12 146

Resubmission of Final Report for NCC2-700

James H. Blaisdell, Ph.D.
Principal Investigator

Thomas F. Brownfield
Lead Programmer

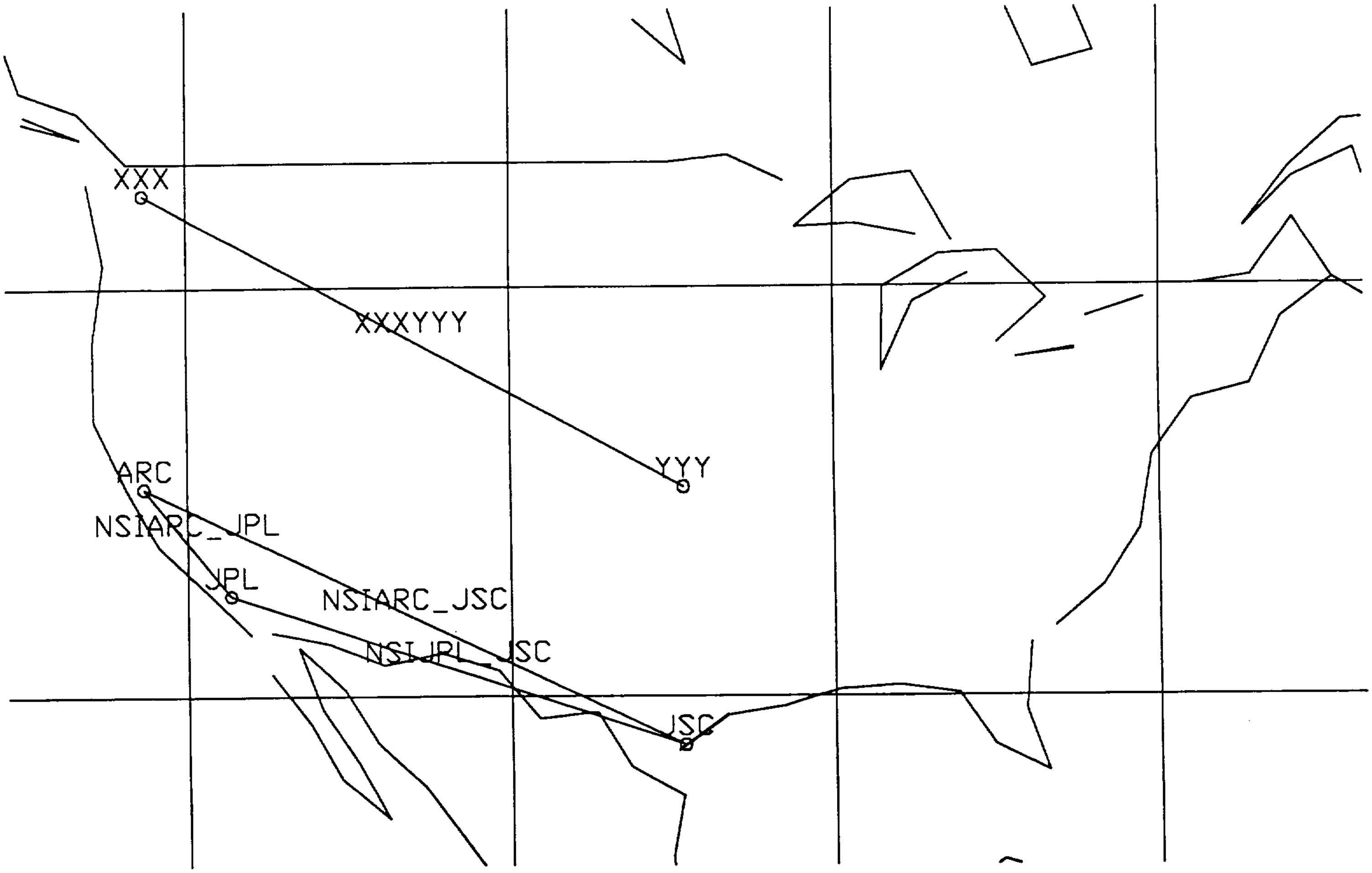
XXX
XXXYYY
YYY
ARC
NSIARC_JPL
JPL
NSIARC_JSC
NSIJPL_JSC
JSC

TABLE OF CONTENTS

INTRODUCTION

While currently available relational database management systems (RDBMS) allow inclusion of spatial information in a data model, they lack tools for presenting this information in an easily comprehensible form. Data models can be constructed to represent tangible objects with attributes describing their characteristics. The locations of these objects can be described with spatial coordinates in units such as degrees of latitude and longitude. Though these coordinates can easily be reported textually, their utility is minimal in that form. A person reading a report of objects and their locating coordinates might, at best, comprehend the locations of one or two objects at any one time. The locations and spatial relationships of several objects would not be apparent. The most useful form for presentation of spatial information is a contextual drawing. On a scale drawing the spatial relationships among selected objects and geographical features become clear. Manually plotting objects on a drawing is time-consuming and expensive, making it impractical for ad hoc reports. Computer- aided design (CAD) software packages provide adequate functions for producing drawings, but still require manual placement of symbols and features. This suggests the need for a bridge between the ad hoc reporting features of the RDBMS and the automated drawing capabilities of the CAD system.

This project is an exercise in solving a specific example of the above problem. Graphic/image diagrams displaying elements of a telecommunications network on a map could prove invaluable as a tool for planning and management of complex or wide-ranging networks. The success of such a tool is dependent on timely access to information about the network and also on the ability to quickly convert that information into a visual form. Given are the Sybase RDBMS containing a data model of the network and the AutoCAD drafting software with suitable maps. The objective of this project is an interface which will allow AutoCAD to function as a front-end to the Sybase server. This interface includes a customized AutoCAD menu, specialized Autolisp commands and functions, and C programs incorporating functions from the Sybase DB-Library. Development was carried out on a Sun 3 workstation, the target hardware, and an IBM PC clone.

The milestones for progress contained in the project proposal were followed throughout the project, and the following sections of this report describe the activities leading to their completion.

PHASE 1: AUTOCAD FROM FILE

The objective of this first phase of the project is the demonstration of a graphic display of telecommunications circuits. Each circuit will be plotted on a map as a line between the circuit's two endpoints. Each endpoint will be identified by the facility code of its location. Each circuit will be identified by its own unique code. The map will be displayed on a CRT screen. All attribute values describing the circuits will be provided in a simple local disk file. The salient process must convert the given attribute values into the graphic display in a way that is convenient and timely for the user.

The AutoCAD drafting software, version 10, was chosen for the graphic display. Version 10 is the first version available for the Sun 3 workstation, which is the target hardware of this project. AutoCAD has a well-established user base on other hardware, however, making it more likely that potential users will have experience with its fundamental features. AutoCAD provides the Autolisp interpreter, which allows creation of new functions and commands. AutoCAD's standard menu is accessible for customization to specialized tasks.

The map used for the display is an equatorial Mercator projection generated on an MSDOS PC using the WorldDXF software purchased from Micro Map & CAD, 9642 W. Virginia Cir, Lakewood, CO 80226. To minimize storage and loading time a thinning radius of 300 Km was used for testing purposes. Utility functions provided with the map generator require that the central meridian value of -75.00 degrees be stored in the AutoCAD variable userr1. For this project the map was saved with a view of the contiguous US. This primary view is shown automatically whenever the map is loaded. In addition, a customized menu was compiled while this map was loaded. The subsequent save of the map causes this menu to be automatically loaded with the map.

The data describing the telecommunications circuits to be plotted was created with a text editor and stored as a disk file in the AutoCAD working directory. The file format was named circuits, and is shown in detail later in this document. Each record contains the attribute values for one circuit. Included in these attributes are the latitude and longitude for location of each endpoint.

The file mapckt.lsp contains all the specialized Autolisp commands and functions for this project. Also included are utility map functions provided by Micro Map & CAD and written by Randy George. These utility functions are specific to the equatorial mercator maps generated by WorldDXF, and aid in converting latitude, longitude coordinates to drawing coordinates on the map. mapckt.lsp must be loaded with the Autolisp load function before use.

To read the circuits file, the Autolisp command lladdckts was created. lladdckts prompts the user for the name of the circuits format file to be plotted. It then reads each record of that file and calls the function insckt with a list of attribute values. The function insckt receives the list of attribute values for a circuit and plots it on the map. It does this by creating new drawing entities in the current AutoCAD drawing "database". This so-called database is the data structure AutoCAD uses to keep representations of all the entities of the drawing currently loaded. To represent each endpoint insckt inserts an llept block with attribute values for the endpoint facility name, the latitude, and the longitude. The visual representation of this block shows the facility name above a small circle centered on the location of the endpoint. The latitude and longitude values are stored internally, but are not normally visible on the display. Next a simple line is drawn between the two endpoints. Finally, an llckt block is inserted. The visual representation of this block shows the circuit name centered above the midpoint of the line. The remaining attributes are stored internally. Among these normally invisible attributes are unique artificial key values identifying the three drawing entities related to this circuit, the two endpoints and the line. These key values are called entity handles by AutoCAD, and they were introduced in version 10. The relationships indicated by these keys will become important for functions that delete all the entities of a single circuit or that extract all the attribute values of a circuit and its endpoints.

The menu mapckt.mnu is a simple modification of the standard AutoCAD menu. Following the example provided by the map vendor, an extra pull-down section was added to the menu file with a text editor. This allows the user with a mouse device to select the pull-down for mapping circuits and then select from the list of specialized functions and commands. Included are a function to easily load the mapckt.lsp command file. Once loaded, any of these commands can be initiated by selection with the mouse.

Phase 1 of the project solution was concluded by a successful demonstration using the elements described above. A file containing four fictitious telecommunications circuits was copied to the AutoCAD working directory. AutoCAD was executed. The contiguous US mercator map was loaded along with its customized menu. The special Autolisp commands were loaded from the menu. Then lladdckts was run from the menu. After entering the name of the circuits file, the circuits were plotted on the map display. Figure 4 illustrates the map display.

Circuits Plotted on Map

Figure 1

PHASE 2: SYBASE TO FILE

The second phase of the project requires selection of data describing telecommunications circuits from a relational database to a disk file. The data selected must include all attribute values necessary to describe each circuit and plot it on the map. The disk file must conform to the format developed for input in Phase 1, and must be local to the graphic display system.

The Sybase Relational Database Management System (RDBMS) was chosen for the database. Sybase is available on the Sun 3 workstation, and provides the necessary development tools for this project. The isql SQL interpreter and the C DB-Library were needed. In addition, Sybase has been used for development of live databases which may benefit from the results of this project.

A local test database, named spatial, was created on the Sun workstation for this phase. Using a SQL script in isql, tables were created to represent circuits ("circuit"), facilities ("facility"), organizations ("torg"), and wire centers ("wire_center"). These tables and their attributes were modeled after similar objects designed for the NASA Science Internet Database. For efficiency, attributes not necessary for this project were eliminated. In addition, rules, triggers, and permissions normally used to maintain database integrity were not implemented at this time. The table circuit includes all attributes of the circuit except for the geographic location of the endpoints. Each endpoint matches a facility name in facility. The organization key of that facility matches that of torg. torg includes a phone number whose area code and exchange match those of a wire center. wire_center includes the latitude and longitude of its location. These wire center locations are readily available, and serve to locate facilities when map resolution is not too fine. Test data was inserted into the database using a SQL script in isql. Details of the test database are shown later in this document.

A second disk file format was created for this phase. It was named results, and consists currently of a single field. The purpose of this file format is to indicate the results of a batch process. At present, a C program would write the message "SUCCEED" into the results file at the end of a normally concluded run.

A C program, gtdbckt, was created to select rows from circuit, lookup the latitude and longitude for each endpoint, and write the values to a circuits format disk file. This program was compiled on the Sun workstation, and includes the Sybase DB- Library interface to the database. This interface includes the definitions and functions necessary to send SQL transactions to the database server and receive the resulting data stream and status information. The program selects all circuits through its primary database process structure.

For each row returned, it uses its secondary database structure to select a latitude and longitude for each endpoint. This is done by matching keys through three tables as described above. The attribute values for each circuit and the latitudes and longitudes in character form are delimited by spaces and written as a record of the circuits disk file. If the program concludes normally, it writes the message "SUCCEED" to the results file. This message does not, however, indicate the success of the database selection process. gtdbckt requires a valid database server password, a name for the circuits file, and a name for the results file. These values are provided as command line arguments.

Phase 2 of the project solution was concluded by a successful demonstration of selection from the test database and creation of a disk file. The SQL scripts bldckt.sql and popckt.sql were used to build and populate the test database. gtdbckt was run from the AutoCAD working directory to produce a new circuits file in that directory. The new circuits file was compared with that created in Phase 1, and found to contain identical data in the correct format.

PHASE 3: SYBASE TO AUTOCAD

The objective of the third phase of the project is a demonstration of the combination of Phase 1 and Phase 2. This requires that the process of selection from the database be easily controlled from the graphic display system. The data resulting from that selection must then be available for plotting on the map.

The Autolisp command seldbckts was created and added to mapckt.lsp for Phase 3. This command prompts the user for a database server password and a circuits file name. It initializes an empty results file and an empty circuits file with the name entered. It then runs the C program gtdbckt with arguments for password, circuits file name, and results file name. This is done with the AutoCAD shell command as an independent process. AutoCAD waits until the process is finished. seldbckts then reads the results file to check for a "SUCCEED" message written by gtdbckt. If the message is found, the command indicates successful selection and the name of the circuits file.

Phase 3 of the project solution was concluded by a successful demonstration of database selection controlled from AutoCAD and plotting of the resulting data. The map, menu, and Autolisp commands were loaded as in Phase 1. seldbckts was run from the menu. After entering a valid password and a circuits file name, seldbckts indicated successful selection from the database. As in Phase 1, lladdckts was run. The circuits file name entered was that created by database selection. The circuits plotted on the map were found to be identical to Phase 1. The entire process was accomplished with a minimum of user input in a reasonably short time.

PHASE 4: AUTOCAD TO SYBASE

Phase 4 of the project is essentially a reversal of the first three phases. It allows the user to plot new circuits on the map by supplying attribute values to the graphic display system. All attribute values may then be extracted from the map to create a disk file. The values from the disk file may inserted into the database by a process controlled from the graphic display system.

The Autolisp command lladdckt prompts the user for all attribute values describing a circuit, including the latitudes and longitudes of the endpoints. It calls the function insckt (described in Phase 1) with a list of attribute values for the circuit to be plotted on the map.

The Autolisp command extckts prompts the user for the name of the circuits file to be created. It then searches the drawing "database" to build a set of the internal entity names of all inserts of the block llckt. It indicates to the user the number of inserts found. With each entity name in the list it calls the function extckt, which returns a list of all attribute values for that circuit with its endpoints. It then writes a record consisting of the attribute values with space delimiters to the circuits file.

The function extckt receives the entity name of an insertion of the llckt block. It extracts the attribute values from the insertion. These values include the entity handles of the two related endpoints. For each endpoint it converts the entity handle into an entity name. It calls the function extept with each entity name, and receives a list of the attribute values for the endpoint. extckt returns a list containing the attribute values from the circuit and the endpoints.

The function extept receives the entity name of an insertion of the llept block. It extracts the attribute values from the insertion and returns them in a list.

A C program, ptdbckt, was created to insert rows into the circuit database table from the circuits disk file. This program was compiled on the Sun workstation, and includes the Sybase DB- Library interface to the database. The program reads each record from the circuits disk file and uses the attribute values to form a SQL insert transaction to add a new row to the circuit table. It does not attempt to insert new facilities, organizations, or wire centers, but assumes that the endpoints of the new circuits are existing facilities. In a live database it would be necessary to implement triggers to prevent insertion of new rows that violate this assumption.

The Autolisp command insdbckts prompts the user for a database server password and a circuits file name. It initializes an empty results file, and opens the circuits file to determine its presence. It then runs the C program ptdbckt with arguments for password, circuits file name, and results file name. This is done with the AutoCAD shell command as an independent process. AutoCAD waits until the process is finished. insdbckts then reads the results file to check for a "SUCCEED" message written by ptdbckt. If the message is found, the command indicates successful insertion to the database.

Phase 4 of the project solution was concluded by a successful demonstration using the new elements described. The map, menu, and Autolisp commands were loaded. lladdckt was used to plot new circuits on the map by prompting the user to enter attribute values. extckts was used to extract attribute values to a circuits file named by the user. insdbckts was then run. After entering a valid password and the circuits file name from extckts, it indicated successful insertion into the database. To verify the success of lladdckt and extckts the circuits file was examined to compare its values with those entered by the user. Selection of the new rows from the database was used to provide a comparison for verification of ptdbckt. Both comparisons produced identical values. Excepting the additional user input needed to plot new circuits on the map, this process was found to function as easily as that of Phase 3.

CONCLUSION

From the beginning of this project it was known that certain limitations would be encountered. The detail of the map display was limited by the processing speed of the hardware platform. For this project this detail was limited by using a large thinning radius when generating the map. In the future it would be desireable to selectively limit the map detail in unused areas. Version 10 of the AutoCAD software was found to have only limited capabilities for controlling external processes. A more interactive capability would improve the interface used in this project. It is hoped that future versions will provide this improvement.

This project has demonstrated a bridge between the data model of an RDBMS and the graphic display of a CAD system. It has been shown that the CAD system can be used to control the selection of data with spatial components from the database and then quickly plot that data on a map display. It has also been shown that the CAD system can be used to extract data from a drawing and then control the insertion of that data into the database. These demonstrations have been successful in a test environment that incorporates many features of known working environments. This suggests that the techniques developed in this project could be adapted for practical use.

SPATIAL DATABASE NORMALIZED SCHEMA

```
circuit(nsi_ckt_id,type_circuit,kbps,signal,tx_medium,end_point1,
          end_point2)

facility(org_key,facility)

torg(org_key,organization,commercial_phone)

wire_center(clli,npa,nxx,lata,locality,state,latitude,longitude)
```

These database tables are intended only for the purpose of testing and demonstrating the functions of the AutoCAD graphic front-end project. Wherever possible their attributes coincide with those of the NASA Science Internet Database (NSI db). Some tables and attributes required for this project have not been implemented in the NSI db. It is hoped that the design of this project will in the future allow transfer of data with a minimum of adaptation.
Please note that all attributes described below are specified not to accept null values. Though some of these attributes are merely descriptive, they require non-null values to protect the integrity of derivative data structures in other parts of the system. The space-delimited disk files used by AutoCAD for input and output of block attributes would be corrupted by null values. In addition, AutoCAD will not allow null values to be entered as block attribute values. Specifying non-null default values for database attributes might be used as an alternative method of solving this problem.

```
Name of Database Server:
     lilmud
Name of the Database:
     spatial
Name of the Table:
     circuit
     1)  Name of Attribute:
          nsi_ckt_id
     2)  Field Characteristics:
          char(20)
     3)  Index Name and Characteristics:
          unique clustered index ckt_index on circuit(nsi_ckt_id)
     4)  Nulls:
          not null
     5)  Keys:
          pk
     6)  Domain Rules:
          Unique identifier for circuit.
          Prevent update by access control.
          Service and material id.
          See NSI db design.
Name of Database Server:
     lilmud
Name of the Database:
     spatial
Name of the Table:
     circuit
     1)  Name of Attribute:
          type_circuit
      2)  Field Characteristics:
          char(10)
      3)  Index Name and Characteristics:
          no index
      4)  Nulls:
          not null
      5)  Keys:
          not a key field
      6)  Domain Rules:
          (e.g. common_carrier, ethernet)
          See NSI db design.
Name of Database Server:
     lilmud
Name of the Database:
     spatial
Name of the Table:
     circuit
     1)  Name of Attribute:
         kbps
     2)  Field Characteristics:
         char(04)
     3)  Index Name and Characteristics:
         no index
     4)  Nulls:
         not null
     5)  Keys:
         not a key field
     6)  Domain Rules:
         Data rate of circuit in thousand bits per second.
         See NSI db design.
Name of Database Server:
     lilmud
Name of the Database:
     spatial
Name of the Table:
     circuit
     1)  Name of Attribute:
         signal
     2)  Field Characteristics:
         char(11)
     3)  Index Name and Characteristics:
         no index
     4)  Nulls:
         not null
     5)  Keys:
          not a key field
     6)  Domain Rules:
          Signal type of circuit (e.g. ANALOG, DIGITAL).
          See NSI db design.
Name of Database Server:
     lilmud
Name of the Database:
     spatial
Name of the Table:
     circuit
     1)  Name of Attribute:
          tx_medium
     2)  Field Characteristics:
          char(20)
     3)  Index Name and Characteristics:
          no index
     4)  Nulls:
          not null
     5)  Keys:
          not a key field
     6)  Domain Rules:
          Transmission medium of circuit (e.g. TERRESTRIAL).
          See NSI db design.
Name of Database Server:
     lilmud
Name of the Database:
     spatial
Name of the Table:
     circuit
     1)  Name of Attribute:
          end_point1
     2)  Field Characteristics:
          char(10)
     3)  Index Name and Characteristics:
          nonclustered index ce1_index on circuit(end_point1)
 4)  Nulls:
          not null
     5)  Keys:
          fk->facility.facility INS RST
                                UPD RST
     6)  Domain Rules:
          Facility of first endpoint of circuit.
Name of Database Server:
     lilmud
Name of the Database:
     spatial
```

```
Name of the Table:
     circuit
     1)  Name of Attribute:
          end_point2
     2)  Field Characteristics:
          char(10)
     3)  Index Name and Characteristics:
          nonclustered index ce2_index on circuit(end_point2)
 4)  Nulls:
          not null
     5)  Keys:
          fk->facility.facility INS RST
                                UPD RST
     6)  Domain Rules:
          Facility of second endpoint of circuit.
Name of Database Server:
     lilmud
Name of the Database:
     spatial
Name of the Table:
     facility
     1)  Name of Attribute:
          org_key
     2)  Field Characteristics:
          int
     3)  Index Name and Characteristics:
          unique clustered index fac_index on
          facility(org_key,facility)
     4)  Nulls:
          not null
     5)  Keys:
          pk (part of composite:  facility.org_key, facility)
          fk->torg.org_key INS RST
     6)  Domain Rules:
          Prevent update by access control.
Name of Database Server:
     lilmud
Name of the Database:
     spatial
Name of the Table:
     facility
     1)  Name of Attribute:
          facility
     2)  Field Characteristics:
          char(20)
     3)  Index Name and Characteristics:
         unique clustered index fac_index on
         facility(org_key,facility)
     4)  Nulls:
         not null
     5)  Keys:
         pk (part of composite:  facility.org_key, facility)
           ->circuit.end_point1 DLT RST
           ->circuit.end_point2 DLT RST
     6)  Domain Rules:
         Facility code of facility.
         Prevent update by access control.
Name of Database Server:
     lilmud
Name of the Database:
     spatial
Name of the Table:
     torg
     1)  Name of Attribute:
         org_key
     2)  Field Characteristics:
         int
     3)  Index Name and Characteristics:
         unique clustered index org_index on
         torg(org_key,organization)
     4)  Nulls:
         not null
     5)  Keys:
         pk (part of composite:  torg.org_key, organization)
           ->facility.org_key DLT RST
     6)  Domain Rules:
         Unique artificial key for torg.
         Prevent update by access control.
         See NSI db design.
Name of Database Server:
     lilmud
Name of the Database:
     spatial
Name of the Table:
     torg
     1)  Name of Attribute:
         organization
     2)  Field Characteristics:
         char(30)
     3)  Index Name and Characteristics:
         unique clustered index org_index on
         torg(org_key,organization)
```

4) Nulls:
not null
5) Keys:
pk (part of composite: torg.org_key, organization)
6) Domain Rules:
Identifier of torg.
Prevent update by access control.
See NSI db design.

