U.S. Department of the Interior
U.S. Geological Survey

SUDS

The Seismic Unified Data System
Version 2.6

Peter L. Ward

U.S. Geological Survey
Menlo Park, CA 94025

May 11, 1994

Open-File Report 94-003

This page left blank intentionally.

Table of Contents

OVERVIEW 1
ACKNOWLEDGEMENTS 2
GETTING STARTED USING SUDS 2
EACH DATA GROUP IS A STRUCTURE 3
TERMINOLOGY 4
AN EXAMPLE 5
COMPATIBILITY WITH OTHER STANDARDS 6
TABLE-DRIVEN PROGRAMMING 6
VARIABLES IN YOUR ENVIRONMENT 7
READING AND WRITING SUDS STRUCTURES 7
A PROGRAMMING EXAMPLE 7
DATA TYPES AND MISSING DATA 8
SUDS FILE FORMAT AND ORGANIZATION 9
COMMENTS 9
CODE LISTS 9
EXCHANGE FORMAT 9
PORTABLE PROGRAMMING 10
RELATIONSHIPS BETWEEN STRUCTURES 11
DEFINING NEW STRUCTURES AND MEMBERS 16
REFERENCES 17
MANUAL PAGES(1): **SUDS** COMMANDS 21
MANUAL PAGES(2): **SUDS** SUBROUTINES 47
MANUAL PAGES(3): SYSTEM STRUCTURE DEFINITIONS 103
MANUAL PAGES(4): USER STRUCTURE DEFINITIONS 133
MANUAL PAGES(5): **PC-SUDS** STRUCTURE DEFINITIONS 261
MANUAL PAGES(6): **SUDS** CODE_LISTS 317
MANUAL PAGES(7): Mappings From Other Formats 355

Availability of the Computer Code

SUDS is in the public domain and is available for unlimited and unrestricted distribution. The source code is available over Internet via anonymous ftp at the computer **dmc.iris.washington.edu** (128.95.166.2) operated by the Incorporated Research Institutions for Seismology in Seattle, Washington.

SUDS is provided "as is" with no expressed or implied guarantee as to its operability in a specific environment. No support can be promised. Please report errors to **ward@andreas.wr.usgs.gov**.

To obtain a copy of the software using **ftp** from a computer connected to Internet, type:

ftp dmc.iris.washington.edu

Your computer should respond:

Connected to dmc.iris.washington.edu.

Name (dmc.iris.washington.edu:your_name):

Type:

anonymous

Your computer types:

Password:

You type your complete email address. It responds:

ftp>

You type:

cd pub/suds/suds_2.6

You can copy the files by typing

prompt

to turn off the interactive prompt and then

mget *

In this directory, the manual is in PostScript format. The README files are in ASCII. The main software is in the file **suds_2.6_tar.Z** and you will need a UNIX system to uncompress the archive (tar) file and take the files out of the archive as described in README.all. This is a complete distribution, primarily for workstations running UNIX, from which distributions for 80386 and 80486 computers running DOS 6 and Macintosh computers running System 7 can be made using the primary Makefile. If you need floppy diskettes useable directly on PCs or Macintosh computers, contact Peter Ward (U.S. Geological Survey, MS 977, 345 Middlefield Road, Menlo Park, CA 94025, email **ward@andreas.wr.usgs.gov**, telphone 415/329-4736, fax 415/329-5163).

OVERVIEW

SUDS stands for the Seismic Unified Data System. **SUDS** is:

* a format for seismic data
* a relational database design and implementation
* a table-driven programming system
* a machine-independent environment for data and programs

SUDS, is a new method for organizing data that promotes efficient storage, exchange, and use of both data and computer programs by seismologists with widely varying needs and interests. **SUDS** is compact, modular, and self-documenting. **SUDS** is machine independent, working well in both files and relational database systems mounted on different types of storage devices and on different types of computers that stand alone or are interconnected by networks. **SUDS** is scalable to meet efficiently the needs of individuals with one seismograph, the needs of operators of major seismograph networks, the needs of seismologists merging data from all over the world, and even the widely varying needs of researchers.

Most existing seismic data formats consist of a large "header", containing many fields that are often not used, followed by waveform data. The header typically describes the data (number of samples, sampling rate, storage type, etc.) as well as information about the way the data were acquired (longitude, latitude, and elevation of the station recording the data, parameters of the sensor, and characteristics of the recording equipment). Few seismic data formats attempt to define in a standard way the results of calculations made from the raw data. Thus the input and output formats of popular analysis programs become the de-facto standards for calculated data and they are typically not compatible with the formats of the raw data.

SUDS takes a different approach. The large, often little used "header" is replaced by many different groups of data, each specializing in a particular type of information, such as information about a seismometer, a recorder, or a waveform. These data groups may be included or not included, depending on whether the information exists and whether the information is needed for a particular application. These groups of data can be organized in any order and in any number of files.

SUDS is also a relational database design. While many seismologists want to use databases to store and rapidly access large volumes of data, few databases have been implemented because of the large amount of work necessary to decide which data to include, how to organize the data, and how to index the data. These issues have all been addressed in the design of **SUDS** so that the definition of the **SUDS** data groups is also a relational database model. In addition, the **SUDS** data groups have been designed so that data from throughout the world can be merged uniquely.

SUDS is also a table-driven programming system. The method of organizing **SUDS** data groups is **unified** from data collection, through routine data processing, and even through specialized, research-oriented processing. All important features of each data group are described in detail in a manual that can be automatically compiled into three machine readable tables. These tables are used by utility programs to read, display, and modify the data at all different stages of processing. This table-driven approach significantly reduces the number of programs needed, promotes modular programming and easy exchange of programs, and assures that new data groups defined in the future will be compatible with existing programs. A library of subroutines makes it easy to read, write, initialize, and interpret **SUDS** data groups and to access the tables.

SUDS is also a machine-independent environment for data and programs. **SUDS** was developed on SUN workstations, but it runs on PCs under MSDOS, on Macintosch computers under system 7, and should be readily portable to any 16 or 32 bit computer with an ANSI C compiler. Data mounted on any one computer can be read and written on any other computer via the Network File System or on portable media.

Thus **SUDS** is more than a data format, it is a method for storing and accessing data and passing the data between programs at all levels of data processing and research. It is a relational database model and it is a table-driven programming system. It is machine independent. A major feature of **SUDS** is that it is expandable to meet currently unknown needs and thus can grow with our research needs. **SUDS** promotes sharing of data and programs.

ACKNOWLEDGEMENTS

SUDS version 1 was designed by Peter Ward (1989) for the SUN-3 computers with considerable assistance from Fred Klein, Chris Stephens, and John Lahr. Version 1 was adopted by Willie Lee as the format for the IASPEI Seismological Library for personal computers and has become known as **PC-SUDS**. The adaptation was done by Dean Tottingham, with additions by John Rogers. The final format used was version 1.31 which is documented in detail by Banfill(1992). Version 1.4 is documented by Banfill (1993). All of these people either work at or were on contract to the U.S. Geological Survey in Menlo Park, CA.

Version 2 is a complete rewrite to make the structures machine independent and relational (Ward, 1992). Fred Williams of the Geophysical Institute, University of Alaska played a major role in the design of the structures, particularly from the database perspective. Ming Jiang of the Computer Science Department of the University of Alaska wrote much of the manual compiler and developed a generic database interface. Also at the Geophysical Institute, John Davies, Cole Sonafrank, and Mitch Robinson helped in specifying the members of many structures and Mark Anderson is working on the Sybase implementation. Nils Lahr of Lafayette College played a major role in modifying SUDS version 2.3 so that it would compile on IBM-compatible and Macintosch personal computers. Bjoern Rugenstein from GeoForschungsZentrum, Potsdam, Germany, provided a detailed analysis of the structures and many suggestions.

GETTING STARTED USING SUDS

Learning any new computer system is a bit bewildering at first. The shear size of this manual is enough to scare off the faint at heart! **SUDS** is very straight-forward and logically quite simple. What makes the manual thick is the implementation of all types of seismic data within this simple framework. Thus do not read this manual from beginning to end. Read the first dozen or so pages of text very carefully.

You can get a feel for **SUDS** using some of the general commands described in chapter 1 of this manual. Here is a simple example for entry on the command line of you computer:

TYPE: cd data

SUDS can exist in files

TYPE: ls -l

To see the contents of the files

TYPE: stdescr waveform.st or stdescr *

This tells you the kind of structures and the amount of data associated with each structure.

To see the members of the structure

TYPE: st2asc waveform.st

To see a verbose description

TYPE: st2asc -v waveform.st

You can access specific structures in a file just like ASCII lines in UNIX

TYPE: sthead -3 calnet.st | st2asc

TYPE: stpart -7 +11 calnet.st | st2asc

You can edit a SUDS file. This may NOT be possible in a COMMAND_TOOL window because the curses terminal library does not always work right in COMMAND_TOOL windows. Either use a SHELL_TOOL window or some dumb terminal.

TYPE: stedit calnet.st

You advance to the next structure with the F7 key or ESC n. You delete a structure with the F9 key or ESC d. You add a new structure into the file in sequence with F5 or ESC a. Note the new structure is initialized to all defaults(see st_init(2)). You quit wth F2 or ESC q. When you quit, answer yes or no to save edits Note codes are described automatically in { }. Note fields are restricted to certain types of input. Stedit is a simple example of the type of forms editor that can be implemented using the central tables of SUDS. A more comprehensive editor using windows one many different types of machines

A sample C program for converting your data into SUDS is found in /suds/cmd/filter.c.

EACH DATA GROUP IS A STRUCTURE

SUDS is based on data stored in structures. A structure is simply a list of variables of specified type, size, and order. A good example of a structure is the summary "card" created by many earthquake location programs. This information was formerly punched on a computer card but now is typically contained on one line within a file. All of the most important information about the location of one earthquake is contained on one card or line in a specific order and format that both people and computer programs can read and write. These lines can be addressed individually or regrouped and reordered. Thus a structure is a single "handle" or reference that makes it easy to grab with one hand or refer to with one word, a group of many different types of variables that explain the attributes of a more complex entity, in this case the location of an earthquake.

