mode is **Percent+VOL**, there is a column labeled "Total Percent" which is the sum of the percentages on the row and there is a column labeled "Total Volume" which is an input data box. When the input mode is **TMC Volume**, there is a column labeled "Total Volume" which is the sum of the volumes on the row.

In the 3, 5, and 6 leg dialog boxes, each outbound leg has a column of input data boxes. To the top of the input data boxes is the leg number and a small diagram of the Alternative with the leg hilited. The inbound legs are sorted clockwise with the north leg at the top. The outbound legs are sorted clockwise with the north leg at the left. Each input data box in the matrix applies to traffic traveling from the inbound leg to the outbound leg. Additionally, there is an option button with the values "U" (u-turn), "L" (left), "S" (straight), or "R" (right) to the right of each input data box for the leg data. This option button indicates the designation of the traffic movement when a vehicle travels from the inbound leg to the outbound leg. Designations considered impossible are disabled. This designation may be changed by the user causing each designation on a row to be re-evaluated.

In the 4 leg dialog box, the rows are labeled "EB" (east bound), "WB" (west bound), "NB" (north bound), and "SB" (south bound) while the columns are labeled "U-Turn", "Left Turn", "Straight", and "Right".

These dialog boxes are used for the 8 **TOOLS - Traffic - Volume** commands that follow.

Primitive Command: TOOLS - Traffic - Volume - Percent+VOL (3 legs)

Primitive Command: TOOLS - Traffic - Volume - Percent+VOL (4 legs)

Primitive Command: TOOLS - Traffic - Volume - Percent+VOL (5 legs)

Primitive Command: TOOLS - Traffic - Volume - Percent+VOL (6 legs)

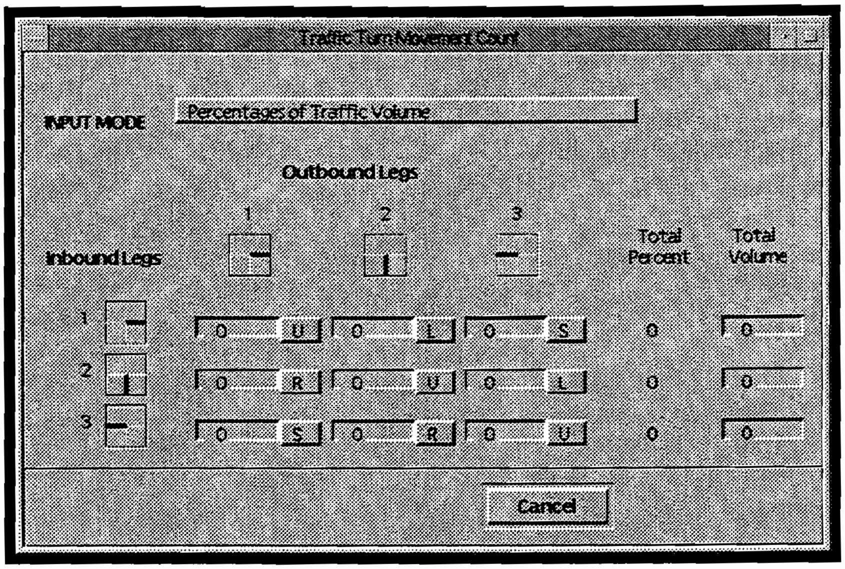
Primitive Command: TOOLS - Traffic - Volume - TMC Volume (3 legs)

Primitive Command: TOOLS - Traffic - Volume - TMC Volume (4 legs)

Primitive Command: TOOLS - Traffic - Volume - TMC Volume (5 legs)

Primitive Command: TOOLS - Traffic - Volume - TMC Volume (6 legs)

Primitive Command: TOOLS - Traffic - Volume - Percent+VOL (3 legs)



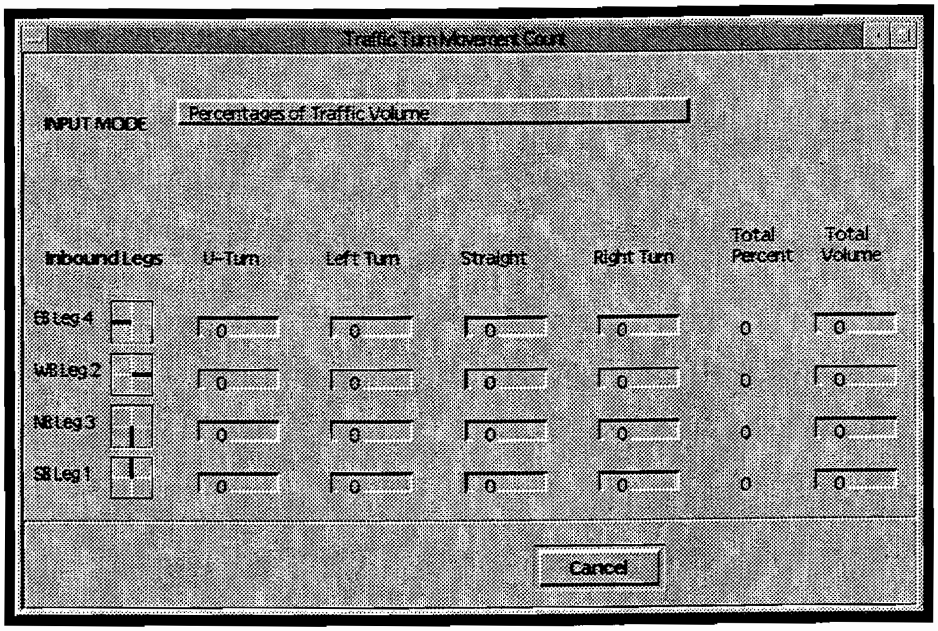
INPUT MODE: Percentages of Traffic Volume

Outbound Legs

Inbound Legs	1	2	3	Total Percent	Total Volume
1	0 U	0 L	0 S	0	0
2	0 R	0 U	0 L	0	0
3	0 S	0 R	0 U	0	0

Specify percentages of traffic volumes plus the traffic volumes for a 3 leg Alternative. See the description of dialog box usage under **Primitive Command: TOOLS - Traffic - Volume**.

