Report Number: WUCS-87-23

1987-08-07

Software for the Standard Linear Format for Digital Cartographic Feature Data

Stephen E. Reichenbach

This paper describes application software and programming tools designed for use with the Defense Mapping Agency's (DMA) Standard Linear Format (SLF) for Digital Cartographic Feature Data. The Standard Linear Format (SLF) is briefly described in this report. It was designed as a standard for the exchange of digital cartographic features on magnetic tape. The format specifies descriptive fields about feature data, as well as specifying the representation of the features. The application software described in this report can transfer files or tapes in this format to relational database maintained under Ingres, graph features in the database, alter the database,... Read complete abstract on page 2.

Software for the Standard Linear Format for Digital Cartographic Feature Data

Stephen E. Reichenbach

Complete Abstract:

This paper describes application software and programming tools designed for use with the Defense Mapping Agency's (DMA) Standard Linear Format (SLF) for Digital Cartographic Feature Data. The Standard Linear Format (SLF) is briefly described in this report. It was designed as a standard for the exchange of digital cartographic features on magnetic tape. The format specifies descriptive fields about feature data, as well as specifying the representation of the features. The application software described in this report can transfer files or tapes in this format to relational database maintained under Ingres, graph features in the database, alter the database, and convert the database back to the SLF. The subroutine library utilized by these application programs is also described. These subroutine should make the task of writing additional application software for the SLF much easier.

SOFTWARE FOR THE STANDARD LINEAR FORMAT FOR DIGITAL CARTOGRAPHIC FEATURE DATA

Stephen E. Reichenbach

WUCS-87-23

Department of Computer Science
Washington University
Campus Box 1045
One Brookings Drive
Saint Louis, MO 63130-4899

Software for the Standard Linear Format for Digital Cartographic Feature Data

Stephen E. Reichenbach

Washington University
St. Louis, MO 63130

ABSTRACT

This paper describes application software and programming tools designed for use with the Defense Mapping Agency's (DMA) Standard Linear Format (SLF) for Digital Cartographic Feature Data. The Standard Linear Format (SLF) is briefly described in this report. It was designed as a standard for the exchange of digital cartographic features on magnetic tape. The format specifies descriptive fields about feature data, as well as specifying the representation of the features. The application software described in this report can transfer files or tapes in this format to a relational database maintained under Ingres, graph features in the database, alter the database, and convert the database back to the SLF. The subroutine library utilized by these application programs is also described. These subroutines should make the task of writing additional application software for the SLF much easier.

Acknowledgments: This work was supported in part by the Defense Mapping Agency under contract DMA800-85-C-0010.

August 7, 1986

Software for the Standard Linear Format for Digital Cartographic Feature Data

Stephen E. Reichenbach

Washington University
St. Louis, MO 63130

1. INTRODUCTION

The *Standard Linear Format for Digital Cartographic Feature Data* (Draft 2nd ed., 18 March 1985) has been developed by the Defense Mapping Agency as a standard for the exchange of digital cartographic feature data on magnetic tape. The DMA is increasingly involved in the production of digital data. They recognize that a standard exchange and offline storage format for digital data could increase the efficiency of DMA by facilitating sharing between all DMA subsystems regardless of computer system or mission.

Institutions working with the DMA can also benefit from this standard. We at Washington University believe that increasing our ability to exchange data with the DMA should benefit both organizations. The DMA can help us to better understand the problems they are interested in solving and we can provide our results in an accessible format. By improving our communication, this software should help draw our two communities closer together.

The SLF software described in this paper includes the essential application programs to receive, display, edit, and generate SLF data. The library of support routines we have developed provides a resource for future application programs. The SLF software has been incorporated into the Washington University/Defense Mapping Agency (WUDMA) Image Processing Software System. This large software system was initially designed for a training program at Washington University for technical personnel of the DMA. It is currently used for research and other educational programs. It is flexible and easily built upon. It now contains about 200 image processing, analysis, and display programs. One of the primary goals of this system was to incorporate the flexibility to handle many image formats. (See "The WUDMA Image Processing System," by Steve Reichenbach and Andrew Laine, submitted for publication.) The SLF software opens this powerful image processing system for use with SLF data.

Section 2 summarizes the SLF as described by the DMA. This section gives some of the motivations and design considerations of the SLF. The format itself is given in brief. Section 3 outlines the structure of the relational database designed to store the SLF data online. Section 4 describes the SLF software developed at Washington University. Four programs have been written. Each program is described--what it does, how it works, and what options are available. The design of each is also outlined. Section 5 gives the data structures used in the programs and subroutines. Section 6 examines the library of subroutines written for programs using the SLF. The final section makes some brief observations about the SLF, the database, the programs, and the library.

2. THE STANDARD LINEAR FORMAT

Portability, flexibility, and efficiency were major considerations in the design of the SLF. The prime motivation was to provide a standard for many users, so increases in efficiency could not be made at the expense of portability or flexibility. In order to be useful, the SLF had to be efficient, so these demands had to be balanced.

Portability is assured by using the Federal Information and Processing Standards (FIPS) and American National Standards Institute (ANSI) standard tape and tape labels and the American Standard Code for Information Interchange (ASCII) character set. The physical block size allows whole word transfers on computers with all common word lengths.

The consistent use of SLF is necessary to insure portability. The SLF document provides several appendices detailing the implementation of specific products (e.g. DFAD). These appendices insure that the SLF maintains the integrity of these products and their portability in the new format.

The SLF is a flexible format. It specifies the structure of a Data Set Record consisting of many fields for descriptive information about the data set. It also allows an optional free-format Text Record. In the Data Set Record, SLF allows specification of different coordinate systems, numbers of dimensions, units and precision of measure, and many other descriptions of the data set. Most applications would not use many of these fields, but they provide the flexibility to handle a variety of applications, products, and data. The Data Set Record also contains a number of additional fields that are unused or reserved. These fileds are provided for special uses or later versions. These fields mean that the SLF has room to grow without making older versions incompatible.

The SLF also defines flexible records for feature data. The descriptive feature headers defined by SLF have no fixed length or format. Therefore, each application can specify the header size and format. The features are stored using chain-node encoding. This method stores line "segments" that define a point or linear feature or delineate an areal feature. Segments are held in separate records and can be shared by multiple features.

Chain-node encoding can be more efficient than simply storing the boundary of an areal feature as a polygon. A segment that separates two features and that consists of many points can be stored more efficiently using chain-node encoding. Polygonal storage would require these boundary points to be stored with each feature. The SLF allows each segment to be stored separately and allows each feature to refer to the segments. The chain-node method also avoids the overlaps and gaps that can occur when the boundary of a feature is specified without regard to its neighbors' boundaries. Appendix VI in the SLF document compares chain-node and polygonal structures.

It is not the purpose of this report to fully describe the SLF. A brief description should be enough for an understanding of the SLF software. More information about the SLF can be found in the DMA's publication.

The magnetic tape is FIPS and ANSI standard recorded at 6250 fpi GCR (preferred), 1600 fpi PE, or 800 fpi NRZI (not preferred). The tape file structure and tape label records conform to the FIPS and ANSI level one labels. The first label is a volume label. Each data set is enclosed by header and end-of-file labels. All data are stored as 7-bit ASCII characters in 8-bit bytes. Only single-reel volumes are defined.

The data set is divided into five logical records that are on the tape in the following order: the Data Set Identifier Record (DSI), the Segment Record (SEG), the Feature Record (FEA), and the Text Record (TXT) which is optional. Each logical record consists of one or more physical blocks of 1980 bytes, each byte an ASCII character. Each physical block begins with an eight-byte header consisting of the left-justified, block type (DSI, SEG, FEA, or TXT) and a right-justified, five-digit block sequence number (with blank fill). Each logical record's physical blocks are sequenced separately beginning with one (1) for the first block of each logical record. The remaining 1972 bytes of each physical block is allocated to the storage of the fields of the logical record. Data fields within a logical record may span more than one physical block, but a physical block may not contain data from different logical records. Any unfilled bytes at the end of physical record following the end of a logical record are filled with the ASCII delete character (octal 177). Figure 1 illustrates the organization of the blocks and records.

The Data Set Identifier record (DSI) consists of eight groups of fields. These groups, their fields, and the field lengths are given in brief below. Two of these eight groups are optional--the Data Set Registration Points Group (DSRG) and the Data Set Accuracy Group (DSAG). The fields of the Data Set Variable Field Address Group (DSVG) are used to indicate the the address of the first characters of the DSRG and DSAG if they are present. Subgroups of fields (such as the

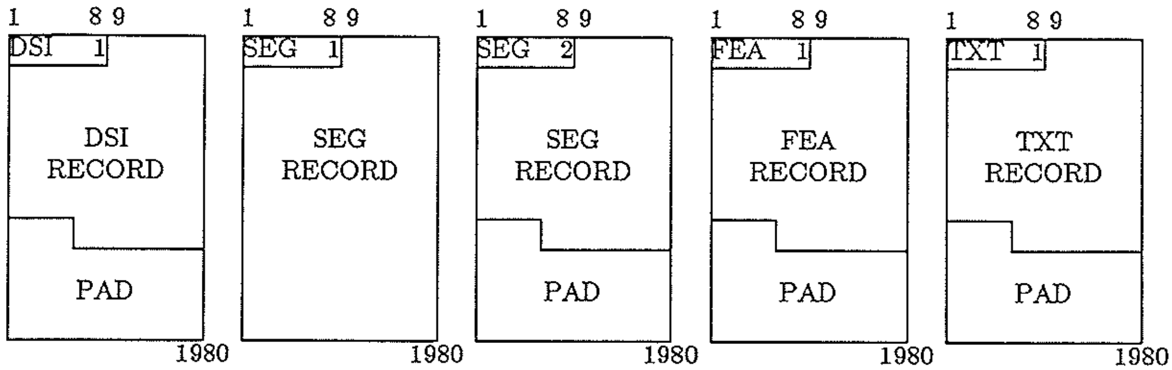


FIGURE 1. EXAMPLE OF PHYSICAL BLOCKS AND LOGICAL RECORDS

information about each of the registration points in the Registration Points Group) may be repeated any number of times. In these cases, the field immediately before the repeating fields indicates the number of occurrences of the subgroup.

A. Data Set Identification Group (DSIG)

1.	DSIG Header	4
2.	Product Type	5
3.	Data Set ID	20
4.	Edition	3
5.	Compilation Date	4
6.	Maintenance Date	4
7.	SLF Version Date	6
8.	FACS Version Date	6
9.	DSIG Reserve	28

B. Data Set Security Group (DSSG)

1.	DSSG Header	4
2.	Security Classification	1
3.	Security Release	2
4.	Downgrading/Reclassification Date	6
5.	Security Handling	21
6.	DSSG Reserve	40

C. Data Set Parameter Group (DSPG)

1.	DSPG Header	4
2.	Data Type	3
3.	Horizontal Units of Measure	3
4.	Horizontal Resolution Units	5
5.	Geodetic Datum	3
6.	Ellipsoid	3
7.	Vertical Units of Measure	3
8.	Vertical Resolution Units	5
9.	Vertical Reference System	4
10.	Sounding Datum	4
11.	Latitude of Origin	9
12.	Longitude of Origin	10
13.	X Coordinate of Origin	10
14.	Y Coordinate of Origin	10
15.	Z Coordinate of Origin	10

16.	Latitude of SW Corner	9
17.	Longitude of SW Corner	10
18.	Latitude of NE Corner	9
19.	Longitude of NE Corner	10
20.	Total Number of Features	6
21.	Number of Point Features	6
22.	Number of Linear Features	6
23.	Number of Areal Features	6
24.	Total Number of Segments	6
25.	DSPG Reserve	40

D. Data Set Map Group (DSMP)

1.	DSMP Header	4
2.	Projection	2
3.	Projection Parameter 1	10
4.	Projection Parameter 2	10
5.	Projection Parameter 3	10
6.	Projection Parameter 4	10
7.	Scale	9
8.	DSMP Reserve	40

E. Data Set History Group (DSHG)

1.	DSHG Header	4
2.	Edition Code	3
3.	Product Specification	15
4.	Specification Date	4
5.	Specification Amendment Number	3
6.	Producer	8
7.	Digitizing System	10
8.	Processing System	2
9.	Absolute Horizontal Accuracy	4
10.	Absolute Vertical Accuracy	4
11.	Relative Horizontal Accuracy	4
12.	Relative Vertical Accuracy	4
13.	Height Accuracy	4
14.	Data Generalization	1
15.	North Match/Merge Number	1
16.	East Match/Merge Number	1
17.	South Match/Merge Number	1
18.	West Match/Merge Number	1
19.	North Match/Merge Date	4
20.	East Match/Merge Date	4
21.	South Match/Merge Date	4
22.	West Match/Merge Date	4
23.	Date of Earliest Source	4
24.	Date of Latest Source	4
25.	Data Collection Code	1
26.	Data Collection Criteria	3
27.	DSHG Reserve	28

F. Data Set Variable Field Address Group (DSVG)

1.	DSVG Header	4
2.	Registration Points Address	5
3.	Accuracy Subset Address	5

4.	DSVG Reserve	40

G. Data Set Registration Points Group (DSRG) (optional)

1.	DSRG Header	4
2.	Number of Registration Points (Number of repetitions of the following seven fields)	3
a.	Point ID	6
b.	Latitude	9
c.	Longitude	10
d.	Elevation	8
e.	X-Coordinate	6
f.	Y-Coordinate	6
g.	Z-Coordinate	6

H. Data Set Accuracy Group (DSAG) (optional)

1.	DSAG Header	4
2.	Multiple Accuracy Outline Count (Number of repetitions of the following fields)	2
a.	Absolute Horizontal Accuracy	4
b.	Absolute Vertical Accuracy	4
c.	Relative Horizontal Accuracy	4
d.	Relative Vertical Accuracy	4
e.	Number of Coordinates (Number of repetitions of the following two fields)	2
i.	Latitude	9
ii.	Longitude	10

The chain-node encoding of features is implemented in two records. The Segment Record (SEG) specifies the line segments that separate features. Each boundary segment is encoded only once and the feature(s) refer to the segments. The Feature Count field of the Segment Record tells how many features the segment defines (in the case of point and linear features) or delineates (in the case of areal features). The Point Count specifies how many points make up the segment.