Summary "cards" are stored in ASCII format, i.e. letters and numbers that are easily understood by people. Structures in **SUDS**, however, are stored in binary format, which is very compact with respect to storage and very efficient when used by computer programs. **SUDS** structures can be readily converted into and out of ASCII, but tools provided with **SUDS**, such as a forms editor, make reading and writing **SUDS** structures easier, more flexible, and less error prone than standard methods of reading and modifying ASCII files in a text editor. **SUDS** structures are encoded in a standard binary format called XDR (eXternal Data Representation) that can be read and written on all popular computers, even those with widely varying native binary formats. Furthermore **SUDS** structures are encoded in a way that avoids the need for any conversion when reading or writing them on most popular work stations.

Several dozen structures are defined in **SUDS** to meet varying needs from data collection, to common types of processing, to maintenance of complex networks of equipment. Each structure stores the most important information about a logical entity such as an earthquake location, a phase reading, a seismogram, or a piece of equipment. Structures referring to data are followed by the data. Related structures can be grouped together in files or directories in any order. Certain structures are also related by keys. These keys are designed to facilitate grouping of related structures in files and programs and also to facilitate storage of structures in database management systems for rapid search and retrieval. The keys have been designed to be uniquely assigned by a local organization but to also have unique meaning throughout the world. Thus worldwide data can be merged readily.

TERMINOLOGY

Throughout this manual there are a few terms that are used repeatedly and to some extent interchangeably. The primary example is the word "structure". In the simplest sense, a structure is simply a group of related variables, such as a "phase card" in older location programs. In FORTRAN a common block is a simple example of a structure. In C, and in new versions of FORTRAN available on nearly all machines used by seismologists, structures are called structures. In Pascal, and in most database systems, they are called records. Each structure is composed of several variables, called members, or fields. The pages in section 5 of this manual describe in detail the members of each SUDS structure. In a database implementation of SUDS, there is a table for each type of SUDS structure, i.e. all *waveform* structures would be put in the *waveform* table, and all *signal_path* structures would be put in the *signal_path* table. Thus rough equivalencies in terminology are as follows:

structure (as a type) = record type = table definition

a structure (as an instance) = a record = a row in a table

member (as a type) = field type = column definition in a table

a member (as an instance) = a field = the contents of a column in a particular row of a table

AN EXAMPLE

This diagram shows a simplified example of **SUDS** structures. Each box represents a data group or structure describing a particular logical entity. Only the members of structures that provide links or keys to other structures are shown and these members are connected by lines with arrows. Keys come in two varieties: primary and foreign. A primary key is a unique identifier of a particular instance of a structure such as a particular *event* or earthquake. A foreign key is a member of many other structures that provide more information about the structure containing the primary key. In other words there may be many *solutions* for one *event*. The *event* structure describing the earthquake would have a primary key that is a member containing a unique number that identifies that particular earthquake. Each of the *solution* structures would contain a foreign key pointing to the primary key. The foreign key has the same value as the primary key. Thus the primary and foreign keys define relationships between structures and are the basic design elements of a relational database system. The arrows point from a foreign key to the primary key. Typically there are many instances of a foreign key pointing to one instance of a primary key. In SUDS, the variables of type **LABEL** are primary keys and the variables of type **REFERS2** are foreign keys.

In the diagram, the *signal_path* structure gives information about a particular sensor located at a *site* and how the data are transmitted to a particular recorder. For each *signal_path*, many *waveform*s or seismograms are recorded. *Waveform*s for the same earthquake, explosion, or period of time belong to a group defined by the structure *data_group*. For each *event* there may be many *solution*s or calculated locations. For each *waveform* there may be many *pick*s or phases. For each *solution* there is typically one *residual* for each phase and there may be a *focal_mech*anism.

These structures can exist in a file, be passed in a data stream between computers or programs, or be contained in a database. In a file or stream each structure is preceded by a small structure called a *structure_tag* that tells which structure follows, how long the structure is, and how much data follows the structure. These pairs of structures can then be organized in any order or grouping. For example some people may prefer to put all of the structures describing a solution in a file. Others may prefer to put all of the structures describing an event in a file. In a relational database system the structures are stored in the respective tables.

This example describes the essence of **SUDS**: many different data groups or structures whose inter-relationships are defined. These structures can occur in whatever order and form is appropriate for the individual scientist. To allow for future growth, new structures can be defined and new members can be added to the end of old structures. Definition of the contents of each structure and the relationships between the members of each structure are contained in three tables that control how the different structures are processed by utility programs. These tables are generated automatically from the pages of this manual. This manual in computer form can be stored with the data and thus provide complete documentation.

COMPATABILITY WITH OTHER STANDARDS

SUDS is a logical extension of many standards that have proven valuable. As described above, **SUDS** is a generalized extension of the well known summary "card" and phase "card" formats to include other formats describing waveforms, focal mechanisms, equipment, etc.

SUDS is a direct and significant extension of the **ah** or "ad-hoc" format developed at Lamont Doherty Geological Observatory in 1987 and used widely for interactive processing of seismic waveforms. **ah** is a single structure of fixed length that contains three sub-structures of fixed length explaining the attributes of the station, the event, and the waveform. **SUDS** provides for dozens of different structures in arbitrary order.

SUDS has some similarities to **SEED**. In **SEED**, data groups are called blockettes, but unlike the structures in **SUDS**, these blockettes must be stored in a specific order. **SEED** is designed for exchange of raw data and is a subset of **SUDS**.

The format designed for the Center for Seismic Studies (**CSS**) is a relational database model for certain types of analysis of seismic data. The **CSS** format is a subset of **SUDS**.

The modularity and interconnectability of **SUDS** extends the very powerful "shell" concept of UNIX, i.e. pipes and standard input and output, for simple ASCII files to complex files of data. General utilities are being written to act on **SUDS** files or streams in ways similar to UNIX commands such as **grep, sed, sort, etc.**

SUDS utility programs are also being written to provide easy ways to convert to and from major standard fixed formats such as **AH, SEG-Y, CSS, SAC, CUSP** and **SEED**. Thus while ultimately many networks may collect data in **SUDS** format, data from other networks and instruments can be converted readily into **SUDS** format at any stage of processing and merged with other **SUDS** data. Since **SUDS** is a superset of other formats, these filters provide a way to convert between two other formats without loosing information.

TABLE-DRIVEN PROGRAMMING

One of the primary features of **SUDS** is that there are three central tables that contain all relevant information about each variable type, each structure, and each member of each structure. These tables are available to programmers so that programs can be written that work on all structures presently defined or to be defined in the future. These tables are themselves arrays of the structures *variable_info, structure_info, and member_info.*

One example of using these tables is the program **st2asc** that converts all structures from binary to ASCII. **st2asc** reads the *structure_tag* structure that gives the number and length of the following structure. Then it reads the structure and decodes each member by looking up in the table the offset to the beginning of a member, the type of the member, and the format for printing the member in ASCII.

Use of these three tables is an easy way to write utility programs that can work on all structures. In this way fewer specialized programs are needed. The basic subroutines for using the tables are described in Chapter 2, primarily in the section **STRUCTURE_PROPERTIES(2)**.

VARIABLES IN YOUR ENVIRONMENT

SUDS uses the following variables that should be set in your environment. Use **setenv** for **UNIX /bin/csh** or **set** for **DOS**. On the Macintosh, these variables are put in a SUDS file named **suds environment** in the system folder.

SUDS_INCLUDE: Usually **/usr/include/suds** in **UNIX** and **C:\msvc\include\suds** or **C:\c700\include\suds** in **DOS**. This is where the include files are found and it is where the **label** files (See make_lab(1) and get_label(2)) are put that are used to define unique values for **LABELS** within a given **DOMAIN.**

HOME: Your home directory. Typically set by the **/bin/csh** in **UNIX**, but must be set explicitly in **DOS**.

LOGNAME: Your login name. Typically set by the **/bin/csh** in **UNIX**, but must be set explicitly in **DOS**. Used for the database.

READING AND WRITING SUDS STRUCTURES

The **SUDS** library provides subroutines for reading and writing **SUDS** structures easily with all error checking. The programmer simply says in the appropriate language:

1) Open a file, stream, or database for reading
2) Read the next structure, the subroutine returns a pointer to the memory dynamically allocated for the structure
3) Decide what to do with this structure
4) Continue reading structures until the end of file
5) Close the file, stream, or database

To write a **SUDS** file or stream,

1) Open a file, stream, or database for writing
2) Write the structures
3) Close the file, stream, or database

All errors are checked, error messages are given, and the programmer can decide what happens on report of each error.

A PROGRAMMING EXAMPLE

Let’s write a simple program in C that reads **SUDS** data from many files, lists the name of each structure read, and extracts the WAVEFORM and PICK structures for use in a waveform analysis program.

```
/* include file with SUDS structures, defines, and externals */
#include <suds/suds.h>
```

```
/* subroutine called by error subroutines for fatal errors */
die(n)  int n; {exit(n);}

main(argc,argv)
        int argc;
        char **argv;
{
        FILE *input;
        char *next_struct,*data;
        int i,type,data_len,num_waves,num_picks;
        /* declare arrays of pointers to waveforms and picks */
        SUDS_WAVEFORM *wv[100];
        SUDS_PICK *pk[200];

        progname=argv[0]; /* initialize program name for error subroutines */
        num_waves=0;
        num_picks=0;

        for(i=1;i<argc;i++) {
                /* read each file listed after program name */
                input=st_open(argv[i],"r");
                /* for each file read each structure until end of file */
                while(st_get(&next_struct,input)!=EOF) {
                        type=type_of_structure(next_struct);
                        printf("read structure %s\n",name_of_structure(type));
                        switch(type) {
                                case WAVEFORMS: wv[num_waves++]=next_struct; break;
                                case PICKS:         pk[num_picks++]=next_struct; break;
                                default:               st_free[next_struct]; break;
                        }
                }
                st_close(input);
        }
        call waveform_processor(wv,num_waves,pk,num_picks);
}
```

Assignment statements in **C** for each structure member are given in the file <**suds/assigns.txt**>.

DATA TYPES AND MISSING DATA

Members of **SUDS** structures can be of many different types described in **variable_info(3)** and further explained in **st_intro(4)**. These types include characters, long and short integers, floating point, double precision floating point, longitudes/latitudes, and two types of time. Programmers are strongly encouraged to use the typedefs defined in **variable_info(3)** and **suds.h** to assure compatibility.