Name of Database Server:
lilmud
Name of the Database:
spatial
Name of the Table:
torg
1) Name of Attribute:
commercial_phone
2) Field Characteristics:
char(20)
3) Index Name and Characteristics:
no index
4) Nulls:
not null
5) Keys:
not a key field
6) Domain Rules:
Phone_rule: (nnn)nnn-nnnn
See NSI db design.

Name of Database Server:
lilmud
Name of the Database:
spatial
Name of the Table:
wire_center
1) Name of Attribute:
clli
2) Field Characteristics:
char(11)
3) Index Name and Characteristics:
unique clustered index wct_index on wire_center(clli)
4) Nulls:
not null
5) Keys:
pk
6) Domain Rules:
Unique identifier for wire_center.
Prevent update by access control.

```
          See NSI db design.
Name of Database Server:
     lilmud
Name of the Database:
     spatial
Name of the Table:
     wire_center
     1)  Name of Attribute:
          npa
     2)  Field Characteristics:
          char(03)
     3)  Index Name and Characteristics:
          nonclustered index wcx_index on wire_center(npa,nxx)
     4)  Nulls:
          not null
     5)  Keys:
          (part of composite: npa, nxx)
          lookup->substring(commercial_phone,2,3)
     6)  Domain Rules:
          Areacode of wire_center.
          See NSI db design.
Name of Database Server:
     lilmud
Name of the Database:
     spatial
Name of the Table:
     wire_center
     1)  Name of Attribute:
          nxx
     2)  Field Characteristics:
          char(03)
     3)  Index Name and Characteristics:
          nonclustered index wcx_index on wire_center(npa,nxx)
     4)  Nulls:
          not null
     5)  Keys:
          (part of composite: npa, nxx)
          lookup->substring(commercial_phone,6,3)
     6)  Domain Rules:
          Exchange of wire_center.
          See NSI db design.
Name of Database Server:
     lilmud
Name of the Database:
     spatial
Name of the Table:
     wire_center
     1)   Name of Attribute:
           lata
     2)   Field Characteristics:
           char(03)
     3)   Index Name and Characteristics:
           no index
     4)   Nulls:
           not null
     5)   Keys:
           not a key field
     6)   Domain Rules:
           Lata of wire_center.
           See NSI db design.
Name of Database Server:
     lilmud
Name of the Database:
     spatial
Name of the Table:
     wire_center
     1)   Name of Attribute:
           locality
     2)   Field Characteristics:
           char(26)
     3)   Index Name and Characteristics:
           no index
     4)   Nulls:
           not null
     5)   Keys:
           not a key field
     6)   Domain Rules:
           Locality of wire_center.
           See NSI db design.
Name of Database Server:
     lilmud
Name of the Database:
     spatial
Name of the Table:
     wire_center
     1)   Name of Attribute:
           state
     2)   Field Characteristics:
           char(02)
     3)   Index Name and Characteristics:
           no index
     4)   Nulls:
          not null
     5)   Keys:
          not a key field
     6)   Domain Rules:
          State of wire_center.
          See NSI db design.
Name of Database Server:
     lilmud
Name of the Database:
     spatial
Name of the Table:
     wire_center
     1)   Name of Attribute:
          latitude
     2)   Field Characteristics:
          float
     3)   Index Name and Characteristics:
          no index
     4)   Nulls:
          not null
     5)   Keys:
          not a key field
     6)   Domain Rules:
          Latitude of wire_center in decimal degrees.
Name of Database Server:
     lilmud
Name of the Database:
     spatial
Name of the Table:
     wire_center
     1)   Name of Attribute:
          longitude
     2)   Field Characteristics:
          float
     3)   Index Name and Characteristics:
          no index
     4)   Nulls:
          not null
     5)   Keys:
          not a key field
     6)   Domain Rules:
          Longitude of wire_center in decimal degrees.
```

SQL SCRIPTS

```
/*
** Script:       bldckt.sql (build circuit)
** Author:       Tom Brownfield 1990
** Interpreter:  Sybase isql
** Purpose:      Create database objects necessary for
**               demonstrating the AutoCAD graphic
**               front-end with circuits data.
*/

/* Use the spatial database. */
use spatial
go

/* Create tables. */
create table circuit
    (nsi_ckt_id         char(20)           not null
    ,type_circuit       char(10)           not null
    ,kbps               char(04)           not null
    ,signal             char(11)           not null
    ,tx_medium          char(20)           not null
    ,end_point1         char(10)           not null
    ,end_point2         char(10)           not null
    )
go

create table facility
    (org_key            int                not null
    ,facility           char(10)           not null
    )
go

create table torg
    (org_key            int                not null
    ,organization       char(30)           not null
    ,commercial_phone   char(20)           not null
    )
go

create table wire_center
    (clli               char(11)           not null
    ,npa                char(03)           not null
    ,nxx                char(03)           not null
    ,lata               char(03)           not null
    ,locality           char(26)           not null
    ,state              char(02)           not null
    ,latitude           float              not null
    ,longitude          float              not null
    )
go

/* Create indexes. */
create unique clustered index ckt_index
    on circuit(nsi_ckt_id)
go

create unique clustered index fac_index
    on facility(org_key, facility)
go

create unique clustered index org_index
    on torg(org_key, organization)
go

create unique clustered index wct_index
    on wire_center(clli)
go

create nonclustered index wcx_index
    on wire_center(npa, nxx)
go

/* end */
/*
** Script:       popckt.sql (populate circuit)
** Author:       Tom Brownfield 1990
** Interpreter:  Sybase isql
** Purpose:      Populate database tables necessary
**               for demonstrating the AutoCAD graphic
**               front-end with circuits data.
*/

/* Use the spatial database. */
use spatial
go

/* Insert wire_center data. */
begin tran
insert into wire_center values
    ('X'
    ,'206'
    ,'111'
    ,'001'
    ,'Xtown'
    ,'WA'
    ,48.0
    ,-122.0
    )
commit tran
go

begin tran
insert into wire_center values
    ('Y'
    ,'402'
    ,'222'
    ,'002'
    ,'Ytown'
    ,'NB'
    ,38.0
    ,-97.0
    )
commit tran
go

begin tran
insert into wire_center values
    ('A'
    ,'415'
    ,'335'
    ,'003'
    ,'Mountain View'
    ,'CA'
    ,38.0
    ,-122.0
    )
commit tran
go

begin tran
insert into wire_center values
    ('B'
    ,'713'
    ,'444'
    ,'004'
    ,'Houston'
    ,'TX'
    ,28.0
    ,-97.0
    )
commit tran
go

begin tran
insert into wire_center values
    ('C'
    ,'818'
    ,'555'
    ,'005'
    ,'Pasadena'
    ,'CA'
    ,34.0
    ,-118.0
    )
commit tran
go

/* Insert torg data. */
begin tran
insert into torg values
    (1
    ,'Xorg'
    ,'(206)111-1111'
    )
commit tran
go

begin tran
insert into torg values
    (2
    ,'Yorg'
    ,'(402)222-2222'
    )
commit tran
go

begin tran
insert into torg values
    (3
    ,'NASA Ames'
    ,'(415)335-3333'
    )
```

```
commit tran
go

begin tran
insert into torg values
     (4
     ,'NASA Johnson'
     ,'(713)444-4444'
     )
commit tran
go

begin tran
insert into torg values
     (5
     ,'NASA JPL'
     ,'(818)555-5555'
     )
commit tran
go

/* Insert facility data. */

begin tran
insert into facility values
     (1
     ,'XXX'
     )
commit tran
go

begin tran
insert into facility values
     (2
     ,'YYY'
     )
commit tran
go

begin tran
insert into facility values
     (3
     ,'ARC'
     )
commit tran
go
```

```
begin tran
insert into facility values
    (4
    ,'JSC'
    )
commit tran
go

begin tran
insert into facility values
    (5
    ,'JPL'
    )
commit tran
go

/* Insert circuit data. */
begin tran
insert into circuit values
    ('XXXYYY'
    ,'NSN'
    ,'224'
    ,'DIGITAL'
    ,'TERRESTRIAL'
    ,'XXX'
    ,'YYY'
    )
commit tran
go

begin tran
insert into circuit values
    ('NSIARC_JSC'
    ,'NSN'
    ,'448'
    ,'DIGITAL'
    ,'TERRESTRIAL'
    ,'ARC'
    ,'JSC'
    )
commit tran
go

begin tran
insert into circuit values
```

```
    ('NSIARC_JPL'
    ,'NSN'
    ,'168'
    ,'DIGITAL'
    ,'TERRESTRIAL'
    ,'ARC'
    ,'JPL'
    )
commit tran
go

begin tran
insert into circuit values
    ('NSIJPL_JSC'
    ,'NSN'
    ,'168'
    ,'DIGITAL'
    ,'TERRESTRIAL'
    ,'JPL'
    ,'JSC'
    )
commit tran
go

/* List all entries in the database. */
print '*** wire_center ***'
select * from wire_center
go

print '*** torg ***'
select * from torg
go

print '*** facility ***'
select * from facility
go

print '*** circuit ***'
select * from circuit
go

/* end */
```

DISK FILE SCHEMA

```
circuits(nsi_ckt_id,type_circuit,kbps,signal,tx_medium,end_point1
         ,latitude1,longitude1,end_point2,latitude2,longitude2)

results(success)
```

These disk files provide the needed data storage between the database system and the AutoCAD system. They are sequential text files contained in the current working directory. All attributes are variable-length character fields delimited by spaces. Spaces are optional at the end of a record.

The circuits file contains latitudes and longitudes which are character representations of values contained in the database as wire_center.latitude and wire_center.longitude. All other attribute values are identical to those in the database.

The results file indicates the results of a DB-LIBRARY C program process. Success will contain the value "SUCCEED" when the program writing it is successfully completed. This does not, however, indicate the success or failure of individual database transactions. At this time such indications are returned as database messages and errors through standard console output. Additional attributes should be added to this file if this interface is to be used in an environment where the user requires improved control information.

AUTOCAD MENU

The following is a partial listing of mapckt.mnu from the beginning of the file through the modified section:

```
***BUTTONS
;
$p1=*
^c^c
^B
^O
^G
^D
^E
^T
***AUX1
;
$p1=*
^C^C
^B
^O
^G
^D
^E
^T
***POP1
[Tools]
[OSNAP]^C^C$p1= $p1=* OSNAP \
[CENter]CENTER
[ENDpoint]ENDPOINT
[INSert]INSERT
[INTersec]INTERSEC
[MIDpoint]MIDPOINT
[NEArest]NEAREST
[NODe]NODE
[PERpend]PERPEND
[QUAdrant]QUADRANT
[QUICK,]QUICK,^Z
[TANgent]TANGEN
NONE
[~--]
[Cancel]^C^C
[U]^C^CU
[Redo]^C^CREDO
[Redraw]'REDRAW
```

```
***POP2
[Draw]
[Line]*^C^C$S=X $s=line line
[Arc]*^C^C$S=X $s=poparc arc
[Circle]*^C^C$S=X $s=popcircl circle
[Polyline]*^C^C$S=X $s=pline pline
[Insert]^C^Csetvar attdia 1 $s=insert insert
[Dtext]*^C^C$S=X $s=Dtext Dtext
[Hatch]^C^C$i=hatch1 $i=*

***POP3
[Edit]
[Erase]*^C^Cerase si auto
[Move]*^C^Cmove si auto
[Copy]*^C^Ccopy si auto
[Trim]*^C^C$S=X $s=trim trim auto
[Extend]*^C^C$S=X $s=extend extend auto
[Stretch]*^C^C$S=X $s=stretch stretch crossing
[Polyedit]*^C^C$S=X $s=pedit pedit

***POP4
[Display]
[Window]'zoom w
[Previous]'zoom p
[Dynamic]'zoom d
[Pan]'pan
[3D View]^C^C$i=3dviews $i=*

***POP5
[Modes]
[Drawing Aids]'ddrmodes
[Entity Creation]'ddemodes
[Modify Layer]'ddlmodes

***POP6
[Options]
[Ashade]^P(cond ((null C:SCENE) +
(vmon) (prompt "Please wait...  Loading ashade.  ") +
(load "ashade")) (T (princ))) ^P$i=as $i=*
[3D Objects]^P(cond ((null C:CONE) +
(vmon) (prompt "Please wait...  Loading 3D Objects.  ") +
(load "3d")) (T (princ))) ^P$i=3DObjects $i=*
[Fonts]^C^C$i=fonts1 $i=*
[DXFScript]^C^CDXFSCRIPT SCRIPT;D:/S/DXF GRAPHSCR;
[Stack]^C^CSTACK
[MOD]^C^CMOD
```

```
***POP7
[File]
[Save]^C^CSave
[End]^C^Cend
[Quit]^C^C$S=X $s=quit quit
[~--]
[Plot]^C^Cplot
[Print]^C^Cprplot


***POP8
[Mercator Map]
[LOAD Merc]^C^C(load "mercator")
[LOAD Merc3D]^C^C(load "merc3D")
[Lat Long]^C^CLatLong
[Map Dist]^C^CMapDist
[Add Pt]^C^CLLaddPt
[Add Pts]^C^CLLaddPts
[Del Pt]^C^CLLdelPt
[Del Pts]^C^CLLdelPts
[Map Radius]^C^CMapRadius
[Add Lines]^C^CLLaddLines
[WDBII]^C^CWDB \\WDBtoDXF script load

***POP9
[Map Circuits]
[LOAD Circuits]^C^C(load "mapckt")
[Add Ckts]^C^CLLaddCkts
[Add Ckt]^C^CLLaddCkt
[Del Ckts]^C^CDelCkts
[Ext Ckts]^C^CExtCkts
[Sel Ckts]^C^CSelDBCkts
[Ins Ckts]^C^CInsDBCkts
```

AUTOLISP COMMANDS AND FUNCTIONS

```
;**********
;* File:         mapckt.lsp (map circuits)
;* Interpreter:  AutoCAD Autolisp version 10
;* Purpose:      Provide AutoCAD commands and functions
;*               to plot circuits on a world-mercator map.
;**********

;**********
;*Global variables and utility map functions from
;*mercator.lsp by Randy George 8/15/88.  Modifications
;*are noted by comments.
;
;* Global variables:
; earth : earth radius in meters 6371
; lat   : latitude
; long  : longitude
; inc   : lat or long increment
; vb    : Blipmode variable
; vc    : Cmdecho variable
```

```
;**********
;* Utility map functions:

(vmon)
(prompt "\nLoading. Please wait...")
(terpri)

(defun MODES (a)
   (setq MLST '())
   (repeat (length a)
      (setq MLST (append MLST (list (list (car a) (getvar
         (car a))))))
      (setq a (cdr a)))
)

(defun MODER ()
   (repeat (length MLST)
      (setvar (caar MLST) (cadar MLST))
      (setq MLST (cdr MLST))
   )
)

(defun *ERROR* (st)
  (moder)
  (terpri)
  (princ "\nError: ")
  (princ st)
  (princ)
)

(defun sqr(x)
  (* x x)
)

(defun arccos (x)
  (if (/= (abs x) 1.0)
      (- (* pi 0.5) (atan (/ x (sqrt (- 1.0 (* x x))))))
      (* (- 1.0 x) pi 0.5)
  )
)

(defun arcsin (x)
  (if (/= (abs x) 1.0)
      (atan (/ x (sqrt (- 1.0 (* x x)))))
      (* x pi 0.5)
  )
)

(defun arctanh (x)
  (if (/= x 1.0)
      (/ (log (/ (+ 1.0 x) (- 1.0 x))) 2.0)
      (list 99999999.0)
  )
)

(defun tanh (x / e1 e2)
  (setq e1 (exp x))
  (setq e2 (exp (* -1.0 x)))
  (/ (- e1 e2) (+ e1 e2))
)

(defun Radian( deg )
  (* deg 0.0174533)
)

(defun Degree( rad)
  (* 57.29578 rad)
)

(defun Meridian (LongR LongR0 / delta)
  (setq delta (- LongR LongR0))
  (if (< delta -3.1415927)
      (setq delta (+ delta 6.2831853))
  )
  (if (> delta 3.1415927)
      (setq delta (- delta 6.2831853))
  )
  (setq delta delta)
)


(defun Mercator (LongR LongR0 LatR radius /)
  (if (< (abs (Radian LatR)) 1.397)
      (progn
        (setq LongR (Meridian (Radian LongR) (Radian LongR0)))
        (setq x (* radius LongR))
        (setq y (* radius (arctanh (sin (Radian LatR)))))
        (list x y)
      )
      (list 0.0 0.0)
  )
)
```

```
(defun InvMeridian (delta LongR0 / )
  (setq delta (+ delta LongR0))
  (if (< delta -3.1415927)
      (setq delta (+ delta 6.2831853))
  )
  (if (> delta 3.1415927)
      (setq delta (- delta 6.2831853))
  )
  (setq delta delta)
)

(defun InvMercator ( pt LongR0 radius / delta )
  (setq delta (/ (car pt) radius))
  (setq LongR (Degree (InvMeridian delta (Radian LongR0))))
  (setq LatR (Degree (arcsin (tanh (/ (cadr pt) radius)))))
  (list LatR LongR)
)

(defun Arc ( pt radius / )
  (setq LongR (Degree (/ (car pt) radius)))
  (setq LatR (Degree (arcsin (tanh (/ (cadr pt) radius)))))
  (list LatR LongR)
)


;  Parse input string into list of strings
(defun sparse (S / LL TMP CNT)
  (setq TMP "" CNT 0)
  (while (< CNT (strlen S))
     (setq CNT (1+ CNT))
     (cond
       ((and (or (= (substr S CNT 1) ",")
                 (= (substr S CNT 1) " ")) (/= TMP ""))
         (setq LL (cons TMP LL) TMP ""))
       ((= (substr S CNT 1) ","))
       ((= (substr S CNT 1) " "))
       (t (setq TMP (strcat TMP (substr S CNT 1))))
     )
  )
  ;*modified to prevent adding null element to end of list
  ;*Tom Brownfield 12/5/90
  (if (/= TMP "")
     (reverse (cons TMP LL))
     (reverse LL)
  )
```