Primitive Command: **TOOLS - Traffic - Volume - Percent+VOL (4 legs)**

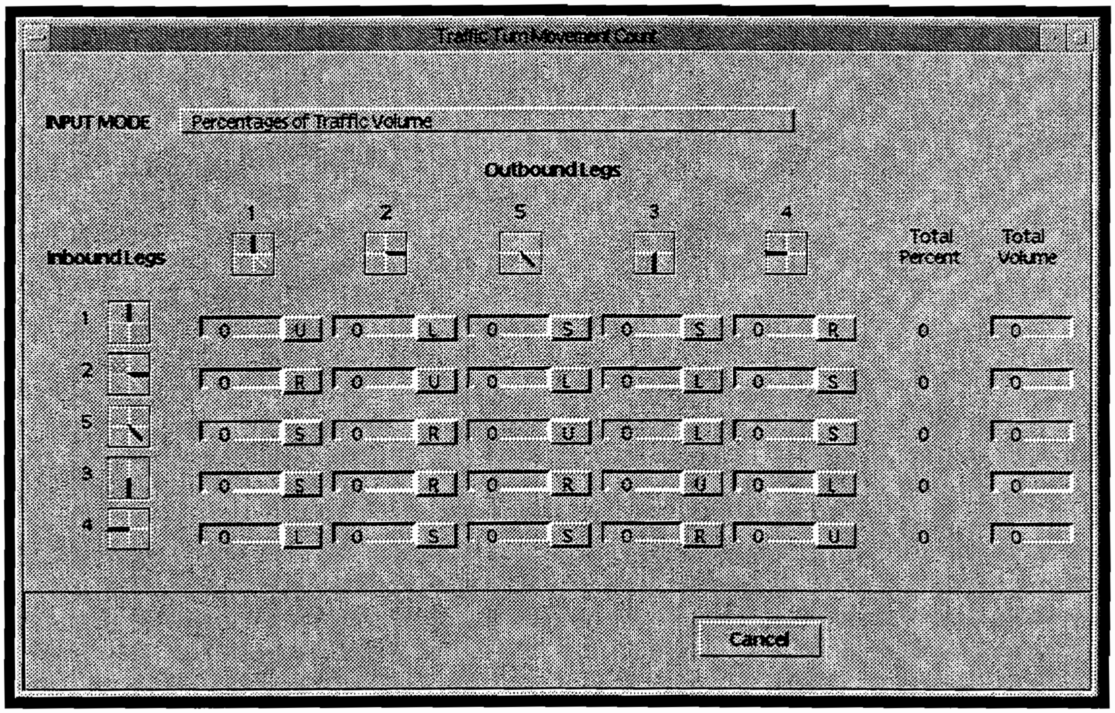


INPUT MODE: Percentages of Traffic Volume

Inbound Legs	U-Turn	Left Turn	Straight	Right Turn	Total Percent	Total Volume
EB Leg 4	0	0	0	0	0	0
WB Leg 2	0	0	0	0	0	0
NB Leg 3	0	0	0	0	0	0
SB Leg 1	0	0	0	0	0	0

Specify percentages of traffic volumes plus the traffic volumes for a 4 leg Alternative. See the description of dialog box usage under **Primitive Command: TOOLS - Traffic - Volume**.

Primitive Command: **TOOLS - Traffic - Volume - Percent+VOL (5 legs)**



Traffic Turn Movement Count

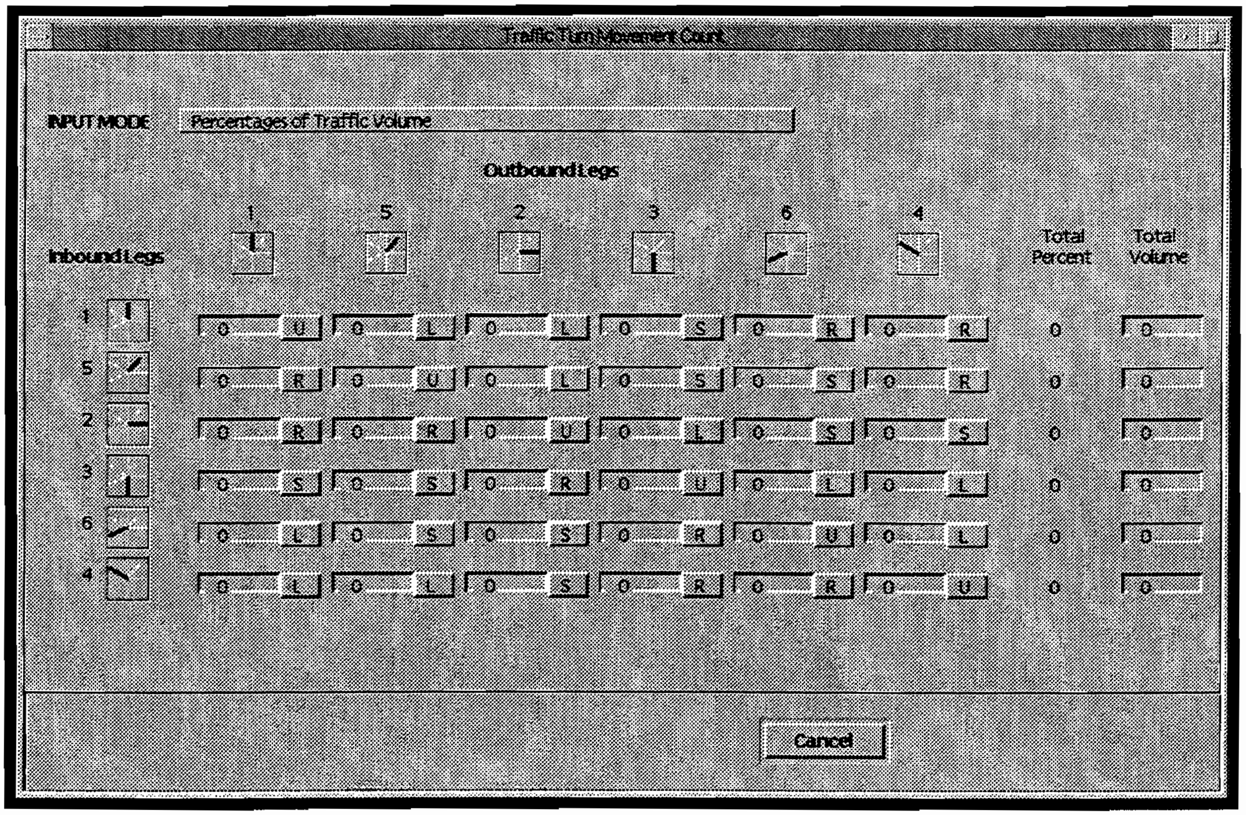
INPUT MODE: Percentages of Traffic Volume

Outbound Legs

Inbound Legs	1	2	5	3	4	Total Percent	Total Volume
1	0 U	0 L	0 S	0 S	0 R	0	0
2	0 R	0 U	0 L	0 L	0 S	0	0
5	0 S	0 R	0 U	0 L	0 S	0	0
3	0 S	0 R	0 R	0 U	0 L	0	0
4	0 L	0 S	0 S	0 R	0 U	0	0

Specify percentages of traffic volumes plus the traffic volumes for a 5 leg Alternative. See the description of dialog box usage under **Primitive Command: TOOLS - Traffic - Volume**.

Primitive Command: **TOOLS - Traffic - Volume - Percent+VOL (6 legs)**



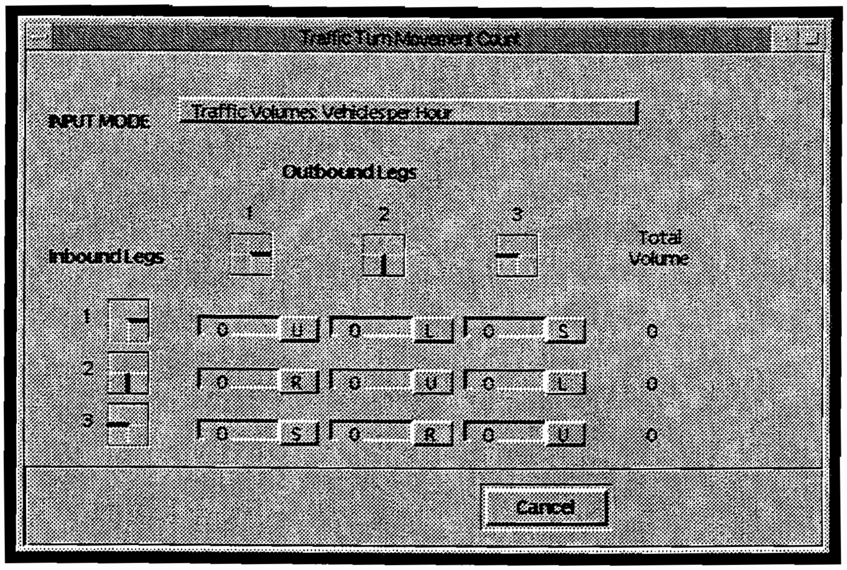
Traffic Turn Movement Count

INPUT MODE: Percentages of Traffic Volume

	Outbound Legs							
Inbound Legs	1	5	2	3	6	4	Total Percent	Total Volume
1	0 U	0 L	0 L	0 S	0 R	0 R	0	0
5	0 R	0 U	0 L	0 S	0 S	0 R	0	0
2	0 R	0 R	0 U	0 L	0 S	0 S	0	0
3	0 S	0 S	0 R	0 U	0 L	0 L	0	0
6	0 L	0 S	0 S	0 R	0 U	0 L	0	0
4	0 L	0 L	0 S	0 R	0 R	0 U	0	0