Any member of a **SUDS** structure can have the value **NODATA** which means no value has been defined for this member. For a number, **NODATA** is distinctly different from zero, since zero could be a reasonable observed value. The numeric value of **NODATA** is different for different data types, but is typically near, but not exactly at a limit for the data type (see **variable_info(3)** and **suds.h**).

SUDS FILE FORMAT AND ORGANIZATION

A SUDS file or stream consists of pairs of SUDS structures. The first structure in each pair is always a **structure_tag**, a special, short structure that serves to identify the type of structure immediately following, so that SUDS files are not dependent upon any particular ordering scheme. Some structures, such as **waveform**, are usually followed by a variable amount of data (in the case of **waveform**, the variable length data are the samples that make up the waveform.) If the second structure in the pair is of a type that is followed by variable length data, it will have members that describe the number of data points or data structures that follow and their type.

All numeric data in SUDS are in binary form. To keep SUDS data files machine-independent, all SUDS data are in XDR format. XDR stands for eXternal Data Representation. XDR specifies a standard for alignment and byte order, and a convention for representing ASCII data in strings. All floating-point members of structures in SUDS and XDR are in IEEE format. Converting the XDR format files to a particular machine's internal binary format is done by the **SUDS** library subroutines when reading or writing a stream. Because certain restrictions are applied to how **SUDS** structures are defined, these structures can be read and written directly, with no conversion required on machines based on the 680X0 or SPARC processors. The **SUDS** manual compiler enforces all of these restrictions.

COMMENTS

Every structure can have a comment of arbitrary length associated with it that describes any or all members (See **comment(4)**).

CODE LISTS

Code lists are used extensively in **SUDS** as a way to standardize commonly used ASCII information in a manner that is efficient for storage and that reduces the chance of operator errors on input. A code list associates a letter or number with an ASCII string. The letter or number is stored in the **SUDS** structure, but the subroutines **get_code(2)** and **list_code** provide easy conversion from and to the ASCII string. **SUDS** utility programs often list the string next to the code and many use a pop-up window to list the strings for choice on inoput. For example, in the code list **instrument_types**, the number 2 represents "sp wwssn" (A short period World Wide Standardized Seismograph Network seismometer) while the number 24 represents an "SMA-1" accelerograph. Code lists are contained in the file **suds_cod.h** and are listed in Appendix I of this manual.

EXCHANGE FORMAT

SUDS structures can be passed between machines via any media writable on one machine and readable on the other. The bit and byte organization is specified by **XDR**. Thus structures can be streamed end to end on a tape, a disk, etc. However, usually structures will be grouped in files and it is most efficient to use a format that retains the file name and grouping. **We strongly encourage that the format to be used for universal exchange is tar(1) or tape archive format used widely on UNIX based systems.** A "tar tape" or file is a series of blocks usually 512 bytes long. The tar representation of a file is a header block which describes the file, followed by zero or more blocks that give the contents of the file. At the end of the tape are two blocks filled with binary zeros. The blocks are grouped for physical I/O operations. Each group of *n* blocks is written with a single system call. The value of *n* is set by the **b** keyletter on the **tar (1)** command line (the default is 20 blocks). The header block is written in ASCII with numbers in octal and is as follows:

```
#define TBLOCK   512
#define NAMSIZ   100
union hblock {
        char dummy[TBLOCK];
        struct header {
                char name[NAMSIZ];      /* file name and path */
                char mode[8];   /* file permissions */
                char uid[8];    /* owner's user identification */
                char gid[8];    /* owner's group identification */
                char size[12];  /* size in bytes */
                char mtime[12]; /* time of last modification */
                char chksum[8]; /* check sum for error detection */
                char linkflag;  /* flag if this is a link */
                char linkname[NAMSIZ];   /*symbolic link name */
        } dbuf;
};
```

See **tar(1)** and **tar(5)** in the **UNIX** manuals for a more detailed description.

PORTABLE PROGRAMMING

Writing code in SUDS that is truly portable among many different types of computers takes only a little extra care. Not taking this care can cause others days of headaches.

INTEGER SIZE: The most common problem arises with the change in integer size between 16- and 32-bit machines. On a 16-bit machine, an int is a short, which is 16 bits. On a 32-bit machine, an int is a long, which is 32 bits. Similarly an integer constant is assumed to be an int so that on a 16-bit machine 9 is a short but 9L is a long. Thus you need to be careful to specify exactly what you want (int, short, or long and 9 or 9L) especially in a call to a subroutine or a function and you must be sure the subroutine or function expects whatever you are calling with. For example, calling a function that expects a long with a constant such as 9 will work on a 32-bit machine and fail on a 16-bit machine. Calling it with 9L will work on both types of machines. Of course an integer value that overflows the storage space (e.g. a short greater than 32767) will fail. System library routines, such as stncmp(), strncpy(), fread(), fwrite(), and others, typically expect ints and will fail if given longs on a 16-bit machine.

PRINTF and SCANF FORMATS: Similarly %ld and %d mean the same on a 32-bit machine but will fail on a 16-bit machine if %d refers to a long or %ld to a short. On almost any machine, %f in scanf will fail for a double, it must be %lf.

CASTING OF POINTERS: All pointers should be explicitly cast if possible. Some compilers provide warnings, others require casting, others may let you hang yourself. When a subroutine call passes a pointer to a function, such as **st_error**, if no function is passed, use

NULL, not 0, since on some machines **NULL** is defined in **stdio.h** as **(CHAR *)0**.

FILE NAMES: Unfortunately MSDOS limits file names to 8 letters (case independent) and a 3-letter suffix. This makes file names rather cryptic. Nevertheless, if portability to MSDOS machines is to be easy, it is best to keep filenames short. Names of files with manual pages in section 3 and 5 of the manual are not shortened since they are in **troff** format not useable in MSDOS and the full filenames are needed to work on UNIX with **man**. Conversion from long names to short names is done by omitting the 5th through eighth characters and all characters after the 12th before the dot. Suffixes are truncated to the first 3 characters. The include file **suds_man.h** contains a cross-reference table between long and short names for manual pages. The program **sudsman** knows how to located appropriate manual pages on different computers.

RELATIONSHIPS BETWEEN STRUCTURES

Structure members ending in **_id** are called keys that identify a particular instance of a structure (See **st_intro(4)**). There are two kinds of keys, primary and foreign. Primary keys usually have the same base name as the table they are in. For example, the event table has a field called **event_id**, which is merely a unique number assigned to each event in that table. The solution table also has a field called **event_id** that identifies for which event these data are a solution. Since the event_id field in the solution table refers to a particular record in the event table, it is called a "foreign" key. In the following diagrams of **SUDS** structures, all primary keys are marked with a capital **P**, and all foreign keys with a capital **F**. The arrows in the diagram always point to a primary key, with the intention of conveying the idea that many foreign keys from the "tail" of the arrow all point to a single record at the "head" of the arrow, namely the one with a primary key.

Event Processing Structures

Access to Network Description

Waveform Data Processing

Miscellaneous Structures

DEFINING NEW STRUCTURES AND MEMBERS

In June of 1994, we intend to establish **SUDS** version 3.0 as an international standard that will be fully supported in the future and will only vary by additions approved by an international standards committee. Version 2.6, described in this manual, is intended to be the beta-test version of 3.0. Any suggested changes should be sent to Peter Ward at the address listed inside the front cover.

While extensions to **SUDS** structures are technically easy to do, they must be done only when clearly required and they must be coordinated to maintain the standard. Additions to code_lists are relatively easy and primarily need to be coordinated to assign unique codes. Additions to structures need to be thought out and debated more carefully. Individuals can use comment structures to store new members of interest while their request for new members or new structures are being debated. This should be done in a standard manner so that a program can later transfer the values from the comment structure to new members. Such use of comment structures should be avoided, however, except in extreme cases because they can easily become non-standard.

When code lists or structures are changed, all programs must currently be recompiled for the new definitions to take effect. Hooks have been included in **SUDS** to allow dynamic update of the tables in future editions.

SUDS is designed so that additions to structures will always be upward compatible. New members may be added to the end of existing structures. When an older, shorter structure is read by **st_get**, the default values of the new members are added to the old structure, automatically updating it. Of course values will need to be assigned to these new members before they can be used by programs that rely on them.

We all have the natural tendency to want to add members and structures that map directly from our existing data formats. After all, these are the variables that we are most familiar with. Many of these formats, however, were constrained by card sizes, printer widths, or tradition and may not map directly into the logical structure of **SUDS**. Any person wishing to define a new structure, should consider whether an existing structure can possibly fit the need. When at all possible, existing structures should be used in order to minimize programming complexity. While many utility programs can work with any structure, most workhorse programs will only utilize a small set of structures. Thus, for example, if there were several structures that described earthquake phase readings, each phase-processing program would need to know about all of these structures or some of these programs would not know how to utilize some of these structures and the data would grow incompatible. THUS TRY TO USE EXISTING STRUCTURES WHENEVER POSSIBLE. Additions and modifications should not be taken lightly and should be viewed as a last resort. Structures may contain members of little interest to you. In some cases these additions may prove useful to you in the future, in other cases they may never be useful. These "useless" members cause little storage and processing overhead in terms of percent of all of the data and do not complicate your programming. Thus they are typically a small price to pay to allow many different people to utilize the same structure for different but related purposes.

SUDS structures come in 4 basic types:

BASIC STRUCTURES contain most information about a logical entity that is time independent or varies very slowly with time. Typically used in a one to many relationship. These structures contain a primary key or LABEL.

ADENDA STRUCTURES contain additional information about a logical entity that is only neede in some cases. For example the **signif_event** structures contains information needed only for large earthquakes. The structures do not contain a primary key because their would only be one instance for a given foreign key and thus the foreign key acts as a primary key.

ASSOCIATIVE STRUCTURES associate two basic structures generally in a time-dependent manner. Typically for a many-to-many relationship between structures. The primary key for these structures is typically a composite of two foreign keys.

DATA STRUCTURES are used for data only following other structures. For example, an array of complex numbers is an array of structures of type complex. Data structures do not have keys since they are associated directly with a main structure.