The Feature record (FEA) specifies the features. Each feature is described by a header and names the segments that delineate it. The Feature Header Block Count tells how many forty-character blocks there are. The format of these blocks is application dependent. The Segment Count specifies how many Direction of Segment and Segment ID pairs follow.

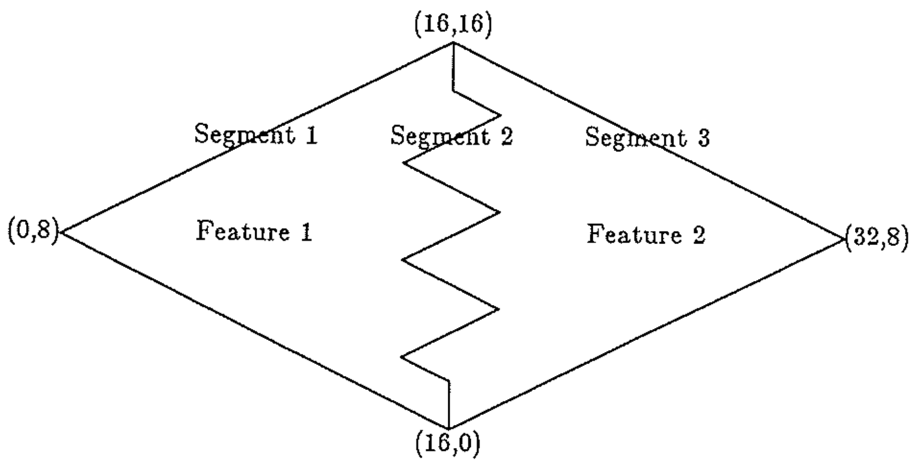
Segment Record

1.	Segment ID	6
2.	Feature Count (Number of repetitions of the following two fields)	2
a.	Feature ID	6
b.	Feature Orientation	1
3.	Point Count (Number of repetitions of the following fields)	5
a.	X-Value	6
b.	Y-Value	6
c.	Z-Value (optional)	6

Feature Record

1.	Feature ID	6
2.	Feature Type	1
3.	Feature Header Block Count (Number of repetitions of the following field)	2
	a. Feature Header Block	40
4.	Segment Count (Number of repetitions of the following two fields)	3
	a. Direction of Segment	1
	b. Segment ID	6

Figure 2 illustrates how these records store the chain-node encoding for a simple example. For a fuller discussion of chain-coding of segments refer to Appendix VI of the DMA specification.



SEGMENTS (ID, Feature Count, (ID, Orientation), Point Count, (X, Y))

1	1	1	R	3	16	16	0	8	16	0					
2	2	1	L	2	R	10	16	0	16	1	...	16	15	16	16
3	1	1	L	3	16	0	32	8	16	16					

FEATURES (ID, Type, Header Count, (Header), Segment Count, (Direction, ID))

1	A	1	xxx . . . xxx	2	F	1	F	2
2	A	1	xxx . . . xxx	2	R	2	F	3

FIGURE 2. EXAMPLE OF CHAIN-NODE ENCODING

The Text record (TXT) is an optional free-format block of up to 1968 characters.

Text Record (TXT) (optional)

1.	Character Count	4
2.	Text	1968

It is not within the scope of this report to detail the possible values for all of the fields of the SLF. The DMA report contains general information about each of the fields and detailed appendices about some of them. Appendix I defines codes and parameters for the Projection Group fields of the DSI Record. Appendix II lists the codes for Geodetic Datum, Sounding Datum, and Vertical Reference System. Appendix III lists the abbreviations for units of measure. Appendix IV gives the codes for the Grid System. Appendix V contains a short glossary.

3. THE DATABASE DESIGN

The SLF is an exchange and storage structure not well-suited to processing. The Ingres relational database system is designed to maintain databases online and provides many tools for manipulating the database and for processing the data in it. Ingres can enforce security, ensure the integrity of the data, synchronize access, and protect the data against machine crashes. It supports a data manipulation language named QUEL (QUEry Language). A form of this language called EQUEL (Embedded QUEry Language) can be embedded in programs written in the high-level programming language C.

Our first task was to specify the structure or scheme of the Ingres database to hold the SLF data. We decided that this structure should closely parallel the structure of the SLF since it must hold the same data. Therefore, all of attributes of the relations in the Ingres database consist of ASCII character strings, just as the fields in the SLF are all character strings. Most of the attributes in the database correspond directly to a SLF field, have the same length, and are named with abbreviated forms of the SLF field name.

The SLF DSI Record is divided into eleven (11) relations. Eight of the relations correspond to the Data Set Groups defined in SLF. These relations have the same names as the SLF abbreviations: *dsig, dssg, dspg, dsmp, dshg, dsvg, dsrg,* and *dsag.* In addition to these, three relations have been added to store the repeating subgroups: the registration points (*dsrg_pts*), the multiple accuracy outlines (*maos*), and the coordinates of the multiple accuracy outlines (*mao_coord*). The data set group headers have been omitted because they are redundant. Some attributes have been added to facilitate access. All of the relations have an attribute to identify the data set (*ds_dssg, ds_dspg, ds_dsmp, ds_dshg,* and so on). The *dsrg_pts* relation has a *dsrg_order* attribute to distinguish each registration point. The *maos* relation has a *mao_id* attribute for the same purpose. The *mao_coord* relation has a *mao_coord_id* attribute to identify the multiple accuracy outline to which the coordinate belongs and a *coord_id* attribute to order the coordinate in the outline.

The database relations for the DSI Record of the SLF are defined as follows.

```
create dsig (
        prod_type = c5,
        ds_id = c20,
        ds_ed = c3,
        comp_date = c4,
        maint_date = c4,
        slf_ver = c6,
        facs_ver = c6,
        dsig_rsv = c28
)

create dssg (
        ds_dssg = c20,
        sec_class = c1,
        sec_rel = c2,
        down_date = c6,
        handling = c21,
        dssg_rsv = c40
)

create dspg (
        ds_dspg = c20,
        data_type = c3,
        hor_units = c3,
        hor_res = c5,
```

```
                geod_data = c3,
                ellipsoid = c3,
                ver_units = c3,
                ver_res = c5,
                ver_ref = c4,
                sounding = c4,
                lat_orig = c9,
                long_orig = c10,
                x_orig = c10,
                y_orig = c10,
                z_orig = c10,
                lat_sw = c9,
                long_sw = c10,
                lat_ne = c9,
                long_ne = c10,
                n_fea = c6,
                n_pt_fea = c6,
                n_ln_fea = c6,
                n_ar_fea = c6,
                n_seg = c6,
                dspg_rsv = c40
)

create dsmp (
                ds_dsmp = c20,
                prjctn = c2,
                parm1 = c10,
                parm2 = c10,
                parm3 = c10,
                parm4 = c10,
                scale = c9,
                dsmp_rsv = c40
)

create dshg (
                ds_dshg = c20,
                dshg_ed = c3,
                prod_spec = c15,
                spec_date = c4,
                amend_no = c3,
                producer = c8,
                dig_sys = c10,
                proc_sys = c10,
                grid_sys = c2,
                abs_h_acc = c4,
                abs_v_acc = c4,
                rel_h_acc = c4,
                rel_v_acc = c4,
                ht_acc = c4,
                data_gen = c1,
                n_mm_no = c1,
                e_mm_no = c1,
                s_mm_no = c1,
                w_mm_no = c1,
```

```
                n_mm_date = c4,
                e_mm_date = c4,
                s_mm_date = c4,
                w_mm_date = c4,
                early_src = c4,
                late_src = c4,
                coll_code = c1,
                coll_crit = c3,
                dshg_rsv = c28
)

create dsvg (
                ds_dsvg = c20,
                reg_pt_add = c5,
                acc_addr = c5,
                dsvg_rsv = c40
)

create dsrg (
                ds_dsrg = c20,
                no_reg_pt = c3
)

create dsrg_pts (
                ds_dsrg_pts = c20,
                dsrg_ord = c3,
                pt_id = c6,
                lat = c9,
                lng = c10,
                elev = c8,
                x_coord = c6,
                y_coord = c6,
                z_coord = c6
)

create dsag (
                ds_dsag = c20,
                no_mao = c2
)

create maos (
                ds_maos = c20,
                mao_id = c2,
                mao_abs_h = c4,
                mao_abs_v = c4,
                mao_rel_h = c4,
                mao_rel_v = c4,
                no_coord = c2
)

create mao_coord (
                ds_mao_coord = c20,
                mao_coord_id = c2,
                coord_id = c2,
```

```
        mao_lat = c9,
        mao_long = c10
)
```

Three feature and three segment relations are used to store the data from the SLF Segment and Feature Records. In addition to a segment (*seg*) relation and a feature (*fea*) relation, relations are defined for the repeating groups of fields in the Segment and Feature Records. Each Segment Record contains a reference to each of the features it defines or delineates. These fields, the Feature ID and the Feature Orientation, are repeated for each feature of the segment. In the database, they are stored in the features of the segment (*fos*) relation. Likewise, the coordinates of points in the segment are listed in the Segment Record. They are recorded in the points of the segment (*pts*) relation. Both these relations also contain an attribute for the segment ID. The Feature Header blocks are stored in a separate relation (*fhd*). The Feature Record names all of its segments in repeated fields specifying the Segment Direction and the Segment ID. These fields are recorded in the segments of the feature (*sof*) relation. Both of these feature relations contain an attribute for the Feature ID. All of these segment and feature relations contain an attribute to order the tuples.

These records tend to be very long, with perhaps hundreds of thousands of segments or features and their constituent points. The Ingres system is not lightening fast so this would definitely present a problem if the segment and feature relations were to contain more than one data set. The solution to this problem was to define separate segment and feature relations for each data set. Each data set has six segment and feature relations whose names are formed by concatenating the three-letter relation type with the data set ID. For example, a data set with the ID *WUTEST1* would have relations named *SEGWUTEST1*, *FOSWUTEST1*, *PTSWUTEST1*, *FEAWUTEST1*, *FHDWUTEST1*, and *SOFWUTEST1*.

The six relations given below would be defined to hold the SLF Segment and Feature Records for each data set.

```
create segrel (
        seg_id = c6,
        fea_cnt = c2,
        pt_cnt = c5
)

create fosrel (
        seg_no = c6,
        ord_fos = c2,
        fos_id = c6,
        fea_orient = c1
)

create ptsrel (
        sop_no = c6,
        ord_pt = c5,
        x_val = c6,
        y_val = c6,
        z_val = c6
)

create fearel (
        fea_id = c6,
        fea_type = c1,
        fea_hdr_cnt = c2,
        seg_cnt = c3
```

```
)

create fhdrel (
            hdr_fea_id = c6,
            ord_hdr = c2,
            block = c40
)

create sofrel (
            fea_no = c6,
            ord_sof = c3,
            direction = c1,
            sof_id = c6
)
```

The limit on the size of a field in Ingres is 255 characters, so the SLF Text Record has to be broken up into blocks. The character count is stored in an attribute of all of the tuples. This is a small redundancy. Other attributes are used to order the blocks of the text and to identify the data set. The relation is created with the following Ingres command.

```
create txt (
            ds_txt = c20,
            txt_cnt = c4,
            ord_txt = c1,
            text = c255
)
```

In order to speed processing, these relations are modified for indexed-sequential access. This is only done for the relations for the segment and feature data, because they are the only relations whose size presents a problem. The segment relation is indexed on the segment ID. The features of the segment relation is indexed on the segment ID and the order of the tuple. The points of the segment relation is indexed on the segment ID and the order of the tuple. The feature relation is indexed on the feature ID. The feature header relation is indexed on the feature ID and the order of the tuple. The segments of the feature relation is indexed on the feature ID and the order of the tuple.

4. THE APPLICATION PROGRAMS

We wanted software that would allow us to input, process, and output SLF data. We wanted software that would enable us to receive SLF tapes and add the data to our database. We wanted to be able to display the data in the database. We wanted to be able to alter the data in the database and to be able to create new data. Finally, we wanted to be able to generate SLF tapes. Our need was not for a production system, but for a smaller-scale research tool. Therefore the software does not contain as many error checks as would be desirable in a production environment. We do not anticipate handling a large number of data sets and our results are usually well-scrutinized. A production system would show a greater concern for speed than our needs dictate. Little emphasis has been placed on the need for optimizing the algorithms. The Ingres relational database system has been used because it is available and familiar. A hierarchical or network database system might be better suited for the SLF.