;*

)

```
;**********
;* Main Program
;**********

;**********
;* Function:  insckt (insert a circuit)
;* Author:    Tom Brownfield 1990
;* Purpose:   Insert blocks for a circuit and two endpoints
;*            and draw a line between the endpoints.
;* Inputs:    central meridian, earth radius, attribute
;*            values for circuit and endpoints.
;* Outputs:   command line.
;
(defun insckt
   ( central earth
     elat1 elong1 fac1 elat2 elong2 fac2
     cktid tckt kbps sgnl txmed
   / errcnt lat long pt1 ep1eh pt2 ep2eh lineeh pt3)
   ;
   ;*Initialize error count.
   (setq errcnt 0)
   ;
   ;*Convert latitude, longitude to float.
   (setq lat (atof elat1))
   (if (null lat) (setq lat 0.0))
   (setq long (atof elong1))
   (if (null long) (setq long 0.0))
   ;*
   ;*Get drawing coordinates for endpoint1 by calling the
   ;*the function mercator with latitude, longitude,
   ;*central meridian as arguments.
   (setq pt1 (mercator long central lat earth))
   (if (and (= (car pt1) 0.0) (= (cadr pt1) 0.0))
      (progn
         (setq errcnt (1+ errcnt))
         (princ "**** Invalid latitude/longitude ***")
         (terpri)
         (princ (rtos lat 2 4))
         (princ ",")
         (princ (rtos long 2 4))
      )
   )
   ;
   ;*Get drawing coordinates for endpoint2 as above.
   (setq lat (atof elat2))
   (if (null lat) (setq lat 0.0))
   (setq long (atof elong2))
   (if (null long) (setq long 0.0))
   (setq pt2 (mercator long central lat earth))
   (if (and (= (car pt2) 0.0) (= (cadr pt2) 0.0))
      (progn
         (setq errcnt (1+ errcnt))
         (princ "**** Invalid latitude/longitude ***")
         (terpri)
         (princ (rtos lat 2 4))
         (princ ",")
         (princ (rtos long 2 4))
      )
   )
   ;
   ;*Execute block inserts of endpoints, at the drawing
   ;*coordinate points, with attribute values.
   ;*Get the entity handles of the endpoint inserts.
   ;*Draw a line between the endpoints.
   ;*Get the entity handle of the line.
   ;*Get the coordinates of the midpoint of the line.
   ;*Execute a block insert of the circuit, at the
   ;*midpoint, with attribute values including the three
   ;*entity handles of the related entities.
   (if (= errcnt 0)
      (progn
         (command "insert" "llept" pt1 "" "" "" fac1 elat1
            elong1)
         (setq ep1eh (cdr (assoc 5 (entget (entlast)))))
         (command "insert" "llept" pt2 "" "" "" fac2 elat2
            elong2)
         (setq ep2eh (cdr (assoc 5 (entget (entlast)))))
         (command "LINE" pt1 pt2 "")
         (setq lineeh (cdr (assoc 5 (entget (entlast)))))
         (setq pt3 (polar pt1 (angle pt1 pt2)
            (/(distance pt1 pt2)2)))
         (command "insert" "llckt" pt3 "" "" "" cktid tckt kbps
            sgnl
            txmed lineeh ep1eh ep2eh)
      )
   )
)

;**********
;* Command:  lladdckts (add circuits at latitude, longitude)
;* Author:   Tom Brownfield 1990
;* Purpose:  Read attributes from a circuits file, and plot
;*           the circuits with endpoints on the map using
;*           latitudes and longitudes from the file.
;* Inputs:   command line
;*           circuits disk file
;* Outputs:  command line
;
(defun C:lladdckts
   (
   / central earth datafl f1
     sln1 slst1 slen1
     cktid tckt kbps sgnl txmed
     fac1 elat1 elong1
     fac2 elat2 elong2
   )
   ;
   ;*Save and turn off echo and blip.
   (modes '("CMDECHO" "BLIPMODE"))
   (setvar "CMDECHO" 0)
   (setvar "BLIPMODE" 0)
   ;
   ;*Set layer to POINTS.
   (command "LAYER" "M" "POINTS" "")
   ;
   ;*Get central meridian of map.
   (setq central (getvar "USERR1"))
   ;
   ;*Set earth radius.
   (setq earth 6371.0 )
   ;
   ;*Be sure entity handles are on.
   (command "HANDLES" "ON")
   ;
   ;*Get the name of the circuits file.
   ;*Open the file for input.
   (setq datafl (getstring "\nEnter name of circuits file: "))
   (setq f1 (open datafl "r"))
   (if (not f1)
      (*ERROR* (strcat "cannot open file: " datafl))
   )
   (if f1
      (progn
         ;
         ;*Read the first line from the circuits file.
         (setq sln1 (read-line f1))
         ;
         ;*Process each line of the circuits file.
```

```
(while sln1
   (progn
      ;
      ;*Get a list of attribute values by calling
      ;*the function sparse with the input line as
      ;*its argument.
      ;*Count the number of attribute values--there
      ;*should be 11.
      (setq slst1 (sparse sln1))
      (setq slen1 (length slst1))
      (if (= slen1 11)
         (progn
            ;
            ;*Set variables to the attribute values
            ;*from the list.
            (setq cktid (car slst1))
            (setq slst1 (cdr slst1))
            (setq tckt (car slst1))
            (setq slst1 (cdr slst1))
            (setq kbps (car slst1))
            (setq slst1 (cdr slst1))
            (setq sgnl (car slst1))
            (setq slst1 (cdr slst1))
            (setq txmed (car slst1))
            (setq slst1 (cdr slst1))
            (setq fac1 (car slst1))
            (setq slst1 (cdr slst1))
            (setq elat1 (car slst1))
            (setq slst1 (cdr slst1))
            (setq elong1 (car slst1))
            (setq slst1 (cdr slst1))
            (setq fac2 (car slst1))
            (setq slst1 (cdr slst1))
            (setq elat2 (car slst1))
            (setq slst1 (cdr slst1))
            (setq elong2 (car slst1))
            (setq slst1 (cdr slst1))
            ;
            ;*Insert circuit entities by calling
            ;*the function insckt with the
            ;*attribute values as arguments.
            (InsCkt
               central earth
               elat1 elong1 fac1 elat2 elong2 fac2
               cktid tckt kbps sgnl txmed
            )
```

```
                  )
                  (progn
                     (*ERROR* "bad input data")
                     (princ (strcat "\nInput item count: "
                        (itoa slen1)))
                  )
               )
               ;
               ;*Read the next line from the
               ;*circuits file.
               (setq sln1 (read-line f1))
            )
         )
         ;
         ;*Close the circuits file.
         (close f1)
      )
   )
   ;
   ;*Restore echo and blip.
   (moder)
)

;**********
;* Command:  lladdckt (add circuit at latitude, longitude)
;* Author:   Tom Brownfield 1990
;* Purpose:  Get attributes from the command line and plot
;*           a circuit with endpoints on the map using
;*           the latitudes and longitudes entered.
;* Inputs:   command line
;* Outputs:  command line
;
(defun C:lladdckt
   (
   / central earth
     clat1 clong1 clat2 clong2 cktid tckt kbps sgnl txmed
     elat1 elong1 eckt1 fac1
     elat2 elong2 eckt2 fac2)
   ;
   ;*Save and turn off echo and blip.
   (modes '("CMDECHO" "BLIPMODE"))
   (setvar "CMDECHO" 0)
   (setvar "BLIPMODE" 0)
   ;
   ;*Set layer to POINTS.
   (command "LAYER" "M" "POINTS" "")
   ;
   ;*Get central meridian of map.
   (setq central (getvar "USERR1"))
   ;
   ;*Set earth radius.
   (setq earth 6371.0 )
   ;
   ;*Be sure entity handles are on.
   (command "HANDLES" "ON")
   ;
   ;*Get attribute values from the command line.
   (setq elat1 (rtos
      (getreal "\nEnter first Latitude in decimal degrees: ")))
   (setq elong1 (rtos
      (getreal "\nEnter first Longitude in decimal degrees : ")))
   (setq fac1 (getstring "\nEnter FACILITY: "))
   (setq elat2 (rtos
      (getreal "\nEnter second Latitude in decimal degrees: ")))
   (setq elong2 (rtos
      (getreal
         "\nEnter second Longitude in decimal degrees : ")))
   (setq fac2 (getstring "\nEnter FACILITY: "))
   (setq cktid (getstring "\nEnter NSI_CKT_ID: "))
   (setq tckt (getstring "\nEnter TYPE_CIRCUIT: "))
   (setq kbps (getstring "\nEnter KBPS: "))
   (setq sgnl (getstring "\nEnter SIGNAL: "))
   (setq txmed (getstring "\nEnter TX_MEDIUM: "))
   ;
   ;*Insert circuit entities by calling the function
   ;*insckt with the attribute values as arguments.
   (InsCkt
      central earth
      elat1 elong1 fac1 elat2 elong2 fac2
      cktid tckt kbps sgnl txmed
   )
   ;
   ;*Restore echo and blip.
   (moder)
)

;**********
;* Function:  delckt (delete circuit)
;* Author:    Tom Brownfield 1990
;* Purpose:   Delete a circuit with its endpoints from
;*            the map.
;* Inputs:    entity name of circuit insert
;* Outputs:   none
;
(defun delckt( ename / ent etype etag lineeh ep1eh ep2eh dname )
   ;
   ;*Save the given circuit entity name.
   (setq dname ename)
   ;
   ;*Get the first entity association list and entity type
   ;*for the given entity name.
   (setq ent (entget ename))
   (setq etype (cdr (assoc 0 ent)))
   ;
   ;*Process each entity association list in the INSERT
   ;*sequence.
   (while (not (equal etype "SEQEND"))
      (progn
         ;
         ;*For attribute entity lists:
         ;*   Get the attribute tags;
         ;*   Set variables to attribute values
         ;*   for the line and the two endpoint
         ;*   entity handles.
         (if (equal etype "ATTRIB")
            (progn
               (setq etag (cdr (assoc 2 ent)))
               (if (equal etag "LINE_EH")
                  (progn
                     (setq lineeh (cdr (assoc 1 ent)))
                  )
               )
               (if (equal etag "EP1_EH")
                  (progn
                     (setq ep1eh (cdr (assoc 1 ent)))
                  )
               )
               (if (equal etag "EP2_EH")
                  (progn
                     (setq ep2eh (cdr (assoc 1 ent)))
                  )
               )
            )
         )
         ;
         ;*Get the next entity name, association list,
         ;*and entity type.
         (setq ename (entnext ename))
         (setq ent (entget ename))
         (setq etype (cdr (assoc 0 ent)))
      )
   )
   ;
   ;*Delete the circuit entity.
   (entdel dname)
   ;
   ;*Get the entity names for the entity handles of
   ;*the line and the two endpoints.
   ;*Delete the line and two endpoints.
   (setq ename (handent lineeh))
   (entdel ename)
   (setq ename (handent ep1eh))
   (entdel ename)
   (setq ename (handent ep2eh))
   (entdel ename)
)

;**********
;* Command:  delckts (delete all circuits)
;* Author:   Tom Brownfield 1990
;* Purpose:  Find all circuits plotted on the map
;*           and delete them.
;* Inputs:   command line
;* Outputs:  command line
;
(defun C:delckts( / ss sscnt ssidx ename)
   ;
   ;*Save and turn off echo.
   (modes '("CMDECHO"))
   (setvar "CMDECHO" 0)
   ;
   ;*Build a selection set of all entity names of
   ;*insertions of the block LLCKT on the layer POINTS.
   (setq ss (ssget "X" (list (cons 0 "INSERT")
                             (cons 2 "llckt")
                             (cons 8 "POINTS"))))
   ;
   ;*Count the number of entities selected.
   (if (null ss)
      (setq sscnt 0)
      (setq sscnt (sslength ss))
   )
   (princ (itoa sscnt))
   (princ " circuits will be deleted.")
   (terpri)
   ;
   ;*Set index for the first entity name in the set.
   (setq ssidx 0)
   ;
   ;*Process each entity name in the set.
   (while (> sscnt 0)
      (progn
         (setq ename (ssname ss ssidx))
         ;
         ;*Delete the circuit by calling the function
         ;*delckt with the entity name as its argument.
         (delckt ename)
         ;
         ;*Set index for the next entity name in the set.
         (setq ssidx (1+ ssidx))
         (setq sscnt (1- sscnt))
      )
   )
   ;
   ;*Redraw the screen to clean up the map.
   (redraw)
   ;
   ;*Restore echo.
   (moder)
   (princ)
)

;**********
;* Function:  extept (extract endpoint)
;* Author:    Tom Brownfield 1990
;* Purpose:   Extract attribute values from an endpoint.
;* Inputs:    entity name of endpoint insertion
;* Outputs:   list of attribute values from endpoint
;
(defun extept
   ( ename
   / ent etype
     efac elat elong
   )
   ;
   ;*Get the first entity association list and entity type
   ;*for the given entity name.
   (setq ent (entget ename))
   (setq etype (cdr (assoc 0 ent)))
   ;
   ;*Process each entity association list in the INSERT
   ;*sequence.
   (while (not (equal etype "SEQEND"))
      (progn
         ;
         ;*For attribute entity lists:
         ;*   Get the attribute tags;
         ;*   Set variables to attribute values.
         (if (equal etype "ATTRIB")
            (progn
               (setq etag (cdr (assoc 2 ent)))
               (if (equal etag "FACILITY")
                  (setq efac (cdr (assoc 1 ent)))
               )
               (if (equal etag "LATITUDE")
                  (setq elat (cdr (assoc 1 ent)))
               )
               (if (equal etag "LONGITUDE")
                  (setq elong (cdr (assoc 1 ent)))
               )
            )
         )
         ;
         ;*Get the next entity name, association list,
         ;*and entity type.
         (setq ename (entnext ename))
         (setq ent (entget ename))
         (setq etype (cdr (assoc 0 ent)))
      )
   )
   ;
   ;*Return return a list of the attribute values for
   ;*the endpoint.
   (list efac elat elong)
)

;************
;* Function:  extckt (extract circuit)
;* Author:    TomBrownfield 1990
;* Purpose:   Extract attribute values from a circuit.
;* Inputs:    entity name of circuit insertion.
;* Outputs:   list of attribute values from circuit
;*            and endpoints
;
(defun extckt
   ( ename
/ ent etype etag
  cktid tckt kbps sgnl txmed lineeh ep1eh ep2eh
  fac1 elat1 elong1 fac2 elat2 elong2
  elist1 elist2
)
;
;*Get the first entity association list and entity type
;*for the given entity name.
(setq ent (entget ename))
(setq etype (cdr (assoc 0 ent)))
;
;*Process each entity association list in the INSERT
;*sequence.
(while (not (equal etype "SEQEND"))
   (progn
      ;
      ;*For attribute entity lists:
      ;*   Get the attribute tags;
      ;*   Set variables to the attribute values.
      (if (equal etype "ATTRIB")
         (progn
            (setq etag (cdr (assoc 2 ent)))
            (if (equal etag "NSI_CKT_ID")
               (setq cktid (cdr (assoc 1 ent)))
            )
            (if (equal etag "TYPE_CIRCUIT")
               (setq tckt (cdr (assoc 1 ent)))
            )
            (if (equal etag "KBPS")
               (setq kbps (cdr (assoc 1 ent)))
            )
            (if (equal etag "SIGNAL")
               (setq sgnl (cdr (assoc 1 ent)))
            )
            (if (equal etag "TX_MEDIUM")
               (setq txmed (cdr (assoc 1 ent)))
            )
            (if (equal etag "LINE_EH")
               (setq lineeh (cdr (assoc 1 ent)))
            )
            (if (equal etag "EP1_EH")
               (setq ep1eh (cdr (assoc 1 ent)))
            )
            (if (equal etag "EP2_EH")
               (setq ep2eh (cdr (assoc 1 ent)))
            )
            )
         )
         ;
         ;*Get the next entity name, association list,
         ;*and entity type.
         (setq ename (entnext ename))
         (setq ent (entget ename))
         (setq etype (cdr (assoc 0 ent)))
      )
   )
   ;
   ;*Get the entity names for the entity handles of
   ;*the two endpoints.
   ;*Get a list of attribute values for each endpoint
   ;*by calling the function extept with each entity
   ;*name as its argument.
   (setq ename (handent ep1eh))
   (setq elist1 (extept ename))
   (setq ename (handent ep2eh))
   (setq elist2 (extept ename))
   ;
   ;*Set variables to the attribute values from the
   ;*two endpoint lists.
   (setq fac1 (car elist1))
   (setq elat1 (cadr elist1))
   (setq elong1 (caddr elist1))
   (setq fac2 (car elist2))
   (setq elat2 (cadr elist2))
   (setq elong2 (caddr elist2))
   ;
   ;*Return a list of the attribute values for the
   ;*circuit and its two endpoints.
   (list cktid tckt kbps sgnl txmed
      fac1 elat1 elong1
      fac2 elat2 elong2)
)

;**********
;* Command:  extckts (extract all circuits)
;* Author:   Tom Brownfield 1990
;* Purpose:  Find all circuits plotted on the map
;*           and extract all their attributes.
;*           Write a line of attributes to the
;*           circuits file for each circuit.
;* Inputs:   command line
;* Outputs:  command line
;*           circuits disk file
;
(defun C:extckts( / ss sscnt ssidx ename clist clcnt cln )
   ;
   ;*Save and turn off echo.
   (modes '("CMDECHO"))
   (setvar "CMDECHO" 0)
   ;
   ;*Get the circuits file name from the command line.
   ;*Open the file for output.
   (setq datafl (getstring "\nEnter name of circuits file: "))
   (setq f1 (open datafl "w"))
   (if (not f1)
      (*ERROR* (strcat "cannot open file: " datafl))
   )
   (if f1
      (progn
         ;
         ;*Build a selection set of all entity names of
         ;*insertions of the block LLCKT on the layer POINTS.
         (setq ss (ssget "X" (list (cons 0 "INSERT")
                                   (cons 2 "llckt")
                                   (cons 8 "POINTS"))))
         ;
         ;*Count the number of entities selected.
         (if (null ss)
            (setq sscnt 0)
            (setq sscnt (sslength ss))
         )
         (princ (itoa sscnt))
         (princ " circuits will be extracted.")
         (terpri)
         ;
         ;*Set index for the first entity name in the set.
         (setq ssidx 0)
         ;
         ;*Process each entity name in the set.
         (while (> sscnt 0)
            (progn
               (setq ename (ssname ss ssidx))
               ;
               ;*Get a list of attribute values by calling
               ;*the function extckt with the entity name
               ;*as its argument.
               (setq clist (extckt ename))
               (setq cln "")
               ;
               ;*Count the number of attribute values in
               ;*the list.
               (if (null clist)
                  (setq clcnt 0)
                  (setq clcnt (length clist))
               )
               ;
               ;*Process each attribute value in the list.
               (while (> clcnt 0)
                  (progn
                     ;
                     ;*Put each value to the output line
                     ;*with a space delimiter.
                     (setq cln (strcat cln (car clist) " "))
                     (setq clist (cdr clist))
                     (setq clcnt (1- clcnt))
                  )
               )
               ;
               ;*Write the output line to the circuits file.
               (write-line cln f1)
               ;
               ;*Set index for the next entity name in
               ;*the set.
               (setq ssidx (1+ ssidx))
               (setq sscnt (1- sscnt))
            )
         )
         ;
         ;*Close the circuits file.
         (close f1)
      )
   )
   ;
   ;*Restore echo.
   (moder)
   (princ)
)

;**********
;* Command:  seldbckts (select database circuits)
;* Author:   Tom Brownfield 1990
;* Purpose:  Run a program to select attribute values
;*           from an external database and write them
;*           to a circuits file.
;* Inputs:   command line
;* Outputs:  command line
;*           circuits disk file
;
(defun C:seldbckts
   (
   / errcnt cname rname pname cf rf cmdline rline rlist succeed )
   ;
   ;*Save and turn off echo.
   (modes '("CMDECHO"))
   (setvar "CMDECHO" 0)
   ;
   ;*Initialize error count.
   (setq errcnt 0)
   (if (= errcnt 0)
      (progn
         ;
         ;*Initialize empty results file.
         (setq rname "selres.txt")
         (setq rf (open rname "w"))
         (if (not rf)
            (progn
               (*ERROR* (strcat "cannot open file: " rname))
               (setq errcnt (1+ errcnt))
            )
            (close rf)
         )
      )
   )
   (if (= errcnt 0)
      (progn
         ;
         ;*Get the circuits file name from the command line.
         ;*Initialize empty circuits file.
         (setq cname
            (getstring "\nEnter name of circuits file: "))
         (setq cf (open cname "w"))
         (if (not cf)
            (progn
               (*ERROR* (strcat "cannot open file: " cname))
               (setq errcnt (1+ errcnt))
            )
            (close cf)
         )
     )
   )
```

```
(if (= errcnt 0)
   (progn
      ;
      ;*Get a db server password from the command line.
      (setq pname (getstring "\nEnter db server password: "))
      (if (= (strlen pname) 0)
         (progn
            (*ERROR* "null password")
            (setq errcnt (1+ errcnt))
         )
      )
   )
)
(if (= errcnt 0)
   (progn
      ;
      ;*Assemble a shell command to run the db selection
      ;*program with arguments for password, circuits file
      ;*name, and results file name.
      ;*Execute the shell command.
      ;*Read the results file to check for SUCCEED message.
      (setq cmdline (strcat "gtdbckt -P" pname " -C" cname
                            " -R" rname))
      (command "SHELL" cmdline)
      (setq rf (open rname "r"))
      (if (not rf)
         (progn
            (*ERROR* (strcat "cannot open file: " rname))
            (setq errcnt (1+ errcnt))
         )
         (progn
            (setq rline (read-line rf))
            (close rf)
            (setq rlist (sparse rline))
            (setq succeed (car rlist))
            (if (/= "SUCCEED" succeed)
               (progn
                  (*ERROR* "db selection unsuccessful")
                  (setq errcnt (1+ errcnt))
               )
               (progn
                  (princ "\nCircuits selected from database")
                  (princ "\n   to ")
                  (princ cname)
                  (princ "\n")
               )
               )
            )
         )
      )
   )
   ;
   ;*Restore echo.
   (moder)
   (princ)
)

;**********
;* Command:  insdbckts (insert database circuits)
;* Author:   Tom Brownfield 1990
;* Purpose:  Run a program to read attribute values
;*           from a circuits file and insert them
;*           into an external database.
;* Inputs:   command line
;*           circuits disk file
;* Outputs:  command line
;
(defun C:insdbckts
   (
   / errcnt cname rname pname cf rf cmdline rline rlist succeed )
   ;
   ;*Save and turn off echo.
   (modes '("CMDECHO"))
   (setvar "CMDECHO" 0)
   ;
   ;*Initialize error count.
   (setq errcnt 0)
   (if (= errcnt 0)
      (progn
         ;
         ;*Initialize empty results file.
         (setq rname "insres.txt")
         (setq rf (open rname "w"))
         (if (not rf)
            (progn
               (*ERROR* (strcat "cannot open file: " rname))
               (setq errcnt (1+ errcnt))
            )
            (close rf)
         )
      )
   )
```

```
(if (= errcnt 0)
   (progn
      ;
      ;*Get the circuits file name from the command line.
      ;*Open the file to check its presence.
      (setq cname
         (getstring "\nEnter name of circuits file: "))
      (setq cf (open cname "r"))
      (if (not cf)
         (progn
            (*ERROR* (strcat "cannot open file: " cname))
            (setq errcnt (1+ errcnt))
         )
         (close cf)
      )
  )
)
(if (= errcnt 0)
   (progn
      ;
      ;*Get a db server password from the command line.
      (setq pname (getstring "\nEnter db server password: "))
      (if (= (strlen pname) 0)
         (progn
            (*ERROR* "null password")
            (setq errcnt (1+ errcnt))
         )
      )
   )
)
(if (= errcnt 0)
   (progn
      ;
      ;*Assemble a shell command to run the db insertion
      ;*program with arguments for password, circuits file
      ;*name, and results file name.
      ;*Execute the shell command.
      ;*Read the results file to check for SUCCEED message.
      (setq cmdline (strcat "ptdbckt -P" pname " -C" cname
                            " -R" rname))
      (command "SHELL" cmdline)
      (setq rf (open rname "r"))
      (if (not rf)
         (progn
            (*ERROR* (strcat "cannot open file: " rname))
            (setq errcnt (1+ errcnt))
            )
            (progn
               (setq rline (read-line rf))
               (close rf)
               (setq rlist (sparse rline))
               (setq succeed (car rlist))
               (if (/= "SUCCEED" succeed)
                  (progn
                     (*ERROR* "db insertion unsuccessful")
                     (setq errcnt (1+ errcnt))
                  )
                  (progn
                     (princ "\nCircuits inserted into database")
                     (princ "\n   from ")
                     (princ cname)
                     (princ "\n")
                  )
               )
            )
         )
      )
   )
   ;
   ;*Restore echo.
   (moder)
   (princ)
)
```