Structures are defined to specify all generally important information about a logical entity. In defining a new structure, you need to step back from your immediate problem and think generally and broadly about the logical concept. Look for commonality of different needs. Some careful and perceptive thought in defining structures will probably save you problems in the future, and will most certainly save others significant effort.

REFERENCES

Banfill, Robert, 1992, **SUDS, Seismic Unified Data System, Version 1.31,** Small Systems Support, Big Water, Utah, available from the IASPEI PC Working Group, send a request by FAX to 415/858-2599, 27 pages.

Banfill, Robert, 1993, **PC-SUDS Utilities, A collection of programs for routine processing of seismic data stored in the Seismic Unified Data System for DOS (PC-SUDS),** Small Systems Support, Big Water, Utah, 84p.

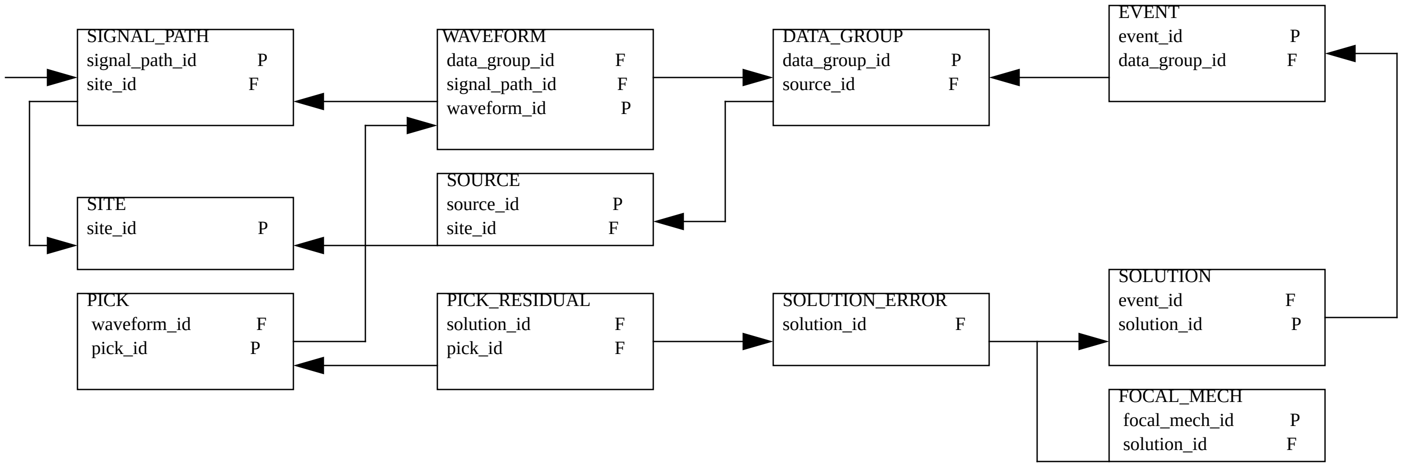
Ward, Peter L., 1989, **SUDS: Seismic Unified Data System,** U.S. Geological Survey Open-File Report 89-188, 123 pages.

Ward, Peter L., 1992, **SUDS: The Seismic Unified Data System,** EOS, Trans. Amer. Geophys. Un., V. 73, No. 35, p. 380.

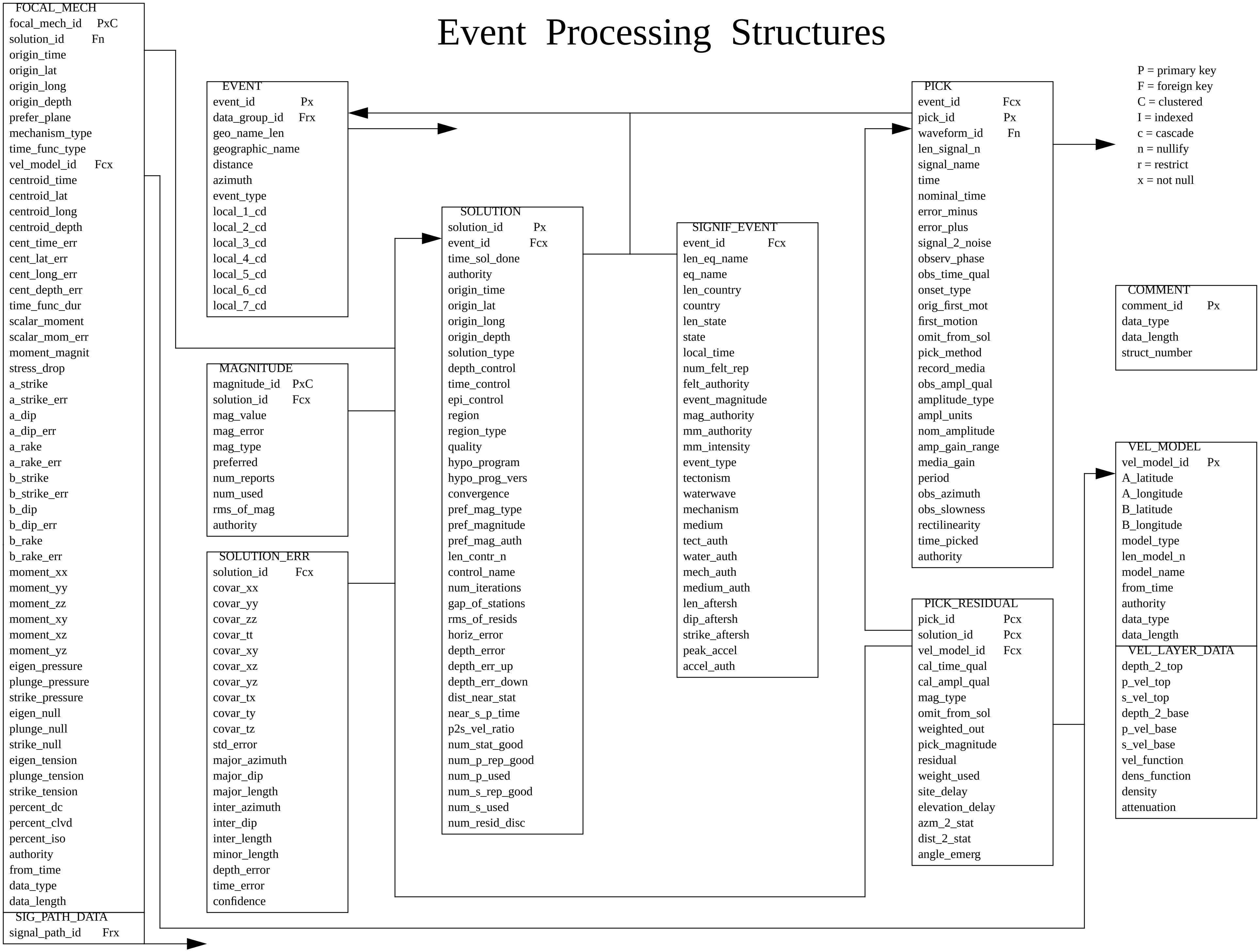
Comparing data formats in seismology

Advantages of SUDS

This page left blank intentionally.


SIGNAL_PATH
signal_path_id P
site_id F
WAVEFORM
data_group_id F
signal_path_id F
waveform_id P
DATA_GROUP
data_group_id P
source_id F
EVENT
event_id P
data_group_id F
SITE
site_id P
SOURCE
source_id P
site_id F
PICK
waveform_id F
pick_id P
PICK_RESIDUAL
solution_id F
pick_id F
SOLUTION_ERROR
solution_id F
SOLUTION
event_id F
solution_id P
FOCAL_MECH
focal_mech_id P
solution_id F