The application programs described in the section are all written in the high-level programming language C. All of them contain embedded queries in EQUEL. Manual pages are available for all four commands. These pages are written in the standard UNIX† manual page form. They

† UNIX is a trademark of Bell Laboratories.

provide the user with the syntax of the command and a general description of the program and its use. These manual pages are included in section one of the Washington University/Defense Mapping Agency (WUDMA) Image Processing Manual.

Our first program was written to receive a tape in the SLF and add the data to our database. This program, named *slftoingres*, relies on several sets of subroutines to carry out its primary tasks. The user must specify the SLF file name and the Ingres database name. The database must already exist. The design hierarchy of *slftoingres* is pictured in figure 3.

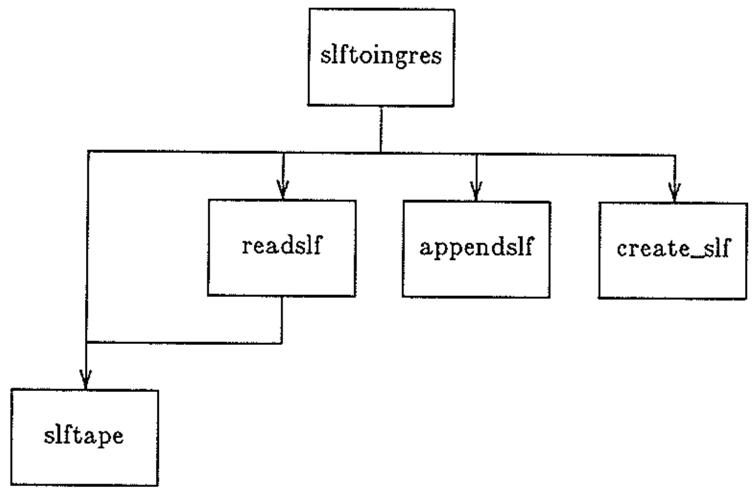


FIGURE 3. DESIGN OF SLFTOINGRES

The tape should first be read into a file (using *dd(1)* for example). *Slftoingres* reads the file and puts the information into the database. The *slftape(3)* subroutines are used for the low-level input from the SLF file. These functions are called directly by *slftoingres* as well as indirectly using the *readslf(3)* subroutines. The *readslf(3)* subroutines read specific groups of fields corresponding to the DSI groups of the subgroups of the Segment or Feature Records. These subroutines verify the group header, but also rely on the *slftape(3)* functions to perform low-level input.

Slftoingres alternates between calls to the *readslf(3)* subroutines and the *appendslf(3)* subroutines. The *appendslf(3)* subroutines add a group of SLF fields to the database by by appending a new tuple to the relation. *Slftoingres* makes a call to a *readslf(3)* subroutine and a call to a *appendslf(3)* subroutine for each group of SLF fields (and database attributes). As explained in the section describing the database, each data set has six of its own segment and feature relations. *Slftoingres* calls on *create_rel(3)* to create these relations.

The command that performs the inverse task, creating a SLF file from data in the database, is named *ingrestoslf*. The user must specify the Ingres database name and the ID of the data set. The data set ID should be specified in capital letters since the SLF data is in capital letters. The output is directed to the standard output. The design hierarchy of *ingrestoslf* is illustrated in figure 4. This program must retrieve data from the database and write it in the SLF to a file. *Ingrestoslf* uses two sets of subroutines to accomplish these tasks. The *retrieveslf(3)* subroutines retrieve groups of fields from the database, keying on the data set ID. Because of the structure of the SLF, the segment and feature data is retrieved directly by the program. The reasons for this are explained in detail in the *retrieveslf(3)* manual page.

The data is written to a file in SLF using the *printslf(3)* subroutines. This group of subroutines builds the block structures, complete with block headers and delete character (ASCII 177) fill, defined in the SLF. A parameter to these routines is available to override construction of the

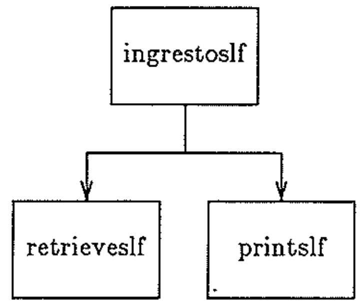


FIGURE 4. DESIGN OF INGRESTOSLF

SLF blocks. This option allows the user to dump the data in a format suitable for display (on the terminal or printer) complete with brief descriptions of the fields.

The command *graphfea* displays features from the database. The user must specify the Ingres database name and the ID of the data set. Output can be generated for several display devices as well as in a generic form. Command options exist to display all of the features of the data set (default is to prompt the user for feature ID's), to erase the screen before displaying the features (default is to display the features over what is there), and to set the logical feature space (default is (0,0) to (511,511)).

Graphfea relies on two sets of subroutine to do its work--get the feature data and display it. Its design hierarchy is given in figure 5. The *retrieveslf(3)* subroutines have already been described. *Graphfea*, like *ingrestoslf* and for the same reasons, retrieves the segment and feature data directly. The subroutines of the UNIX *plot(3)* library are called to actually plot the data. This means that *graphfea* can be immediately used with any device that has a set of plot subroutines written for it. Several new *plot(3)* "flavors" have been written at Washington University. We can display features on the DeAnza IP8500, Vectrix 385 graphics devices, and the Ergo and Graphon graphics terminals. Of course, files of the *plot(5)* format can also be created. Refer to the UNIX documentation for more information about *plot*.

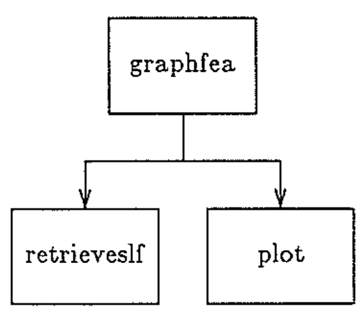


FIGURE 5. DESIGN OF GRAPHFEA

The fourth command, *access_slf*, allows a user to update selected fields of the database. The user must specify the Ingres database name and the data set ID. If a data set of that name does not already exist in the database, then a default data set is created. The existing data set or the default data set, if one is created, can then be altered by the user. The program is menu-driven and provides prompts that include the current data.

Using the program, the user can change data in the Data Set Identification Group (except the data set ID), Security Group, Parameter Group, Map Projection Group, and History Group, as well as in the Text Record. In the current version, changes can not be made in the Variable Field Address Group, the Registration Points Group, or the Accuracy Group. The user may also append features to the data set (but not alter existing segments or features). At this time, only additional point features are allowed and the user may specify only the point's coordinates and not its feature ID, segment ID, or header. The segment and feature ID's are assigned unused numbers ID's and a default header is used.

Access_slf uses many subroutines. Its design hierarchy is pictured in figure 6. If the data set ID specified by the user identifies one of the data sets, then the *retrieveslf(3)* subroutines are used to get the data from the existing database. If no such data set is found, then the *defaultslf(3)*, *appendslf(3)*, and *create_rel(3)* subroutines are used to create a default data set. The *appendslf(3)* and *create_rel(3)* subroutines have already been described. The *default(3)* subroutines are used to give default values to groups of the SLF fields before they are appended to the relations. These values are given in the *defaultslf(3)* manual page in section 3 of the manual.

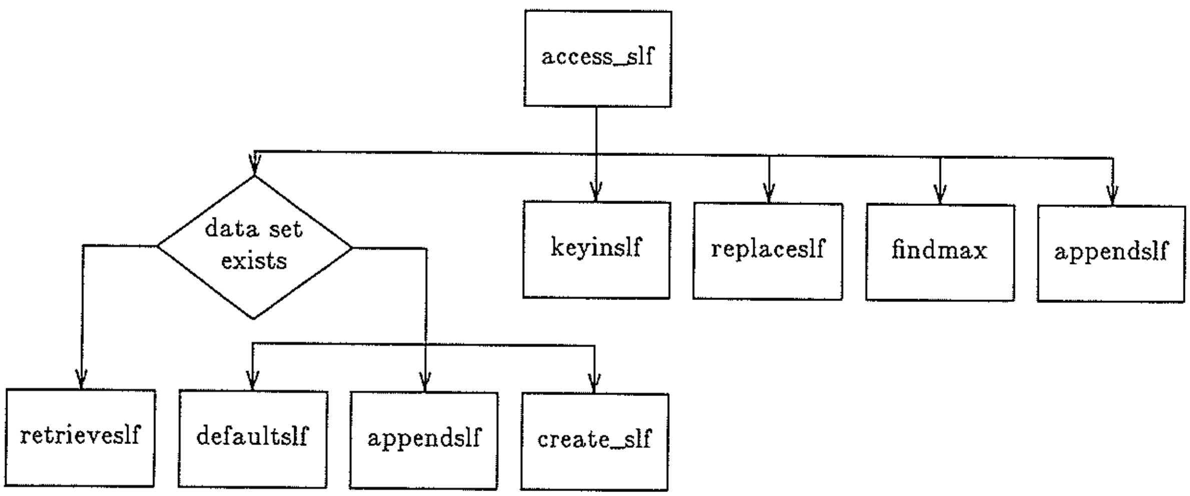


FIGURE 6. DESIGN OF ACCESS_SLF

The Data Set Identifier record (DSI) consists of eight groups

The *keyinslf(3)* subroutines are called upon to allow the user to specify changes. The *keyin(3)* subroutines print a field description and the current contents of the field and ask the user to specify a new value or indicate that the current data is to be left unchanged. Once a group of fields has been "keyed in," the *replaceslf(3)* subroutines are used to replace the old tuple in the database with the new. These subroutines use the data set ID to key the tuples to be replaced.

Point features can be added to a data set. *Access_slf* uses the *findmax(3)* subroutines to select segment and feature ID's greater than any existing ID's and the *appendslf(3)* subroutines to add these features (with all their data) to the database. The user specifies the features coordinates and the *defaultslf(3)* subroutines are used to assign all of the other attributes.

These commands have been incorporated into the WUDMA Image Processing Software System. The source code, manual pages, and executable modules have been integrated into this UNIX subsystem. The programs are maintained under the UNIX *make* facility so that the versions can be easily kept up to date and regenerating the executable modules is simple.

5. THE SLF DATA STRUCTURES

Data structures in the high-level programming language C are used in the application programs and subroutines. These structures closely correspond to the SLF described by DMA. These structures have fields of the DMA standard that are not included in the database (the data set group headers) and attributes of the database relations that are not defined in SLF (the identifying and ordering attributes). The field names are the same as the database attribute names. This means that the Equel programs must use the non-referencing operator (#) for all database attribute references, but having the names the same makes them easier to remember. The length of the structure fields is one character longer than the fields of the SLF and the attributes of the database because the strings used with Equel require an extra character for the null-terminator.

Eighteen structures are defined--one for each of the relations in the database. The data set and text structures have the same names, but in capital letters, as the corresponding relations: DSIG, DSSG, DSPG, DSMP, DSHG, DSVG, DSRG, DSRG_PTS, DSAG, MAOS, MAO_COORD, and TXT. The segment and feature structures are named with the three-letter type in capitals: SEG, FOS, PTS, FEA, FHD, and SOF. These structure definitions are contained in the header files *slf.q.h* and *slf.c.h* and can be found in the INCLUDE subdirectory. *Slf.c.h* is generated by the Equel preprocessor from *slf.q.h*. These structures are described more fully in *slfstruct(5)*. Section 5 of the manual describes special formats and files.

6. THE SLF LIBRARY

A library of subroutines facilitates access to SLF files and the database. There are seven main sets of subroutines and several miscellaneous routines. The main sets provide the ability to *append* tuples containing SLF fields to the database, create *default* groups of SLF fields, *keyin* groups of SLF fields, *print* groups of SLF fields, *read* sets of fields from a SLF file, *replace* tuples in a database, and *retrieve* tuples from a SLF database. The other functions provide additional capabilities to the application programmer. Manual pages are provided for all of the functions are included in section 3 of the manual. A summary of the SLF library is given in *slf(3)*.

The *appendslf* routines add the data in a SLF structure to a relation in the database by appending a tuple. Their is a subroutine for each type of relation. The subroutine is named by concatenating *append* with the relationtype. For example, *appendtxt* is used to append data to the *txt* relation. The database must already be opened under Equel. Because these routines contain embedded queries in Equel, they must be linked to the Equel library using -lq with the compiler.

Most of the *appendslf* functions require only the address of the SLF structure as a parameter. The functions *appendseg(3)*, *appendfos(3)*, *appendpts(3)*, *appendfea(3)*, *appendfhd(3)*, and *appendsof(3)*, require an additional parameter to identify the relation. (Recall that each data set has its own segment and feature relations.)

The *defaultslf* routines set default values in SLF structures. The default values follow the guidelines for DFAD given in Appendix XII of the Defense Mapping Agency's *Standard Linear Format for Digital Cartographic Feature Data* (Draft 2nd ed., 18 March 1985) and the Defense Mapping Agency's *Product Specifications for Digital Landmass System (DLMS) Data Base* (2nd ed., April 1983).