Specify percentages of traffic volumes plus the traffic volumes for a 6 leg Alternative. See the description of dialog box usage under **Primitive Command: TOOLS - Traffic - Volume**.

Primitive Command: **TOOLS - Traffic - Volume - TMC Volume (3 legs)**



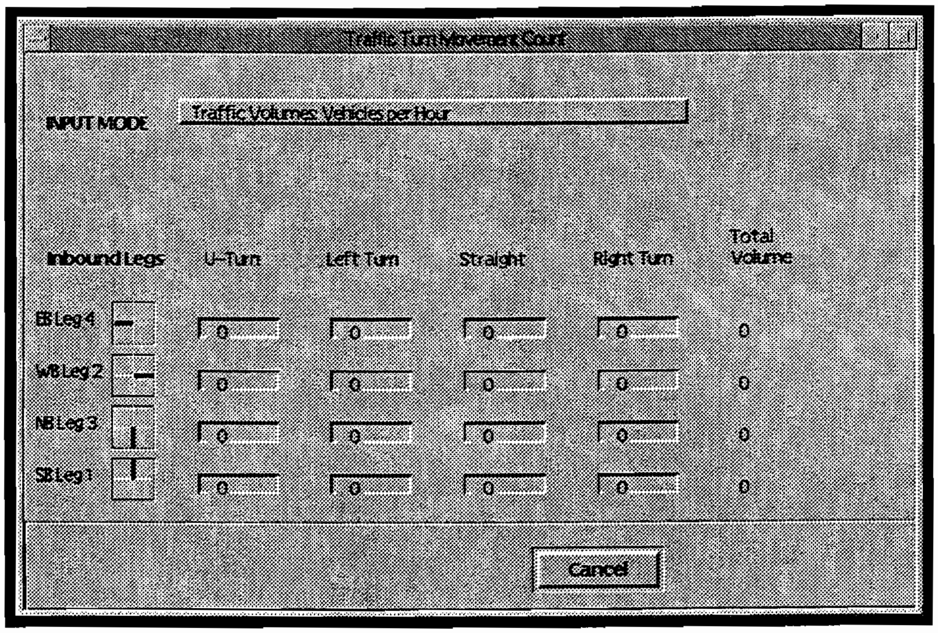
Traffic Turn Movement Count

INPUT MODE: Traffic Volumes: Vehicles per Hour

Inbound Legs	Outbound Legs 1	Outbound Legs 2	Outbound Legs 3	Total Volume
1	0 U	0 L	0 S	0
2	0 R	0 U	0 L	0
3	0 S	0 R	0 U	0

Specify turn movement count volumes for a 3 leg Alternative. See the description of dialog box usage under **Primitive Command: TOOLS - Traffic - Volume**.

Primitive Command: **TOOLS - Traffic - Volume - TMC Volume (4 legs)**

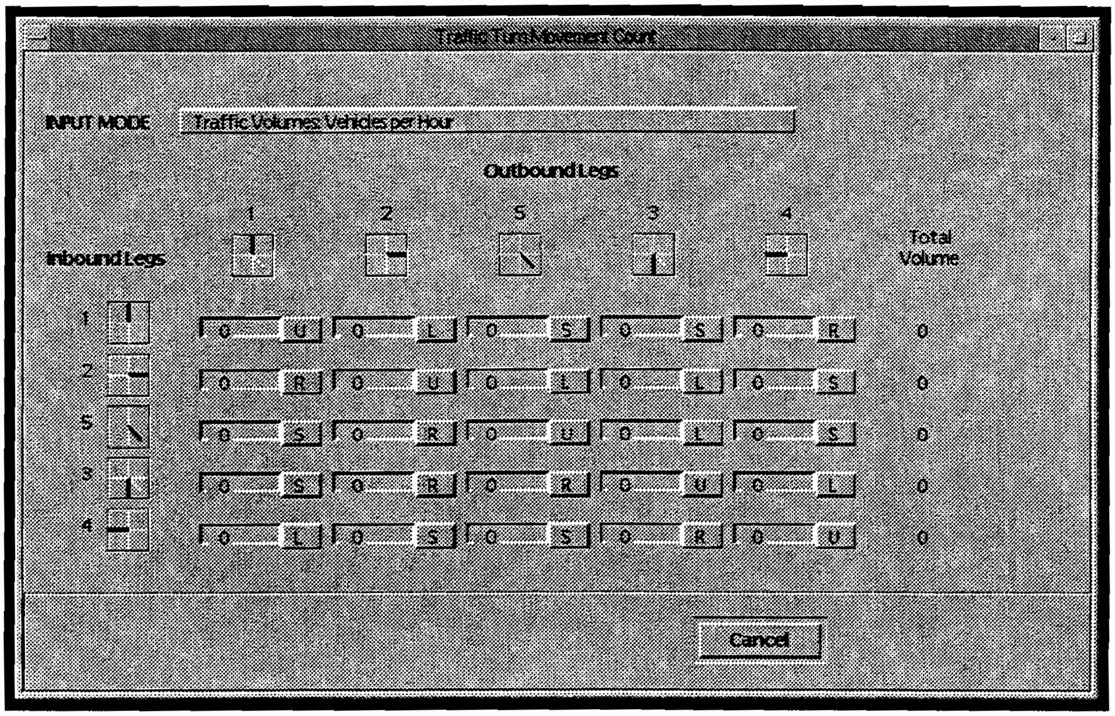


INPUT MODE: Traffic Volumes Vehicles per Hour

Inbound Legs	U-Turn	Left Turn	Straight	Right Turn	Total Volume
EB Leg 4	0	0	0	0	0
WB Leg 2	0	0	0	0	0
NB Leg 3	0	0	0	0	0
SB Leg 1	0	0	0	0	0

Specify turn movement count volumes for a 4 leg Alternative. See the description of dialog box usage under **Primitive Command: TOOLS - Traffic - Volume**.