C PROGRAMS

```
/*
** Program:   gtdbckt (get circuits from database)
** Author:    Tom Brownfield 1990
** Compiler:  Sun UNIX C with Sybase DB-Library
** Purpose:   Select rows from circuit database table,
**            lookup latitude and longitude for each
**            endpoint, and write values to circuits
**            disk file.
** Inputs:    Optional program arguments:
**               -C<circuits_file_name>
**               -R<results_file_name>
**               -P<password>
**            circuit, facility, torg, wire_center
**              in spatial database
** Outputs:   standard output messages
**            circuits disk file
**            results disk file
*/

#include <stdio.h>
#include <sybfront.h>
#include <sybdb.h>
#include <strings.h>

#define BUFLEN      200
#define PWDLEN      64
#define NCILEN      20
#define TCLEN       10
#define KBPLEN      04
#define SIGLEN      11
#define TXMLEN      20
#define EPLEN       10
#define FLTLEN      20

/*
** Forward declarations of the error handler, message handler,
** and lookup function for latitudes and longitudes.
*/
int err_handler();
int msg_handler();
int lookup_ll();

main(argc, argv)
int         argc;
```

```
char          *argv[];
{
    DBPROCESS    *dbproc1;      /* Primary connection with SQL */
                                /* server.                     */
    DBPROCESS    *dbproc2;      /* Secondary connection with   */
                                /* SQL server to be used by    */
                                /* lookup_ll().                */
    LOGINREC     *login;        /* Our login information. */
    RETCODE      return_code1;
    RETCODE      return_code2;
    DBINT        lookup_ok1;
    DBINT        lookup_ok2;

    char         outbuf[BUFLEN];
    FILE         *circuitsfile;
    FILE         *resultsfile;

    char         dbpwd[PWDLEN+1];
    char         cfname[BUFLEN];
    char         rfname[BUFLEN];
    int          i,j,k;
    unsigned
        int      count;
    int          argerr;

    /*
    ** These are the variables used to store the returning data.
    ** <length>+1 allows for a null character.
    ** <length>+1 allows for a space and a null character.
    */
    DBCHAR       nsi_ckt_id[NCILEN+2];
    DBCHAR       type_circuit[TCLEN+2];
    DBCHAR       kbps[KBPLEN+2];
    DBCHAR       signal[SIGLEN+2];
    DBCHAR       tx_medium[TXMLEN+2];
    DBCHAR       end_point1[EPLEN+2];
    DBCHAR       end_point2[EPLEN+2];
    DBCHAR       epts1[EPLEN+2];
    DBCHAR       epts2[EPLEN+2];
    DBCHAR       lats[FLTLEN+1];
    DBCHAR       longs[FLTLEN+1];

    /* Initialize password and file names to defaults. */
    strcpy(dbpwd,"server_password");
    strcpy(cfname,"circuits");
    strcpy(rfname,"results");
```

```
/*
** Scan program arguments for valid options:
** -C<circuits_file_name>
** -R<results_file_name>
** -P<password>
*/
argerr = 0;
for (i = 1; i < argc; i++) {
    printf("argument: %s\n", argv[i]);
    if (argv[i][0] == '-') {
        switch (argv[i][1]) {
        case 'C':
            j = 2;
            k = 0;
            while (argv[i][j]) {
                cfname[k] = argv[i][j];
                ++j;
                ++k;
            }
            cfname[k] = '\0';
            break;
        case 'R':
            j = 2;
            k = 0;
            while (argv[i][j]) {
                rfname[k] = argv[i][j];
                ++j;
                ++k;
            }
            rfname[k] = '\0';
            break;
        case 'P':
            j = 2;
            k = 0;
            while (argv[i][j]) {
                dbpwd[k] = argv[i][j];
                ++j;
                ++k;
            }
            dbpwd[k] = '\0';
            break;
        default:
            argerr = 1;
            printf("illegal option %c\n",argv[i][1]);
            break;
        }
    }
    else {
        argerr = 1;
        printf("option must begin with '-'\n");
    }
}

/* Check for program argument errors. */
if (argerr) {
    printf("%s had argument errors--discontinued.\n",
            argv[0]);
    exit(STDEXIT);
}

/* Open results file. */
if ((resultsfile = fopen(rfname, "w")) == NULL) {
    printf("Unable to open file %s\n",rfname);
    exit(STDEXIT);
}

/* Open circuits file. */
if ((circuitsfile = fopen(cfname, "w")) == NULL) {
    printf("Unable to open file %s\n",cfname);
    exit(STDEXIT);
}

/* Initialize DB-lIBRARY. */
if (dbinit() == FAIL)
    exit(ERREXIT);

/*
** Install the user-supplied error-handling and
** message-handling routines.  They are defined at
** the bottom of this source file.
*/
dberrhandle(err_handler);
dbmsghandle(msg_handler);

/*
** Get a LOGINREC structure and fill it with the necessary
** login information.
*/
login = dblogin();
DBSETLPWD(login, dbpwd);
DBSETLAPP(login, "getdbces");
```

```
/*
** Get two DBPROCESS structures for communicating with SQL
** Server.  A NULL servername defaults to the server
** specified by DSQUERY.
*/
dbproc1 = dbopen(login, NULL);
dbproc2 = dbopen(login, NULL);

/* Use the spatial db. */
dbuse(dbproc1, "spatial");
dbuse(dbproc2, "spatial");

printf("Selecting rows from the 'circuit' table:\n");

/* Assemble SQL select transaction in command buffer. */
dbcmd(dbproc1, "select * from circuit");

/* Send transaction to SQL server. */
dbsqlexec(dbproc1);

/* Get results from transaction. */
while ((return_code1 = dbresults(dbproc1)) !=
        NO_MORE_RESULTS) {
    if (return_code1 == SUCCEED) {

        /* Bind columns to program variables. */
        dbbind(dbproc1, 1, CHARBIND, 0, nsi_ckt_id);
        dbbind(dbproc1, 2, CHARBIND, 0, type_circuit);
        dbbind(dbproc1, 3, CHARBIND, 0, kbps);
        dbbind(dbproc1, 4, CHARBIND, 0, signal);
        dbbind(dbproc1, 5, CHARBIND, 0, tx_medium);
        dbbind(dbproc1, 6, CHARBIND, 0, end_point1);
        dbbind(dbproc1, 7, CHARBIND, 0, end_point2);

        /*
        ** Initialize space in arrays
        ** to serve as delimiter in circuits file.
        */
        nsi_ckt_id[NCILEN] = ' ';
        type_circuit[TCLEN] = ' ';
        kbps[KBPLEN] = ' ';
        signal[SIGLEN] = ' ';
        tx_medium[TXMLEN] = ' ';
        end_point1[EPLEN] = ' ';
        end_point2[EPLEN] = ' ';
```

```
/*
** Initialize null terminator in arrays
** since CHARBIND does not add one.
*/
nsi_ckt_id[NCILEN+1] = '\0';
type_circuit[TCLEN+1] = '\0';
kbps[KBPLEN+1] = '\0';
signal[SIGLEN+1] = '\0';
tx_medium[TXMLEN+1] = '\0';
end_point1[EPLEN+1] = '\0';
end_point2[EPLEN+1] = '\0';

/* Get each selected row. */
while (dbnextrow(dbproc1) != NO_MORE_ROWS) {
    printf
        ("%s%s%s%s%s\n",
        nsi_ckt_id, type_circuit, kbps, signal,
        tx_medium);

    /* Copy variables to outbuf. */
    strcpy(outbuf, nsi_ckt_id);
    strcat(outbuf,type_circuit);
    strcat(outbuf,kbps);
    strcat(outbuf,signal);
    strcat(outbuf,tx_medium);
    strcat(outbuf, end_point1);

    /*
    ** Copy end_point1 to epts1, and remove trailing
    ** spaces by replacing the first space with a
    ** null character.
    */
    strcpy(epts1, end_point1);
    epts1[strcspn(epts1," ")] = '\0';

    /*
    ** Call lookup_ll() to get latitude, longitude
    ** of epts1 using the secondary DBPROCESS.
    ** Check results of lookup.
    */
    if ((lookup_ok1 =
        lookup_ll(dbproc2, epts1, lats, longs)) == 0)
        printf("BAD LOOKUP %s\n", end_point1);
    else {
        printf("%s%s %s \n", end_point1, lats,
            longs);

    /*
    ** if lookup was ok,
    ** copy variables to outbuf.
    */
    strcat(outbuf, lats);
    strcat(outbuf, " ");
    strcat(outbuf, longs);
    strcat(outbuf, " ");
}

/* Copy variables to outbuf. */
strcat(outbuf, end_point2);

/*
** Copy end_point2 to epts2, and remove trailing
** spaces by replacing the first space with a
** null character.
*/
strcpy(epts2, end_point2);
epts2[strcspn(epts2," ")] = '\0';

/*
** Call lookup_ll() to get latitude, longitude
** of epts2 using the secondary DBPROCESS.
** Check results of lookup.
*/
if ((lookup_ok2 =
    lookup_ll(dbproc2, epts2, lats, longs)) == 0)
    printf("BAD LOOKUP %s\n", end_point2);
else {

    /*
    ** if lookup was ok,
    ** copy variables to outbuf.
    */
    printf("%s%s %s \n", end_point2, lats,
            longs);
    strcat(outbuf, lats);
    strcat(outbuf, " ");
    strcat(outbuf, longs);
    strcat(outbuf, " ");
}

/* Copy newline character to outbuf. */
```

```
                strcat(outbuf, "\n");

                /*
                ** If both lookups were ok, write record
                ** to circuits file.
                */
                if (lookup_ok1 == 1 && lookup_ok2 == 1)
                    fputs(outbuf, circuitsfile);
            }
        }
    }

    /* Close DBPROCESS structure. */
    dbexit();

    /* Close circuits file. */
    fclose(circuitsfile);

    /* Write "SUCCEED" message to results file and close. */
    fputs("SUCCEED\n", resultsfile);
    fclose(resultsfile);

    /* Normal program exit. */
    exit(STDEXIT);
}

/*
** Function:  lookup_ll (lookup latitude, longitude)
** Author:    Tom Brownfield 1990
** Purpose:   Select the latitude and longitude of the
**            wire center serving an endpoint facility.
** Inputs:    Arguments:
**               dbproc--pointer to DBPROCESS structure
**               epts--pointer to string containing endpoint
**                     facility
**               rlats--pointer to return string for latitude
**               rlongs--pointer to return string for longitude
** Outputs:  latitude string value
**           longitude string value
**           function return error status
*/

int lookup_ll(dbproc, epts, rlats, rlongs)
DBPROCESS   *dbproc;
DBCHAR      *epts;
DBCHAR      *rlats;
DBCHAR      *rlongs;
{
    RETCODE     return_code;
    DBCHAR      lats[FLTLEN+1];
    DBCHAR      longs[FLTLEN+1];
    DBINT       dbcount;

    /*
    ** Assemble SQL select transaction in command buffer
    ** using epts for condition value.
    */
    dbcmd (dbproc, "select lats = str(w.latitude,10,4),     ");
    dbcmd (dbproc, "       longs = str(w.longitude,10,4)    ");
    dbcmd (dbproc, " from facility f,torg o,               ");
    dbcmd (dbproc, "      wire_center w                     ");
    dbfcmd(dbproc, " where f.facility = '%s'          ", epts);
    dbcmd (dbproc, "   and o.org_key = f.org_key            ");
    dbcmd (dbproc, "   and w.npa =                          ");
    dbcmd (dbproc, "       substring(o.commercial_phone,2,3)");
    dbcmd (dbproc, "   and w.nxx =                          ");
    dbcmd (dbproc, "       substring(o.commercial_phone,6,3)");

    /* Send transaction to SQL server. */
    dbsqlexec(dbproc);

    /* Get results from transaction. */
    while ((return_code = dbresults(dbproc)) !=
            NO_MORE_RESULTS) {
        if (return_code == SUCCEED) {

            /* Bind columns to function variables. */
            dbbind(dbproc, 1, STRINGBIND, 0, lats);
            dbbind(dbproc, 2, STRINGBIND, 0, longs);

            /* Get each selected row. */
            while (dbnextrow(dbproc) != NO_MORE_ROWS) {

                /* Ignore any rows after the first. */
                if (DBCURROW(dbproc) > 1)
                    continue;

                /* copy function variables to outputs. */
                strcpy(rlats, lats);
                strcpy(rlongs, longs);
            }
        }
    }

    /*
    ** If one row was selected return 1 for success;
    ** else return 0 for failure.
    */
    if ((dbcount = DBCOUNT(dbproc)) == 1)
        return(1);
    else
        return(0);
}

/*
** Function:  err_handler
** Author:    (from Sybase DBLibrary Examples)
*/

int err_handler(dbproc, severity, dberr, oserr, dberrstr,
oserrstr)
DBPROCESS   *dbproc;
int         severity;
int         dberr;
int         oserr;
char        *dberrstr;
char        *oserrstr;
{
    if ((dbproc == NULL) || (DBDEAD(dbproc)))
        return(INT_EXIT);
    else {
        printf("DB-Library error:\n\t%s\n", dberrstr);

        if (oserr != DBNOERR)
            printf("Operating-system error:\n\t%s\n", oserrstr);

        return(INT_CANCEL);
    }
}

/*
** Function:  msg_handler
** Author:    (from Sybase DBLibrary Examples)
*/

int    msg_handler(dbproc, msgno, msgstate, severity, msgtext,
srvname, procname, line)
```

```
DBPROCESS   *dbproc;
DBINT       msgno;
int         msgstate;
int         severity;
char        *msgtext;
char        *srvname;
char        *procname;
DBUSMALLINT line;

{
    printf ("Msg %ld, Level %d, State %d\n",
        msgno, severity, msgstate);

    if (strlen(srvname) > 0)
        printf ("Server '%s', ", srvname);
    if (strlen(procname) > 0)
        printf ("Procedure '%s', ", procname);
    if (line > 0)
        printf ("Line %d", line);

    printf("\n\t%s\n", msgtext);

    return(0);
}

/*
** Program:   ptdbckt (put circuits into database)
** Author:    Tom Brownfield 1990
** Compiler:  Sun UNIX C with Sybase DB-Library
** Purpose:   Read circuits disk file, and use the values
**            to insert new rows into circuit database
**            table.
** Inputs:    Optional program arguments:
**               -C<circuits_file_name>
**               -R<results_file_name>
**               -P<password>
**            circuits disk file
** Outputs:   standard output messages
**            circuit in spatial database
**            results disk file
*/

#include <stdio.h>
#include <sybfront.h>
#include <sybdb.h>
#include <strings.h>
```

```
#define BUFLEN      200
#define PWDLEN      64

/*
** Forward declarations of the error handler and message
** handler.
*/
int err_handler();
int msg_handler();

main(argc, argv)
int         argc;
char        *argv[];
{
    DBPROCESS   *dbproc1;       /* Our connection with SQL */
                                /* server.                 */
    LOGINREC    *login;         /* Our login information. */
    RETCODE     return_code1;

    char        valbuf[BUFLEN];
    char        inbuf[BUFLEN];
    char        *ibp;
    int         tokcnt;
    FILE        *circuitsfile;
    FILE        *resultsfile;

    char        dbpwd[PWDLEN+1];
    char        cfname[BUFLEN];
    char        rfname[BUFLEN];
    int         i,j,k;
    unsigned
        int     count;
    int         argerr;

    /* Initialize password and file names to defaults. */
    strcpy(dbpwd,"server_password");
    strcpy(cfname,"circuits");
    strcpy(rfname,"results");

    /*
    ** Scan program arguments for valid options:
    ** -C<circuits_file_name>
    ** -R<results_file_name>
    ** -P<password>
    */
```

```
argerr = 0;
for (i = 1; i < argc; i++) {
    printf("argument: %s\n", argv[i]);
    if (argv[i][0] == '-') {
        switch (argv[i][1]) {
        case 'C':
            j = 2;
            k = 0;
            while (argv[i][j]) {
                cfname[k] = argv[i][j];
                ++j;
                ++k;
            }
            cfname[k] = '\0';
            break;
        case 'R':
            j = 2;
            k = 0;
            while (argv[i][j]) {
                rfname[k] = argv[i][j];
                ++j;
                ++k;
            }
            rfname[k] = '\0';
            break;
        case 'P':
            j = 2;
            k = 0;
            while (argv[i][j]) {
                dbpwd[k] = argv[i][j];
                ++j;
                ++k;
            }
            dbpwd[k] = '\0';
            break;
        default:
            argerr = 1;
            printf("illegal option %c\n",argv[i][1]);
            break;
        }
    }
    else {
        argerr = 1;
        printf("option must begin with '-'\n");
    }
}
```

```
/* Check for program argument errors. */
if (argerr) {
    printf("%s had argument errors--discontinued.\n",
            argv[0]);
    exit(STDEXIT);
}

/* Open results file. */
if ((resultsfile = fopen(rfname, "w")) == NULL) {
    printf("Unable to open file %s\n",rfname);
    exit(STDEXIT);
}

/* Open circuits file. */
if ((circuitsfile = fopen(cfname, "r")) == NULL) {
    printf("Unable to open file %s\n",cfname);
    exit(STDEXIT);
}

/* Initialize DB-lIBRARY. */
if (dbinit() == FAIL)
    exit(ERREXIT);

/*
** Install the user-supplied error-handling and
** message-handling routines. They are defined at
** the bottom of this source file.
*/
dberrhandle(err_handler);
dbmsghandle(msg_handler);

/*
** Get a LOGINREC structure and fill it with the necessary
** login information.
*/

login = dblogin();
DBSETLPWD(login, dbpwd);
DBSETLAPP(login, "putdbckts");

/*
** Get a DBPROCESS structure for communicating with SQL
** Server.  A NULL servername defaults to the server
** specified by DSQUERY.
*/
```

```
dbproc1 = dbopen(login, NULL);

/* Use the spatial db. */
dbuse(dbproc1, "spatial");

printf("Inserting rows into the 'circuit' table:\n");

/* Read each record from circuits. */
while ((fgets(inbuf,BUFLEN,circuitsfile)) != NULL) {

    /*
    ** Scan the first space-delimited token.
    ** Copy it to valbuf surrounded by quotes.
    */
    tokcnt = 0;
    if (ibp = (char *) strtok(inbuf," ")) {
        tokcnt++;
        strcpy(valbuf,"'");
        strcat(valbuf,ibp);
        strcat(valbuf,"'");
    }

    /*
    ** Scan the remaining tokens.
    ** Copy the second through sixth and the ninth
    ** (excluding latitudes and longitudes) to valbuf
    ** separated by commas and surrounded by quotes.
    */
    while (ibp = (char *) strtok(NULL," ")) {
        tokcnt++;
        if ((tokcnt < 7) || (tokcnt == 9)) {
            strcat(valbuf,",'");
            strcat(valbuf,ibp);
            strcat(valbuf,"'");
        }
    }
    printf("%s\n",valbuf);

    /*
    ** Assemble SQL insert transaction in command
    ** buffer using valbuf for values.
    */
    dbcmd (dbproc1, "begin tran             ");
    dbcmd (dbproc1, "insert into circuit    ");
    dbfcmd(dbproc1, "   values(%s) \n",valbuf);
    dbcmd (dbproc1, "commit tran            ");
```

```
        /* Send transaction to SQL server. */
        dbsqlexec(dbproc1);

        /* Get results from transaction. */
        while ((return_code1 = dbresults(dbproc1)) !=
                NO_MORE_RESULTS) {

            /* Check results for failed transaction. */
            if (return_code1 == FAIL) {
                printf("DB insert failed:\n");
                printf("%s\n",valbuf);
            }
        }
    }

    /* Close DBPROCESS structure. */
    dbexit();

    /* Close circuits file. */
    fclose(circuitsfile);

    /* Write "SUCCEED" message to results file and close. */
    fputs("SUCCEED\n", resultsfile);
    fclose(resultsfile);

    /* Normal program exit. */
    exit(STDEXIT);
}

/*
** Function:  err_handler
** Author:    (from Sybase DBLibrary Examples)
*/

int err_handler(dbproc, severity, dberr, oserr, dberrstr,
oserrstr)
DBPROCESS   *dbproc;
int         severity;
int         dberr;
int         oserr;
char        *dberrstr;
char        *oserrstr;
{
    if ((dbproc == NULL) || (DBDEAD(dbproc)))
        return(INT_EXIT);
    else {
        printf("DB-Library error:\n\t%s\n", dberrstr);

        if (oserr != DBNOERR)
            printf("Operating-system error:\n\t%s\n", oserrstr);

        return(INT_CANCEL);
    }
}

/*
** Function:  msg_handler
** Author:    (from Sybase DBLibrary Examples)
*/

int msg_handler(dbproc, msgno, msgstate, severity, msgtext,
srvname, procname, line)

DBPROCESS   *dbproc;
DBINT       msgno;
int         msgstate;
int         severity;
char        *msgtext;
char        *srvname;
char        *procname;
DBUSMALLINT line;

{
    printf ("Msg %ld, Level %d, State %d\n",
        msgno, severity, msgstate);

    if (strlen(srvname) > 0)
        printf ("Server '%s', ", srvname);
    if (strlen(procname) > 0)
        printf ("Procedure '%s', ", procname);
    if (line > 0)
        printf ("Line %d", line);

    printf("\n\t%s\n", msgtext);

    return(0);
}
?
```