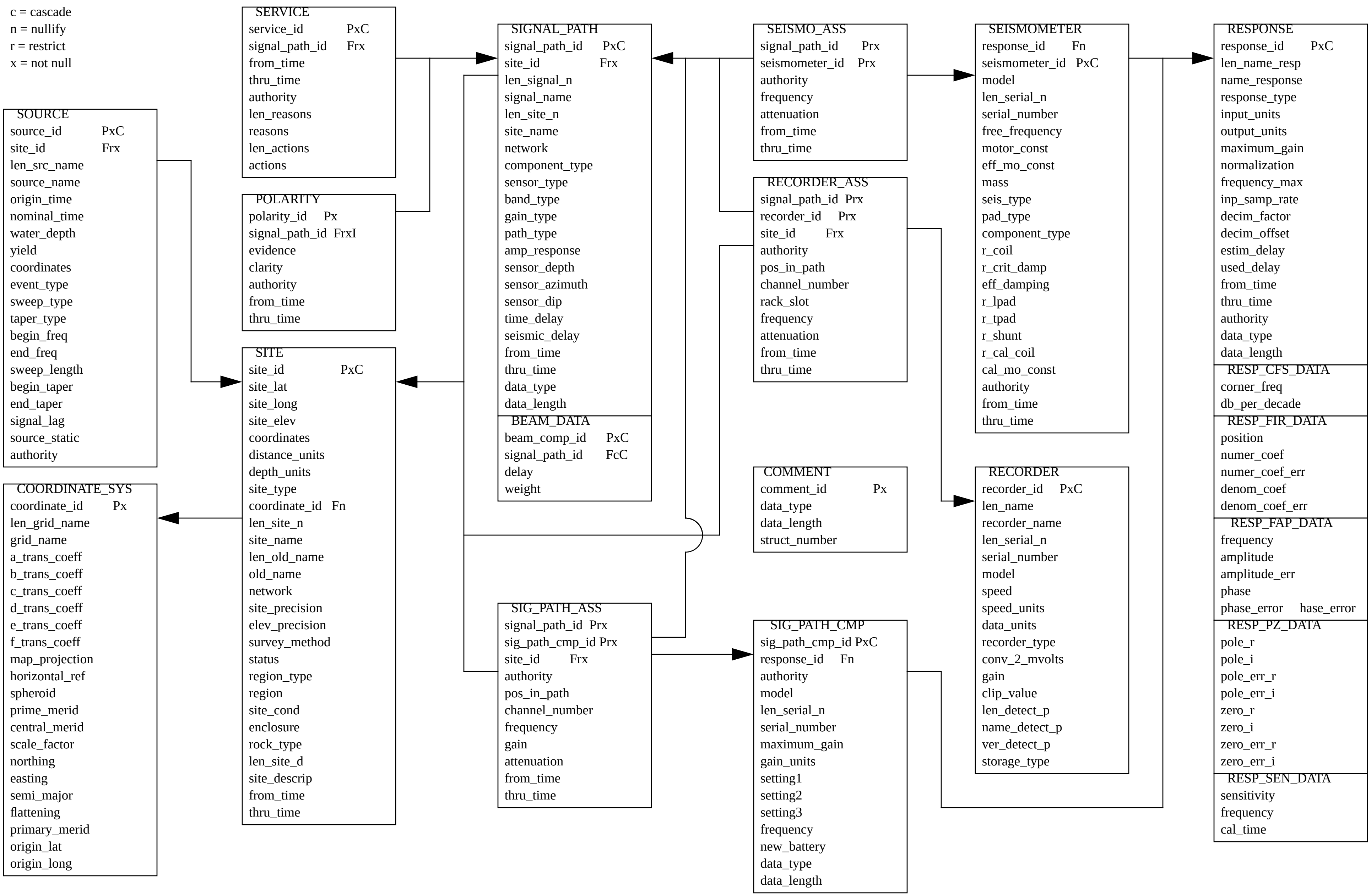
Event Processing Structures



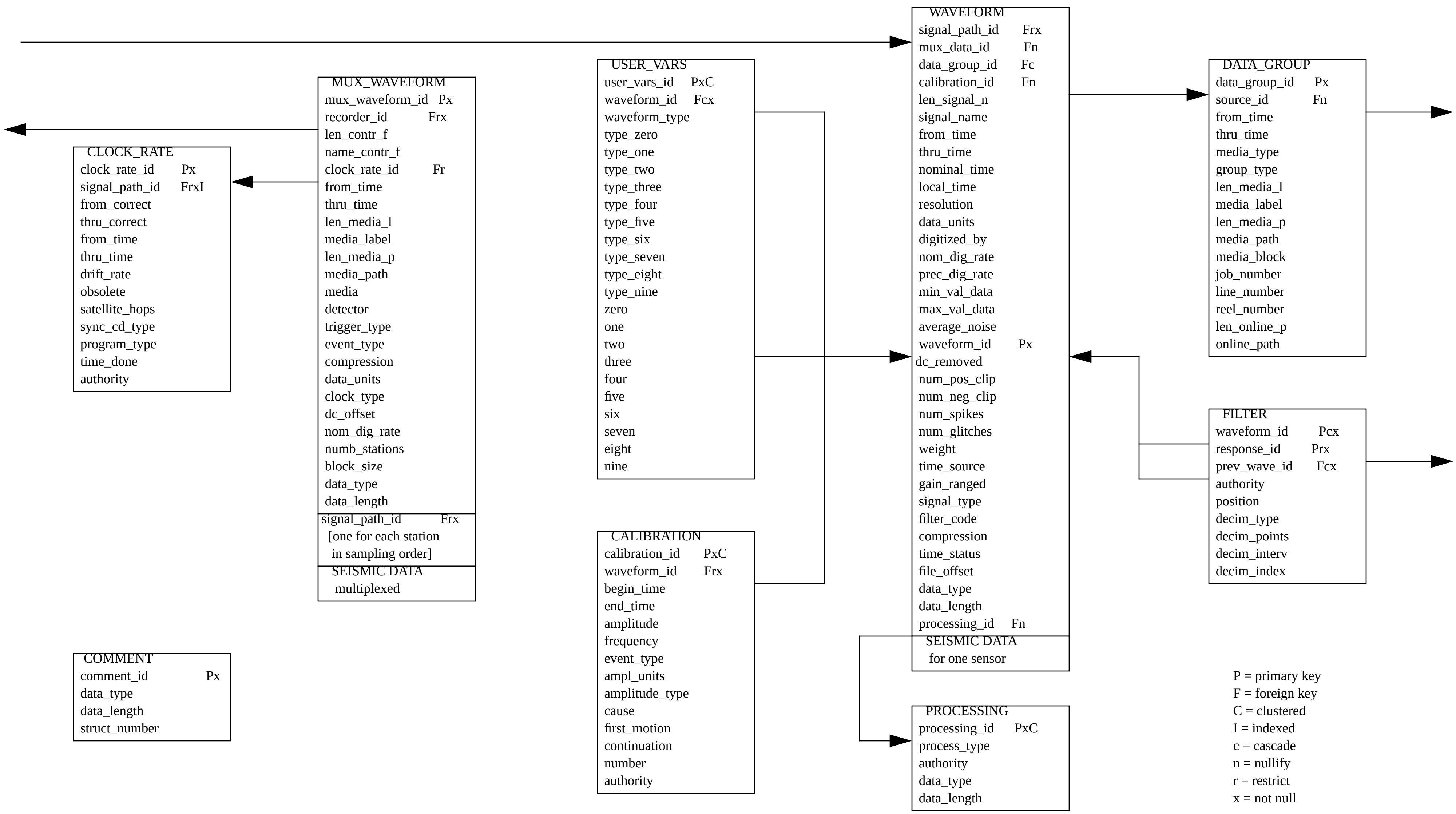
P = primary key
F = foreign key
C = clustered
I = indexed
c = cascade
n = nullify
r = restrict
x = not null

Network Description, Calibration, and Source

Response (Calibration) Information



Waveform Data Processing, Filtering,
Association by Time into Data Groups



Miscellaneous Structures

SPECTRA	
spectra_id	Px
waveform_id	Fcx
spectra_type	
x_units	
y_units	
taper_type	
low_taper_from	
low_taper_to	
high_taper_from	
high_taper_to	
damping	
corner_freq	
prec_dig_rate	
authority	
data_type	
data_length	

MAP_ELEMENT	
map_element_id	PxC
latitude	
longitude	
elevation	
element	
map_source	
map_scale	
time_mapped	
time_encoded	
authority	
importance	
compression	
data_type	
data_length	

SSAM_SETUP	
ssam_setup_id	Px
nom_dig_rate	
num_band_chan	
samp_per_fft	
data_type	
data_length	

SSAM_BAND_DATA	
upper_freq	
lower_freq	

SSAM_DATA	
ssam_setup_id	Fcx
num_band_chan	
data_type	
data_length	

LSA_DETECTION	
detection_id	Px
signal_path_id	Frx
lsa_setting_id	Frx
lsa_onset_time	
amplitude	
frequency	
signal_2_noise	
longterm_ave	
shortterm_ave	
other_ave	
level	
local_1	
local_2	
local_3	
local_4	
local_5	
local_6	
event_type	
first_motion	
num_detections	
authority	

LSA_SETTING	
lsa_setting_id	PxC
algorithm	
decimation	
num_intervals	
start	
before_secs	
after_secs	
begin_level	
end_level	
sweep	
aperture	
constant_1	
constant_2	
constant_3	
constant_4	
constant_5	
constant_6	
constant_7	
authority	
from_time	
thru_time	
data_type	
data_length	

LSA_SET_DATA	
signal_path_id	Frx
subnet	
min_channels	

P = primary key
F = foreign key
C = clustered
I = indexed
c = cascade
n = nullify
r = restrict
x = not null

COMMENT	
comment_id	Px
data_type	
data_length	
struct_number	

SUDS

Chapter 1: Commands

This page left blank intentionally.

NAME

st_intro – **SUDS** commands

DESCRIPTION

Section 1 of this manual describes commands that exist to select and modify **SUDS** structures or to convert to or from other data formats. **SUDS**, the Seismic Unified Data System, consists of an extensible set of structures that associate related variables into logical groups.

In the general case, **SUDS** structures can be thought of as existing in a **stream** or sequence. These commands operate on such a stream either by reading one or more files, by input and output using standard I/O, by a pipe from or to another program perhaps running on another computer, etc.

Basic conversion routines include:

st2asc: Convert suds streams to ascii streams.

asc2st: Convert ascii streams to suds streams.

ah2st: Convert Lamont ah streams to suds streams.

Many other conversion routines are planned including:

st2ah: Convert suds streams to Lamont ah streams.

ping2st: Convert ping (Carl Johnson and University of Washington) streams to suds streams.

st2ping: Convert suds streams to ping streams.

segy2st: Convert SEG-Y format streams to suds streams.

st2segy: Convert suds streams to SEG-Y format streams.

st2seed Convert suds streams to the Standard for Exchange of Earthquake Data.

seed2st Convert SEED data to suds streams.

sac2st Convert data for the Seismic Analysis Code to suds streams.

st2sac Convert suds streams to data for the Seismic Analysis Code.

css2st Convert the data format for the Center for Sesimic Studies to suds streams.

st2css Convert suds streams to the data format for the Center for Sesimic Studies.

st2db: Load suds streams into a database.

db2st: Extract structures from a database into a suds stream.

General commands for handling suds streams include:

stdescribe: List the names and sizes of structures in a suds stream.

stedit: Edit a suds stream.

General commands planned include:

stgrep Extract structures with certain values for certain fields.

stsort Sort structures based on the values for certain fields.

stsubset: Extract a subset of structures from a suds stream.

SEE ALSO

st_intro(2), st_intro(4)

NAME

ah2st – convert an **ah** stream into a **suds** stream

SYNOPSIS

ah2st domain options file_names or directory_names

DESCRIPTION

ah2st converts a stream of **ah** or **ad hoc** structures in the Lamont-Doherty format into **suds** structures. **ah2st** reads data in the original **ah** format for SUN-3 computers. On some computers such as Masscomps and SUN-4s, there are several memory alignment problems with this original format. Therefore the header is read for each segment between bytes where there is a problem, so that this original format can be read on all computers. At Lamont-Doherty, the alignment problem was solved by converting to the eXternal Data Representation (XDR). Data in **ah-xdr** can be read by specifying the **-x** flag.