Most of the fields are blank or zero. The data set ID must be passed as well as the structure address. The group headers are given the specified values. (For example, dssgrec->ds_dssg is set to "DSSG".) The map group and variable field group contain only null fields. No functions exist for the the registration points group, and accuracy subset group because they are optional. The default text record has a length of zero. The non-null fields of the remaining fields of the Data Set Identification Record are:

dsigrec->prod_type:	"DFAD2"
dsigrec->ds_ed:	"1"

dsigrec->maint_date:	"0000"
dsigrec->slf_ver:	"850315"
dsigrec->facs_ver:	"000000"
dssgrec->sec_class:	"U"
dssgrec->handling:	"DISTRIBUTION LIMITED"
dspgrec->data_type:	"GEO"
dspgrec->hor_units:	"SEC"
dspgrec->hor_res:	"0.100"
dspgrec->geod_data:	"WGC"
dspgrec->ellipsoid:	"WGC"
dspgrec->lat_orig:	"00000000S"
dspgrec->long_orig:	"000000000W"
dspgrec->lat_sw:	"00000000S"
dspgrec->long_sw:	"000000000W"
dspgrec->lat_ne:	"00000000N"
dspgrec->long_ne:	"000000000E"
dspgrec->n_fea:	"0"
dspgrec->n_pt_fea:	"0"
dspgrec->n_ln_fea:	"0"
dspgrec->n_ar_fea:	"0"
dspgrec->n_seg:	"0"
dshgrec->prod_spec:	"SPEXDLMS2"
dshgrec->spec_date:	"8304"
dshgrec->amend_no:	"000"
dshgrec->producer:	"USWASHU"

Default features can be created using *defaultseg*, *defaultfos*, *defaultpts*, *defaultfea*, *defaultfhd*, and *defaultsof*. The feature ID and segment ID, as well as the relation name and structure address, must be parameters to these functions. The default feature is a point with coordinates (0,0,0). The default field values are:

segrec->fea_cnt:	"1"
segrec->pt_cnt:	"1"
fosrec->ord_fos:	"1"
fosrec->fea_orient:	"C"
ptsrec->ord_pt:	"1"
ptsrec->x_val:	"0"
ptsrec->y_val:	"0"
ptsrec->z_val:	"0"
fearec->fea_type:	"P"
fearec->fea_hdr_cnt:	"1"
fearec->seg_cnt:	"1"
fhdrec->ord_hdr:	"1"
fhdrec->block:	All but the first character of the feature ID followed by "010 0 0 0 09013603 0 0 "
sofrec->ord_sof:	"1"

sofrec->direction: "F"

The *keyin* subroutines allow the user to specify data to be placed in the fields of SLF structures. The function *keyin* prompts the user with a field description and the field's current value. The user can change the value or leave it unchanged. *Keyin* checks the length of the new value, and if necessary justifies the value and blank fills. Usually, the function *keyin* is used indirectly to perform these tasks. The user typically calls one of the subroutines that handles an entire structure such as *appenddssg* and that routine would call *keyin* for each of the fields of the DSSG structure. The only parameter necessary for the routines to key in a structure is the structures address. *Keyin* requires the prompt, string address, length of the field, and justification. The justification may be "F" meaning the field must be filled, "L" for left justification, or "R" for right justification. The justifications are not specified for all fields in the Defense Mapping Agency's *Standard Linear Format for Digital Cartographic Feature Data* (Draft 2nd ed., 18 March 1985), so in some cases the choices are speculative. There are no subroutines to key in the segment and feature structures.

The *printslf* routines are used to print SLF structures to the standard output file. *Cprint(3)* is used to create the blocks (size 1980 characters) defined in the SLF. This subroutine requires the address of the string to be printed, the number of characters printed in the current block, the block type, and the block sequence number. There is an additional prarameter that can be used to override the block formatting. If this parameter is set, then the string is printed followed by an end-of-line.

There are individual subroutines to print each of the SLF Data Set and Text structures. For example, *printdssg* would be used to print the DSSG structure and would call *cprint* to print each of its fields. These routines have the same parameters as *cprint*. If the option for overriding the block formatting is set, then a field description is given with each field value.

The functions *printseg(3)*, *printfos(3)*, *printpts(3)*, *printfea(3)*, *printfhd(3)*, and *printsof(3)* are somewhat different. Because the slf specification mandates that the segment and field records are not contiguous, generation of slf output requires that portions of the segment and field records be delayed. This and the fact that there are usually many tuples in these relations (indicating the need for speed) and the fact that there are few fields in the records (which means printing the fields individually is relatively easy) led to the decision in implementing *ingrestoslf(1)* to print these records directly rather than via calls to these functions. The functions remain as a debugging aid, but are not used by any of the existing application programs. If they were to be used similarly to the other routines, it would be desirable to add the *p* parameter and use *cprint(3)*.

The *readslf* routines read groups of fields from a SLF file. They verify the group header (if appropriate), but rely on the *slftape* functions to do the low-level input and verify the record type. The only parameter is the address of the structure. There are functions for all of the eighteen field groups corresponding to database relations.

Due to the structure of the SLF, the field for number of points in a segment is not contiguous with the rest of the data for the segment record and the field for the number of segments in a feature is not contiguous with the rest of the feature record. For this reason, the read for these two fields is not accomplished during the *readseg* and *readfea* calls and must be done after reading the intervening data.

The *replaceslf* functions replace tuples in a database. The address of the structure containing the new data and the data set ID to be used as the replacement key are required parameters. If there are multiple records with the same data set ID, then all such records are replaced. If no record with the proper key is found, then no change is made to the relation. Only subroutines to replace tuples in the *dsig*, *dssg*, *dspg*, *dsmp*, *dshg*, and *txt* are provided. The functions return an integer equal to the number of records replaced. The database must already be opened Equel and because these routines contain embedded queries they must be linked to the Equel library (-lq).

The *retrieveslf* functions key on the data set ID requested to retrieve a data from the database. The data set ID and the address of the structure to return the data are parameters. Most of the functions use only the data set ID to key the relation, but *retrievedsrg_pts* also uses the order of the registration point, *retrievemaos* also uses the order of the multiple accuracy outline, and *retrievemao_coord* also uses the order of the multiple accuracy outline and the order of the coordinate. The functions set the value of the structure to that of the last tuple found in the relation. If no record with the key(s) is found, the record parameter is unchanged. The functions return an integer equal to the number of records with the key(s) in the relation. The database must already be opened under Equel and because these routines contain embedded queries they must be linked to the Equel library (-lq).

The functions to retrieve feature and segment data are not currently used by any of the application programs. These relations tend to be very large, but easy to access because they have few fields. Nested queries are the most natural way to access this data, but one limitation of our version of Equel is that it does not allow nested queries. Therefore these calls can not be nested since they contain queries.

Several subroutines provide the low-level interface to read the block structure of the SLF. There are several variables (hereafter referred to as the static tape variables) that are global to these four routines. They maintain information about the SLF file being read. *Initializetape* opens a file, initializes the static tape variables about the file, and reads the first block. This function should always be called first in order to set-up things for the other functions. It requires only the file name as a parameter. *Readblock* is an internal routine used to fetch new blocks and maintain the static tape variables. It needs no *readtape* compares the type of record requested with the type of the block which is being read. If they are different an error occurs. Otherwise, it copies a number of bytes from the current block into a buffer. This string is terminated with the null character. If during the process of reading, a block is exhausted, another is read (using *readblock*) and the *readtape* process continues. *Readtape* requires parameters specifying the type of record desired, the number of bytes to be read, and the buffer address. *Unread(3)* is a kludge to "unread" what may have been mistakenly read from a block. There are better ways to look ahead.

Findmaxfea(3) examines a feature relation and returns the largest feature ID. *Findmaxseg(3)* examines a segment relation and returns the larges segment ID. The relation name is the only parameter. These subroutines assume ID's must be greater than zero. The database must already be opened under Equel. Because these routines contain embedded queries in Equel, they must be linked to the Equel library (-lq).

Create_rel(3) creates the *seg, fos, pts, fea, fhd,* and *sof* relations for a given data set (whose ID is given as a parameter). These relations are maintained separately for each data set because they can be quite large. The relations are named by the string created by appending the data set ID to the relation type. These relations would be modified to indexed sequential access mode using the keys described above. The database must already be opened under Equel. Because these routines contain embedded queries in Equel, they must be linked to the Equel library (-lq).

These functions should provide useful tools for future application programmers. The sources and manual pages are located in the WUDMA Image Processing Software. The archive is located in the *lib* subdirectory and can be linked using -lslf with the compiler.

7. CONCLUSIONS

The SLF should provide useful standard for transmitting feature data. The SLF is designed to meet DMA's needs. It is bulkier than we would need, it is limited to two or three dimensional data, and features must be specified using chain-node encoding. We can discard unneeded data and we usually work with two and three dimensional data. We have not solved the problems involved with automated generation of the chain-node encoding. This is an area that we hope to work on soon.

The database might work better for some applications if some of the attributes were stored as integers rather than strings. This would involve more processing when creating or dumping the data, but would save time in some computations. Our experience at this time is insufficient to gauge our use of the database and the balance of the tradeoffs. The modular design of the SLF software should make modifications to the programs relatively simple. Such changes would involve minor changes in some of the subroutines.

Currently only point features can be added under program control. Also, these features can only be given a default header. Because this is not a production site and because the data that feature headers can contain are so diverse, tools to create feature headers will probably be written as needed. The ability to create linear and areal feature data should be a high priority in improving this package.

Our research goal is to develop a semi-automated photograph interpretation system. We hope that our system will automate much of what must now be done by hand. If we are to be successful, it is important that we understand the DMA's needs. Communication is an important condition for success. Using the SLF has already helped us better understand the task of the photo-interpreter and the kind of products that are generated. Using the SLF, our results should be immediately accessible to the DMA. We believe that the ability to easily exchange data will benefit us in automating photo-interpretation.

APPENDIX A
MANUAL PAGES

NAME

access_slf — updates a slf database under ingres

SYNOPSIS

access_slf <*database*> <*DATASET*>

DESCRIPTION

Access_slf allows a user to update selected fields of an slf database maintained under the Ingres relational database system. This database is modeled on the Defense Mapping Agency's *Standard Linear Format for Digital Cartographic Feature Data* (Draft 2nd ed., 18 March 1985). The user specifies the Ingres database name as *database* and the data set ID as *DATASET* (in capital letters). If a data set of that name does not already exist in the database, then a default data set is created (see *default(8)*). This default data set can then be altered by the user. If the data set already exists, any subsequent changes are made on the existing database.

The program is menu-driven and provides prompts for the user. The fields in the data set identification group (except the data set ID), security group, parameter group, map projection group, and history group may be specified by the user. For example, a user can change the maintenance date. The user may also append features to the data set. At this time, only additional point features are allowed and the user may specify only the point's coordinates (and not its feature ID, segment ID, or header).

SEE ALSO

graphfea(1), ingrestoslf(1), slftoingres(1), slf(3), slf(5)

AUTHOR

Steve Reichenbach 2/86

NAME

graphfea – graphs features from an ingres database

SYNOPSIS

graphfea <-T*termtype*> [*options*] <*database*> <*DATASET*>

DESCRIPTION

Graphfea graphs features in the Ingres database named *database* and the relation **fea***DATASET*. This database is based on the Defense Mapping Agency's *Standard Linear Format for Digital Cartographic Feature Data* (Draft 2nd ed., 18 March 1985). The user must specify the database name and the data set ID in the command line. The data set ID should be in capital letters. The user must also specify the terminal type for display. Current choices are:

-Tdz DeAnza.

-Tvx Vectrix.

-Tgr Graphon.

-Tplot *Plot(5)* output.

If any type but the DeAnza is specified, then the output from *graphfea* is produced on standard output.

Options:

-a Display all features. Default is to prompt for individual features where the user must give the feature ID.

-e Erase previous graphics before display.

-s *xmin ymin xmax ymax*
Defines the space for the features.

SEE ALSO

access_slf(1), ingrestoslf(1), slftoingres(1), slf(3), slf(5)

AUTHOR

Steve Reichenbach 8/85

NAME

ingrestoslf — writes a slf file from an ingres database

SYNOPSIS

ingrestoslf [-p] <*database*> <*DATASET*>

DESCRIPTION

Ingrestoslf writes the data for a particular data set from a database maintained under the Ingres relational database system to the standard output file. *Database* specifies the Ingres database name. *DATASET*, the data set ID, is used as the key and should be specified in capital letters. The default is to create output formated according to the Defense Mapping Agency's *Standard Linear Format for Digital Cartographic Feature Data* (Draft 2nd ed., 18 March 1985). If the -p option is specified, output suitable for viewing (and accompanied with explanatory notes for each field) is generated.

SEE ALSO

access_slf(1), graphfea(1), slftoingres(1), slf(3), slf(5)

AUTHOR

Steve Reichenbach 12/85

NAME

slftoingres – reads a slf file into an ingres database

SYNOPSIS

slftoingres <*filename*> <*database*>

DESCRIPTION

Slftoingres reads the data from a file of the form of the Defense Mapping Agency's *Standard Linear Format for Digital Cartographic Feature Data* (Draft 2nd ed., 18 March 1985) and puts it into a database maintained under the Ingres relational database system. *Filename* specifies the location of the slf file. *Database* specifies the Ingres database name.