Primitive Command: TOOLS - Traffic - Volume - TMC Volume (5 legs)



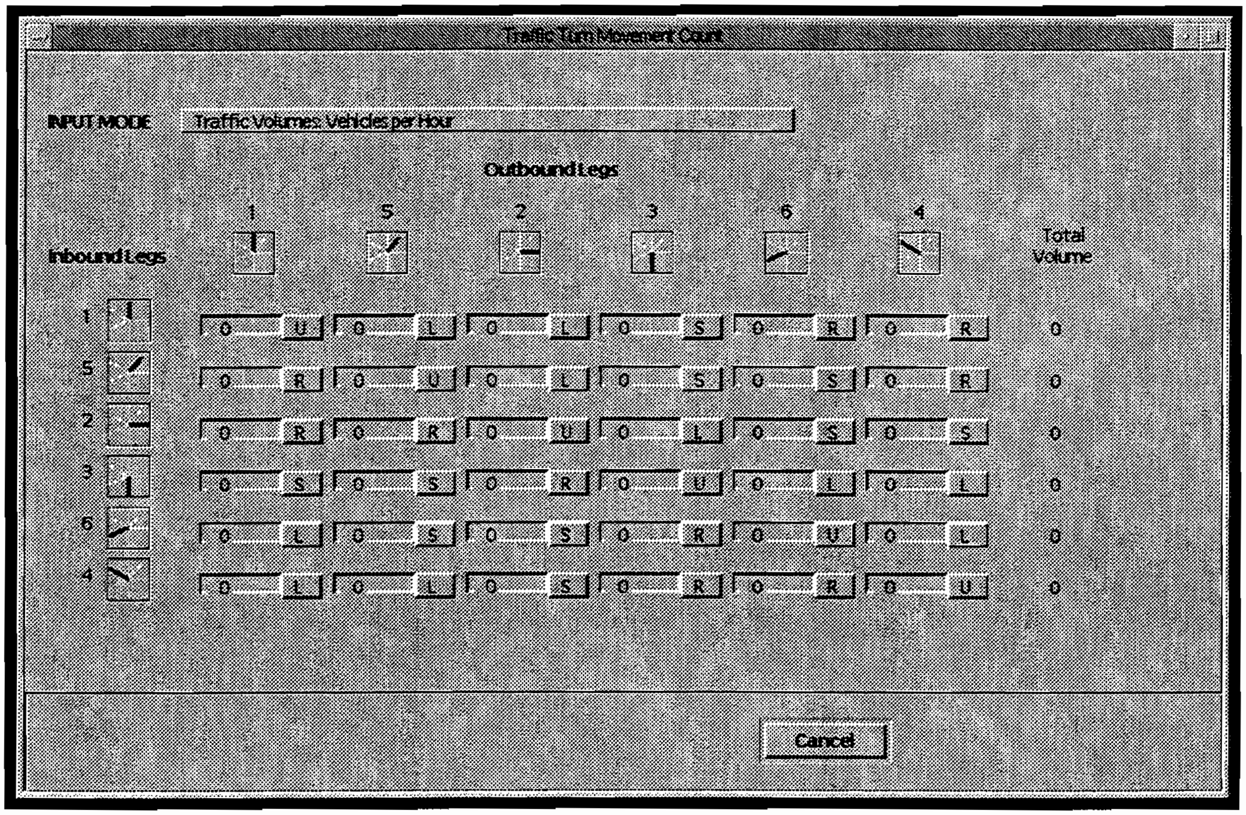
INPUT MODE: Traffic Volumes: Vehicles per Hour

Outbound Legs

Inbound Legs	1	2	5	3	4	Total Volume
1	0 U	0 L	0 S	0 S	0 R	0
2	0 R	0 U	0 L	0 L	0 S	0
5	0 S	0 R	0 U	0 L	0 S	0
3	0 S	0 R	0 R	0 U	0 L	0
4	0 L	0 S	0 S	0 R	0 U	0

Specify turn movement count volumes for a 5 leg Alternative. See the description of dialog box usage under **Primitive Command: TOOLS - Traffic - Volume**.

Primitive Command: **TOOLS - Traffic - Volume - TMC Volume (6 legs)**

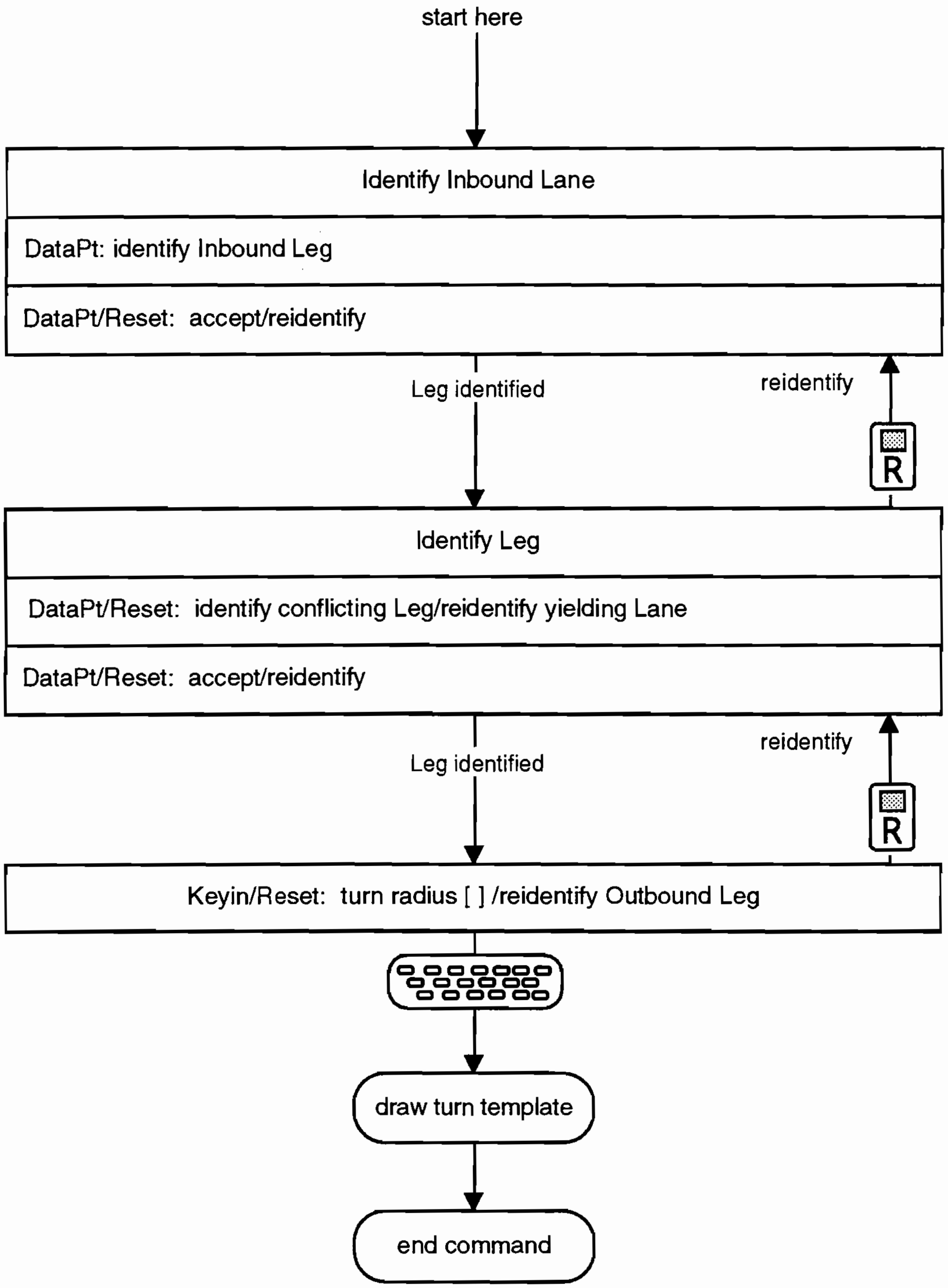


INPUT MODE: Traffic Volumes: Vehicles per Hour

	Outbound Legs						
Inbound Legs	1	5	2	3	6	4	Total Volume
1	0 U	0 L	0 L	0 S	0 R	0 R	0
5	0 R	0 U	0 L	0 S	0 S	0 R	0
2	0 R	0 R	0 U	0 L	0 S	0 S	0
3	0 S	0 S	0 R	0 U	0 L	0 L	0
6	0 L	0 S	0 S	0 R	0 U	0 L	0
4	0 L	0 L	0 S	0 R	0 R	0 U	0

Specify turn movement count volumes for a 6 leg Alternative. See the description of dialog box usage under **Primitive Command: TOOLS - Traffic - Volume**.