The **ah** format consists of a header followed by a waveform. The 1024 byte header contains information about the waveform and may also contain information about the station name, station location and instrument type, a calibration, an event hypocenter, an event comment, a waveform comment, a processing comment, and 21 floating-point numbers to be used in any manner desired (See **/usr/include/suds/ahheader.h**). The header-waveform units may be grouped one or more to a file of any name. Typically header-waveform units for an event are either stored in one file with a name with the prefix **ah.** followed by the date or in individual files with names based on the station names typically contained in an event directory with a name based on the date. The date is usually close to the time of the beginning of the waveform in the format of YrMoDyHrMnSc or YrMoDy.HrMnSc.

domain, on the command line, is the integer designating the domain in which the LABELS in **suds** structures are unique. The domain must be included. Use the value **0** if you do not want LABELS defined, but this is not recommended. A fatal error is given if the domain is not defined in the **authorities code_list(6)**.

ah2st processes the options, which may be mixed with file or directory names, and tries to open each name given on the command line as a directory. If this fails, it assumes the name is a file name. If the name is a directory, the directory is assumed to contain files in **ah** format and possibly a file with phase data (see option **-p**). All files with the same name as the directory and a suffix other than **p** or **P** and files whose name begins with a dot (**.**) are ignored.

ah2st assumes that all the data in one file name as typed on the command line or in one directory are for one event, are assigned to one **data_group** in **suds**, and are output in one **suds** file with the name **st.**∗ where ∗ is the name of the input file or directory with the prefix (**ah.**) deleted where appropriate. If a file exists, the new data are added to the end of the file. The structures may be output on **stdout** or in a file of a non-standard name using the **-o** option.

The minimum output of **ah2st** is **waveform** structures if the **-d** or **-e** options are used. Otherwise a **data_group** structure is output before the **waveform** structures.

If any of the **ah** headers that are input contain information about an event, the solution of an event or an event comment, **ah2st** creates and outputs one **event** structure, one **solution** structure, and, if needed, one **comment** structure that is associated with the **solution** structure. If the **-e** option is used, either a **data_group** or an **event** structure is assumed to be in that file and the **data_group_id** and **data_group_dc** are read and used in any **waveform** structures output. With the **-e** option, all event information in the **ah** headers are ignored.

If any of the input **ah** headers contain information about the location of a station, **ah2st** creates one **station** structure and one **signal_path** structure for that station. If the **-s** option is used, the **station** and **signal_path** structures are assumed to be in that file. The **signal_path_id**, **signal_path_dc**, and **signal_name** are read and used in the **waveform** structures output. A fatal error message is given if any of these structures do not exist for a station whose data are included in the **ah** structures.

If any of the input **ah** headers contain information about the calibration of a station, **ah2st** creates one **response** structure and up to 30 **response_pz** structures for that station. If the **-c** option is used, all calibration information in the **ah** structures is ignored.

If **record.comment** or **record.log**, contain information, this information is put in a **comment** structure associated with the **waveform** structure. If any of the **float extra[21]** "Freebies" are not equal to 0.0, they are also added to the same comment structure. The format of the **comment** structure is (See **comment(4)**) {waveform_id} record.comment {processing} record.log {digitized_by} extra[21] in the format of 21 floating point numbers separated by spaces.

OPTIONS

–c Ignor any calibration information in the **ah** structures.

–d number
Assign number to **data_group_id** in the **waveform** structures. The **data_group_dc** will be set to **domain** on the command line. This argument is only needed if you wish to assign these waveforms to an existing **data_group**. Otherwise a unique **data_group_id** will be assigned.

–e file Where file is the name of a file containing an **event** and a **solution** structure. Only the last **event** and **solution** structures in the file will be used. Any event information in the **ah** structures, including any comment, will be ignored.

–n name
Where name is the network name either as the ASCII abbreviation from the **authorities code_list(6)** or as the corresponding number. The default is the number 0 or the name NONE, which mean that no network is specified. A fatal error is given if the name or number is not in the **authorities code_list(6)**.

–o file Put the output in **file**. Output is normally put in a file of the name **st.**∗ where ∗ is the name of the input file or directory with any prefix (**ah.**) deleted. **-o stdout** will put the output on **stdout**.

–p If a file exists in the directoryAll files with the directory name and the suffix **p** or **P**, read the pick information using the subroutine **picks2st(2)** and add the **suds** structures to the end of the output stream.

–s file Where file is the name of a file containing **station** and **signal_path** structures for the waveforms. Any station information in the **ah** structures will be ignored.

–S factor
Convert the waveform to short integers multiplying each point by factor.

–x Input data is in eXternal Data Representation format.

EXAMPLE

Command-line arguments are processed in order. Thus the command

ah2st 10000 -n SUDS -s my.stations -e event1 -d ah.event1 -o stdout -e event2 -d ah.event2

will create the output file st.event1 and put the structures for event2 on **stdout**.

EFFICIENCY

If we assume an **ah** file contains a waveform for 30 seconds at 100 samples per second or 3,000 samples, then since the waveform values are usually stored as floating point numbers, there will be 12,000 bytes of waveform and 1,024 bytes of header. If event, station, and calibration information are not included in the **ah** file, the corresponding **suds** file will be 7% smaller if the waveform is kept in floating point format, and 53% smaller if the waveform is converted to short integer format with the **-S** option.

If the **ah** header contains all possible information, then the corresponding **suds** file would be 8% larger if the waveform is kept in floating point format and 47% smaller if the waveform is converted to short integer format. In **suds**, it is not necessary or appropriate to carry the calibration, station, and event information with every waveform. Thus when the space is added up over all the waveforms in a file system, the savings in space by using **suds** will be closer to the 7% or 53% decrease for just the **waveform** structures discussed above.

SEE ALSO

st_intro(1), st2ah(1)

BUGS

The **ah** variable **rmin** is not translated into the **suds** structures because there does not seem to be a standard definition of what it means.

AUTHOR

Peter L. Ward, U.S. Geological Survey, Menlo Park, CA 94025

NAME

asc2st – convert an ascii stream to a stream of suds structures

SYNOPSIS

asc2st [-o *file*] [*file...*]

DESCRIPTION

asc2st converts a stream of ascii data into **suds** structures. Each input line is assumed to begin with the integer number of the structure type and to contain one data field for each variable within the structure in order. The data fields may be separated by blank spaces, tab characters, or commas. Presently this routine simply reads the non-verbose output of *st2asc.*

OPTIONS

–c The following file contains a control file similar to **/usr/include/suds/suds_descr.h**. The input format and order can be changed from this control file. NOT IMPLEMENTED YET.

–o Put the output in the following file instead of *stdout*.

–s Field separators may only be one of the characters in the following string. NOT IMPLEMENTED YET.

–v Ascii data are in verbose format output by *st2asc*. The input routines will assume any characters on a line before an unquoted colon are part of the field label. The label is matched to the standard list to determine which field follows. Thus lines for fields may be in any order within a structure and may be left out. NOT IMPLEMENTED YET.

–V List input as read. If *asc2st* fails, this option can show on what field it fails.

SEE ALSO

st_intro(2), st2asc(1)

DIAGNOSTICS

BUGS

This routine is presently bare bones and should be expanded to cover a variety of input styles.

EXAMPLES

AUTHOR

Peter L. Ward, U.S. Geological Survey, Menlo Park, CA 94025

NAME

comp_man – create the suds include files from the manual

SYNOPSIS

cd suds/comp_man
make

DESCRIPTION

The **suds** include files **suds.h suds_var.h suds_str.h suds_mem.h suds_cod.h** are compiled from the manual pages. The compiler is in the directory **suds/comp_man** and is controlled by **Makefile.** Type **make** to compile the manual.

The compiler extracts the **typedefs** and variable defines for **suds** variable types from **variable_info.5**. Then it extracts all #**define** statements from the manual. It extracts all **extern** definitions by looking at all subroutines in section 3 of the manual and all code_lists in **code_lists.5**. The compiler determines the machine type and writes a #**define** statement. It then extracts all structure definitions from the **SYNOPSIS** sections of the manual. All of these defines, externs, and structure definitions are combined to form **suds.h**.

suds_var.h and **suds_cod.h** are generated directly from **variable_info.5**.

The manual pages are then scanned by a preprocessor that extracts all of the relevant information and puts it in a formal pattern of **X=Y** in the file **fields**. This latter file is then scanned by a **LEX** and **YACC** parser/compiler to create **suds_str.h suds_mem.h**.

The input is checked in many ways while creating the include files, but further checks of alignment, uniqueness, length, code_lists, keys, etc. are done by the program **check_tabs**.

Finally the program **sizes** is run to print the sizes in bytes of the compiled tables **variables, structures,** and **members** contained in the include files.

By typing **make install**, the include files are copied to **suds/include** and a check is make to be sure a symbolic link exists between **/usr/include/suds** and **suds/include**.

AUTHOR

Peter L. Ward, U.S. Geological Survey, Menlo Park, CA 94025 and Ming Jiang, Department of Computer Science, Geophysical Institute, University of Alaska Fairbanks, Fairbanks, AK 99775-0800.

NAME

dbload – load **SUDS** structures in a **SUDS** database.

SYNOPSIS

dbload dbase_name files

DESCRIPTION

dbload loads **SUDS** structures from one or more files into a database or another file. If *dbase_name* ends in *.db*, then it is assumed to be the name of a database. Otherwise it is assumed to be a file name. It may be *stdout* or *stderr*. *files* is one or more file names to be read. One of these names may be *stdin*.

The pathname to the database and to data files are specified in the file **.suds_defaults** (See **st_intro(2)**).

OPTIONS

SEE ALSO

st_intro(2), st_intro(4), dbsearch(1)

BUGS

AUTHOR

Peter L. Ward, U.S. Geological Survey, Menlo Park, CA 94025

NAME

dbsearch – get suds structures from a **SUDS** database.

SYNOPSIS

dbsearch dbase_name sql_cmd output_file

DESCRIPTION

dbsearch reads **SUDS** structures from a **SUDS** database or another file and puts them in an output file. If *dbase_name* ends in *.db*, then it is assumed to be the name of a database. Otherwise it is assumed to be a file name. It may be *stdin*. The structures are put in *output_file* which may be *stdout* or *stderr*.

The pathname to the database and to data files are specified in the file **.suds_defaults** (See **st_intro(3)**).