SEE ALSO

access_slf(1), graphfea(1), ingrestoslf(1), slf(3), slf(5)

AUTHOR

Steve Reichenbach 8/85

NAME

slf – introduction to the slf library functions

DESCRIPTION

This section of the manual describes the slf library functions. These functions are provided to facilitate access to slf format files and slf databases maintained under the Ingres relational database system. In addition to several miscellaneous routines, there are seven groups of files that manipulate slf files, slf databases, and the slf records described in slf(5). These groups provide the ability to *append* records to a slf database, create *default* slf records, *keyin* slf records, *print* slf records, *read* records from a slf file, *replace* records in a slf database, and *retrieve* records from a slf database. The other functions provide additional capabilities to the application programmer. Manual pages are provided for all of the functions.

SEE ALSO

access_slf(1), graphfea(1), slftoingres(1), slf(3), slf(5)

FILES

/usr/lib/libslf.a

SEE ALSO

access_slf(1), graphfea(1), ingrestoslf(1), slftoingres(1)

LIST OF FUNCTIONS

Name	*Appears on Page*	*Description*
appenddsag	appendslf.3	append a dsag record to a slf relation
appenddshg	appendslf.3	append a dshg record to a slf relation
appenddsig	appendslf.3	append a dsig record to a slf relation
appenddsmp	appendslf.3	append a dsmp record to a slf relation
appenddspg	appendslf.3	append a dspg record to a slf relation
appenddsrg	appendslf.3	append a dsrg record to a slf relation
appenddsrg_pts	appendslf.3	append a dsrg_pts record to a slf relation
appenddssg	appendslf.3	append a dssg record to a slf relation
appenddsvg	appendslf.3	append a dsvg record to a slf relation
appendfea	appendslf.3	append a fea record to a slf relation
appendfhd	appendslf.3	append a fhd record to a slf relation
appendfos	appendslf.3	append a fos record to a slf relation
appendmao_coord	appendslf.3	append a mao_coord record to a slf relation
appendmaos	appendslf.3	append a maos record to a slf relation
appendpts	appendslf.3	append a pts record to a slf relation
appendseg	appendslf.3	append a seg record to a slf relation
appendsof	appendslf.3	append a sof record to a slf relation
appendtxt	appendslf.3	append a txt record to a slf relation
cprint	printslf.3	print a slf record field
create_rel	create_rel.3	create slf feature and segment relations
defaultdshg	defaultslf.3	return a default dshg record
defaultdsig	defaultslf.3	return a default dsig record
defaultdsmp	defaultslf.3	return a default dsmp record
defaultdspg	defaultslf.3	return a default dspg record
defaultdssg	defaultslf.3	return a default dssg record
defaultdsvg	defaultslf.3	return a default dsvg record
defaultfea	defaultslf.3	return a default fea record
defaultfhd	defaultslf.3	return a default fhd record
defaultfos	defaultslf.3	return a default fos record
defaultpts	defaultslf.3	return a default pts record
defaultseg	defaultslf.3	return a default seg record

defaultsof	defaultslf.3	return a default sof record
defaulttxt	defaultslf.3	return a default txt record
findmaxfea	findmax.3	find maximum feature id
findmaxseg	findmax.3	find maximum segment id
initializetape	slftape.3	initialize slf file for reading
keyin	keyinslf.3	key in a slf record field
keyindsag	keyinslf.3	key in a dsag record
keyindshg	keyinslf.3	key in a dshg record
keyindsig	keyinslf.3	key in a dsig record
keyindsmp	keyinslf.3	key in a dsmp record
keyindspg	keyinslf.3	key in a dspg record
keyindsrg	keyinslf.3	key in a dsrg record
keyindsrg_pts	keyinslf.3	key in a dsrg_pts record
keyindssg	keyinslf.3	key in a dssg record
keyindsvg	keyinslf.3	key in a dsvg record
keyinmao_coord	keyinslf.3	key in a mao_coord record
keyinmaos	keyinslf.3	key in a maos record
keyintxt	keyinslf.3	key in a txt record
n_itoa	n_itoa.3	convert an integer to a fixed-length string
printdsag	printslf.3	print a dsag record
printdshg	printslf.3	print a dshg record
printdsig	printslf.3	print a dsig record
printdsmp	printslf.3	print a dsmp record
printdspg	printslf.3	print a dspg record
printdsrg	printslf.3	print a dsrg record
printdsrg_pts	printslf.3	print a dsrg_pts record
printdssg	printslf.3	print a dssg record
printdsvg	printslf.3	print a dsvg record
printfea	printslf.3	print a fea record
printfhd	printslf.3	print a fhd record
printfos	printslf.3	print a fos record
printmao_coord	printslf.3	print a mao_coord record
printmaos	printslf.3	print a maos record
printpts	printslf.3	print a pts record
printseg	printslf.3	print a seg record
printsof	printslf.3	print a sof record
printtxt	printslf.3	print a txt record
readblock	slftape.3	read a block from a slf file
readdsag	readslf.3	read a dsag record from a slf file
readdshg	readslf.3	read a dshg record from a slf file
readdsig	readslf.3	read a dsig record from a slf file
readdsmp	readslf.3	read a dsmp record from a slf file
readdspg	readslf.3	read a dspg record from a slf file
readdsrg	readslf.3	read a dsrg record from a slf file
readdsrg_pts	readslf.3	read a dsrg_pts record from a slf file
readdssg	readslf.3	read a dssg record from a slf file
readdsvg	readslf.3	read a dsvg record from a slf file
readfea	readslf.3	read a fea record from a slf file
readfhd	readslf.3	read a fhd record from a slf file
readfos	readslf.3	read a fos record from a slf file
readmao_coord	readslf.3	read a mao_coord record from a slf file
readmaos	readslf.3	read a maos record from a slf file

readpts	readslf.3	read a pts record from a slf file
readseg	readslf.3	read a seg record from a slf file
readsof	readslf.3	read a sof record from a slf file
readtape	slftape.3	read from a slf file
readtxt	readslf.3	read a txt record from a slf file
replacedshg	replaceslf.3	replace a dshg record in a slf relation
replacedsig	replaceslf.3	replace a dsig record in a slf relation
replacedsmp	replaceslf.3	replace a dsmp record in a slf relation
replacedspg	replaceslf.3	replace a dspg record in a slf relation
replacedssg	replaceslf.3	replace a dssg record in a slf relation
replacetxt	replaceslf.3	replace a txt record in a slf relation
retrievedsag	retrieveslf.3	retrieve a dsag record from a slf relation
retrievedshg	retrieveslf.3	retrieve a dshg record from a slf relation
retrievedsig	retrieveslf.3	retrieve a dsig record from a slf relation
retrievedsmp	retrieveslf.3	retrieve a dsmp record from a slf relation
retrievedspg	retrieveslf.3	retrieve a dspg record from a slf relation
retrievedsrg	retrieveslf.3	retrieve a dsrg record from a slf relation
retrievedsrg_pts	retrieveslf.3	retrieve a dsrg_pts record from a slf relation
retrievedssg	retrieveslf.3	retrieve a dssg record from a slf relation
retrievedsvg	retrieveslf.3	retrieve a dsvg record from a slf relation
retrievefea	retrieveslf.3	retrieve a fea record from a slf relation
retrievefhd	retrieveslf.3	retrieve a fhd record from a slf relation
retrievefos	retrieveslf.3	retrieve a fos record from a slf relation
retrievemao_coord	retrieveslf.3	retrieve a mao_coord record from a slf relation
retrievemaos	retrieveslf.3	retrieve a maos record from a slf relation
retrievepts	retrieveslf.3	retrieve a pts record from a slf relation
retrieveseg	retrieveslf.3	retrieve a seg record from a slf relation
retrievesof	retrieveslf.3	retrieve a sof record from a slf relation
retrievetxt	retrieveslf.3	retrieve a txt record from a slf relation
unread	slftape.3	unread a characters in a slf block

NAME

appenddsig, appenddssg, appenddspg, appenddsmp, appenddshg, appenddsvg, appenddsrg, appenddsrg_pts, appenddsag, appendmaos, appendmao_coord, appendseg, appendfos, appendpts, appendfea, appendfhd, appendsof, appendtxt – append a slf record to a slf relation

SYNOPSIS

```
#include "slf.q.h"      /* Needed for all appendslf functions */

appenddsig(dsigrec)
struct DSIG *dsigrec;

appenddssg(dssgrec)
struct DSSG *dssgrec;

appenddspg(dspgrec)
struct DSPG *dspgrec;

appenddsmp(dsmprec)
struct DSMG *dsmprec;

appenddshg(dshgrec)
struct DSHG *dshgrec;

appenddsvg(dsvgrec)
struct DSVG *dsvgrec;

appenddsrg(dsrgrec)
struct DSRG *dsrgrec;

appenddsrg_pts(dsrg_ptsrec)
struct DSRG_PTS *dsrg_ptsrec;

appenddsag(dsagrec)
struct DSAG *dsagrec;

appendmaos(maosrec)
struct MAOS *maosrec;

appendmao_coord(mao_coordrec)
struct MAO_COORD *mao_coordrec;

appendseg(relname, segrec)
char *relname;
struct SEG *segrec;

appendfos(relname, fosrec)
char *relname;
struct FOS *fosrec;

appendpts(relname, ptsrec)
char *relname;
struct PTS *ptsrec;
```

appendfea(relname, fearec)
char *relname;
struct FEA *fearec;

appendfhd(relname, fhdrec)
char *relname;
struct FHD *fhdrec;

appendsof(relname, sofrec)
char *relname;
struct SOF *sofrec;

appendtxt(txtrec)
struct TXT *txtrec;

DESCRIPTION

These routines append a slf record to a relation in a slf database maintained under Ingres. The database must already be opened under Equel. Because these routines contain embedded queries in Equel, they must be linked to the Equel library (-lq). These routines are used whenever a new record is added to a slf relation. If a tuple is to be replaced then the functions of *replaceslf(3)* should be used. These functions are used in application programs *access_slf(1)* and *slftoingres(1)*.

The functions *appendseg(3)*, *appendfos(3)*, *appendpts(3)*, *appendfea(3)*, *appendfhd(3)*, and *appendsof(3)*, require an additional parameter to identify the relation. These relations tend to be very large and therefore each data set is given its own relation. See *slf(5)* for a further explanation.

SEE ALSO

access_slf(1), ingrestoslf(1), slf(3), replaceslf(3), slf(5)

AUTHOR

Steve Reichenbach 8/85

NAME

create_rel — create slf feature and segment relations

SYNOPSIS

create_rel(dataset_id)
char *dataset_id;

DESCRIPTION

Create_rel(3) creates the *seg*, *fos*, *pts*, *fea*, *fhd*, and *sof* relations for a given data set (whose ID is given by *dataset_id*). These relations are maintained separately for each data set because they can be quite large. (See *slf(5)* for a further explanation.) The relations are named by the string created by appending the data set ID to the relation type. For example, a call to *creat_rel* with a data set ID of "DTEST1" would generate relations named "SEGDTEST1", "FOSDTEST1", and so on. These relations would be modified to indexed sequential access mode keyed on the fields as described in *slf(5)*. The database must already be opened under Equel. Because these routines contain embedded queries in Equel, they must be linked to the Equel library (-lq).

AUTHOR

Steve Reichenbach 1/86

NAME

defaultdsig, defaultdssg, defaultdspg, defaultdsmp, defaultdshg, defaultdsvg, defaultseg, defaultfos, defaultpts, defaultfea, defaultfhd, defaultsof, defaulttxt — return a default slf record

SYNOPSIS

```
#include "slf.q.h"          /* Needed for all defaultslf functions */
#include <sys/time.h>       /* Needed for defaultdsig only */

defaultdsig(dsigrec, ds)
struct DSIG *dsigrec;
char *ds;

defaultdssg(dssgrec, ds)
struct DSSG *dssgrec;
char *ds;

defaultdspg(dspgrec, ds)
struct DSPG *dspgrec;
char *ds;

defaultdsmp(dsmprec, ds)
struct DSMG *dsmprec;
char *ds;

defaultdshg(dshgrec, ds)
struct DSHG *dshgrec;
char *ds;

defaultdsvg(dsvgrec, ds)
struct DSVG *dsvgrec;
char *ds;
defaultseg(segrec, sid)
struct SEG *segrec;
char *sid;

defaultfos(fosrec, sid, fid)
struct FOS *fosrec;
char *sid, *fid;

defaultpts(ptsrec, sid)
struct PTS *ptsrec;
char *sid;

defaultfea(fearec, fid)
struct FEA *fearec;
char *fid;

defaultfhd(fhdrec, fid)
struct FHD *fhdrec;
char *fid;

defaultsof(sofrec, fid, sid)
struct SOF *sofrec;
```

```
char *fid, *sid;

defaulttxt(txtrec, ds)
struct TXT *txtrec;
char *ds;
```

DESCRIPTION

These routines set default values in slf records. The default values follow the guidelines given in Appendix XII of the Defense Mapping Agency's *Standard Linear Format for Digital Cartographic Feature Data* (Draft 2nd ed., 18 March 1985) and the Defense Mapping Agency's *Product Specifications for Digital Landmass System (DLMS) Data Base* (2nd ed., April 1983).