Resubmission of Final Report for NCC2-700

James H. Blaisdell, Ph.D.
Principal Investigator

Thomas F. Brownfield
Lead Programmer

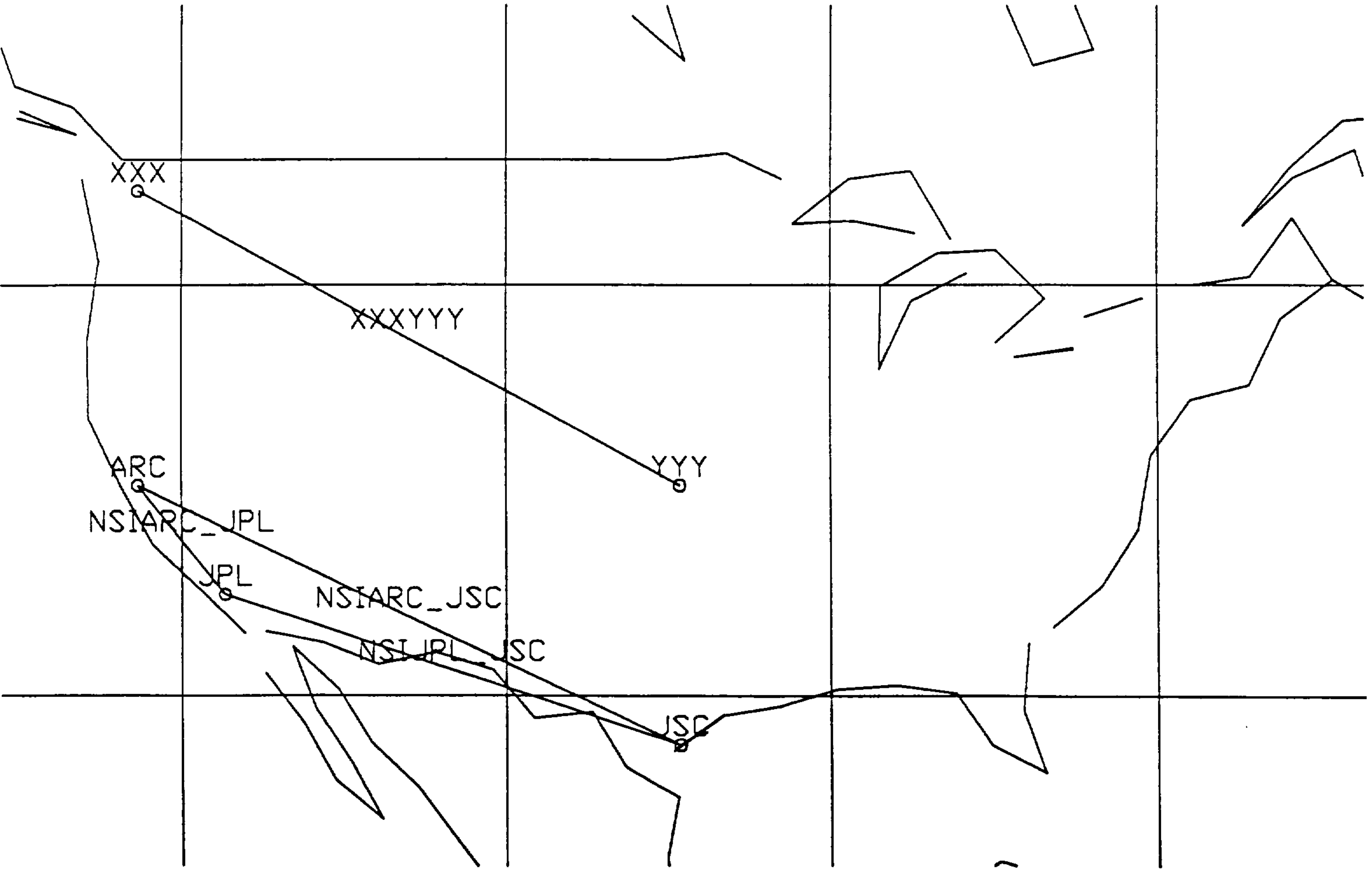
XXX
XXXYYY
YYY
ARC
NSIARC_JPL
JPL
NSIARC_JSC
NSIJPL_JSC
JSC

TABLE OF CONTENTS

INTRODUCTION

While currently available relational database management systems (RDBMS) allow inclusion of spatial information in a data model, they lack tools for presenting this information in an easily comprehensible form. Data models can be constructed to represent tangible objects with attributes describing their characteristics. The locations of these objects can be described with spatial coordinates in units such as degrees of latitude and longitude. Though these coordinates can easily be reported textually, their utility is minimal in that form. A person reading a report of objects and their locating coordinates might, at best, comprehend the locations of one or two objects at any one time. The locations and spatial relationships of several objects would not be apparent. The most useful form for presentation of spatial information is a contextual drawing. On a scale drawing the spatial relationships among selected objects and geographical features become clear. Manually plotting objects on a drawing is time-consuming and expensive, making it impractical for ad hoc reports. Computer- aided design (CAD) software packages provide adequate functions for producing drawings, but still require manual placement of symbols and features. This suggests the need for a bridge between the ad hoc reporting features of the RDBMS and the automated drawing capabilities of the CAD system.

This project is an exercise in solving a specific example of the above problem. Graphic/image diagrams displaying elements of a telecommunications network on a map could prove invaluable as a tool for planning and management of complex or wide-ranging networks. The success of such a tool is dependent on timely access to information about the network and also on the ability to quickly convert that information into a visual form. Given are the Sybase RDBMS containing a data model of the network and the AutoCAD drafting software with suitable maps. The objective of this project is an interface which will allow AutoCAD to function as a front-end to the Sybase server. This interface includes a customized AutoCAD menu, specialized Autolisp commands and functions, and C programs incorporating functions from the Sybase DB-Library. Development was carried out on a Sun 3 workstation, the target hardware, and an IBM PC clone.

The milestones for progress contained in the project proposal were followed throughout the project, and the following sections of this report describe the activities leading to their completion.

PHASE 1: AUTOCAD FROM FILE

The objective of this first phase of the project is the demonstration of a graphic display of telecommunications circuits. Each circuit will be plotted on a map as a line between the circuit's two endpoints. Each endpoint will be identified by the facility code of its location. Each circuit will be identified by its own unique code. The map will be displayed on a CRT screen. All attribute values describing the circuits will be provided in a simple local disk file. The salient process must convert the given attribute values into the graphic display in a way that is convenient and timely for the user.

The AutoCAD drafting software, version 10, was chosen for the graphic display. Version 10 is the first version available for the Sun 3 workstation, which is the target hardware of this project. AutoCAD has a well-established user base on other hardware, however, making it more likely that potential users will have experience with its fundamental features. AutoCAD provides the Autolisp interpreter, which allows creation of new functions and commands. AutoCAD's standard menu is accessible for customization to specialized tasks.

The map used for the display is an equatorial Mercator projection generated on an MSDOS PC using the WorldDXF software purchased from Micro Map & CAD, 9642 W. Virginia Cir, Lakewood, CO 80226. To minimize storage and loading time a thinning radius of 300 Km was used for testing purposes. Utility functions provided with the map generator require that the central meridian value of -75.00 degrees be stored in the AutoCAD variable userr1. For this project the map was saved with a view of the contiguous US. This primary view is shown automatically whenever the map is loaded. In addition, a customized menu was compiled while this map was loaded. The subsequent save of the map causes this menu to be automatically loaded with the map.

The data describing the telecommunications circuits to be plotted was created with a text editor and stored as a disk file in the AutoCAD working directory. The file format was named circuits, and is shown in detail later in this document. Each record contains the attribute values for one circuit. Included in these attributes are the latitude and longitude for location of each endpoint.

The file mapckt.lsp contains all the specialized Autolisp commands and functions for this project. Also included are utility map functions provided by Micro Map & CAD and written by Randy George. These utility functions are specific to the equatorial mercator maps generated by WorldDXF, and aid in converting latitude, longitude coordinates to drawing coordinates on the map. mapckt.lsp must be loaded with the Autolisp load function before use.

To read the circuits file, the Autolisp command lladdckts was created. lladdckts prompts the user for the name of the circuits format file to be plotted. It then reads each record of that file and calls the function insckt with a list of attribute values.

The function insckt receives the list of attribute values for a circuit and plots it on the map. It does this by creating new drawing entities in the current AutoCAD drawing "database". This so-called database is the data structure AutoCAD uses to keep representations of all the entities of the drawing currently loaded. To represent each endpoint insckt inserts an llept block with attribute values for the endpoint facility name, the latitude, and the longitude. The visual representation of this block shows the facility name above a small circle centered on the location of the endpoint. The latitude and longitude values are stored internally, but are not normally visible on the display. Next a simple line is drawn between the two endpoints. Finally, an llckt block is inserted. The visual representation of this block shows the circuit name centered above the midpoint of the line. The remaining attributes are stored internally. Among these normally invisible attributes are unique artificial key values identifying the three drawing entities related to this circuit, the two endpoints and the line. These key values are called entity handles by AutoCAD, and they were introduced in version 10. The relationships indicated by these keys will become important for functions that delete all the entities of a single circuit or that extract all the attribute values of a circuit and its endpoints.

The menu mapckt.mnu is a simple modification of the standard AutoCAD menu. Following the example provided by the map vendor, an extra pull-down section was added to the menu file with a text editor. This allows the user with a mouse device to select the pull-down for mapping circuits and then select from the list of specialized functions and commands. Included are a function to easily load the mapckt.lsp command file. Once loaded, any of these commands can be initiated by selection with the mouse.

Phase 1 of the project solution was concluded by a successful demonstration using the elements described above. A file containing four fictitious telecommunications circuits was copied to the AutoCAD working directory. AutoCAD was executed. The contiguous US mercator map was loaded along with its customized menu. The special Autolisp commands were loaded from the menu. Then lladdckts was run from the menu. After entering the name of the circuits file, the circuits were plotted on the map display. Figure 4 illustrates the map display.

Circuits Plotted on Map

Figure 1

PHASE 2: SYBASE TO FILE

The second phase of the project requires selection of data describing telecommunications circuits from a relational database to a disk file. The data selected must include all attribute values necessary to describe each circuit and plot it on the map. The disk file must conform to the format developed for input in Phase 1, and must be local to the graphic display system.

The Sybase Relational Database Management System (RDBMS) was chosen for the database. Sybase is available on the Sun 3 workstation, and provides the necessary development tools for this project. The isql SQL interpreter and the C DB-Library were needed. In addition, Sybase has been used for development of live databases which may benefit from the results of this project.

A local test database, named spatial, was created on the Sun workstation for this phase. Using a SQL script in isql, tables were created to represent circuits ("circuit"), facilities ("facility"), organizations ("torg"), and wire centers ("wire_center"). These tables and their attributes were modeled after similar objects designed for the NASA Science Internet Database. For efficiency, attributes not necessary for this project were eliminated. In addition, rules, triggers, and permissions normally used to maintain database integrity were not implemented at this time. The table circuit includes all attributes of the circuit except for the geographic location of the endpoints. Each endpoint matches a facility name in facility. The organization key of that facility matches that of torg. torg includes a phone number whose area code and exchange match those of a wire center. wire_center includes the latitude and longitude of its location. These wire center locations are readily available, and serve to locate facilities when map resolution is not too fine. Test data was inserted into the database using a SQL script in isql. Details of the test database are shown later in this document.

A second disk file format was created for this phase. It was named results, and consists currently of a single field. The purpose of this file format is to indicate the results of a batch process. At present, a C program would write the message "SUCCEED" into the results file at the end of a normally concluded run.

A C program, gtdbckt, was created to select rows from circuit, lookup the latitude and longitude for each endpoint, and write the values to a circuits format disk file. This program was compiled on the Sun workstation, and includes the Sybase DB- Library interface to the database. This interface includes the definitions and functions necessary to send SQL transactions to the database server and receive the resulting data stream and status information. The program selects all circuits through its primary database process structure.

For each row returned, it uses its secondary database structure to select a latitude and longitude for each endpoint. This is done by matching keys through three tables as described above. The attribute values for each circuit and the latitudes and longitudes in character form are delimited by spaces and written as a record of the circuits disk file. If the program concludes normally, it writes the message "SUCCEED" to the results file. This message does not, however, indicate the success of the database selection process. gtdbckt requires a valid database server password, a name for the circuits file, and a name for the results file. These values are provided as command line arguments.

Phase 2 of the project solution was concluded by a successful demonstration of selection from the test database and creation of a disk file. The SQL scripts bldckt.sql and popckt.sql were used to build and populate the test database. gtdbckt was run from the AutoCAD working directory to produce a new circuits file in that directory. The new circuits file was compared with that created in Phase 1, and found to contain identical data in the correct format.

PHASE 3: SYBASE TO AUTOCAD

The objective of the third phase of the project is a demonstration of the combination of Phase 1 and Phase 2. This requires that the process of selection from the database be easily controlled from the graphic display system. The data resulting from that selection must then be available for plotting on the map.

The Autolisp command seldbckts was created and added to mapckt.lsp for Phase 3. This command prompts the user for a database server password and a circuits file name. It initializes an empty results file and an empty circuits file with the name entered. It then runs the C program gtdbckt with arguments for password, circuits file name, and results file name. This is done with the AutoCAD shell command as an independent process. AutoCAD waits until the process is finished. seldbckts then reads the results file to check for a "SUCCEED" message written by gtdbckt. If the message is found, the command indicates successful selection and the name of the circuits file.

Phase 3 of the project solution was concluded by a successful demonstration of database selection controlled from AutoCAD and plotting of the resulting data. The map, menu, and Autolisp commands were loaded as in Phase 1. seldbckts was run from the menu. After entering a valid password and a circuits file name, seldbckts indicated successful selection from the database. As in Phase 1, lladdckts was run. The circuits file name entered was that created by database selection. The circuits plotted on the map were found to be identical to Phase 1. The entire process was accomplished with a minimum of user input in a reasonably short time.

PHASE 4: AUTOCAD TO SYBASE

Phase 4 of the project is essentially a reversal of the first three phases. It allows the user to plot new circuits on the map by supplying attribute values to the graphic display system. All attribute values may then be extracted from the map to create a disk file. The values from the disk file may inserted into the database by a process controlled from the graphic display system.

The Autolisp command lladdckt prompts the user for all attribute values describing a circuit, including the latitudes and longitudes of the endpoints. It calls the function insckt (described in Phase 1) with a list of attribute values for the circuit to be plotted on the map.

The Autolisp command extckts prompts the user for the name of the circuits file to be created. It then searches the drawing "database" to build a set of the internal entity names of all inserts of the block llckt. It indicates to the user the number of inserts found. With each entity name in the list it calls the function extckt, which returns a list of all attribute values for that circuit with its endpoints. It then writes a record consisting of the attribute values with space delimiters to the circuits file.

The function extckt receives the entity name of an insertion of the llckt block. It extracts the attribute values from the insertion. These values include the entity handles of the two related endpoints. For each endpoint it converts the entity handle into an entity name. It calls the function extept with each entity name, and receives a list of the attribute values for the endpoint. extckt returns a list containing the attribute values from the circuit and the endpoints.

The function extept receives the entity name of an insertion of the llept block. It extracts the attribute values from the insertion and returns them in a list.

A C program, ptdbckt, was created to insert rows into the circuit database table from the circuits disk file. This program was compiled on the Sun workstation, and includes the Sybase DB- Library interface to the database. The program reads each record from the circuits disk file and uses the attribute values to form a SQL insert transaction to add a new row to the circuit table. It does not attempt to insert new facilities, organizations, or wire centers, but assumes that the endpoints of the new circuits are existing facilities. In a live database it would be necessary to implement triggers to prevent insertion of new rows that violate this assumption.

The Autolisp command insdbckts prompts the user for a database server password and a circuits file name. It initializes an empty results file, and opens the circuits file to determine its presence. It then runs the C program ptdbckt with arguments for password, circuits file name, and results file name. This is done with the AutoCAD shell command as an independent process. AutoCAD waits until the process is finished. insdbckts then reads the results file to check for a "SUCCEED" message written by ptdbckt. If the message is found, the command indicates successful insertion to the database.

Phase 4 of the project solution was concluded by a successful demonstration using the new elements described. The map, menu, and Autolisp commands were loaded. lladdckt was used to plot new circuits on the map by prompting the user to enter attribute values. extckts was used to extract attribute values to a circuits file named by the user. insdbckts was then run. After entering a valid password and the circuits file name from extckts, it indicated successful insertion into the database. To verify the success of lladdckt and extckts the circuits file was examined to compare its values with those entered by the user. Selection of the new rows from the database was used to provide a comparison for verification of ptdbckt. Both comparisons produced identical values. Excepting the additional user input needed to plot new circuits on the map, this process was found to function as easily as that of Phase 3.

CONCLUSION

From the beginning of this project it was known that certain limitations would be encountered. The detail of the map display was limited by the processing speed of the hardware platform. For this project this detail was limited by using a large thinning radius when generating the map. In the future it would be desireable to selectively limit the map detail in unused areas. Version 10 of the AutoCAD software was found to have only limited capabilities for controlling external processes. A more interactive capability would improve the interface used in this project. It is hoped that future versions will provide this improvement.

This project has demonstrated a bridge between the data model of an RDBMS and the graphic display of a CAD system. It has been shown that the CAD system can be used to control the selection of data with spatial components from the database and then quickly plot that data on a map display. It has also been shown that the CAD system can be used to extract data from a drawing and then control the insertion of that data into the database. These demonstrations have been successful in a test environment that incorporates many features of known working environments. This suggests that the techniques developed in this project could be adapted for practical use.

SPATIAL DATABASE NORMALIZED SCHEMA

circuit(nsi_ckt_id,type_circuit,kbps,signal,tx_medium,end_point1,
end_point2)

facility(org_key,facility)

torg(org_key,organization,commercial_phone)

wire_center(clli,npa,nxx,lata,locality,state,latitude,longitude)