OPTIONS

SEE ALSO

st_intro(2), st_intro(4), dbload(1)

BUGS

AUTHOR

Peter L. Ward, U.S. Geological Survey, Menlo Park, CA 94025

NAME

is_suds – determine which files are in SUDS format

SYNOPSIS

is_suds [-s] [-p] [filename...]

DESCRIPTION

List all files and tell whether they are in SUDS format, PC_SUDS format or neither.

OPTIONS

The following options can be used alone or together.

n   No new line after each name.

p   List names only of files in PC_SUDS format.

s   List names only of files in SUDS format.

EXAMPLE

Make a long listing of all SUDS files in this directory:

ls -l `is_suds -s -p -n *`

SEE ALSO

is_suds_file(2)

AUTHOR

Peter L. Ward, U.S. Geological Survey, Menlo Park, CA 94025

NAME

make_lab – create /usr/include/suds/00000.lab

SYNOPSIS

make_lab domain_number

DESCRIPTION

LABELS are unique integers that identify a particular instance of a structure within a given domain. These numbers are assigned in increasing order starting at 1. The largest value assigned to date is kept in a file **00000.lab** where the 00000 is the **domain_number** . This file is stored in the directory specified by the environmental variable **SUDS_INCLUDE**. If this variable does not exist, **make_lab** tries to put the file in the directory **/usr/include/suds**. The format is one long integer in **XDR** binary format for each label in the order listed in the **code_list labels** (See **code_lists(6)**). This file is created with the command **make_lab** and accessed with the subroutine **get_label**.

SEE ALSO

get_label(2)

AUTHOR

Peter L. Ward, U.S. Geological Survey, Menlo Park, CA 94025

NAME

make_rdm – create a **SUDS** database in rdm (DB_Vista).

SYNOPSIS

make_rdm directory database_name structure_names

DESCRIPTION

make_rdm generates the two files necessary to create and access the Raima Data Manager III database, formerly known ad DB_vista. The first file, **database_name.ddl**, is a Data Description Language file used by the DB_Vista command **ddlp** to generate the database. This file specifies the records or structures, the sets, file names, etc. based on the **structure_info** and **member_info** tables in **SUDS**.

The second file, **database_name_suds.h**, contains tables that cross-reference the Db_Vista and SUDS constants for use by the database input/output library.

directory is the name of the directory where the database files will be stored.

The pathname to the database and to data files are specified in the file **.suds_defaults** (See **st_intro(2)**).

OPTIONS

SEE ALSO

st_intro(2), st_intro(4)

BUGS

AUTHOR

Peter L. Ward, U.S. Geological Survey, Menlo Park, CA 94025

NAME

make_syb – generate SYBASE SQL statements needed to create a **SUDS** database.

SYNOPSIS

make_syb -D datadevice size -L logdevice

DESCRIPTION

make_sybase generates a file containing all SYBASE Transact-SQL statements needed to create a SUDS database using SYBASE's "SQL Server" relational database management system.

SUDS datatypes are defined in terms of SQL base datatypes. SQL create table statements are generated for each structure, with FIXED fields eliminated, DOMAIN fields optional, and date and user stamp fields added for insert and last update. All code lists are added as database tables, and range checking rules are bound to all CODE fields. Table-wide and field-specific select, insert, update, and delete permissions are granted/revoked for four different classes of users: public, network_tech, analyst, and manager. Unique indexes are created for all primary keys, and non-unique indexes are created for other frequently searched upon fields. Insert, update, and delete triggers are created for each table. These automatically enforce referential integrity between tables, set date and user stamps, and check convenience fields for correctness. The output is an ascii file which can be further edited before running it through the SYBASE *isql* interactive SQL interpreter to actually create the database.

OPTIONS

–D The following name and size are, respectively, the Sybase device where the database will be created and the database size in megabytes. Available Sybase device names can be found with the Sybase **sp_helpdevice** command. **–L** The following name and size are, respectively, the Sybase device where the database transaction log will be created and the log size in megabytes.

SEE ALSO

st_intro(4)

BUGS

The capability to drop and recreate separate portions of SUDS that change frequently, such as code_lists and their associated range checks, is not yet implemented. The structure_names given on the command line should include all of, and only, the structures which are present in the table diagrams in the introduction to part II of the manual.

AUTHOR

Fred Williams and Mark Anderson, Geophysical Institute, University of Alaska Fairbanks, Fairbanks, AK 99775-0800

NAME

sgy2suds – convert SEGY files into a stream of suds structures.

SYNOPSIS

sgy2suds -res file sgy2suds [-o outputfile] [[-d domain] [-v

DESCRIPTION

sgy2suds converts one or more SEGY files into suds structures. All input SEGY files are combined into a single output file of suds structures. A response file is optional to specify all other command line arguments. sgy2suds.ini should be in the same directory as sgy2suds and contains two lines. These values are only used if they are not specified on the command line or in a response file:
Domain=370000 [replace with local domain number]
Version=3 [replace with prefered default version number]

OPTIONS

-res The following file is the response file. It contains all the command line arguments and is used as an override.

-o The following file is the name of the output file containing the suds structures created. If not specified, it is assumed to be the same root as the first input SEGY file with a ".st" extension.

-s The start time for the output samples. Default is start of each trace.

-e The end time for the outut samples. Default is end of each trace.

-red The reduction velocity to use. Default is no reduction.

-d Number of the local domain. Default is specified in sgy2suds.ini.

-v The SEGY version number. Default is specified in sgy2suds.ini.

-p Generate a profile of the input SEGY files. Creates a default response file.

-?

-h Generate a simple summary of command line arguments.

BUGS

There is no implementation of start and end times or reduction. Profile doesn't do anything but produce the help message.

EXAMPLES

sgy2suds -v 2 mysegy.sgy sgy2suds -o outsuds.st mysegy1.sgy mysegy2.sgy sgy2suds -res mysegy.rsp

AUTHOR

Bruce Kirby, Geological Survey of Canada

NAME

st2ah – convert a **suds** stream into an **ah** stream

SYNOPSIS

st2ah [*files*]

DESCRIPTION

st2ah converts a stream of **suds** structures into **ah or ad hoc** structures in the Lamont-Doherty format. The only structures processed are ORIGIN, STATION and WAVEFORM. The ORIGIN structure, if it exists, must preceed the STATION and WAVEFORM structures and the STATION structure must preceed the WAVEFORM structure. Data types allowed for waveforms are short, long, and float.

SEE ALSO

st_intro(2), ah2st(1)

BUGS

This is a quick hack for moving local network data into sunpick and needs to be made more general. THIS ROUTINE NOT IMPLEMENTED YET IN SUDS 2.0

AUTHOR

Peter L. Ward, U.S. Geological Survey, Menlo Park, CA 94025

NAME

st2asc – convert a stream of suds structures to ascii

SYNOPSIS

st2asc [-c *file*] [-h string] [-i string] [-l number] [-L] [-o *file*] [-s string] [-v] [*file...*]

DESCRIPTION

st2asc converts a stream of **suds** structures from binary to ascii. One structure is output per line. The number of the structure is ouput followed by the length in bytes of the structure, the length in bytes of any ensuing data, and then by the value of each field in the structure in order. The values are separated by a space, characters are included within single quotes, and character strings are included within double quotes.

OPTIONS

–h Place the next argument as a header at the beginning of the line for each new primary structure.

–i Make the next argument the string by which lines are indented.

–l When the output line is greater than the following number, it will be output and the next line will be indented by the indent string.

–L Label each structure with file name, position in the file, and structure name.

–n Only list headers. Do not list data following structures except for comments.

–o Put the output in the following file instead of *stdout*.

–s Separate the fields in the non verbose option by the following string.

–t Print values of structure_tag in addition to structure members.

–v Verbose option. Output each field on one line preceded by the field title. Structures nested within a structure are indented three spaces. Numeric codes defined in **suds_cod.h** are followed by the appropriate explanatory string within brackets.

SEE ALSO

st_intro(2), asc2st(1)

DIAGNOSTICS

BUGS

EXAMPLE

Separate the fields by commas:

st2asc -s "," myfile

AUTHOR

Peter L. Ward, U.S. Geological Survey, Menlo Park, CA 94025

NAME

stdescr – describe the suds structures in a file

SYNOPSIS

stdescr [-o *file*] [*file...*]

DESCRIPTION

stdescr lists the file name followed by a list of structures within the file. The number of the structure from the beginning of the file, counting from 0, is given followed by the structure number, structure name, structure length in bytes, and the length of any ensuing data in bytes. The member of the structure identified by **index_string=true** in Chapter 5 of the SUDS manuals is listed at the end of the line.

OPTIONS

–o Put the output in the following file instead of *stdout*.

SEE ALSO

st_intro(2), st2asc(1)

AUTHOR

Peter L. Ward, U.S. Geological Survey, Menlo Park, CA 94025

NAME

stedit – edit the suds structures in a file

SYNOPSIS

stedit *file...*

DESCRIPTION

stedit displays one structure in a stream at a time and allows changes to be made to the values. Certain characters and values are not allowed for some fields where appropriate based on the value of **allow_char** (See **st_intro(4)**) in the structures **variables** and **members**. The next structure is displayed by pushing the key **F7** or **ESC n**. A new structure may be appended after the current structure by pushing the key **F5** or **ESC a**. The current structure may be deleted by pushing the key **F9** or **ESC d** and typing the letter **y**. Any other character will cancel the delete request. To quit the editor push the key **F2** or **ESC q**. To save the changes type the letter **y**. Any other character will cause all changes during the current session to be deleted. These options are listed on the top line of the display. Error messages are displayed on the second line. The third line shows which structure is being edited.

To change individual members of the structures, press the **Tab, Return,** or the arrow keys. If any character is typed or changed in a field and **Return** is pressed, all characters after the last one typed or changed are deleted. **Tab** is equivalent to a **Return** except that all characters after the last one typed or changed are saved. After a member is changed, if the member is a code or a time variable, the ASCII string for the code or time is displayed after the member within curly brackets.

Note that you may only type within certain parts of the screen. The position of each field is set by the members **ed_row** and **ed_col** (See **st_intro(4)**) in the structure **members**.

stedit writes its output into a temporary file with a name of the form *stedit.XXXXX*. After completion of the input file, the editor asks whether to save the changes. If the answer is yes the temporary file is moved to be the input file.