Most of the fields are blank or zero. The data set ID must be passed as the parameter *ds*. The group headers are the specified values. (For example, dssgrec->ds_dssg is set to "DSSG".) The map group and variable field group contain only null fields. No functions exist for the the registration points group, and accuracy subset group because they are optional. The default text record has a length of zero. The non-null fields of the remaining groups are:

```
dsigrec->prod_type:    "DFAD2"
dsigrec->ds_ed:        "1"
dsigrec->maint_date:   "0000"
dsigrec->slf_ver:      "850315"
dsigrec->facs_ver:     "000000"

dssgrec->sec_class:    "U"
dssgrec->handling:     "DISTRIBUTION LIMITED"

dspgrec->data_type:    "GEO"
dspgrec->hor_units:    "SEC"
dspgrec->hor_res:      "0.100"
dspgrec->geod_data:    "WGC"
dspgrec->ellipsoid:    "WGC"
dspgrec->lat_orig:     "00000000S"
dspgrec->long_orig:    "000000000W"
dspgrec->lat_sw:       "00000000S"
dspgrec->long_sw:      "000000000W"
dspgrec->lat_ne:       "00000000N"
dspgrec->long_ne:      "000000000E"
dspgrec->n_fea:        "0"
dspgrec->n_pt_fea:     "0"
dspgrec->n_ln_fea:     "0"
dspgrec->n_ar_fea:     "0"
dspgrec->n_seg:        "0"

dshgrec->prod_spec:    "SPEXDLMS2"
dshgrec->spec_date:    "8304"
dshgrec->amend_no:     "000"
dshgrec->producer:     "USWASHU"
```

Default features can also be created using *defaultseg(3)*, *defaultfos(3)*, fIdefaultpts(3), *defaultfea(3)*, *defaultfhd(3)*, and *defaultsof(3)*. The feature (*fid*) and segment (*sid*) ID's must be parameters to these functions. The default feature is a point with coordinates (0,0,0). The default field values are:

segrec->fea_cnt:	"1"
segrec->pt_cnt:	"1"
fosrec->ord_fos:	"1"
fosrec->fea_orient:	"C"
ptsrec->ord_pt:	"1"
ptsrec->x_val:	"0"
ptsrec->y_val:	"0"
ptsrec->z_val:	"0"
fearec->fea_type:	"P"
fearec->fea_hdr_cnt:	"1"
fearec->seg_cnt:	"1"
fhdrec->ord_hdr:	"1"
fhdrec->block:	All but the first character of the feature ID followed by "010 0 0 0 09013603 0 0 "
sofrec->ord_sof:	"1"
sofrec->direction:	"F"

These functions are used by *access_slf(1)* to create a default data set and the default structure of new features.

SEE ALSO

access_slf(1), slf(3), slf(5)

AUTHOR

Steve Reichenbach 1/86

NAME

findmaxfea, findmaxseg – find maximum feature or segment id

SYNOPSIS

```
#include "slf.q.h"      /* Needed for both findmaxfea and findmaxseg */

findmaxfea(relname)
char *relname;

findmaxseg(relname)
char *relname;
```

DESCRIPTION

Findmaxfea(3) examines a feature relation and returns the largest feature ID. *Findmaxseg(3)* examines a segment relation and returns the larges segment ID. The relation name is passed by *relname*. Both functions search only for ID's greater than zero. The database must already be opened under Equel. Because these routines contain embedded queries in Equel, they must be linked to the Equel library (-lq).

SEE ALSO

access_slf(1), slf(3), slf(5)

AUTHOR

Steve Reichenbach 1/86

NAME

keyin, keyindsig, keyindssg, keyindspg, keyindsmp, keyindshg, keyindsvg, keyindsrg, keyindsrg_pts, keyindsag, keyinmaos, keyinmao_coord, keyintxt – keyin a slf record

SYNOPSIS

```
keyin(prompt, s, n, fill)
char *prompt, *s, *fill;
int n;

#include "slf.c.h"        /* Needed for all keyinslf functions */

keyindsig(dsigrec)
struct DSIG *dsigrec;

keyindssg(dssgrec)
struct DSSG *dssgrec;

keyindspg(dspgrec)
struct DSPG *dspgrec;

keyindsmp(dsmprec)
struct DSMG *dsmprec;

keyindshg(dshgrec)
struct DSHG *dshgrec;

keyindsvg(dsvgrec)
struct DSVG *dsvgrec;

keyindsrg(dsrgrec)
struct DSRG *dsrgrec;

keyindsrg_pts(dsrg_ptsrec)
struct DSRG_PTS *dsrg_ptsrec;

keyindsag(dsagrec)
struct DSAG *dsagrec;

keyinmaos(maosrec)
struct MAOS *maosrec;

keyinmao_coord(mao_coordrec)
struct MAO_COORD *mao_coordrec;

keyintxt(txtrec)
struct TXT *txtrec;
```

DESCRIPTION

These routines allow values to be placed in the fields of slf records by prompting the user, reading the new value, checking the length of the value, and if necessary justifying the value and blank filling. The function *keyin(3)* is used to perform these tasks. The other routines provide the necessary parameters to *keyin(3)*.

Keyin(3) expects four parameters: the user prompt (*prompt*), the string location to be filled (*s*), the size of the field (*n*), and the justification (*fill*). The justication may be "F" meaning the field must be filled, "L" for left justification, or "R" for right justification. The justifications are not specified for all fields in the Defense Mapping Agency's *Standard Linear Format for Digital Cartographic Feature Data* (Draft 2nd ed., 18 March 1985), so in some cases I just chose what made sense to me.

The primary use of these functions is in *access_slf(1)* where users may specify new values for data set fields. These routines are used when the fields' values are to be specified without regard to length or justification and separated by end-of-line markers. When the values are specified without end-of-line markers and with proper length and justification, the *readslf(3)* functions should be used.

SEE ALSO

access_slf(1), slf(3), slf(5)

AUTHOR

Steve Reichenbach 1/85

NAME

n_itoa – convert an integer to a fixed-length character string

SYNOPSIS

```
n_itoa(s, x, n)
char *s;
int x, n;
```

DESCRIPTION

N_itoa converts an integer (specified by the parameter *x*) to a character string (beginning at the location pointed to by *s*) of a fixed length (of *n* characters). If the length of the string is less than *n*, then the string is blank-filled and right justified. If the length of the string is greater than *n*, then the string is truncated at the left (most significant).

AUTHOR

Steve Reichenbach 8/85

NAME

cprint, printdsig, printdssg, printdspg, printdsmp, printdshg, printdsvg, printdsrg, printdsrg_pts, printdsag, printmaos, printmao_coord, printseg, printfos, printpts, printfea, printfhd, printsof, printtxf – print a slf record

SYNOPSIS

```
cprint(p, str, len, pgtype, pgnum)
int p, *len, *pgnum;
char *str, *pgtype;

#include "slf.c.h"      /* Needed for all following printslf functions */

printdsig(p, dsigrec, len, pgtype, pgnum)
int p, *len, *pgnum;
char *pgtype;
struct DSIG *dsigrec;

printdssg(p, dssgrec, len, pgtype, pgnum)
int p, *len, *pgnum;
char *pgtype;
struct DSSG *dssgrec;

printdspg(p, dspgrec, len, pgtype, pgnum)
int p, *len, *pgnum;
char *pgtype;
struct DSPG *dspgrec;

printdsmp(p, dsmprec, len, pgtype, pgnum)
int p, *len, *pgnum;
char *pgtype;
struct DSMG *dsmprec;

printdshg(p, dshgrec, len, pgtype, pgnum)
int p, *len, *pgnum;
char *pgtype;
struct DSHG *dshgrec;

printdsvg(p, dsvgrec, len, pgtype, pgnum)
int p, *len, *pgnum;
char *pgtype;
struct DSVG *dsvgrec;

printdsrg(p, dsrgrec, len, pgtype, pgnum)
int p, *len, *pgnum;
char *pgtype;
struct DSRG *dsrgrec;

printdsrg_pts(p, dsrg_ptsrec, len, pgtype, pgnum)
int p, *len, *pgnum;
char *pgtype;
struct DSRG_PTS *dsrg_ptsrec;

printdsag(p, dsagrec, len, pgtype, pgnum)
```

```
int p, *len, *pgnum;
char *pgtype;
struct DSAG *dsagrec;

printmaos(p, maosrec, len, pgtype, pgnum)
int p, *len, *pgnum;
char *pgtype;
struct MAOS *maosrec;

printmao_coord(p, mao_coordrec, len, pgtype, pgnum)
int p, *len, *pgnum;
char *pgtype;
struct MAO_COORD *mao_coordrec;

printseg(relname, segrec)
char *relname;
struct SEG *segrec;

printfos(relname, fosrec)
char *relname;
struct FOS *fosrec;

printpts(relname, ptsrec)
char *relname;
struct PTS *ptsrec;

printfea(relname, fearec)
char *relname;
struct FEA *fearec;

printfhd(relname, fhdrec)
char *relname;
struct FHD *fhdrec;

printsof(relname, sofrec)
char *relname;
struct SOF *sofrec;

printtxt(p, txtrec, len, pgtype, pgnum)
int p, *len, *pgnum;
char *pgtype;
struct TXT *txtrec;
```

DESCRIPTION

These routines are used to print slf records to the standard output file. *Cprint(3)* is used to create the blocks (size 1980 characters) defined in the Defense Mapping Agency's *Standard Linear Format for Digital Cartographic Feature Data* (Draft 2nd ed., 18 March 1985). If the parameter *p* is nonzero (TRUE) then string pointed to by *str* is printed with an end-of-line appended. If the parameter is zero (FALSE) then *cprint(3)* prints only as many characters from the string as can fit in the block. If the block cannot hold the string, then a new block is begun with the proper page type (*pgtype*) and page number (*pgnum*). If a new block is begun, the value of *pgnum* is incremented. In any case, the length of the block thus far (*len*) is adjusted accordingly.

The functions *printseg(3)*, *printfos(3)*, *printpts(3)*, *printfea(3)*, *printfhd(3)*, and *printsof(3)* are different than the remaining functions and are described in more detail in the *BUGS* section below.

For the other functions, if the value of the parameter *p* is nonzero (TRUE) then a brief description of each field is attached to the beginning of each field's value. Otherwise, (*p* is zero or FALSE) the data is written in block form according to the DMA's specification using *cprint(3)*. The primary use of these functions is to display the contents of a record or to create an slf tape (see *ingrestoslf(1)*).

SEE ALSO

ingrestoslf(1), slf(3), slf(5)

AUTHOR

Steve Reichenbach 8/85

BUGS

The functions *printseg(3)*, *printfos(3)*, *printpts(3)*, *printfea(3)*, *printfhd(3)*, and *printsof(3)* are somewhat different. Because the slf specification mandates that the segment and field records are not contiguous, generation of slf output requires that portions of the segment and field records be delayed. This and the fact that there are usually many tuples in these relations (indicating the need for speed) and the fact that there are few fields in the records (which means printing the fields individually is relatively easy) led to the decision in implementing *ingrestoslf(1)* to print these records directly rather than via calls to these functions. The functions remain as a debugging aid, but are not used by any of the existing application programs. If they were to be used similarly to the other routines, it would be desirable to add the *p* parameter and use *cprint(3)*.

NAME

readdsig, readdssg, readdspg, readdsmp, readdshg, readdsvg, readdsrg, readdsrg_pts, readdsag, readmaos, readmao_coord, readseg, readfos, readpts, readfea, readfhd, readsof, readtxf – read a record from a slf file

SYNOPSIS

```
#include "slf.c.h"        /* Needed for all readslf functions */

readdsig(dsigrec)
struct DSIG *dsigrec;

readdssg(dssgrec)
struct DSSG *dssgrec;

readdspg(dspgrec)
struct DSPG *dspgrec;

readdsmp(dsmprec)
struct DSMG *dsmprec;

readdshg(dshgrec)
struct DSHG *dshgrec;

readdsvg(dsvgrec)
struct DSVG *dsvgrec;

readdsrg(dsrgrec)
struct DSRG *dsrgrec;

readdsrg_pts(dsrg_ptsrec)
struct DSRG_PTS *dsrg_ptsrec;

readdsag(dsagrec)
struct DSAG *dsagrec;

readmaos(maosrec)
struct MAOS *maosrec;

readmao_coord(mao_coordrec)
struct MAO_COORD *mao_coordrec;

readseg(segrec)
struct SEG *segrec;

readfos(fosrec)
struct FOS *fosrec;

readpts(ptsrec)
struct PTS *ptsrec;

readfea(fearec)
struct FEA *fearec;
```

```
readfhd(fhdrec)
struct FHD *fhdrec;

readsof(sofrec)
struct SOF *sofrec;

readtxt(txtrec)
struct TXT *txtrec;
```

DESCRIPTION

These routines read groups of fields from a file formated according to the Defense Mapping Agency's *Standard Linear Format for Digital Cartographic Feature Data* (Draft 2nd ed., 18 March 1985). They verify the group header (if appropriate), but rely on the tape functions (see *slftape(3)*) to do the low-level I/O and verify the record type. The primary use of these routines is in *slftoingres(1)*, which reads a slf tape into an relational database under Ingres.