These database tables are intended only for the purpose of testing and demonstrating the functions of the AutoCAD graphic front-end project. Wherever possible their attributes coincide with those of the NASA Science Internet Database (NSI db). Some tables and attributes required for this project have not been implemented in the NSI db. It is hoped that the design of this project will in the future allow transfer of data with a minimum of adaptation.
Please note that all attributes described below are specified not to accept null values. Though some of these attributes are merely descriptive, they require non-null values to protect the integrity of derivative data structures in other parts of the system. The space-delimited disk files used by AutoCAD for input and output of block attributes would be corrupted by null values. In addition, AutoCAD will not allow null values to be entered as block attribute values. Specifying non-null default values for database attributes might be used as an alternative method of solving this problem.

```
Name of Database Server:
     lilmud
Name of the Database:
     spatial
Name of the Table:
     circuit
     1)  Name of Attribute:
          nsi_ckt_id
     2)  Field Characteristics:
          char(20)
     3)  Index Name and Characteristics:
          unique clustered index ckt_index on circuit(nsi_ckt_id)
     4)  Nulls:
          not null
     5)  Keys:
          pk
     6)  Domain Rules:
          Unique identifier for circuit.
          Prevent update by access control.
          Service and material id.
          See NSI db design.
Name of Database Server:
     lilmud
Name of the Database:
     spatial
Name of the Table:
     circuit
     1)  Name of Attribute:
          type_circuit
     2)  Field Characteristics:
          char(10)
     3)  Index Name and Characteristics:
          no index
     4)  Nulls:
          not null
     5)  Keys:
          not a key field
     6)  Domain Rules:
          (e.g. common_carrier, ethernet)
          See NSI db design.
Name of Database Server:
     lilmud
Name of the Database:
     spatial
Name of the Table:
     circuit
     1)  Name of Attribute:
          kbps
     2)  Field Characteristics:
          char(04)
     3)  Index Name and Characteristics:
          no index
     4)  Nulls:
          not null
     5)  Keys:
          not a key field
     6)  Domain Rules:
          Data rate of circuit in thousand bits per second.
          See NSI db design.
Name of Database Server:
     lilmud
Name of the Database:
     spatial
Name of the Table:
     circuit
     1)  Name of Attribute:
          signal
     2)  Field Characteristics:
          char(11)
     3)  Index Name and Characteristics:
          no index
     4)  Nulls:
          not null
     5)  Keys:
          not a key field
     6)  Domain Rules:
          Signal type of circuit (e.g. ANALOG, DIGITAL).
          See NSI db design.
Name of Database Server:
     lilmud
Name of the Database:
     spatial
Name of the Table:
     circuit
     1)  Name of Attribute:
          tx_medium
     2)  Field Characteristics:
          char(20)
     3)  Index Name and Characteristics:
          no index
     4)  Nulls:
          not null
     5)  Keys:
          not a key field
     6)  Domain Rules:
          Transmission medium of circuit (e.g. TERRESTRIAL).
          See NSI db design.
Name of Database Server:
     lilmud
Name of the Database:
     spatial
Name of the Table:
     circuit
     1)  Name of Attribute:
          end_point1
     2)  Field Characteristics:
          char(10)
     3)  Index Name and Characteristics:
          nonclustered index ce1_index on circuit(end_point1)
 4)  Nulls:
          not null
     5)  Keys:
          fk->facility.facility INS RST
                                UPD RST
     6)  Domain Rules:
          Facility of first endpoint of circuit.
Name of Database Server:
     lilmud
Name of the Database:
     spatial
```

```
Name of the Table:
     circuit
     1)  Name of Attribute:
          end_point2
     2)  Field Characteristics:
          char(10)
     3)  Index Name and Characteristics:
          nonclustered index ce2_index on circuit(end_point2)
 4)  Nulls:
          not null
     5)  Keys:
          fk->facility.facility INS RST
                                UPD RST
     6)  Domain Rules:
          Facility of second endpoint of circuit.
Name of Database Server:
     lilmud
Name of the Database:
     spatial
Name of the Table:
     facility
     1)  Name of Attribute:
          org_key
     2)  Field Characteristics:
          int
     3)  Index Name and Characteristics:
          unique clustered index fac_index on
          facility(org_key,facility)
     4)  Nulls:
          not null
     5)  Keys:
          pk (part of composite:  facility.org_key, facility)
          fk->torg.org_key INS RST
     6)  Domain Rules:
          Prevent update by access control.
Name of Database Server:
     lilmud
Name of the Database:
     spatial
Name of the Table:
     facility
     1)  Name of Attribute:
          facility
     2)  Field Characteristics:
          char(20)
     3)  Index Name and Characteristics:
         unique clustered index fac_index on
         facility(org_key,facility)
     4)  Nulls:
         not null
     5)  Keys:
         pk (part of composite:  facility.org_key, facility)
           ->circuit.end_point1 DLT RST
           ->circuit.end_point2 DLT RST
     6)  Domain Rules:
         Facility code of facility.
         Prevent update by access control.
Name of Database Server:
     lilmud
Name of the Database:
     spatial
Name of the Table:
     torg
     1)  Name of Attribute:
         org_key
     2)  Field Characteristics:
         int
     3)  Index Name and Characteristics:
         unique clustered index org_index on
         torg(org_key,organization)
     4)  Nulls:
         not null
     5)  Keys:
         pk (part of composite:  torg.org_key, organization)
           ->facility.org_key DLT RST
     6)  Domain Rules:
         Unique artificial key for torg.
         Prevent update by access control.
         See NSI db design.
Name of Database Server:
     lilmud
Name of the Database:
     spatial
Name of the Table:
     torg
     1)  Name of Attribute:
         organization
     2)  Field Characteristics:
         char(30)
     3)  Index Name and Characteristics:
         unique clustered index org_index on
         torg(org_key,organization)
     4)  Nulls:
          not null
     5)  Keys:
          pk (part of composite:  torg.org_key, organization)
 6)  Domain Rules:
          Identifier of torg.
          Prevent update by access control.
          See NSI db design.
Name of Database Server:
     lilmud
Name of the Database:
     spatial
Name of the Table:
     torg
     1)  Name of Attribute:
          commercial_phone
     2)  Field Characteristics:
          char(20)
     3)  Index Name and Characteristics:
          no index
     4)  Nulls:
          not null
     5)  Keys:
          not a key field
     6)  Domain Rules:
          Phone_rule:  (nnn)nnn-nnnn
          See NSI db design.
Name of Database Server:
     lilmud
Name of the Database:
     spatial
Name of the Table:
     wire_center
     1)  Name of Attribute:
          clli
     2)  Field Characteristics:
          char(11)
     3)  Index Name and Characteristics:
          unique clustered index wct_index on wire_center(clli)
     4)  Nulls:
          not null
     5)  Keys:
          pk
     6)  Domain Rules:
          Unique identifier for wire_center.
          Prevent update by access control.
          See NSI db design.
Name of Database Server:
     lilmud
Name of the Database:
     spatial
Name of the Table:
     wire_center
     1)  Name of Attribute:
          npa
     2)  Field Characteristics:
          char(03)
     3)  Index Name and Characteristics:
          nonclustered index wcx_index on wire_center(npa,nxx)
     4)  Nulls:
          not null
     5)  Keys:
          (part of composite: npa, nxx)
          lookup->substring(commercial_phone,2,3)
     6)  Domain Rules:
          Areacode of wire_center.
          See NSI db design.
Name of Database Server:
     lilmud
Name of the Database:
     spatial
Name of the Table:
     wire_center
     1)  Name of Attribute:
          nxx
     2)  Field Characteristics:
          char(03)
     3)  Index Name and Characteristics:
          nonclustered index wcx_index on wire_center(npa,nxx)
     4)  Nulls:
          not null
     5)  Keys:
          (part of composite: npa, nxx)
          lookup->substring(commercial_phone,6,3)
     6)  Domain Rules:
          Exchange of wire_center.
          See NSI db design.
Name of Database Server:
     lilmud
Name of the Database:
     spatial
Name of the Table:
     wire_center
     1)  Name of Attribute:
          lata
     2)  Field Characteristics:
          char(03)
     3)  Index Name and Characteristics:
          no index
     4)  Nulls:
          not null
     5)  Keys:
          not a key field
     6)  Domain Rules:
          Lata of wire_center.
          See NSI db design.
Name of Database Server:
     lilmud
Name of the Database:
     spatial
Name of the Table:
     wire_center
     1)  Name of Attribute:
          locality
     2)  Field Characteristics:
          char(26)
     3)  Index Name and Characteristics:
          no index
     4)  Nulls:
          not null
     5)  Keys:
          not a key field
     6)  Domain Rules:
          Locality of wire_center.
          See NSI db design.
Name of Database Server:
     lilmud
Name of the Database:
     spatial
Name of the Table:
     wire_center
     1)  Name of Attribute:
          state
     2)  Field Characteristics:
          char(02)
     3)  Index Name and Characteristics:
          no index
     4)  Nulls:
          not null
     5)  Keys:
          not a key field
     6)  Domain Rules:
          State of wire_center.
          See NSI db design.
Name of Database Server:
     lilmud
Name of the Database:
     spatial
Name of the Table:
     wire_center
     1)  Name of Attribute:
          latitude
     2)  Field Characteristics:
          float
     3)  Index Name and Characteristics:
          no index
     4)  Nulls:
          not null
     5)  Keys:
          not a key field
     6)  Domain Rules:
          Latitude of wire_center in decimal degrees.
Name of Database Server:
     lilmud
Name of the Database:
     spatial
Name of the Table:
     wire_center
     1)  Name of Attribute:
          longitude
     2)  Field Characteristics:
          float
     3)  Index Name and Characteristics:
          no index
     4)  Nulls:
          not null
     5)  Keys:
          not a key field
     6)  Domain Rules:
          Longitude of wire_center in decimal degrees.
```

SQL SCRIPTS

```
/*
** Script:       bldckt.sql (build circuit)
** Author:       Tom Brownfield 1990
** Interpreter:  Sybase isql
** Purpose:      Create database objects necessary for
**               demonstrating the AutoCAD graphic
**               front-end with circuits data.
*/

/* Use the spatial database. */
use spatial
go

/* Create tables. */
create table circuit
    (nsi_ckt_id         char(20)            not null
    ,type_circuit       char(10)            not null
    ,kbps               char(04)            not null
    ,signal             char(11)            not null
    ,tx_medium          char(20)            not null
    ,end_point1         char(10)            not null
    ,end_point2         char(10)            not null
    )
go

create table facility
    (org_key            int                 not null
    ,facility           char(10)            not null
    )
go

create table torg
    (org_key            int                 not null
    ,organization       char(30)            not null
    ,commercial_phone   char(20)            not null
    )
go

create table wire_center
    (clli               char(11)            not null
    ,npa                char(03)            not null
    ,nxx                char(03)            not null
    ,lata               char(03)            not null
    ,locality           char(26)            not null
    ,state              char(02)            not null
    ,latitude           float               not null
    ,longitude          float               not null
    )
go

/* Create indexes. */
create unique clustered index ckt_index
    on circuit(nsi_ckt_id)
go

create unique clustered index fac_index
    on facility(org_key, facility)
go

create unique clustered index org_index
    on torg(org_key, organization)
go

create unique clustered index wct_index
    on wire_center(clli)
go

create nonclustered index wcx_index
    on wire_center(npa, nxx)
go

/* end */
/*
** Script:       popckt.sql (populate circuit)
** Author:       Tom Brownfield 1990
** Interpreter:  Sybase isql
** Purpose:      Populate database tables necessary
**               for demonstrating the AutoCAD graphic
**               front-end with circuits data.
*/

/* Use the spatial database. */
use spatial
go

/* Insert wire_center data. */
begin tran
insert into wire_center values
     ('X'
    ,'206'
    ,'111'
    ,'001'
    ,'Xtown'
    ,'WA'
    ,48.0
    ,-122.0
    )
commit tran
go

begin tran
insert into wire_center values
    ('Y'
    ,'402'
    ,'222'
    ,'002'
    ,'Ytown'
    ,'NB'
    ,38.0
    ,-97.0
    )
commit tran
go

begin tran
insert into wire_center values
    ('A'
    ,'415'
    ,'335'
    ,'003'
    ,'Mountain View'
    ,'CA'
    ,38.0
    ,-122.0
    )
commit tran
go

begin tran
insert into wire_center values
    ('B'
    ,'713'
    ,'444'
    ,'004'
    ,'Houston'
    ,'TX'
    ,28.0
    ,-97.0
    )
commit tran
go

begin tran
insert into wire_center values
    ('C'
    ,'818'
    ,'555'
    ,'005'
    ,'Pasadena'
    ,'CA'
    ,34.0
    ,-118.0
    )
commit tran
go

/* Insert torg data. */
begin tran
insert into torg values
    (1
    ,'Xorg'
    ,'(206)111-1111'
    )
commit tran
go

begin tran
insert into torg values
    (2
    ,'Yorg'
    ,'(402)222-2222'
    )
commit tran
go

begin tran
insert into torg values
    (3
    ,'NASA Ames'
    ,'(415)335-3333'
    )
```

```
commit tran
go

begin tran
insert into torg values
    (4
    ,'NASA Johnson'
    ,'(713)444-4444'
    )
commit tran
go

begin tran
insert into torg values
    (5
    ,'NASA JPL'
    ,'(818)555-5555'
    )
commit tran
go

/* Insert facility data. */

begin tran
insert into facility values
    (1
    ,'XXX'
    )
commit tran
go

begin tran
insert into facility values
    (2
    ,'YYY'
    )
commit tran
go

begin tran
insert into facility values
    (3
    ,'ARC'
    )
commit tran
go
```

```
begin tran
insert into facility values
    (4
    ,'JSC'
    )
commit tran
go

begin tran
insert into facility values
    (5
    ,'JPL'
    )
commit tran
go

/* Insert circuit data. */
begin tran
insert into circuit values
    ('XXXYYY'
    ,'NSN'
    ,'224'
    ,'DIGITAL'
    ,'TERRESTRIAL'
    ,'XXX'
    ,'YYY'
    )
commit tran
go

begin tran
insert into circuit values
    ('NSIARC_JSC'
    ,'NSN'
    ,'448'
    ,'DIGITAL'
    ,'TERRESTRIAL'
    ,'ARC'
    ,'JSC'
    )
commit tran
go

begin tran
insert into circuit values
```

```
    ('NSIARC_JPL'
    ,'NSN'
    ,'168'
    ,'DIGITAL'
    ,'TERRESTRIAL'
    ,'ARC'
    ,'JPL'
    )
commit tran
go

begin tran
insert into circuit values
    ('NSIJPL_JSC'
    ,'NSN'
    ,'168'
    ,'DIGITAL'
    ,'TERRESTRIAL'
    ,'JPL'
    ,'JSC'
    )
commit tran
go

/* List all entries in the database. */
print '*** wire_center ***'
select * from wire_center
go

print '*** torg ***'
select * from torg
go

print '*** facility ***'
select * from facility
go

print '*** circuit ***'
select * from circuit
go

/* end */
```

DISK FILE SCHEMA

```
circuits(nsi_ckt_id,type_circuit,kbps,signal,tx_medium,end_point1
         ,latitude1,longitude1,end_point2,latitude2,longitude2)

results(success)
```

These disk files provide the needed data storage between the database system and the AutoCAD system. They are sequential text files contained in the current working directory. All attributes are variable-length character fields delimited by spaces. Spaces are optional at the end of a record.

The circuits file contains latitudes and longitudes which are character representations of values contained in the database as wire_center.latitude and wire_center.longitude. All other attribute values are identical to those in the database.

The results file indicates the results of a DB-LIBRARY C program process. Success will contain the value "SUCCEED" when the program writing it is successfully completed. This does not, however, indicate the success or failure of individual database transactions. At this time such indications are returned as database messages and errors through standard console output. Additional attributes should be added to this file if this interface is to be used in an environment where the user requires improved control information.

AUTOCAD MENU

The following is a partial listing of mapckt.mnu from the beginning of the file through the modified section:

```
***BUTTONS
;
$p1=*
^c^c
^B
^O
^G
^D
^E
^T
***AUX1
;
$p1=*
^C^C
^B
^O
^G
^D
^E
^T
***POP1
[Tools]
[OSNAP]^C^C$p1= $p1=* OSNAP \
[CENter]CENTER
[ENDpoint]ENDPOINT
[INSert]INSERT
[INTersec]INTERSEC
[MIDpoint]MIDPOINT
[NEArest]NEAREST
[NODe]NODE
[PERpend]PERPEND
[QUAdrant]QUADRANT
[QUICK,]QUICK,^Z
[TANgent]TANGEN
NONE
[~--]
[Cancel]^C^C
[U]^C^CU
[Redo]^C^CREDO
[Redraw]'REDRAW
```

```
***POP2
[Draw]
[Line]*^C^C$S=X $s=line line
[Arc]*^C^C$S=X $s=poparc arc
[Circle]*^C^C$S=X $s=popcircl circle
[Polyline]*^C^C$S=X $s=pline pline
[Insert]^C^Csetvar attdia 1 $s=insert insert
[Dtext]*^C^C$S=X $s=Dtext Dtext
[Hatch]^C^C$i=hatch1 $i=*

***POP3
[Edit]
[Erase]*^C^Cerase si auto
[Move]*^C^Cmove si auto
[Copy]*^C^Ccopy si auto
[Trim]*^C^C$S=X $s=trim trim auto
[Extend]*^C^C$S=X $s=extend extend auto
[Stretch]*^C^C$S=X $s=stretch stretch crossing
[Polyedit]*^C^C$S=X $s=pedit pedit

***POP4
[Display]
[Window]'zoom w
[Previous]'zoom p
[Dynamic]'zoom d
[Pan]'pan
[3D View]^C^C$i=3dviews $i=*

***POP5
[Modes]
[Drawing Aids]'ddrmodes
[Entity Creation]'ddemodes
[Modify Layer]'ddlmodes

***POP6
[Options]
[Ashade]^P(cond ((null C:SCENE) +
(vmon) (prompt "Please wait...  Loading ashade.  ") +
(load "ashade")) (T (princ))) ^P$i=as $i=*
[3D Objects]^P(cond ((null C:CONE) +
(vmon) (prompt "Please wait...  Loading 3D Objects.  ") +
(load "3d")) (T (princ))) ^P$i=3DObjects $i=*
[Fonts]^C^C$i=fonts1 $i=*
[DXFScript]^C^CDXFSCRIPT SCRIPT;D:/S/DXF GRAPHSCR;
[Stack]^C^CSTACK
[MOD]^C^CMOD
```

```
***POP7
[File]
[Save]^C^CSave
[End]^C^Cend
[Quit]^C^C$S=X $s=quit quit
[~--]
[Plot]^C^Cplot
[Print]^C^Cprplot


***POP8
[Mercator Map]
[LOAD Merc]^C^C(load "mercator")
[LOAD Merc3D]^C^C(load "merc3D")
[Lat Long]^C^CLatLong
[Map Dist]^C^CMapDist
[Add Pt]^C^CLLaddPt
[Add Pts]^C^CLLaddPts
[Del Pt]^C^CLLdelPt
[Del Pts]^C^CLLdelPts
[Map Radius]^C^CMapRadius
[Add Lines]^C^CLLaddLines
[WDBII]^C^CWDB \\WDBtoDXF script load

***POP9
[Map Circuits]
[LOAD Circuits]^C^C(load "mapckt")
[Add Ckts]^C^CLLaddCkts
[Add Ckt]^C^CLLaddCkt
[Del Ckts]^C^CDelCkts
[Ext Ckts]^C^CExtCkts
[Sel Ckts]^C^CSelDBCkts
[Ins Ckts]^C^CInsDBCkts
```

AUTOLISP COMMANDS AND FUNCTIONS

```
;**********
;* File:         mapckt.lsp (map circuits)
;* Interpreter:  AutoCAD Autolisp version 10
;* Purpose:      Provide AutoCAD commands and functions
;*               to plot circuits on a world-mercator map.
;**********

;**********
;*Global variables and utility map functions from
;*mercator.lsp by Randy George 8/15/88.  Modifications
;*are noted by comments.
;
;* Global variables:
; earth : earth radius in meters 6371
; lat   : latitude
; long  : longitude
; inc   : lat or long increment
; vb    : Blipmode variable
; vc    : Cmdecho variable
```

```
;**********
;* Utility map functions:

(vmon)
(prompt "\nLoading. Please wait...")
(terpri)

(defun MODES (a)
   (setq MLST '())
   (repeat (length a)
      (setq MLST (append MLST (list (list (car a) (getvar
         (car a))))))
      (setq a (cdr a)))
)

(defun MODER ()
   (repeat (length MLST)
      (setvar (caar MLST) (cadar MLST))
      (setq MLST (cdr MLST))
   )
)

(defun *ERROR* (st)
  (moder)
  (terpri)
  (princ "\nError: ")
  (princ st)
  (princ)
)

(defun sqr(x)
  (* x x)
)

(defun arccos (x)
  (if (/= (abs x) 1.0)
      (- (* pi 0.5) (atan (/ x (sqrt (- 1.0 (* x x))))))
      (* (- 1.0 x) pi 0.5)
  )
)

(defun arcsin (x)
  (if (/= (abs x) 1.0)
      (atan (/ x (sqrt (- 1.0 (* x x)))))
      (* x pi 0.5)
  )
)

(defun arctanh (x)
  (if (/= x 1.0)
      (/ (log (/ (+ 1.0 x) (- 1.0 x))) 2.0)
      (list 99999999.0)
  )
)

(defun tanh (x / e1 e2)
  (setq e1 (exp x))
  (setq e2 (exp (* -1.0 x)))
  (/ (- e1 e2) (+ e1 e2))
)

(defun Radian( deg )
  (* deg 0.0174533)
)

(defun Degree( rad)
  (* 57.29578 rad)
)

(defun Meridian (LongR LongR0 / delta)
  (setq delta (- LongR LongR0))
  (if (< delta -3.1415927)
      (setq delta (+ delta 6.2831853))
  )
  (if (> delta 3.1415927)
      (setq delta (- delta 6.2831853))
  )
  (setq delta delta)
)


(defun Mercator (LongR LongR0 LatR radius /)
  (if (< (abs (Radian LatR)) 1.397)
      (progn
        (setq LongR (Meridian (Radian LongR) (Radian LongR0)))
        (setq x (* radius LongR))
        (setq y (* radius (arctanh (sin (Radian LatR)))))
        (list x y)
      )
      (list 0.0 0.0)
  )
)
```

```
(defun InvMeridian (delta LongR0 / )
  (setq delta (+ delta LongR0))
  (if (< delta -3.1415927)
      (setq delta (+ delta 6.2831853))
  )
  (if (> delta 3.1415927)
      (setq delta (- delta 6.2831853))
  )
  (setq delta delta)
)

(defun InvMercator ( pt LongR0 radius / delta )
  (setq delta (/ (car pt) radius))
  (setq LongR (Degree (InvMeridian delta (Radian LongR0))))
  (setq LatR (Degree (arcsin (tanh (/ (cadr pt) radius)))))
  (list LatR LongR)
)

(defun Arc ( pt radius / )
  (setq LongR (Degree (/ (car pt) radius)))
  (setq LatR (Degree (arcsin (tanh (/ (cadr pt) radius)))))
  (list LatR LongR)
)


;  Parse input string into list of strings
(defun sparse (S / LL TMP CNT)
  (setq TMP "" CNT 0)
  (while (< CNT (strlen S))
     (setq CNT (1+ CNT))
     (cond
       ((and (or (= (substr S CNT 1) ",")
                 (= (substr S CNT 1) " ")) (/= TMP ""))
         (setq LL (cons TMP LL) TMP ""))
       ((= (substr S CNT 1) ","))
       ((= (substr S CNT 1) " "))
       (t (setq TMP (strcat TMP (substr S CNT 1))))
     )
  )
  ;*modified to prevent adding null element to end of list
  ;*Tom Brownfield 12/5/90
  (if (/= TMP "")
     (reverse (cons TMP LL))
     (reverse LL)
  )
```