Primitive Command: TOOLS - Turn Template


start here
Identify Inbound Lane
DataPt: identify Inbound Leg
DataPt/Reset: accept/reidentify
Leg identified
reidentify
R
Identify Leg
DataPt/Reset: identify conflicting Leg/reidentify yielding Lane
DataPt/Reset: accept/reidentify
Leg identified
reidentify
R
Keyin/Reset: turn radius [ ] /reidentify Outbound Leg
draw turn template
end command

Use this diagram for the 14 **TOOLS - Turn Template** commands that follow.

Primitive Command: TOOLS - Turn Template - A-Bus
Draw a turning vehicle template for an articulated bus.

Primitive Command: TOOLS - Turn Template - Bus
Draw a turning vehicle template for a bus.

Primitive Command: TOOLS - Turn Template - MH
Draw a turning vehicle template for a motor home.

Primitive Command: TOOLS - Turn Template - P
Draw a turning vehicle template for a passenger vehicle.

Primitive Command: TOOLS - Turn Template - PB
Draw a turning vehicle template for a passenger vehicle pulling a boat.

Primitive Command: TOOLS - Turn Template - PT
Draw a turning vehicle template for a passenger vehicle pulling a trailer.

Primitive Command: TOOLS - Turn Template - RMD
Draw a turning vehicle template for a rocky mountain double truck.

Primitive Command: TOOLS - Turn Template - SU
Draw a turning vehicle template for a single unit truck.

Primitive Command: TOOLS - Turn Template - WB-40
Draw a turning vehicle template for a medium tractor-semitrailer.

Primitive Command: TOOLS - Turn Template - WB-50
Draw a turning vehicle template for a larger tractor-semitrailer combination.

Primitive Command: TOOLS - Turn Template - WB-60
Draw a turning vehicle template for tractor-semitrailer-full trailer combinations.

Primitive Command: TOOLS - Turn Template - WB-62
Draw a turning vehicle template for tractor-semitrailer combinations.

Primitive Command: TOOLS - Turn Template - WB-96
Draw a turning vehicle template for tractor-semitrailer-full trailer combinations.

Primitive Command: TOOLS - Turn Template - WB-114
Draw a turning vehicle template for tractor-semitrailer-full trailer-full trailer combinations.

Transient Command: TOOLS - Turn Template - Del Graphics
Delete all of the turning vehicle template graphics.

Transient Command: Verb-Noun or Noun-Verb
This command toggles between the two methods of command processing. The **Verb-Noun** method always prompts the user to identify an Object for processing. The **Noun-Verb** method always uses the selected IGIDS Object as the default choice when identifying an Object for processing.

Transient Command: VIEW - ALTERNATIVES - ALL OFF
For all Alternatives, make all graphics invisible.

Transient Command: VIEW - ALTERNATIVES - ALL ON
For all Alternatives, make all graphics visible.

Transient Command: VIEW - ALTERNATIVES - CURRENT OFF
For the selected Alternative, make all graphics invisible.

Transient Command: VIEW - ALTERNATIVES - CURRENT ON
For the selected Alternative, make all graphics visible.

Transient Command: VIEW - LANE - CURRENT OFF
For the selected Alternative, make all Inbound Lanes and Outbound Lanes invisible.

Transient Command: VIEW - LANE - CURRENT ON
For the selected Alternative, make all Inbound Lanes and Outbound Lanes visible.

Transient Command: VIEW - LEG CNTRLINE - CURRENT OFF
For the selected Alternative, make all Centerline Segs invisible.

Transient Command: VIEW - LEG CNTRLINE - CURRENT ON
For the selected Alternative, make all Centerline Segs visible.

Transient Command: VIEW - TEXT - CURRENT OFF
For the selected Alternative, make all Text invisible.

Transient Command: VIEW - TEXT - CURRENT ON
For the selected Alternative, make all Text visible.

Transient Command: VIEW - TRAF CONTROL - CURRENT OFF
For the selected Alternative, make all of the traffic control device graphics invisible.

Transient Command: VIEW - TRAF CONTROL - CURRENT ON
For the selected Alternative, make all of the traffic control device graphics visible.

Transient Command: Yes
In reply to a prompt requesting a "yes" or "no" response, send "yes" to IGIDS. This is the same as entering "yes" through the keyboard.

Transient Command: [default]
In response to a prompt, send the default value to IGIDS. When a default value is acceptable, it will be shown in the prompt, enclosed by square brackets. The default value may also be sent through the keyboard by pressing only the return key. For example, the prompt "Keyln: Lane width[12]" indicates that IGIDS will use 12 as the default value for lane width.

APPENDIX D

IGIDS Vehicle Turn Template Parameters

This appendix contains tables of the parameters used for the AASHTO vehicles in the Vehicle Turn Template commands within IGIDS.Command Menus. The parameters are for the Texas Truck Off-Tracking program TxTom. The tables are in alphabetical order of the AASHTO vehicle designation as follows:

(1) A-Bus,
(2) Bus,
(3) MH,
(4) P,
(5) PB,
(6) PT,
(7) RMD,
(8) SU,
(9) WB-40,
(10) WB-50,
(11) WB-60,
(12) WB-62,
(13) WB-96, and
(14) WB-114.

Table D.1 AASHTO DESIGN VEHICLE A-Bus

NO. OF UNITS FT02 C2F1	BEGIN PATH NO. FT04 C5F1	RUNOFF DIST FT01 C4F1	OFFSET DIST FT03 C2F3	VEHICLE WIDTH FT02 C2F2	FRONT OVERHANG FT02 C4F1	OFFSET TO FRONT OVERHANG FT02 C4F2
2	4	150	4.25	8.5	8.5	4.25

UNIT 1	UNIT 2	UNIT 3	UNIT 4	UNIT 5	UNIT 6	VARIABLE
18.00	24.00	0.00	0.00	0.00	0.00	(wheelbase) FT02 C3F2
0.00	0.00	0.00	0.00	0.00	0.00	(5th wheel or hitch) FT02 C3F3
26.50	0.00	0.00	0.00	0.00	0.00	(AVR dist from rear axle) FT04 C4F2
-4.25	4.25	0.00	0.00	0.00	0.00	(AVR dist from vehicle centerline) FT04 C4F3
1	2	0	0	0	0	(AVR unit number) FT04 C4F1