SEE ALSO

slftoingres(1), slf(3), slftape(3), slf(5)

AUTHOR

Steve Reichenbach 8/85

BUGS

Due to the structure of the slf tape, the field for number of points in a segment is not contiguous with the rest of the data for the segment record and the field for the number of segments in a feature is not contiguous with the rest of the feature record. For this reason, the read for these two fields is not accomplished during the *readseg(3)* and *readfea(3)* calls and must be done after reading the intervening data.

NAME

replacedsig, replacedssg, replacedspg, replacedsmp, replacedshg, replacetxt – replace a slf record in a relation

SYNOPSIS

```
#include "slf.q.h"      /* Needed for all replaceslf functions */

replacedsig(dsigrec, ds)
struct DSIG *dsigrec;
char *ds;

replacedssg(dssgrec, ds)
struct DSSG *dssgrec;
char *ds;

replacedspg(dspgrec, ds)
struct DSPG *dspgrec;
char *ds;

replacedsmp(dsmprec, ds)
struct DSMG *dsmprec;
char *ds;

replacedshg(dshgrec, ds)
struct DSHG *dshgrec;
char *ds;

replacetxt(txtrec, ds)
struct TXT *txtrec;
char *ds;
```

DESCRIPTION

These functions replace slf records in a slf database. The data set specified by the character string *ds* is used to key the record to be replaced. If there are multiple records with the same data set ID, then all such records are replaced. If no record with the proper key is found, then no change is made to the relation. The functions return an integer equal to the number of records replaced. The database must already be opened Equel and because these routines contain embedded queries they must be linked to the Equel library (-lq). These functions are used by *access_slf(1)* to alter the values of fields in the data set.

SEE ALSO

access_slf(1), slf(3), slf(5)

AUTHOR

Steve Reichenbach 1/86

NAME

retrievedsig, retrievedssg, retrievedspg, retrievedsmp, retrievedshg, retrievedsvg, retrievedsrg, retrievedsrg_pts, retrievedsag, retrievemaos, retrievemao_coord, retrieveseg, retrievefos, retrievepts, retrievefea, retrievefhd, retrievesof, retrievetxt – retrieve a slf record from a relation

SYNOPSIS

```
#include "slf.q.h"      /* Needed for all retrieveslf functions */

retrievedsig(dsigrec, idreq)
struct DSIG *dsigrec;
char *idreq;

retrievedssg(dssgrec, idreq)
struct DSSG *dssgrec;
char *idreq;

retrievedspg(dspgrec, idreq)
struct DSPG *dspgrec;
char *idreq;

retrievedsmp(dsmprec, idreq)
struct DSMG *dsmprec;
char *idreq;

retrievedshg(dshgrec, idreq)
struct DSHG *dshgrec;
char *idreq;

retrievedsvg(dsvgrec, idreq)
struct DSVG *dsvgrec;
char *idreq;

retrievedsrg(dsrgrec, idreq)
struct DSRG *dsrgrec;
char *idreq;

retrievedsrg_pts(dsrg_ptsrec, idreq, no1req)
struct DSRG_PTS *dsrg_ptsrec;
char *idreq, *no1req;

retrievedsag(dsagrec, idreq)
struct DSAG *dsagrec;
char *idreq;

retrievemaos(maosrec, idreq, no1req)
struct MAOS *maosrec;
char *idreq, *no1req;

retrievemao_coord(mao_coordrec, idreq, no1req, no2req)
struct MAO_COORD *mao_coordrec;
char *idreq, *no1req, *no2req;
```

```
retrieveseg(relname, segrec)
char *relname;
struct SEG *segrec;

retrievefos(relname, fosrec)
char *relname;
struct FOS *fosrec;

retrievepts(relname, ptsrec)
char *relname;
struct PTS *ptsrec;

retrievefea(relname, fearec)
char *relname;
struct FEA *fearec;

retrievefhd(relname, fhdrec)
char *relname;
struct FHD *fhdrec;

retrievesof(relname, sofrec)
char *relname;
struct SOF *sofrec;

retrievetxt(txtrec, idreq)
struct TXT *txtrec;
char *idreq;
```

DESCRIPTION

These functions key on the data set ID requested to retrieve a record from a slf database. The data set ID is specified by *idreq*. Most of the functions use only this one key, but *retrievedsrg_pts* also uses the order of the registration point, *retrievemaos* also uses the order of the multiple accuracy outline, and *retrievemao_coord* also uses the order of the multiple accuracy outline and the order of the coordinate.

The functions set the value of the record parameter to that of the last record found in the relation. If no record with the key(s) is found, the record parameter is unchanged. The functions return an integer equal to the number of records with the key(s) in the relation.

The database must already be opened under Equel and because these routines contain embedded queries they must be linked to the Equel library (-lq).

SEE ALSO

access_slf(1), ingrestoslf(1), slf(3), slf(5)

AUTHOR

Steve Reichenbach 8/85

BUGS

The functions to retrieve feature and segment data are not currently used by any of the application programs. These relations tend to be very large, but easy to access because they have few fields. Nested queries are the most natural way to access this data, but one limitation of our version of Equel is that it does not allow nested queries. Therefore these calls can not be nested since they contain queries.

NAME

initializetape, readblock, readtape, unread – low-level access to slf file

SYNOPSIS

```
initializetape(name)
char *name;

readblock()

readtape(recordtype, buf, n)
char *recordtype, *buf;
int n;

unread(n)
int n;
```

DESCRIPTION

These functions provide the low-level interface to the block structure of the Defense Mapping Agency's *Standard Linear Format for Digital Cartographic Feature Data* (Draft 2nd ed., 18 March 1985). There are several data structures (hereafter referred to as the static tape variables) that are global to these four routines. They maintain information about the slf file being read.

Initializetape(3) opens the named file (*name*), initializes the static tape variables about the file, and reads the first block. This function should always be called first in order to set-up things for the other functions.

Readblock(3) is an internal routine used to fetch new blocks and maintain the static tape variables. It needs no parameters. It checks block types and sequence numbers.

Readtape(3) compares the type of record requested (*recordtype*) with the type of the block which is being read. If they are different an error occurs. Otherwise, *n* bytes are copied from the current block into the buffer specified by the pointer *buf*. This string is terminated with the null character. If during the process of reading, a block is exhausted, another is read (using *readblock(3)*) and the *readtape(3)* process continues.

Unread(3) is a kluge to discard what may have been mistakenly read from a block. For example, if the block is read for the presence of an optional group and it turned out that the optional group did not exist, then the characters could be "unread." There are better ways to do this, but that's often true.

SEE ALSO

slftoingres(1), slf(3), slf(5)

AUTHOR

Steve Reichenbach 8/85

NAME

slf – introduction to the slf tape format, database organization, and structures

DESCRIPTION

These manual pages describe three separate organizational formats: 1) the Defense Mapping Agency's *Standard Linear Format for Digital Cartographic Feature Data* (Draft 2nd ed., 18 March 1985), 2) the structure of the relational database under Ingres that has been implemented to hold this data, and 3) the C language record structures that are used in the programs dealing with the slf files and databases. Although these pages are extensive, they are by no means a complete description. Because the standard is new (and in draft form), it can be expected that there may be changes or revisions.

Some files that are useful for manipulating the slf database can be found in the INGRES subdirectory. The C structures are defined in the header files *slf.q.h* and *slf.c.h* in the INCLUDE subdirectory.

SEE ALSO

access_slf(1), graphfea(1), ingrestoslf(1), slftoingres(1), slf(3)

LIST OF FORMATS

Name	*Appears on Page*	*Description*
slf database	slfdb	ingres relational database for slf data
slf structures	slfstruct	C structures for slf data
slf tape	slftape	standard linear format definition

NAME

slfdb – ingres relational database for slf data

DESCRIPTION

This relational database was designed under Ingres to receive and maintain data in the SLF. The fields of the relations of the database closely follow DMA's specification of the format. All of the fields are ascii character strings. Most correspond directly to a SLF field, having an abbreviated name and the same character length.

The DSI record has been broken into eleven (11) relations. Eight of the relations correspond to the Data Set Groups defined in SLF. These relations have the same names as the SLF abreviations: *dsig, dssg, dspg, dsmp, dshg, dsvg, dsrg,* and *dsag*. In addition to these, three relations have been added to store the repeating subgroups: the registration points (*dsrg_pts*), the multiple accuracy outlines (*maos*), and the coordinates of the multiple accuracy outlines (*mao_coord*). The data set group headers have been omitted because they are redundant. Some fields have been added to facilitate access. All of the relations have a field to identify the data set (*ds_dssg, ds_dspg, ds_dsmp, ds_dshg,* and so on). The *dsrg_pts* relation has a *dsrg_order* field to distinguish each registration point. The *maos* relation has a *mao_id* field for the same purpose. The *mao_coord* relation has a *mao_coord_id* field to identify the multiple accuracy outline to which the coordinate belongs and a *coord_id* field to order the coordinate in the outline.

These fields are fully described by the Ingres commands used to create them.

```
create dsig (
        prod_type = c5,
        ds_id = c20,
        ds_ed = c3,
        comp_date = c4,
        maint_date = c4,
        slf_ver = c6,
        facs_ver = c6,
        dsig_rsv = c28
)

create dssg (
        ds_dssg = c20,
        sec_class = c1,
        sec_rel = c2,
        down_date = c6,
        handling = c21,
        dssg_rsv = c40
)

create dspg (
        ds_dspg = c20,
        data_type = c3,
        hor_units = c3,
        hor_res = c5,
        geod_data = c3,
        ellipsoid = c3,
        ver_units = c3,
        ver_res = c5,
        ver_ref = c4,
```

```
                sounding = c4,
                lat_orig = c9,
                long_orig = c10,
                x_orig = c10,
                y_orig = c10,
                z_orig = c10,
                lat_sw = c9,
                long_sw = c10,
                lat_ne = c9,
                long_ne = c10,
                n_fea = c6,
                n_pt_fea = c6,
                n_ln_fea = c6,
                n_ar_fea = c6,
                n_seg = c6,
                dspg_rsv = c40
)

create dsmp (
                ds_dsmp = c20,
                prjctn = c2,
                parm1 = c10,
                parm2 = c10,
                parm3 = c10,
                parm4 = c10,
                scale = c9,
                dsmp_rsv = c40
)

create dshg (
                ds_dshg = c20,
                dshg_ed = c3,
                prod_spec = c15,
                spec_date = c4,
                amend_no = c3,
                producer = c8,
                dig_sys = c10,
                proc_sys = c10,
                grid_sys = c2,
                abs_h_acc = c4,
                abs_v_acc = c4,
                rel_h_acc = c4,
                rel_v_acc = c4,
                ht_acc = c4,
                data_gen = c1,
                n_mm_no = c1,
                e_mm_no = c1,
                s_mm_no = c1,
                w_mm_no = c1,
                n_mm_date = c4,
                e_mm_date = c4,
                s_mm_date = c4,
```

```
        w_mm_date = c4,
        early_src = c4,
        late_src = c4,
        coll_code = c1,
        coll_crit = c3,
        dshg_rsv = c28
)

create dsvg (
        ds_dsvg = c20,
        reg_pt_add = c5,
        acc_addr = c5,
        dsvg_rsv = c40
)

create dsrg (
        ds_dsrg = c20,
        no_reg_pt = c3
)

create dsrg_pts (
        ds_dsrg_pts = c20,
        dsrg_ord = c3,
        pt_id = c6,
        lat = c9,
        lng = c10,
        elev = c8,
        x_coord = c6,
        y_coord = c6,
        z_coord = c6
)

create dsag (
        ds_dsag = c20,
        no_mao = c2
)

create maos (
        ds_maos = c20,
        mao_id = c2,
        mao_abs_h = c4,
        mao_abs_v = c4,
        mao_rel_h = c4,
        mao_rel_v = c4,
        no_coord = c2
)

create mao_coord (
        ds_mao_coord = c20,
        mao_coord_id = c2,
        coord_id = c2,
        mao_lat = c9,
```

```
            mao_long = c10
      )
```

The Segment and Feature records tend to be very long, with perhaps hundreds of thousands segments or features and their constituent points. The Ingres system is not lightening fast so this would definitely present a problem if the database were to contain more than one data set. The solution to this problem was to define six separate relations for each data set's segments and features. The names of these relations are formed by concatenating a three-letter relation type with the data set ID. The relations and their three-letter identifiers are: the segments relation (*seg*), the features of the segments relation (*fos*), the points of the segments relation (*pts*), the features relation (*fea*), the headers of the features relation (*fhd*), and the segments of the features relation (*sof*). The feature ID's and Segment ID's are used to tie these relations together. Also, some fields were added to maintain the order of the repeated fields. The fos relation uses the order of the feature of the segment (*ord_fos*). The pts relation uses the order of the point (*ord_pt*). The fhd relation uses the order of the header block (*ord_hdr*). The sof relation uses the order of the segment of the feature (*ord_sof*). As with the Data Set record, these relations can be described by the Ingres commands to create them, but the relation names would not be known until the data set ID of the data set is known.

```
create segrel (
        seg_id = c6,
        fea_cnt = c2,
        pt_cnt = c5
)

create fosrel (
        seg_no = c6,
        ord_fos = c2,
        fos_id = c6,
        fea_orient = c1
)

create ptsrel (
        sop_no = c6,
        ord_pt = c5,
        x_val = c6,
        y_val = c6,
        z_val = c6
)

create fearel (
        fea_id = c6,
        fea_type = c1,
        fea_hdr_cnt = c2,
        seg_cnt = c3
)

create fhdrel (
        hdr_fea_id = c6,
        ord_hdr = c2,
        block = c40
)
```

```
create sofrel (
        fea_no = c6,
        ord_sof = c3,
        direction = c1,
        sof_id = c6
)
```

The limit on the size of a field in Ingres is 255 characters, so the text record had to be broken up into blocks. The character count is stored in the *txt_cnt* field of all of the tuples. This is a small redundancy. Another field is used to order the blocks of the text (*ord_txt*). The relation is created with the following Ingres command.

```
create txt (
        ds_txt = c20,
        txt_cnt = c4,
        ord_txt = c1,
        text = c255
)
```

In order to speed processing, these relations are modified for indexed-sequential access. This is only done for the Segment and Feature relations, because they are the only relations whose size presents a problem. The following Ingres commands are used to modify these relations.

```
modify segrel to isam on seg_id
modify fosrel to isam on seg_no, ord_fos
modify ptsrel to isam on sop_no, ord_pt
modify fearel to isam on fea_id
modify fhdrel to isam on hdr_fea_id, ord_hdr
modify sofrel to isam on fea_no, ord_sof
```

FILES

Some files useful for manipulating the slf database (delete, modify, create, etc.) can be found in the INGRES subdirectory.