;*

)

```
;**********
;* Main Program
;**********

;**********
;* Function:  insckt (insert a circuit)
;* Author:    Tom Brownfield 1990
;* Purpose:   Insert blocks for a circuit and two endpoints
;*            and draw a line between the endpoints.
;* Inputs:    central meridian, earth radius, attribute
;*            values for circuit and endpoints.
;* Outputs:   command line.
;
(defun insckt
   ( central earth
     elat1 elong1 fac1 elat2 elong2 fac2
     cktid tckt kbps sgnl txmed
   / errcnt lat long pt1 ep1eh pt2 ep2eh lineeh pt3)
   ;
   ;*Initialize error count.
   (setq errcnt 0)
   ;
   ;*Convert latitude, longitude to float.
   (setq lat (atof elat1))
   (if (null lat) (setq lat 0.0))
   (setq long (atof elong1))
   (if (null long) (setq long 0.0))
   ;*
   ;*Get drawing coordinates for endpoint1 by calling the
   ;*the function mercator with latitude, longitude,
   ;*central meridian as arguments.
   (setq pt1 (mercator long central lat earth))
   (if (and (= (car pt1) 0.0) (= (cadr pt1) 0.0))
      (progn
         (setq errcnt (1+ errcnt))
         (princ "**** Invalid latitude/longitude ***")
         (terpri)
         (princ (rtos lat 2 4))
         (princ ",")
         (princ (rtos long 2 4))
      )
   )
   ;
   ;*Get drawing coordinates for endpoint2 as above.
   (setq lat (atof elat2))
   (if (null lat) (setq lat 0.0))
   (setq long (atof elong2))
   (if (null long) (setq long 0.0))
   (setq pt2 (mercator long central lat earth))
   (if (and (= (car pt2) 0.0) (= (cadr pt2) 0.0))
      (progn
         (setq errcnt (1+ errcnt))
         (princ "**** Invalid latitude/longitude ***")
         (terpri)
         (princ (rtos lat 2 4))
         (princ ",")
         (princ (rtos long 2 4))
      )
   )
   ;
   ;*Execute block inserts of endpoints, at the drawing
   ;*coordinate points, with attribute values.
   ;*Get the entity handles of the endpoint inserts.
   ;*Draw a line between the endpoints.
   ;*Get the entity handle of the line.
   ;*Get the coordinates of the midpoint of the line.
   ;*Execute a block insert of the circuit, at the
   ;*midpoint, with attribute values including the three
   ;*entity handles of the related entities.
   (if (= errcnt 0)
      (progn
         (command "insert" "llept" pt1 "" "" "" fac1 elat1
            elong1)
         (setq ep1eh (cdr (assoc 5 (entget (entlast)))))
         (command "insert" "llept" pt2 "" "" "" fac2 elat2
            elong2)
         (setq ep2eh (cdr (assoc 5 (entget (entlast)))))
         (command "LINE" pt1 pt2 "")
         (setq lineeh (cdr (assoc 5 (entget (entlast)))))
         (setq pt3 (polar pt1 (angle pt1 pt2)
            (/(distance pt1 pt2)2)))
         (command "insert" "llckt" pt3 "" "" "" cktid tckt kbps
            sgnl
            txmed lineeh ep1eh ep2eh)
      )
   )
)

;**********
;* Command:  lladdckts (add circuits at latitude, longitude)
;* Author:   Tom Brownfield 1990
;* Purpose:  Read attributes from a circuits file, and plot
;*           the circuits with endpoints on the map using
;*           latitudes and longitudes from the file.
;* Inputs:   command line
;*           circuits disk file
;* Outputs:  command line
;
(defun C:lladdckts
   (
   / central earth datafl f1
     sln1 slst1 slen1
     cktid tckt kbps sgnl txmed
     fac1 elat1 elong1
     fac2 elat2 elong2
   )
   ;
   ;*Save and turn off echo and blip.
   (modes '("CMDECHO" "BLIPMODE"))
   (setvar "CMDECHO" 0)
   (setvar "BLIPMODE" 0)
   ;
   ;*Set layer to POINTS.
   (command "LAYER" "M" "POINTS" "")
   ;
   ;*Get central meridian of map.
   (setq central (getvar "USERR1"))
   ;
   ;*Set earth radius.
   (setq earth 6371.0 )
   ;
   ;*Be sure entity handles are on.
   (command "HANDLES" "ON")
   ;
   ;*Get the name of the circuits file.
   ;*Open the file for input.
   (setq datafl (getstring "\nEnter name of circuits file: "))
   (setq f1 (open datafl "r"))
   (if (not f1)
      (*ERROR* (strcat "cannot open file: " datafl))
   )
   (if f1
      (progn
         ;
         ;*Read the first line from the circuits file.
         (setq sln1 (read-line f1))
         ;
         ;*Process each line of the circuits file.
```

```
(while sln1
   (progn
      ;
      ;*Get a list of attribute values by calling
      ;*the function sparse with the input line as
      ;*its argument.
      ;*Count the number of attribute values--there
      ;*should be 11.
      (setq slst1 (sparse sln1))
      (setq slen1 (length slst1))
      (if (= slen1 11)
         (progn
            ;
            ;*Set variables to the attribute values
            ;*from the list.
            (setq cktid (car slst1))
            (setq slst1 (cdr slst1))
            (setq tckt (car slst1))
            (setq slst1 (cdr slst1))
            (setq kbps (car slst1))
            (setq slst1 (cdr slst1))
            (setq sgnl (car slst1))
            (setq slst1 (cdr slst1))
            (setq txmed (car slst1))
            (setq slst1 (cdr slst1))
            (setq fac1 (car slst1))
            (setq slst1 (cdr slst1))
            (setq elat1 (car slst1))
            (setq slst1 (cdr slst1))
            (setq elong1 (car slst1))
            (setq slst1 (cdr slst1))
            (setq fac2 (car slst1))
            (setq slst1 (cdr slst1))
            (setq elat2 (car slst1))
            (setq slst1 (cdr slst1))
            (setq elong2 (car slst1))
            (setq slst1 (cdr slst1))
            ;
            ;*Insert circuit entities by calling
            ;*the function insckt with the
            ;*attribute values as arguments.
            (InsCkt
               central earth
               elat1 elong1 fac1 elat2 elong2 fac2
               cktid tckt kbps sgnl txmed
            )
```

```
                  )
                  (progn
                     (*ERROR* "bad input data")
                     (princ (strcat "\nInput item count: "
                        (itoa slen1)))
                  )
               )
               ;
               ;*Read the next line from the
               ;*circuits file.
               (setq sln1 (read-line f1))
            )
         )
         ;
         ;*Close the circuits file.
         (close f1)
      )
   )
   ;
   ;*Restore echo and blip.
   (moder)
)

;**********
;* Command:  lladdckt (add circuit at latitude, longitude)
;* Author:   Tom Brownfield 1990
;* Purpose:  Get attributes from the command line and plot
;*           a circuit with endpoints on the map using
;*           the latitudes and longitudes entered.
;* Inputs:   command line
;* Outputs:  command line
;
(defun C:lladdckt
   (
   / central earth
     clat1 clong1 clat2 clong2 cktid tckt kbps sgnl txmed
     elat1 elong1 eckt1 fac1
     elat2 elong2 eckt2 fac2)
   ;
   ;*Save and turn off echo and blip.
   (modes '("CMDECHO" "BLIPMODE"))
   (setvar "CMDECHO" 0)
   (setvar "BLIPMODE" 0)
   ;
   ;*Set layer to POINTS.
   (command "LAYER" "M" "POINTS" "")
   ;
   ;*Get central meridian of map.
   (setq central (getvar "USERR1"))
   ;
   ;*Set earth radius.
   (setq earth 6371.0 )
   ;
   ;*Be sure entity handles are on.
   (command "HANDLES" "ON")
   ;
   ;*Get attribute values from the command line.
   (setq elat1 (rtos
      (getreal "\nEnter first Latitude in decimal degrees: ")))
   (setq elong1 (rtos
      (getreal "\nEnter first Longitude in decimal degrees : ")))
   (setq fac1 (getstring "\nEnter FACILITY: "))
   (setq elat2 (rtos
      (getreal "\nEnter second Latitude in decimal degrees: ")))
   (setq elong2 (rtos
      (getreal
         "\nEnter second Longitude in decimal degrees : ")))
   (setq fac2 (getstring "\nEnter FACILITY: "))
   (setq cktid (getstring "\nEnter NSI_CKT_ID: "))
   (setq tckt (getstring "\nEnter TYPE_CIRCUIT: "))
   (setq kbps (getstring "\nEnter KBPS: "))
   (setq sgnl (getstring "\nEnter SIGNAL: "))
   (setq txmed (getstring "\nEnter TX_MEDIUM: "))
   ;
   ;*Insert circuit entities by calling the function
   ;*insckt with the attribute values as arguments.
   (InsCkt
      central earth
      elat1 elong1 fac1 elat2 elong2 fac2
      cktid tckt kbps sgnl txmed
   )
   ;
   ;*Restore echo and blip.
   (moder)
)

;**********
;* Function:  delckt (delete circuit)
;* Author:    Tom Brownfield 1990
;* Purpose:   Delete a circuit with its endpoints from
;*            the map.
;* Inputs:    entity name of circuit insert
;* Outputs:   none
;
(defun delckt( ename / ent etype etag lineeh ep1eh ep2eh dname )
   ;
   ;*Save the given circuit entity name.
   (setq dname ename)
   ;
   ;*Get the first entity association list and entity type
   ;*for the given entity name.
   (setq ent (entget ename))
   (setq etype (cdr (assoc 0 ent)))
   ;
   ;*Process each entity association list in the INSERT
   ;*sequence.
   (while (not (equal etype "SEQEND"))
      (progn
         ;
         ;*For attribute entity lists:
         ;*   Get the attribute tags;
         ;*   Set variables to attribute values
         ;*   for the line and the two endpoint
         ;*   entity handles.
         (if (equal etype "ATTRIB")
            (progn
               (setq etag (cdr (assoc 2 ent)))
               (if (equal etag "LINE_EH")
                  (progn
                     (setq lineeh (cdr (assoc 1 ent)))
                  )
               )
               (if (equal etag "EP1_EH")
                  (progn
                     (setq ep1eh (cdr (assoc 1 ent)))
                  )
               )
               (if (equal etag "EP2_EH")
                  (progn
                     (setq ep2eh (cdr (assoc 1 ent)))
                  )
               )
            )
         )
         ;
         ;*Get the next entity name, association list,
         ;*and entity type.
         (setq ename (entnext ename))
         (setq ent (entget ename))
         (setq etype (cdr (assoc 0 ent)))
      )
   )
   ;
   ;*Delete the circuit entity.
   (entdel dname)
   ;
   ;*Get the entity names for the entity handles of
   ;*the line and the two endpoints.
   ;*Delete the line and two endpoints.
   (setq ename (handent lineeh))
   (entdel ename)
   (setq ename (handent ep1eh))
   (entdel ename)
   (setq ename (handent ep2eh))
   (entdel ename)
)

;**********
;* Command:  delckts (delete all circuits)
;* Author:   Tom Brownfield 1990
;* Purpose:  Find all circuits plotted on the map
;*           and delete them.
;* Inputs:   command line
;* Outputs:  command line
;
(defun C:delckts( / ss sscnt ssidx ename)
   ;
   ;*Save and turn off echo.
   (modes '("CMDECHO"))
   (setvar "CMDECHO" 0)
   ;
   ;*Build a selection set of all entity names of
   ;*insertions of the block LLCKT on the layer POINTS.
   (setq ss (ssget "X" (list (cons 0 "INSERT")
                             (cons 2 "llckt")
                             (cons 8 "POINTS"))))
   ;
   ;*Count the number of entities selected.
   (if (null ss)
      (setq sscnt 0)
      (setq sscnt (sslength ss))
   )
   (princ (itoa sscnt))
   (princ " circuits will be deleted.")
   (terpri)
   ;
   ;*Set index for the first entity name in the set.
   (setq ssidx 0)
   ;
   ;*Process each entity name in the set.
   (while (> sscnt 0)
      (progn
         (setq ename (ssname ss ssidx))
         ;
         ;*Delete the circuit by calling the function
         ;*delckt with the entity name as its argument.
         (delckt ename)
         ;
         ;*Set index for the next entity name in the set.
         (setq ssidx (1+ ssidx))
         (setq sscnt (1- sscnt))
      )
   )
   ;
   ;*Redraw the screen to clean up the map.
   (redraw)
   ;
   ;*Restore echo.
   (moder)
   (princ)
)

;**********
;* Function:  extept (extract endpoint)
;* Author:    Tom Brownfield 1990
;* Purpose:   Extract attribute values from an endpoint.
;* Inputs:    entity name of endpoint insertion
;* Outputs:   list of attribute values from endpoint
;
(defun extept
   ( ename
   / ent etype
     efac elat elong
   )
   ;
   ;*Get the first entity association list and entity type
   ;*for the given entity name.
   (setq ent (entget ename))
   (setq etype (cdr (assoc 0 ent)))
   ;
   ;*Process each entity association list in the INSERT
   ;*sequence.
   (while (not (equal etype "SEQEND"))
      (progn
         ;
         ;*For attribute entity lists:
         ;*   Get the attribute tags;
         ;*   Set variables to attribute values.
         (if (equal etype "ATTRIB")
            (progn
               (setq etag (cdr (assoc 2 ent)))
               (if (equal etag "FACILITY")
                  (setq efac (cdr (assoc 1 ent)))
               )
               (if (equal etag "LATITUDE")
                  (setq elat (cdr (assoc 1 ent)))
               )
               (if (equal etag "LONGITUDE")
                  (setq elong (cdr (assoc 1 ent)))
               )
            )
         )
         ;
         ;*Get the next entity name, association list,
         ;*and entity type.
         (setq ename (entnext ename))
         (setq ent (entget ename))
         (setq etype (cdr (assoc 0 ent)))
      )
   )
   ;
   ;*Return return a list of the attribute values for
   ;*the endpoint.
   (list efac elat elong)
)

;************
;* Function:  extckt (extract circuit)
;* Author:    TomBrownfield 1990
;* Purpose:   Extract attribute values from a circuit.
;* Inputs:    entity name of circuit insertion.
;* Outputs:   list of attribute values from circuit
;*            and endpoints
;
(defun extckt
   ( ename
/ ent etype etag
  cktid tckt kbps sgnl txmed lineeh ep1eh ep2eh
  fac1 elat1 elong1 fac2 elat2 elong2
  elist1 elist2
)
;
;*Get the first entity association list and entity type
;*for the given entity name.
(setq ent (entget ename))
(setq etype (cdr (assoc 0 ent)))
;
;*Process each entity association list in the INSERT
;*sequence.
(while (not (equal etype "SEQEND"))
   (progn
      ;
      ;*For attribute entity lists:
      ;*   Get the attribute tags;
      ;*   Set variables to the attribute values.
      (if (equal etype "ATTRIB")
         (progn
            (setq etag (cdr (assoc 2 ent)))
            (if (equal etag "NSI_CKT_ID")
               (setq cktid (cdr (assoc 1 ent)))
            )
            (if (equal etag "TYPE_CIRCUIT")
               (setq tckt (cdr (assoc 1 ent)))
            )
            (if (equal etag "KBPS")
               (setq kbps (cdr (assoc 1 ent)))
            )
            (if (equal etag "SIGNAL")
               (setq sgnl (cdr (assoc 1 ent)))
            )
            (if (equal etag "TX_MEDIUM")
               (setq txmed (cdr (assoc 1 ent)))
            )
            (if (equal etag "LINE_EH")
               (setq lineeh (cdr (assoc 1 ent)))
            )
            (if (equal etag "EP1_EH")
               (setq ep1eh (cdr (assoc 1 ent)))
            )
            (if (equal etag "EP2_EH")
               (setq ep2eh (cdr (assoc 1 ent)))
            )
            )
         )
         ;
         ;*Get the next entity name, association list,
         ;*and entity type.
         (setq ename (entnext ename))
         (setq ent (entget ename))
         (setq etype (cdr (assoc 0 ent)))
      )
   )
   ;
   ;*Get the entity names for the entity handles of
   ;*the two endpoints.
   ;*Get a list of attribute values for each endpoint
   ;*by calling the function extept with each entity
   ;*name as its argument.
   (setq ename (handent ep1eh))
   (setq elist1 (extept ename))
   (setq ename (handent ep2eh))
   (setq elist2 (extept ename))
   ;
   ;*Set variables to the attribute values from the
   ;*two endpoint lists.
   (setq fac1 (car elist1))
   (setq elat1 (cadr elist1))
   (setq elong1 (caddr elist1))
   (setq fac2 (car elist2))
   (setq elat2 (cadr elist2))
   (setq elong2 (caddr elist2))
   ;
   ;*Return a list of the attribute values for the
   ;*circuit and its two endpoints.
   (list cktid tckt kbps sgnl txmed
      fac1 elat1 elong1
      fac2 elat2 elong2)
)

;**********
;* Command:  extckts (extract all circuits)
;* Author:   Tom Brownfield 1990
;* Purpose:  Find all circuits plotted on the map
;*           and extract all their attributes.
;*           Write a line of attributes to the
;*           circuits file for each circuit.
;* Inputs:   command line
;* Outputs:  command line
;*           circuits disk file
;
(defun C:extckts( / ss sscnt ssidx ename clist clcnt cln )
   ;
   ;*Save and turn off echo.
   (modes '("CMDECHO"))
   (setvar "CMDECHO" 0)
   ;
   ;*Get the circuits file name from the command line.
   ;*Open the file for output.
   (setq datafl (getstring "\nEnter name of circuits file: "))
   (setq fl (open datafl "w"))
   (if (not fl)
      (*ERROR* (strcat "cannot open file: " datafl))
   )
   (if fl
      (progn
         ;
         ;*Build a selection set of all entity names of
         ;*insertions of the block LLCKT on the layer POINTS.
         (setq ss (ssget "X" (list (cons 0 "INSERT")
                                   (cons 2 "llckt")
                                   (cons 8 "POINTS"))))
         ;
         ;*Count the number of entities selected.
         (if (null ss)
            (setq sscnt 0)
            (setq sscnt (sslength ss))
         )
         (princ (itoa sscnt))
         (princ " circuits will be extracted.")
         (terpri)
         ;
         ;*Set index for the first entity name in the set.
         (setq ssidx 0)
         ;
         ;*Process each entity name in the set.
         (while (> sscnt 0)
            (progn
               (setq ename (ssname ss ssidx))
               ;
               ;*Get a list of attribute values by calling
               ;*the function extckt with the entity name
               ;*as its argument.
               (setq clist (extckt ename))
               (setq cln "")
               ;
               ;*Count the number of attribute values in
               ;*the list.
               (if (null clist)
                  (setq clcnt 0)
                  (setq clcnt (length clist))
               )
               ;
               ;*Process each attribute value in the list.
               (while (> clcnt 0)
                  (progn
                     ;
                     ;*Put each value to the output line
                     ;*with a space delimiter.
                     (setq cln (strcat cln (car clist) " "))
                     (setq clist (cdr clist))
                     (setq clcnt (1- clcnt))
                  )
               )
               ;
               ;*Write the output line to the circuits file.
               (write-line cln f1)
               ;
               ;*Set index for the next entity name in
               ;*the set.
               (setq ssidx (1+ ssidx))
               (setq sscnt (1- sscnt))
            )
         )
         ;
         ;*Close the circuits file.
         (close f1)
      )
   )
   ;
   ;*Restore echo.
   (moder)
   (princ)
)

;**********
;* Command:  seldbckts (select database circuits)
;* Author:   Tom Brownfield 1990
;* Purpose:  Run a program to select attribute values
;*           from an external database and write them
;*           to a circuits file.
;* Inputs:   command line
;* Outputs:  command line
;*           circuits disk file
;
(defun C:seldbckts
   (
   / errcnt cname rname pname cf rf cmdline rline rlist succeed )
   ;
   ;*Save and turn off echo.
   (modes '("CMDECHO"))
   (setvar "CMDECHO" 0)
   ;
   ;*Initialize error count.
   (setq errcnt 0)
   (if (= errcnt 0)
      (progn
         ;
         ;*Initialize empty results file.
         (setq rname "selres.txt")
         (setq rf (open rname "w"))
         (if (not rf)
            (progn
               (*ERROR* (strcat "cannot open file: " rname))
               (setq errcnt (1+ errcnt))
            )
            (close rf)
         )
      )
   )
   (if (= errcnt 0)
      (progn
         ;
         ;*Get the circuits file name from the command line.
         ;*Initialize empty circuits file.
         (setq cname
            (getstring "\nEnter name of circuits file: "))
         (setq cf (open cname "w"))
         (if (not cf)
            (progn
               (*ERROR* (strcat "cannot open file: " cname))
               (setq errcnt (1+ errcnt))
            )
            (close cf)
         )
      )
   )
```

```
(if (= errcnt 0)
   (progn
      ;
      ;*Get a db server password from the command line.
      (setq pname (getstring "\nEnter db server password: "))
      (if (= (strlen pname) 0)
         (progn
            (*ERROR* "null password")
            (setq errcnt (1+ errcnt))
         )
      )
   )
)
(if (= errcnt 0)
   (progn
      ;
      ;*Assemble a shell command to run the db selection
      ;*program with arguments for password, circuits file
      ;*name, and results file name.
      ;*Execute the shell command.
      ;*Read the results file to check for SUCCEED message.
      (setq cmdline (strcat "gtdbckt -P" pname " -C" cname
                            " -R" rname))
      (command "SHELL" cmdline)
      (setq rf (open rname "r"))
      (if (not rf)
         (progn
            (*ERROR* (strcat "cannot open file: " rname))
            (setq errcnt (1+ errcnt))
         )
         (progn
            (setq rline (read-line rf))
            (close rf)
            (setq rlist (sparse rline))
            (setq succeed (car rlist))
            (if (/= "SUCCEED" succeed)
               (progn
                  (*ERROR* "db selection unsuccessful")
                  (setq errcnt (1+ errcnt))
               )
               (progn
                  (princ "\nCircuits selected from database")
                  (princ "\n   to ")
                  (princ cname)
                  (princ "\n")
               )
               )
            )
         )
      )
   )
   ;
   ;*Restore echo.
   (moder)
   (princ)
)

;**********
;* Command:  insdbckts (insert database circuits)
;* Author:   Tom Brownfield 1990
;* Purpose:  Run a program to read attribute values
;*           from a circuits file and insert them
;*           into an external database.
;* Inputs:   command line
;*           circuits disk file
;* Outputs:  command line
;
(defun C:insdbckts
   (
   / errcnt cname rname pname cf rf cmdline rline rlist succeed )
   ;
   ;*Save and turn off echo.
   (modes '("CMDECHO"))
   (setvar "CMDECHO" 0)
   ;
   ;*Initialize error count.
   (setq errcnt 0)
   (if (= errcnt 0)
      (progn
         ;
         ;*Initialize empty results file.
         (setq rname "insres.txt")
         (setq rf (open rname "w"))
         (if (not rf)
            (progn
               (*ERROR* (strcat "cannot open file: " rname))
               (setq errcnt (1+ errcnt))
            )
            (close rf)
         )
      )
   )
```

```
(if (= errcnt 0)
   (progn
      ;
      ;*Get the circuits file name from the command line.
      ;*Open the file to check its presence.
      (setq cname
         (getstring "\nEnter name of circuits file: "))
      (setq cf (open cname "r"))
      (if (not cf)
         (progn
            (*ERROR* (strcat "cannot open file: " cname))
            (setq errcnt (1+ errcnt))
         )
         (close cf)
      )
  )
)
(if (= errcnt 0)
   (progn
      ;
      ;*Get a db server password from the command line.
      (setq pname (getstring "\nEnter db server password: "))
      (if (= (strlen pname) 0)
         (progn
            (*ERROR* "null password")
            (setq errcnt (1+ errcnt))
         )
      )
   )
)
(if (= errcnt 0)
   (progn
      ;
      ;*Assemble a shell command to run the db insertion
      ;*program with arguments for password, circuits file
      ;*name, and results file name.
      ;*Execute the shell command.
      ;*Read the results file to check for SUCCEED message.
      (setq cmdline (strcat "ptdbckt -P" pname " -C" cname
                            " -R" rname))
      (command "SHELL" cmdline)
      (setq rf (open rname "r"))
      (if (not rf)
         (progn
            (*ERROR* (strcat "cannot open file: " rname))
            (setq errcnt (1+ errcnt))
            )
            (progn
               (setq rline (read-line rf))
               (close rf)
               (setq rlist (sparse rline))
               (setq succeed (car rlist))
               (if (/= "SUCCEED" succeed)
                  (progn
                     (*ERROR* "db insertion unsuccessful")
                     (setq errcnt (1+ errcnt))
                  )
                  (progn
                     (princ "\nCircuits inserted into database")
                     (princ "\n   from ")
                     (princ cname)
                     (princ "\n")
                  )
               )
            )
         )
      )
   )
   ;
   ;*Restore echo.
   (moder)
   (princ)
)
```