Table D.2 AASHTO DESIGN VEHICLE Bus

NO. OF UNITS FT02 C2F1	BEGIN PATH NO. FT04 C5F1	RUNOFF DIST FT01 C4F1	OFFSET DIST FT03 C2F3	VEHICLE WIDTH FT02 C2F2	FRONT OVERHANG FT02 C4F1	OFFSET TO FRONT OVERHANG FT02 C4F2
1	3	100	4.25	8.5	7	4.25

UNIT 1	UNIT 2	UNIT 3	UNIT 4	UNIT 5	UNIT 6	VARIABLE
25.00	0.00	0.00	0.00	0.00	0.00	(wheelbase) FT02 C3F2
0.00	0.00	0.00	0.00	0.00	0.00	(5th wheel or hitch) FT02 C3F3
32.00	0.00	0.00	0.00	0.00	0.00	(AVR dist from rear axle) FT04 C4F2
-4.25	4.25	0.00	0.00	0.00	0.00	(AVR dist from vehicle centerline) FT04 C4F3
1	1	0	0	0	0	(AVR unit number) FT04 C4F1

*NOTE: example FT02 C2F1 defined as TXTOM Input //DATA.FT02F001 DD *, Card 2 Field 1

Table D.3 AASHTO DESIGN VEHICLE MH

NO. OF UNITS FT02 C2F1	BEGIN PATH NO. FT04 C5F1	RUNOFF DIST FT01 C4F1	OFFSET DIST FT03 C2F3	VEHICLE WIDTH FT02 C2F2	FRONT OVERHANG FT02 C4F1	OFFSET TO FRONT OVERHANG FT02 C4F2
1	3	100	4	8	4	4

UNIT 1	UNIT 2	UNIT 3	UNIT 4	UNIT 5	UNIT 6	VARIABLE
20.00	0.00	0.00	0.00	0.00	0.00	(wheelbase) FT02 C3F2
0.00	0.00	0.00	0.00	0.00	0.00	(5th wheel or hitch) FT02 C3F3
24.00	0.00	0.00	0.00	0.00	0.00	(AVR dist from rear axle) FT04 C4F2
-4.00	4.00	0.00	0.00	0.00	0.00	(AVR dist from vehicle centerline) FT04 C4F3
1	1	0	0	0	0	(AVR unit number) FT04 C4F1

Table D.4 AASHTO DESIGN VEHICLE P

NO. OF UNITS FT02 C2F1	BEGIN PATH NO. FT04 C5F1	RUNOFF DIST FT01 C4F1	OFFSET DIST FT03 C2F3	VEHICLE WIDTH FT02 C2F2	FRONT OVERHANG FT02 C4F1	OFFSET TO FRONT OVERHANG FT02 C4F2
1	3	50	3.5	7	3	3.5

UNIT 1	UNIT 2	UNIT 3	UNIT 4	UNIT 5	UNIT 6	VARIABLE
11.00	0.00	0.00	0.00	0.00	0.00	(wheelbase) FT02 C3F2
0.00	0.00	0.00	0.00	0.00	0.00	(5th wheel or hitch) FT02 C3F3
14.00	0.00	0.00	0.00	0.00	0.00	(AVR dist from rear axle) FT04 C4F2
-3.50	3.50	0.00	0.00	0.00	0.00	(AVR dist from vehicle centerline) FT04 C4F3
1	1	0	0	0	0	(AVR unit number) FT04 C4F1

*NOTE: example FT02 C2F1 defined as TXTOM Input //DATA.FT02F001 DD *, Card 2 Field 1

Table D.5 AASHTO DESIGN VEHICLE PB

NO. OF UNITS FT02 C2F1	BEGIN PATH NO. FT04 C5F1	RUNOFF DIST FT01 C4F1	OFFSET DIST FT03 C2F3	VEHICLE WIDTH FT02 C2F2	FRONT OVERHANG FT02 C4F1	OFFSET TO FRONT OVERHANG FT02 C4F2
2	4	100	3.5	7	3	3.5

UNIT 1	UNIT 2	UNIT 3	UNIT 4	UNIT 5	UNIT 6	VARIABLE
11.00	15.00	0.00	0.00	0.00	0.00	(wheelbase) FT02 C3F2
-5.00	0.00	0.00	0.00	0.00	0.00	(5th wheel or hitch) FT02 C3F3
14.00	0.00	0.00	0.00	0.00	0.00	(AVR dist from rear axle) FT04 C4F2
-3.50	4.00	0.00	0.00	0.00	0.00	(AVR dist from vehicle centerline) FT04 C4F3
1	2	0	0	0	0	(AVR unit number) FT04 C4F1

Table D.6 AASHTO DESIGN VEHICLE PT

NO. OF UNITS FT02 C2F1	BEGIN PATH NO. FT04 C5F1	RUNOFF DIST FT01 C4F1	OFFSET DIST FT03 C2F3	VEHICLE WIDTH FT02 C2F2	FRONT OVERHANG FT02 C4F1	OFFSET TO FRONT OVERHANG FT02 C4F2
2	4	100	3.5	7	3	3.5

UNIT 1	UNIT 2	UNIT 3	UNIT 4	UNIT 5	UNIT 6	VARIABLE
11.00	18.00	0.00	0.00	0.00	0.00	(wheelbase) FT02 C3F2
-5.00	0.00	0.00	0.00	0.00	0.00	(5th wheel or hitch) FT02 C3F3
14.00	0.00	0.00	0.00	0.00	0.00	(AVR dist from rear axle) FT04 C4F2
-3.50	4.00	0.00	0.00	0.00	0.00	(AVR dist from vehicle centerline) FT04 C4F3
1	2	0	0	0	0	(AVR unit number) FT04 C4F1

*NOTE: example FT02 C2F1 defined as TXTOM Input //DATA.FT02F001 DD *, Card 2 Field 1

Table D.7 AASHTO DESIGN VEHICLE RMD

NO. OF UNITS FT02 C2F1	BEGIN PATH NO. FT04 C5F1	RUNOFF DIST FT01 C4F1	OFFSET DIST FT03 C2F3	VEHICLE WIDTH FT02 C2F2	FRONT OVERHANG FT02 C4F1	OFFSET TO FRONT OVERHANG FT02 C4F2
4	6	200	4.25	8.5	2	4.25

UNIT 1	UNIT 2	UNIT 3	UNIT 4	UNIT 5	UNIT 6	VARIABLE
15.50	37.30	6.30	22.30	0.00	0.00	(wheelbase) FT02 C3F2
0.00	-6.60	0.00	0.00	0.00	0.00	(5th wheel or hitch) FT02 C3F3
17.70	0.00	0.00	0.00	0.00	0.00	(AVR dist from rear axle) FT04 C4F2
-4.25	4.25	0.00	0.00	0.00	0.00	(AVR dist from vehicle centerline) FT04 C4F3
1	4	0	0	0	0	(AVR unit number) FT04 C4F1

Table D.8 AASHTO DESIGN VEHICLE SU

NO. OF UNITS FT02 C2F1	BEGIN PATH NO. FT04 C5F1	RUNOFF DIST FT01 C4F1	OFFSET DIST FT03 C2F3	VEHICLE WIDTH FT02 C2F2	FRONT OVERHANG FT02 C4F1	OFFSET TO FRONT OVERHANG FT02 C4F2
1	3	100	4.25	8.5	4	4.25