SEE ALSO

access_slf(1), graphfea(1), ingrestoslf(1), slftoingres(1), slf(3), slf(5)

AUTHOR

Steve Reichenbach 8/85

NAME

slfstruct – C structures for slf data

DESCRIPTION

The C data structures used to read, write, and process slf data also closely correspond to the SLF described by DMA. These structures have fields of the DMA standard that are not included in the database (the data set group records) and fields of the database relations that are not defined in SLF (the identifying and ordering fields). The field names are the same as the database field names. This means that the Equel programs must use the non-referencing operator (#) for all database field references, but having the names the same makes them easier to remember. The length of the fields is one character longer than the fields of the SLF and the database because the strings used with Equel require an extra character for the null-terminator.

```
struct DSIG {
        char        dsig_hdr[5];
        char        prod_type[6];
        char        ds_id[21];
        char        ds_ed[4];
        char        comp_date[5];
        char        maint_date[5];
        char        slf_ver[7];
        char        facs_ver[7];
        char        dsig_rsv[29];
};

struct DSSG {
        char        ds_dssg[21];
        char        dssg_hdr[5];
        char        sec_class[2];
        char        sec_rel[3];
        char        down_date[7];
        char        handling[22];
        char        dssg_rsv[41];
};

struct DSPG {
        char        ds_dspg[21];
        char        dspg_hdr[5];
        char        data_type[4];
        char        hor_units[4];
        char        hor_res[6];
        char        geod_data[4];
        char        ellipsoid[4];
        char        ver_units[4];
        char        ver_res[6];
        char        ver_ref[5];
        char        sounding[5];
        char        lat_orig[10];
        char        long_orig[11];
        char        x_orig[11];
        char        y_orig[11];
        char        z_orig[11];
        char        lat_sw[10];
```

```
        char        long_sw[11];
        char        lat_ne[10];
        char        long_ne[11];
        char        n_fea[7];
        char        n_pt_fea[7];
        char        n_ln_fea[7];
        char        n_ar_fea[7];
        char        n_seg[7];
        char        dspg_rsv[41];
};

struct DSMP {
        char        ds_dsmp[21];
        char        dsmp_hdr[5];
        char        prjctn[3];
        char        parm1[11];
        char        parm2[11];
        char        parm3[11];
        char        parm4[11];
        char        scale[10];
        char        dsmp_rsv[41];
};

struct DSHG {
        char        ds_dshg[21];
        char        dshg_hdr[5];
        char        dshg_ed[4];
        char        prod_spec[16];
        char        spec_date[5];
        char        amend_no[4];
        char        producer[9];
        char        dig_sys[11];
        char        proc_sys[11];
        char        grid_sys[3];
        char        abs_h_acc[5];
        char        abs_v_acc[5];
        char        rel_h_acc[5];
        char        rel_v_acc[5];
        char        ht_acc[5];
        char        data_gen[2];
        char        n_mm_no[2];
        char        e_mm_no[2];
        char        s_mm_no[2];
        char        w_mm_no[2];
        char        n_mm_date[5];
        char        e_mm_date[5];
        char        s_mm_date[5];
        char        w_mm_date[5];
        char        early_src[5];
        char        late_src[5];
        char        coll_code[2];
        char        coll_crit[4];
```

```
        char            dshg_rsv[29];
};

struct DSVG {
        char            ds_dsvg[21];
        char            dsvg_hdr[5];
        char            reg_pt_add[6];
        char            acc_addr[6];
        char            dsvg_rsv[41];
};

struct DSRG {
        char            ds_dsrg[21];
        char            dsrg_hdr[5];
        char            no_reg_pt[4];
};

struct DSRG_PTS {
        char            ds_dsrg_pts[21];
        char            dsrg_ord[4];
        char            pt_id[7];
        char            lat[10];
        char            lng[11];
        char            elev[9];
        char            x_coord[7];
        char            y_coord[7];
        char            z_coord[7];
};

struct DSAG {
        char            ds_dsag[21];
        char            dsag_hdr[5];
        char            no_mao[3];
};

struct MAOS {
        char            ds_maos[21];
        char            mao_id[3];
        char            mao_abs_h[5];
        char            mao_abs_v[5];
        char            mao_rel_h[5];
        char            mao_rel_v[5];
        char            no_coord[3];
};

struct MAO_COORD {
        char            ds_mao_coord[21];
        char            mao_coord_id[3];
        char            coord_id[3];
        char            mao_lat[10];
        char            mao_long[11];
};
```

```
struct SEG {
        char            seg_id[7];
        char            fea_cnt[3];
        char            pt_cnt[6];
};

struct  FOS {
        char            seg_no[7];
        char            ord_fos[3];
        char            fos_id[7];
        char            fea_orient[2];
};

struct PTS {
        char            sop_no[7];
        char            ord_pt[6];
        char            x_val[7];
        char            y_val[7];
        char            z_val[7];
};

struct FEA {
        char            fea_id[7];
        char            fea_type[2];
        char            fea_hdr_cnt[3];
        char            seg_cnt[4];
};

struct FHD {
        char            hdr_fea_id[7];
        char            ord_hdr[3];
        char            block[41];
};

struct SOF {
        char            fea_no[7];
        char            ord_sof[4];
        char            direction[2];
        char            sof_id[7];
};

struct  TXT {
        char            ds_txt[21];
        char            txt_cnt[5];
        char            text[1969];
};
```

These structure definitions are contained in the header files *slf.q.h* and *slf.c.h*. *Slf.c.h* is generated by the Equel preprocessor from *slf.q.h*. In addition to these structure definitions, the header file also contains the constant definitions for the sizes of the SLF block and its components (see *slftape(5)*) and an error macro.

```
#define error(msg) {printf(msg); exit(-1);}
```

FILES

~INCLUDE/slf.q.h
~INCLUDE/slf.c.h

SEE ALSO

access_slf(1), graphfea(1), ingrestoslf(1), slftoingres(1), slf(3), slf(5)

AUTHOR

Steve Reichenbach 8/85

NAME

slftape – standard linear format definition

DESCRIPTION

The Standard Linear Format (SLF) was designed by the Defense Mapping Agency as a standard for the exchange of digital cartographic features on magnetic tape. The format specifies descriptive fields about feature data, as well as specifying the representation of the features. It represents features as chain-node data structures, by specifying common boundaries between areal features (as well as specifying point and linear features).

The SLF tape is divided into five logical records with the order: the Data Set Identifier record (DSI), the Segment record (SEG), the Feature record (FEA), and the Text Record (TXT) which is optional. The order of the logical records must be that given above. Each logical record consists of one or more physical blocks of 1980 bytes, each byte an ascii character. Each physical block begins with an eight-byte header consisting of the left-justified, block type (DSI, SEG, FEA, or TXT) and a right-justified, five-digit block sequence number (with blank fill). Each logical record's physical blocks are sequenced separately beginning with one (1) for the first block of each logical record. The remaining 1972 bytes of each physical block is allocated to the storage of the fields of the logical record. Data fields within a logical record may span more than one physical block, but a physical block may not contain data from different logical records. Any unfilled bytes at the end of physical record following the end of a logical record are filled with the ascii delete character (octal 177).

The Data Set Identifier record (DSI) consists of eight groups of fields. These groups, their fields, and the field lengths are given in brief below. Subgroups of fields (such as the information about each of the registration points in the Registration Points Group) may be repeated any number of times. In these cases, the field immediately before the repeating fields indicates the number of occurrences of the subgroup. For a more details, consult *Standard Linear Format for Digital Cartographic Feature Data* (Draft 2nd ed., 18 March 1985).

A. Data Set Identification Group (DSIG)

1.	DSIG Header	4
2.	Product Type	5
3.	Data Set ID	20
4.	Edition	3
5.	Compilation Date	4
6.	Maintenance Date	4
7.	SLF Version Date	6
8.	FACS Version Date	6
9.	DSIG Reserve	28

B. Data Set Security Group (DSSG)

1.	DSSG Header	4
2.	Security Classification	1
3.	Security Release	2
4.	Downgrading/Reclassification Date	6
5.	Security Handling	21
6.	DSSG Reserve	40

C. Data Set Parameter Group (DSPG)

1.	DSPG Header	4
2.	Data Type	3
3.	Horizontal Units of Measure	3
4.	Horizontal Resolution Units	5

5.	Geodetic Datum	3
6.	Ellipsoid	3
7.	Vertical Units of Measure	3
8.	Vertical Resolution Units	5
9.	Vertical Reference System	4
10.	Sounding Datum	4
11.	Latitude of Origin	9
12.	Longitude of Origin	10
13.	X Coordinate of Origin	10
14.	Y Coordinate of Origin	10
15.	Z Coordinate of Origin	10
16.	Latitude of SW Corner	9
17.	Longitude of SW Corner	10
18.	Latitude of NE Corner	9
19.	Longitude of NE Corner	10
20.	Total Number of Features	6
21.	Number of Point Features	6
22.	Number of Linear Features	6
23.	Number of Areal Features	6
24.	Total Number of Segments	6
25.	DSPG Reserve	40

D. Data Set Map Group (DSMP)

1.	DSMP Header	4
2.	Projection	2
3.	Projection Parameter 1	10
4.	Projection Parameter 2	10
5.	Projection Parameter 3	10
6.	Projection Parameter 4	10
7.	Scale	9
8.	DSMP Reserve	40

E. Data Set History Group (DSHG)

1.	DSHG Header	40
2.	Edition Code	3
3.	Product Specification	15
4.	Specification Date	4
5.	Specification Amendment Number	3
6.	Producer	8
7.	Digitizing System	10
8.	Processing System	2
9.	Absolute Horizontal Accuracy	4
10.	Absolute Vertical Accuracy	4
11.	Relative Horizontal Accuracy	4
12.	Relative Vertical Accuracy	4
13.	Height Accuracy	4
14.	Data Generalization	1
15.	North Match/Merge Number	1
16.	East Match/Merge Number	1
17.	South Match/Merge Number	1
18.	West Match/Merge Number	1
19.	North Match/Merge Date	4

20.	East Match/Merge Date	4
21.	South Match/Merge Date	4
22.	West Match/Merge Date	4
23.	Date of Earliest Source	4
24.	Date of Latest Source	4
25.	Data Collection Code	1
26.	Data Collection Criteria	3
27.	DSHG Reserve	28

F. Data Set Variable Field Address Group (DSVG)

1.	DSVG Header	40
2.	Registration Points Address	5
3.	Accuracy Subset Address	5
4.	DSVG Reserve	40

G. Data Set Registration Points Group (DSRG) (optional)

1.	DSRG Header	40
2.	Number of Registration Points	3
	a. Point ID	6
	b. Latitude	9
	c. Longitude	10
	d. Elevation	8
	e. X-Coordinate	6
	f. Y-Coordinate	6
	g. Z-Coordinate	6

H. Data Set Accuracy Group (DSAG) (optional)

1.	DSAG Header	40
2.	Multiple Accuracy Outline Count	2
	a. Absolute Horizontal Accuracy	4
	b. Absolute Vertical Accuracy	4
	c. Relative Horizontal Accuracy	4
	d. Relative Vertical Accuracy	4
	e. Number of Coordinates	2
	i. Latitude	9
	ii. Longitude	10

The Segment record (SEG) specifies the line segments that separate features. Each boundary segment is encoded only once and the feature(s) it delineates refer to that segment. For a fuller discussion of chain-coding of segments refer to Appendix VI of the DMA specification.

Segment Record

1.	Segment ID	6
2.	Feature Count	2
	a. Feature ID	6
	b. Feature Orientation	1
3.	Point Count	5
	a. X-Value	6
	b. Y-Value	6
	c. Z-Value (optional)	6

The Feature record (FEA) specifies the features. Each feature is described by a header and names the segments that delineate it. The Feature Header Block Count specifies how many forty (40) character blocks follow it. The Segment Count specifies how many Direction of