C PROGRAMS

```
/*
** Program:   gtdbckt (get circuits from database)
** Author:    Tom Brownfield 1990
** Compiler:  Sun UNIX C with Sybase DB-Library
** Purpose:   Select rows from circuit database table,
**            lookup latitude and longitude for each
**            endpoint, and write values to circuits
**            disk file.
** Inputs:    Optional program arguments:
**               -C<circuits_file_name>
**               -R<results_file_name>
**               -P<password>
**            circuit, facility, torg, wire_center
**              in spatial database
** Outputs:   standard output messages
**            circuits disk file
**            results disk file
*/

#include <stdio.h>
#include <sybfront.h>
#include <sybdb.h>
#include <strings.h>

#define BUFLEN      200
#define PWDLEN      64
#define NCILEN      20
#define TCLEN       10
#define KBPLEN      04
#define SIGLEN      11
#define TXMLEN      20
#define EPLEN       10
#define FLTLEN      20

/*
** Forward declarations of the error handler, message handler,
** and lookup function for latitudes and longitudes.
*/
int err_handler();
int msg_handler();
int lookup_ll();

main(argc, argv)
int         argc;
```

```
char         *argv[];
{
    DBPROCESS    *dbproc1;      /* Primary connection with SQL */
                                /* server.                     */
    DBPROCESS    *dbproc2;      /* Secondary connection with   */
                                /* SQL server to be used by    */
                                /* lookup_ll().                */
    LOGINREC     *login;        /* Our login information. */
    RETCODE      return_code1;
    RETCODE      return_code2;
    DBINT        lookup_ok1;
    DBINT        lookup_ok2;

    char         outbuf[BUFLEN];
    FILE         *circuitsfile;
    FILE         *resultsfile;

    char         dbpwd[PWDLEN+1];
    char         cfname[BUFLEN];
    char         rfname[BUFLEN];
    int          i,j,k;
    unsigned
        int      count;
    int          argerr;

    /*
    ** These are the variables used to store the returning data.
    ** <length>+1 allows for a null character.
    ** <length>+1 allows for a space and a null character.
    */
    DBCHAR       nsi_ckt_id[NCILEN+2];
    DBCHAR       type_circuit[TCLEN+2];
    DBCHAR       kbps[KBPLEN+2];
    DBCHAR       signal[SIGLEN+2];
    DBCHAR       tx_medium[TXMLEN+2];
    DBCHAR       end_point1[EPLEN+2];
    DBCHAR       end_point2[EPLEN+2];
    DBCHAR       epts1[EPLEN+2];
    DBCHAR       epts2[EPLEN+2];
    DBCHAR       lats[FLTLEN+1];
    DBCHAR       longs[FLTLEN+1];

    /* Initialize password and file names to defaults. */
    strcpy(dbpwd,"server_password");
    strcpy(cfname,"circuits");
    strcpy(rfname,"results");
```

```
/*
** Scan program arguments for valid options:
** -C<circuits_file_name>
** -R<results_file_name>
** -P<password>
*/
argerr = 0;
for (i = 1; i < argc; i++) {
    printf("argument: %s\n", argv[i]);
    if (argv[i][0] == '-') {
        switch (argv[i][1]) {
        case 'C':
            j = 2;
            k = 0;
            while (argv[i][j]) {
                cfname[k] = argv[i][j];
                ++j;
                ++k;
            }
            cfname[k] = '\0';
            break;
        case 'R':
            j = 2;
            k = 0;
            while (argv[i][j]) {
                rfname[k] = argv[i][j];
                ++j;
                ++k;
            }
            rfname[k] = '\0';
            break;
        case 'P':
            j = 2;
            k = 0;
            while (argv[i][j]) {
                dbpwd[k] = argv[i][j];
                ++j;
                ++k;
            }
            dbpwd[k] = '\0';
            break;
        default:
            argerr = 1;
            printf("illegal option %c\n",argv[i][1]);
            break;
        }
    }
    else {
        argerr = 1;
        printf("option must begin with '-'\n");
    }
}

/* Check for program argument errors. */
if (argerr) {
    printf("%s had argument errors--discontinued.\n",
            argv[0]);
    exit(STDEXIT);
}

/* Open results file. */
if ((resultsfile = fopen(rfname, "w")) == NULL) {
    printf("Unable to open file %s\n",rfname);
    exit(STDEXIT);
}

/* Open circuits file. */
if ((circuitsfile = fopen(cfname, "w")) == NULL) {
    printf("Unable to open file %s\n",cfname);
    exit(STDEXIT);
}

/* Initialize DB-lIBRARY. */
if (dbinit() == FAIL)
    exit(ERREXIT);

/*
** Install the user-supplied error-handling and
** message-handling routines.  They are defined at
** the bottom of this source file.
*/
dberrhandle(err_handler);
dbmsghandle(msg_handler);

/*
** Get a LOGINREC structure and fill it with the necessary
** login information.
*/
login = dblogin();
DBSETLPWD(login, dbpwd);
DBSETLAPP(login, "getdbces");
```

```
/*
** Get two DBPROCESS structures for communicating with SQL
** Server.  A NULL servername defaults to the server
** specified by DSQUERY.
*/
dbproc1 = dbopen(login, NULL);
dbproc2 = dbopen(login, NULL);

/* Use the spatial db. */
dbuse(dbproc1, "spatial");
dbuse(dbproc2, "spatial");

printf("Selecting rows from the 'circuit' table:\n");

/* Assemble SQL select transaction in command buffer. */
dbcmd(dbproc1, "select * from circuit");

/* Send transaction to SQL server. */
dbsqlexec(dbproc1);

/* Get results from transaction. */
while ((return_code1 = dbresults(dbproc1)) !=
        NO_MORE_RESULTS) {
    if (return_code1 == SUCCEED) {

        /* Bind columns to program variables. */
        dbbind(dbproc1, 1, CHARBIND, 0, nsi_ckt_id);
        dbbind(dbproc1, 2, CHARBIND, 0, type_circuit);
        dbbind(dbproc1, 3, CHARBIND, 0, kbps);
        dbbind(dbproc1, 4, CHARBIND, 0, signal);
        dbbind(dbproc1, 5, CHARBIND, 0, tx_medium);
        dbbind(dbproc1, 6, CHARBIND, 0, end_point1);
        dbbind(dbproc1, 7, CHARBIND, 0, end_point2);

        /*
        ** Initialize space in arrays
        ** to serve as delimiter in circuits file.
        */
        nsi_ckt_id[NCILEN] = ' ';
        type_circuit[TCLEN] = ' ';
        kbps[KBPLEN] = ' ';
        signal[SIGLEN] = ' ';
        tx_medium[TXMLEN] = ' ';
        end_point1[EPLEN] = ' ';
        end_point2[EPLEN] = ' ';
```

```
/*
** Initialize null terminator in arrays
** since CHARBIND does not add one.
*/
nsi_ckt_id[NCILEN+1] = '\0';
type_circuit[TCLEN+1] = '\0';
kbps[KBPLEN+1] = '\0';
signal[SIGLEN+1] = '\0';
tx_medium[TXMLEN+1] = '\0';
end_point1[EPLEN+1] = '\0';
end_point2[EPLEN+1] = '\0';

/* Get each selected row. */
while (dbnextrow(dbproc1) != NO_MORE_ROWS) {
    printf
        ("%s%s%s%s%s\n",
        nsi_ckt_id, type_circuit, kbps, signal,
        tx_medium);

    /* Copy variables to outbuf. */
    strcpy(outbuf, nsi_ckt_id);
    strcat(outbuf,type_circuit);
    strcat(outbuf,kbps);
    strcat(outbuf,signal);
    strcat(outbuf,tx_medium);
    strcat(outbuf, end_point1);

    /*
    ** Copy end_point1 to epts1, and remove trailing
    ** spaces by replacing the first space with a
    ** null character.
    */
    strcpy(epts1, end_point1);
    epts1[strcspn(epts1," ")] = '\0';

    /*
    ** Call lookup_ll() to get latitude, longitude
    ** of epts1 using the secondary DBPROCESS.
    ** Check results of lookup.
    */
    if ((lookup_ok1 =
        lookup_ll(dbproc2, epts1, lats, longs)) == 0)
        printf("BAD LOOKUP %s\n", end_point1);
    else {
        printf("%s%s %s \n", end_point1, lats,
            longs);

    /*
    ** if lookup was ok,
    ** copy variables to outbuf.
    */
    strcat(outbuf, lats);
    strcat(outbuf, " ");
    strcat(outbuf, longs);
    strcat(outbuf, " ");
}

/* Copy variables to outbuf. */
strcat(outbuf, end_point2);

/*
** Copy end_point2 to epts2, and remove trailing
** spaces by replacing the first space with a
** null character.
*/
strcpy(epts2, end_point2);
epts2[strcspn(epts2," ")] = '\0';

/*
** Call lookup_ll() to get latitude, longitude
** of epts2 using the secondary DBPROCESS.
** Check results of lookup.
*/
if ((lookup_ok2 =
    lookup_ll(dbproc2, epts2, lats, longs)) == 0)
    printf("BAD LOOKUP %s\n", end_point2);
else {

    /*
    ** if lookup was ok,
    ** copy variables to outbuf.
    */
    printf("%s%s %s \n", end_point2, lats,
            longs);
    strcat(outbuf, lats);
    strcat(outbuf, " ");
    strcat(outbuf, longs);
    strcat(outbuf, " ");
}

/* Copy newline character to outbuf. */
```

```
                strcat(outbuf, "\n");

                /*
                ** If both lookups were ok, write record
                ** to circuits file.
                */
                if (lookup_ok1 == 1 && lookup_ok2 == 1)
                    fputs(outbuf, circuitsfile);
            }
        }
    }

    /* Close DBPROCESS structure. */
    dbexit();

    /* Close circuits file. */
    fclose(circuitsfile);

    /* Write "SUCCEED" message to results file and close. */
    fputs("SUCCEED\n", resultsfile);
    fclose(resultsfile);

    /* Normal program exit. */
    exit(STDEXIT);
}

/*
** Function:  lookup_ll (lookup latitude, longitude)
** Author:    Tom Brownfield 1990
** Purpose:   Select the latitude and longitude of the
**            wire center serving an endpoint facility.
** Inputs:    Arguments:
**               dbproc--pointer to DBPROCESS structure
**               epts--pointer to string containing endpoint
**                     facility
**               rlats--pointer to return string for latitude
**               rlongs--pointer to return string for longitude
** Outputs:  latitude string value
**           longitude string value
**           function return error status
*/

int lookup_ll(dbproc, epts, rlats, rlongs)
DBPROCESS   *dbproc;
DBCHAR      *epts;
DBCHAR      *rlats;
DBCHAR      *rlongs;
{
    RETCODE     return_code;
    DBCHAR      lats[FLTLEN+1];
    DBCHAR      longs[FLTLEN+1];
    DBINT       dbcount;

    /*
    ** Assemble SQL select transaction in command buffer
    ** using epts for condition value.
    */
    dbcmd (dbproc, "select lats = str(w.latitude,10,4),     ");
    dbcmd (dbproc, "       longs = str(w.longitude,10,4)    ");
    dbcmd (dbproc, " from facility f,torg o,                ");
    dbcmd (dbproc, "      wire_center w                     ");
    dbfcmd(dbproc, " where f.facility = '%s'          ", epts);
    dbcmd (dbproc, "   and o.org_key = f.org_key            ");
    dbcmd (dbproc, "   and w.npa =                          ");
    dbcmd (dbproc, "       substring(o.commercial_phone,2,3)");
    dbcmd (dbproc, "   and w.nxx =                          ");
    dbcmd (dbproc, "       substring(o.commercial_phone,6,3)");

    /* Send transaction to SQL server. */
    dbsqlexec(dbproc);

    /* Get results from transaction. */
    while ((return_code = dbresults(dbproc)) !=
            NO_MORE_RESULTS) {
        if (return_code == SUCCEED) {

            /* Bind columns to function variables. */
            dbbind(dbproc, 1, STRINGBIND, 0, lats);
            dbbind(dbproc, 2, STRINGBIND, 0, longs);

            /* Get each selected row. */
            while (dbnextrow(dbproc) != NO_MORE_ROWS) {

                /* Ignore any rows after the first. */
                if (DBCURROW(dbproc) > 1)
                    continue;

                /* copy function variables to outputs. */
                strcpy(rlats, lats);
                strcpy(rlongs, longs);
            }
        }
    }

    /*
    ** If one row was selected return 1 for success;
    ** else return 0 for failure.
    */
    if ((dbcount = DBCOUNT(dbproc)) == 1)
        return(1);
    else
        return(0);
}

/*
** Function:  err_handler
** Author:    (from Sybase DBLibrary Examples)
*/

int err_handler(dbproc, severity, dberr, oserr, dberrstr,
oserrstr)
DBPROCESS   *dbproc;
int         severity;
int         dberr;
int         oserr;
char        *dberrstr;
char        *oserrstr;
{
    if ((dbproc == NULL) || (DBDEAD(dbproc)))
        return(INT_EXIT);
    else {
        printf("DB-Library error:\n\t%s\n", dberrstr);

        if (oserr != DBNOERR)
            printf("Operating-system error:\n\t%s\n", oserrstr);

        return(INT_CANCEL);
    }
}

/*
** Function:  msg_handler
** Author:    (from Sybase DBLibrary Examples)
*/

int    msg_handler(dbproc, msgno, msgstate, severity, msgtext,
srvname, procname, line)
```

```
DBPROCESS   *dbproc;
DBINT       msgno;
int         msgstate;
int         severity;
char        *msgtext;
char        *srvname;
char        *procname;
DBUSMALLINT line;

{
    printf ("Msg %ld, Level %d, State %d\n",
        msgno, severity, msgstate);

    if (strlen(srvname) > 0)
        printf ("Server '%s', ", srvname);
    if (strlen(procname) > 0)
        printf ("Procedure '%s', ", procname);
    if (line > 0)
        printf ("Line %d", line);

    printf("\n\t%s\n", msgtext);

    return(0);
}

/*
** Program:   ptdbckt (put circuits into database)
** Author:    Tom Brownfield 1990
** Compiler:  Sun UNIX C with Sybase DB-Library
** Purpose:   Read circuits disk file, and use the values
**            to insert new rows into circuit database
**            table.
** Inputs:    Optional program arguments:
**               -C<circuits_file_name>
**               -R<results_file_name>
**               -P<password>
**            circuits disk file
** Outputs:   standard output messages
**            circuit in spatial database
**            results disk file
*/

#include <stdio.h>
#include <sybfront.h>
#include <sybdb.h>
#include <strings.h>
```

```
#define BUFLEN      200
#define PWDLEN      64

/*
** Forward declarations of the error handler and message
** handler.
*/
int err_handler();
int msg_handler();

main(argc, argv)
int         argc;
char        *argv[];
{
    DBPROCESS   *dbproc1;      /* Our connection with SQL */
                               /* server.                 */
    LOGINREC    *login;        /* Our login information. */
    RETCODE     return_code1;

    char        valbuf[BUFLEN];
    char        inbuf[BUFLEN];
    char        *ibp;
    int         tokcnt;
    FILE        *circuitsfile;
    FILE        *resultsfile;

    char        dbpwd[PWDLEN+1];
    char        cfname[BUFLEN];
    char        rfname[BUFLEN];
    int         i,j,k;
    unsigned
        int     count;
    int         argerr;

    /* Initialize password and file names to defaults. */
    strcpy(dbpwd,"server_password");
    strcpy(cfname,"circuits");
    strcpy(rfname,"results");

    /*
    ** Scan program arguments for valid options:
    ** -C<circuits_file_name>
    ** -R<results_file_name>
    ** -P<password>
    */
```

```
argerr = 0;
for (i = 1; i < argc; i++) {
    printf("argument: %s\n", argv[i]);
    if (argv[i][0] == '-') {
        switch (argv[i][1]) {
        case 'C':
            j = 2;
            k = 0;
            while (argv[i][j]) {
                cfname[k] = argv[i][j];
                ++j;
                ++k;
            }
            cfname[k] = '\0';
            break;
        case 'R':
            j = 2;
            k = 0;
            while (argv[i][j]) {
                rfname[k] = argv[i][j];
                ++j;
                ++k;
            }
            rfname[k] = '\0';
            break;
        case 'P':
            j = 2;
            k = 0;
            while (argv[i][j]) {
                dbpwd[k] = argv[i][j];
                ++j;
                ++k;
            }
            dbpwd[k] = '\0';
            break;
        default:
            argerr = 1;
            printf("illegal option %c\n",argv[i][1]);
            break;
        }
    }
    else {
        argerr = 1;
        printf("option must begin with '-'\n");
    }
}
```

```
/* Check for program argument errors. */
if (argerr) {
    printf("%s had argument errors--discontinued.\n",
            argv[0]);
    exit(STDEXIT);
}

/* Open results file. */
if ((resultsfile = fopen(rfname, "w")) == NULL) {
    printf("Unable to open file %s\n",rfname);
    exit(STDEXIT);
}

/* Open circuits file. */
if ((circuitsfile = fopen(cfname, "r")) == NULL) {
    printf("Unable to open file %s\n",cfname);
    exit(STDEXIT);
}

/* Initialize DB-lIBRARY. */
if (dbinit() == FAIL)
    exit(ERREXIT);

/*
** Install the user-supplied error-handling and
** message-handling routines. They are defined at
** the bottom of this source file.
*/
dberrhandle(err_handler);
dbmsghandle(msg_handler);

/*
** Get a LOGINREC structure and fill it with the necessary
** login information.
*/

login = dblogin();
DBSETLPWD(login, dbpwd);
DBSETLAPP(login, "putdbckts");

/*
** Get a DBPROCESS structure for communicating with SQL
** Server.  A NULL servername defaults to the server
** specified by DSQUERY.
*/
```

```
dbproc1 = dbopen(login, NULL);

/* Use the spatial db. */
dbuse(dbproc1, "spatial");

printf("Inserting rows into the 'circuit' table:\n");

/* Read each record from circuits. */
while ((fgets(inbuf,BUFLEN,circuitsfile)) != NULL) {

    /*
    ** Scan the first space-delimited token.
    ** Copy it to valbuf surrounded by quotes.
    */
    tokcnt = 0;
    if (ibp = (char *) strtok(inbuf," ")) {
        tokcnt++;
        strcpy(valbuf,"'");
        strcat(valbuf,ibp);
        strcat(valbuf,"'");
    }

    /*
    ** Scan the remaining tokens.
    ** Copy the second through sixth and the ninth
    ** (excluding latitudes and longitudes) to valbuf
    ** separated by commas and surrounded by quotes.
    */
    while (ibp = (char *) strtok(NULL," ")) {
        tokcnt++;
        if ((tokcnt < 7) || (tokcnt == 9)) {
            strcat(valbuf,",'");
            strcat(valbuf,ibp);
            strcat(valbuf,"'");
        }
    }
    printf("%s\n",valbuf);

    /*
    ** Assemble SQL insert transaction in command
    ** buffer using valbuf for values.
    */
    dbcmd (dbproc1, "begin tran             ");
    dbcmd (dbproc1, "insert into circuit    ");
    dbfcmd(dbproc1, "   values(%s) \n",valbuf);
    dbcmd (dbproc1, "commit tran            ");
```

```
        /* Send transaction to SQL server. */
        dbsqlexec(dbproc1);

        /* Get results from transaction. */
        while ((return_code1 = dbresults(dbproc1)) !=
                NO_MORE_RESULTS) {

            /* Check results for failed transaction. */
            if (return_code1 == FAIL) {
                printf("DB insert failed:\n");
                printf("%s\n",valbuf);
            }
        }
    }

    /* Close DBPROCESS structure. */
    dbexit();

    /* Close circuits file. */
    fclose(circuitsfile);

    /* Write "SUCCEED" message to results file and close. */
    fputs("SUCCEED\n", resultsfile);
    fclose(resultsfile);

    /* Normal program exit. */
    exit(STDEXIT);
}

/*
** Function:  err_handler
** Author:    (from Sybase DBLibrary Examples)
*/

int err_handler(dbproc, severity, dberr, oserr, dberrstr,
oserrstr)
DBPROCESS   *dbproc;
int         severity;
int         dberr;
int         oserr;
char        *dberrstr;
char        *oserrstr;
{
    if ((dbproc == NULL) || (DBDEAD(dbproc)))
        return(INT_EXIT);
    else {
        printf("DB-Library error:\n\t%s\n", dberrstr);

        if (oserr != DBNOERR)
            printf("Operating-system error:\n\t%s\n", oserrstr);

        return(INT_CANCEL);
    }
}

/*
** Function:  msg_handler
** Author:    (from Sybase DBLibrary Examples)
*/

int msg_handler(dbproc, msgno, msgstate, severity, msgtext,
srvname, procname, line)

DBPROCESS   *dbproc;
DBINT       msgno;
int         msgstate;
int         severity;
char        *msgtext;
char        *srvname;
char        *procname;
DBUSMALLINT line;

{
    printf ("Msg %ld, Level %d, State %d\n",
        msgno, severity, msgstate);

    if (strlen(srvname) > 0)
        printf ("Server '%s', ", srvname);
    if (strlen(procname) > 0)
        printf ("Procedure '%s', ", procname);
    if (line > 0)
        printf ("Line %d", line);

    printf("\n\t%s\n", msgtext);

    return(0);
}
?
```