UNIT 1	UNIT 2	UNIT 3	UNIT 4	UNIT 5	UNIT 6	VARIABLE
20.00	0.00	0.00	0.00	0.00	0.00	(wheelbase) FT02 C3F2
0.00	0.00	0.00	0.00	0.00	0.00	(5th wheel or hitch) FT02 C3F3
24.00	0.00	0.00	0.00	0.00	0.00	(AVR dist from rear axle) FT04 C4F2
-4.25	4.25	0.00	0.00	0.00	0.00	(AVR dist from vehicle centerline) FT04 C4F3
1	1	0	0	0	0	(AVR unit number) FT04 C4F1

*NOTE: example FT02 C2F1 defined as TXTOM Input //DATA.FT02F001 DD *, Card 2 Field 1

Table D.9 AASHTO DESIGN VEHICLE WB-40

NO. OF UNITS FT02 C2F1	BEGIN PATH NO. FT04 C5F1	RUNOFF DIST FT01 C4F1	OFFSET DIST FT03 C2F3	VEHICLE WIDTH FT02 C2F2	FRONT OVERHANG FT02 C4F1	OFFSET TO FRONT OVERHANG FT02 C4F2
2	4	100	4.25	8.5	4	4.25
UNIT 1	UNIT 2	UNIT 3	UNIT 4	UNIT 5	UNIT 6	VARIABLE
13.00	25.00	0.00	0.00	0.00	0.00	(wheelbase) FT02 C3F2
0.00	0.00	0.00	0.00	0.00	0.00	(5th wheel or hitch) FT02 C3F3
17.00	0.00	0.00	0.00	0.00	0.00	(AVR dist from rear axle) FT04 C4F2
-4.25	4.25	0.00	0.00	0.00	0.00	(AVR dist from vehicle centerline) FT04 C4F3
1	2	0	0	0	0	(AVR unit number) FT04 C4F1

Table D.10 AASHTO DESIGN VEHICLE WB-50

NO. OF UNITS FT02 C2F1	BEGIN PATH NO. FT04 C5F1	RUNOFF DIST FT01 C4F1	OFFSET DIST FT03 C2F3	VEHICLE WIDTH FT02 C2F2	FRONT OVERHANG FT02 C4F1	OFFSET TO FRONT OVERHANG FT02 C4F2
2	4	150	4.25	8.5	3	4.25

UNIT 1	UNIT 2	UNIT 3	UNIT 4	UNIT 5	UNIT 6	VARIABLE
18.00	30.00	0.00	0.00	0.00	0.00	(wheelbase) FT02 C3F2
0.00	0.00	0.00	0.00	0.00	0.00	(5th wheel or hitch) FT02 C3F3
21.00	0.00	0.00	0.00	0.00	0.00	(AVR dist from rear axle) FT04 C4F2
-4.25	4.25	0.00	0.00	0.00	0.00	(AVR dist from vehicle centerline) FT04 C4F3
1	2	0	0	0	0	(AVR unit number) FT04 C4F1

*NOTE: example FT02 C2F1 defined as TXTOM Input //DATA.FT02F001 DD *, Card 2 Field 1

Table D.11 AASHTO DESIGN VEHICLE WB-60

NO. OF UNITS FT02 C2F1	BEGIN PATH NO. FT04 C5F1	RUNOFF DIST FT01 C4F1	OFFSET DIST FT03 C2F3	VEHICLE WIDTH FT02 C2F2	FRONT OVERHANG FT02 C4F1	OFFSET TO FRONT OVERHANG FT02 C4F2
4	6	150	4.25	8.5	2	4.25

UNIT 1	UNIT 2	UNIT 3	UNIT 4	UNIT 5	UNIT 6	VARIABLE
9.70	20.00	5.40	20.90	0.00	0.00	(wheelbase) FT02 C3F2
1.00	-4.00	0.80	0.00	0.00	0.00	(5th wheel or hitch) FT02 C3F3
11.70	0.00	0.00	0.00	0.00	0.00	(AVR dist from rear axle) FT04 C4F2
-4.25	4.25	0.00	0.00	0.00	0.00	(AVR dist from vehicle centerline) FT04 C4F3
1	4	0	0	0	0	(AVR unit number) FT04 C4F1

Table D.12 AASHTO DESIGN VEHICLE WB-62

NO. OF UNITS FT02 C2F1	BEGIN PATH NO. FT04 C5F1	RUNOFF DIST FT01 C4F1	OFFSET DIST FT03 C2F3	VEHICLE WIDTH FT02 C2F2	FRONT OVERHANG FT02 C4F1	OFFSET TO FRONT OVERHANG FT02 C4F2
2	4	150	4.25	8.5	3	4.25

UNIT 1	UNIT 2	UNIT 3	UNIT 4	UNIT 5	UNIT 6	VARIABLE
18.00	42.00	0.00	0.00	0.00	0.00	(wheelbase) FT02 C3F2
0.00	0.00	0.00	0.00	0.00	0.00	(5th wheel or hitch) FT02 C3F3
21.00	0.00	0.00	0.00	0.00	0.00	(AVR dist from rear axle) FT04 C4F2
-4.25	4.25	0.00	0.00	0.00	0.00	(AVR dist from vehicle centerline) FT04 C4F3
1	2	0	0	0	0	(AVR unit number) FT04 C4F1

*NOTE: example FT02 C2F1 defined as TXTOM Input //DATA.FT02F001 DD *, Card 2 Field 1

Table D.13 AASHTO DESIGN VEHICLE WB-96

NO. OF UNITS FT02 C2F1	BEGIN PATH NO. FT04 C5F1	RUNOFF DIST FT01 C4F1	OFFSET DIST FT03 C2F3	VEHICLE WIDTH FT02 C2F2	FRONT OVERHANG FT02 C4F1	OFFSET TO FRONT OVERHANG FT02 C4F2
6	8	200	4.25	8.5	2.5	4.25

UNIT 1	UNIT 2	UNIT 3	UNIT 4	UNIT 5	UNIT 6	VARIABLE
13.50	20.70	6.10	21.60	6.10	21.60	(wheelbase) FT02 C3F2
2.00	-3.30	0.00	-3.30	0.00	0.00	(5th wheel or hitch) FT02 C3F3
16.00	0.00	0.00	0.00	0.00	0.00	(AVR dist from rear axle) FT04 C4F2
-4.25	4.25	0.00	0.00	0.00	0.00	(AVR dist from vehicle centerline) FT04 C4F3
1	6	0	0	0	0	(AVR unit number) FT04 C4F1

Table D.14 AASHTO DESIGN VEHICLE WB-114

NO. OF UNITS FT02 C2F1	BEGIN PATH NO. FT04 C5F1	RUNOFF DIST FT01 C4F1	OFFSET DIST FT03 C2F3	VEHICLE WIDTH FT02 C2F2	FRONT OVERHANG FT02 C4F1	OFFSET TO FRONT OVERHANG FT02 C4F2
4	6	250	4.25	8.5	2	4.25

UNIT 1	UNIT 2	UNIT 3	UNIT 4	UNIT 5	UNIT 6	VARIABLE
20.00	40.00	8.00	40.00	0.00	0.00	(wheelbase) FT02 C3F2
0.00	-4.00	0.00	0.00	0.00	0.00	(5th wheel or hitch) FT02 C3F3
22.00	0.00	0.00	0.00	0.00	0.00	(AVR dist from rear axle) FT04 C4F2
-4.25	4.25	0.00	0.00	0.00	0.00	(AVR dist from vehicle centerline) FT04 C4F3
1	4	0	0	0	0	(AVR unit number) FT04 C4F1

*NOTE: example FT02 C2F1 defined as TXTOM Input //DATA.FT02F001 DD *, Card 2 Field 1

APPENDIX E

IGIDS Abbreviations

Several abbreviations are used throughout this document. The following list is included here for the convenience of the reader:

(1)	AASHTO	American Association of State Highway and Transportation Officials
(2)	alt	alternative
(3)	AT&T	American Telephone and Telegraph
(4)	CAD	Computer Aided Design
(5)	CPU	Central Processor Unit
(6)	DEC	Digital Equipment Corporation
(7)	DOT	Department of Transportation
(8)	ID	identification; the ID is a unique number defined by IGIDS and is the entry number or instance number in the appropriate structure
(9)	ID_NULL	a "#define" constant which stands for an invalid ID and has a value of -1
(10)	IGIDS	Interactive Graphics Intersection Design System
(11)	IGrds	AASHTO's Interactive Graphics Roadway Design System
(12)	inter	intersection
(13)	MS-DOS	Microsoft's Disk Operating System
(14)	NULL	a pointer to void with a value of zero which is an invalid address
(15)	OSF	Open Software Foundation
(16)	OS1	Operating System 1
(17)	seg	segment

APPENDIX F

IGIDS Terminology

Several terms have been adopted and are used throughout this document and are defined below:

absolute text angle

Text angle remains constant, independent of the parent Seg rotation.

Alternative

An IGIDS Object that contains a set of Legs and, optionally, descriptive Text. An Alternative specifies one particular configuration of the Intersection being designed. The parent of an alternative is the intersection.

channelization symbol

A graphical symbol used to indicate non-standard traffic channelization for a Lane. Channelization symbols are available to represent u-turns, lefts, straight through and rights. They may be used alone or in any suitable combination. Standard channelization is: (a) lefts and straights from a median Lane, (b) straights and rights from a curb Lane, and (c) straights from all other Lanes.

click

A user input to IGIDS. The sequence of moving the cursor to the desired position on the screen, pressing and then releasing the data button.

command

A request from the user for IGIDS to take some action. Commands are initiated by a Click on an IGIDS menu item.

DataPt

A user input of coordinate data to IGIDS . Executed by using the mouse to position the cursor at the desired coordinates in a graphics window, then pressing and releasing the mouse data button. This sends the X and Y coordinates of the geometric point to IGIDS. A DataPt may also be accomplished by entering any of the MicroStation precision keyin commands. A DataPt is represented in the process diagrams by this symbol:

Graphics Engine

The software that is used by IGIDS to draw graphics and provide the user interface. A commercial product which performs all interactive graphics operations and maintains the graphics database. Micro Station is the graphics engine for IGIDS.

ID number

An integer number assigned to each IGIDS Object. These numbers must be unique for any of the logical groupings of Objects. The logical groupings are: (1) all Alternatives, (2) all Legs on an Alternative, (3) all Inbound Lanes on a Leg, (4) all Outbound Lanes on a Leg, (5) all Centerline Segs on a Leg, (6) all Curb Return Segs on a Leg, (7) all Inner Edge Segs on a Lane, (8) all Outer Edge Segs on a Lane, (9) all Stopline Segs on a Lane, (10) all Texts on an Alternative, and (11) all Texts on a Seg. Numbers are assigned to Alternatives and Legs by the user at the time of creation. ID numbers are assigned automatically to Lanes and Segs. For Lanes, ID number 1 is assigned to the median Lane. The assigned ID numbers then increase by 1 for the next adjacent Lane, in the direction toward the curb Lane. This numbering process is applied independently to the set of Inbound Lanes and to the set of Outbound Lanes. Segs are numbered in increasing order within each logical group, with the Seg nearest the intersection being ID number 1. Text numbers are used only internally by IGIDS and are of no user interest.

Intersection

An IGIDS Object that is composed of all currently defined Alternatives.

KeyIn

A user input to IGIDS of alpha numeric data. A sequence of key presses that are ended by pressing the return key. A keyin may also be accomplished by clicking the "Yes," "No," or "[default]" command on the IGIDS menu. A Keyin is represented in the process diagrams by this symbol:

⌨

Lane

An IGIDS Object that contains Inner Edge Segs, Outer Edge Segs, and Stopline Segs. Lanes are classified as either Inbound or Outbound.

Leg

An IGIDS Object that contains Centerline Segs, Inbound Lanes, Outbound Lanes, Inner Edge Curb Return Segs, and Outer Edge Curb Return Segs.

Object

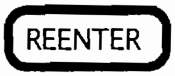
A set of graphic elements or other IGIDS data that are grouped together by IGIDS. All information needed to describe an Object is contained in the IGIDS database. Objects can be identified and manipulated only through IGIDS commands. The Objects are Intersection, Alternative, Leg, Lane, Seg and Text.

primitive command

An IGIDS command that initiates an action that requires user interaction. When issued during the execution of another IGIDS Command or a MicroStation Command, it cancels any active IGIDS or MicroStation command.

Reenter

A transient command requesting that IGIDS re-prompt for the most recently keyed-in data. A Reenter is represented in the process diagrams by this symbol:

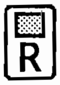


relative text angle

Text angle as measured from the parent Seg. It remains constant. Text rotates with the parent Seg.

Reset

A user input to IGIDS. Executed by pressing the mouse reset button. A Reset is represented in the process diagrams by this symbol:



scratch graphics

The portion of the drawing space that is available for use by the user. This space may also be used by IGIDS to draw graphics that are not a part of the IGIDS database.

Seg

An IGIDS Object that is either a straight line segment or an arc of a circle. Segs are classified according to their use and may be Inner Edge Segs, Outer Edge Segs, Stopline Segs, Centerline Segs, Inner Edge Curb Return Segs, or Outer Edge Curb Return Segs. Seg is an abbreviated form of the word segment. The parent of a seg is a leg or a lane. The parent of a leg is an alternative.

selected Object

an Object that has been identified to be used as the default Object for purposes of command processing.

signal face

A graphical symbol used to indicate a signal indication to be shown to a leg during the green interval of a controller phase.

temporary command

An IGIDS command that initiates an action that requires user interaction. When issued during the execution of another IGIDS Command, it temporarily suspends the command in progress. When the temporary command is ended, the suspended command continues from the point where it was suspended.

Text

An IGIDS Object that is defined in one of the fonts that is currently known to the Graphics Engine. Text is classified at the time that it is placed and will be either Text on an Alternative or Text on a Seg. The parent of text in an Alternative or a Seg. Text may be oriented with a fixed rotation angle or, if it is a Text on a Seg, may be oriented with respect to the current rotation angle of the associated Seg.

traffic control device A traffic signal controller, channelization symbol, signal face, stop sign, or yield sign.

traffic signal controller A graphical symbol used to indicate the type of controller to be used with the Alternative.

transient command An IGIDS command that initiates an action that requires no user interaction. When issued during the execution of another IGIDS Command, it does not end the command in progress.

turning vehicle template Scratch graphics that show the paths followed by the outside front bumper and the inside rearmost wheel of a specified type of vehicle that is turning from an Inbound Leg to an Outbound Leg and is using a specified turning radius.

APPENDIX G

Index to Commands