The editor uses the **members** structure in the include file **suds_mem.h** to get field labels, lengths, types, etc. The codes are listed in **suds_cod.h**

Errors are reported by **st_error(2)** on the device **/dev/console**.

EDITORIAL

This editor is an example of some of the features that might be nice in a real structure editor. It uses **curses** and is thus terminal independent on output but the input from cursor keys and function keys uses SUN conventions and thus is not device independent. UNIX System V **curses** might improve this. If *stedit* does not start properly in a SUN-CMD window. Use Shelltool.

This editor does not allow access to the data following structures. It should be integrated with the Lamont waveform editor to allow display and editing of waveforms in sequence. Waveforms are presently passed through but not noted. It should allow global changes, that is changes of a specific field for all instances of the given structure. It should allow input of code fields as numbers or strings. It should be written in X-windows for versatility and portability.

This general type of editor could improve the quality control of data in seismology immensely and would help enforce a standardization that will allow computer processing of even the ancillary fields. It also provides a way for unskilled operators to get work done reliably.

BUGS

Many. Be patient.

AUTHOR

Peter L. Ward, U.S. Geological Survey, Menlo Park, CA 94025

NAME

sthead – get first few suds structures from a **SUDS** file.

SYNOPSIS

sthead [-n] [-o output_file] [filename...]

DESCRIPTION

sthead reads **n SUDS** structures from files or *stdin*.

OPTIONS

–n Number of structures to get. The default value is 10.

–o file Put output in file.

SEE ALSO

stpart(1)

AUTHOR

Peter L. Ward, U.S. Geological Survey, Menlo Park, CA 94025

NAME

stload – create a **SUDS** database in DB_Vista.

SYNOPSIS

stload output files

DESCRIPTION

stload loads **SUDS** structures from one or more files into a database or another file. If *output* ends in *.db*, then it is assumed to be the name of a database. Otherwise it is assumed to be a file name. It may be *stdout* or *stderr*. *files* is one or more file names to be read. One of these names may be *stdin*.

The pathname to the database and to data files are specified in the file **.suds_defaults** (See **st_intro(2)**).

OPTIONS

SEE ALSO

st_intro(2), st_intro(4)

BUGS

AUTHOR

Peter L. Ward, U.S. Geological Survey, Menlo Park, CA 94025

NAME

stpart – get a sequence of suds structures from a **SUDS** file.

SYNOPSIS

stpart [-begin] [+end] [-o output_file]

DESCRIPTION

stpart reads a sequence of **SUDS** structures from files or *stdin*.

OPTIONS

–begin Number of beginning structure counting from 0. The default value is 0.

+end Number of ending structure counting from 0. The default value is 10.

–o file Put output in file.

SEE ALSO

sthead(1)

AUTHOR

Peter L. Ward, U.S. Geological Survey, Menlo Park, CA 94025

NAME

stsort – sort suds structures

SYNOPSIS

stsort -d -f file -l list

DESCRIPTION

stsort reads the input SUDS files as a single stream, sorts the structures according to the order given by a list of structure names in the **order_file**, and writes the structures to the output_file or stdout if no output_file is specified. If a stream contains one or more TERMINATOR structures, the stream is divided into sub-streams delimited by the beginning of the first input file, the TERMINATOR structures, and the end of the last input file, and each substream is sorted separately.

OPTIONS

–d Do not write out any structures that are in the ordered list.

–f The **order_file** is the filename of a list of SUDS structure names, one per line.

–l A quoted list of structure names separated by spaces follows this argument on the command line, e.g. -l"site signal_path pick"

–o The **output_file** is the filename of an output SUDS file or database. The filenames **stdout** and **stderr** are acceptable. If omitted, stdout is assumed.

–t Terminators (terminator(3)) are used in this file, so sort groups of structures between terminators separately.

AUTHOR

Peter L. Ward, U.S. Geological Survey, Menlo Park, CA 94025

NAME

sudsman – display suds manual pages

SYNOPSIS

sudsman [command, subroutine, or structure names]

DESCRIPTION

sudsman looks for **suds** commands, subroutines, or structures that begin with the words typed on the command line and lists the appropriate manual pages. If no names are given, the main text of the manual is listed. Names may be in upper, lower, or mixed case. Normal use should pipe the output to **more(1)**. For example:

sudsman station stedit | more

Or the similarly

sudsman stat sted | more

To print the output in unix:

sudsman station | lpr

To print the output in dos:

sudsman station > lpt1:

sudsman uses an alphabetized and indexed cross-reference list of names to files found in <**suds/suds_man.h**> which is created by **comp_man(1)** and lists the files from the **suds/man.txt** directories, which are created on a UNIX system using **nroff(1)** and **sed(1)** from the main **suds/makefile**. On **dos** systems, the file names are truncated to 8 characters before the period and 3 characters after the period by the method explained in **texttran(1)**.

AUTHOR

Peter L. Ward, U.S. Geological Survey, Menlo Park, CA 94025

NAME

texttran – translate text files to unix, dos, or mac types

SYNOPSIS

texttran intype to outtype to_directory files....

DESCRIPTION

Each line in a unix file ends with a line feed (**n** or **LF**). Each line in a macintosh file ends with a carriage return (**r** or **CR**). Each line in an msdos file ends in a **r****n** (**CR LF**), except the last line, which ends in a control **Z**. This program copies each **file** to the **to_directory** converting the end-of-line characters from **intype** to **outtype** where types are **unix, dos,** or **mac**.

If the output type is **dos**, then the output file name is shortened to 8 characters or less before the period and 3 characters or less after the period. Shortening before the period is done by removing the 5th through 8th characters and truncating if there are still more than 8 characters. Shortening after the period is done by truncating the suffix after the 3rd character.

EXAMPLE

On a UNIX machine:

cd src
texttran unix to dos /pcfs ∗.c

AUTHOR

Peter L. Ward, U.S. Geological Survey, Menlo Park, CA 94025

SUDS

Chapter 2: Subroutines

This page left blank intentionally.

NAME

st_intro – standard IO package and other subroutines for accessing **SUDS** structures

DESCRIPTION

Section 3 of this manual describes subroutines that ease access to and manipulation of **SUDS** structures. Programs using these subroutines should be compiled with the **-lsuds** flag.

SUDS is built around streams of data where a stream is simply a sequence of structures contained in a file, in a pipe, on a tape, as output from an indexed database, etc. The basic input and output routines for reading and writing the structures are:

st_abort_trans	cancel a pending transaction
st_begin_trans	mark the beginning of a database transaction
st_close	close a stream of SUDS structures
st_command	Send a command to a database management system for execution
st_delete	delete a SUDS structure from a database SUDS stream
st_die	Close all opened SUDS_STREAMS gracefully when a program must terminate
st_end_trans, st_flush	mark the end of a transaction
st_free	deallocate memory occupied by a SUDS structure
st_get, st_load	get the next suds structure and data from a stream
st_index, st_position, st_tell, st_peek	index and reposition a stream of suds structures
st_open	open a stream containing suds structures
st_peek	return structure type of next structure to be read by st_get
st_put, st_put_mux	put a suds structure on a stream
st_seek, st_tell, st_rewind, st_peek	reposition a stream of suds structures
st_unget	push a suds structure back into input stream

These routines are designed to look like standard, buffered I/O except that errors are handled using **st_error**. Fatal errors are reported and then call a user supplied subroutine **die** which may simply call **exit** or may also clear buffers, reset terminal characteristics, etc. **PLEASE NOTE: your must define a routine called die.** It may simply be **INT4 die(n) INT4 n;** {**exit(n)**}**; st_error** provides an easy to use, general error handling capability

WARNING: Do not mix these routines with stdio(3s) or rawio(2) routines for the same file at the same time. Follow **st_open** and other routines with **st_close** before using **open** or **fopen**, and so forth.

Defaults for input and output can be set in the file **suds.def**. When **st_open(3)** is called for the first time, it looks for a file with this name first in the present directory, then if not found it looks in the effective user's home directory, then in the directory specified by the environmental variable **SUDS_INCLUDE**, and if still not found it looks finally in **/usr/include/suds**. Each line in this file is expected to contain three strings. The only values presently allowed are

waveform_path domain path

where waveform_path is a key word, domain is the abbreviation for some domain in the authorities code_list (See code_lists(6)), and path is an absolute pathname to be prefaced to the pathname for waveform files from this domain (See calc_pathname(2)).

Other subroutines for use with **SUDS** structures include:

asc2member, member2asc	convert ascii string to suds member and the inverse
calc_pathname	calculate waveform pathname from data_group_id and data_group_dc
copy_struct	copy a structure
descr_trace	calculate minimum, maximum, average noise, and number of clipped values
get_code, list_code, list_authority–	**get or list codes used in suds structures**
get_label	find next unique value for this label
is_suds_file	determine if file is in SUDS format
isnearf, isneard	test if a floating point variable is equal to NODATF
make_comment, add_comment, replace_comment, get_comment	write and read suds comments
make_mstime, scan_mstime, decode_mstime, list_mstime, get_mstime	suds time and date utilities

make_sig_name	make up the signal_name from its components
set_label	assign label from previous values or return a unique new value
st_error, die	general purpose error reporting and handling
st_init, st_create	initialize or create and initialize a suds structure
st_seek, st_tell, st_rewind, st_peek	reposition a stream of suds structures
structure_properties	get information about the properties of structures

DIAGNOSTICS

Mand of these subroutines return either SUCCESSFUL, FAILED, or IGNORED which are defined as follows:

#define SUCCESSFUL 0
#define FAILED (-2)
#define IGNORED (-9)

Some return FOUND or NOTFOUND which are defined as follows:

#define FOUND (-7)
#define NOTFOUND (-8)

Some return EOF meaning end of file, which is defined in <**stdio.h**> as (-1). Test values often used include

#define TRUE (1)
#define FALSE (0)

SEE ALSO

st_intro(1), st_intro(5)

FILES

/usr/lib/libsuds.a

AUTHOR

Peter L. Ward, U.S. Geological Survey, Menlo Park, CA 94025

NAME

asc2member, member2asc – convert ascii string to suds member and the inverse

C_SYNOPSIS

#include <suds/suds.h>
void asc2member(CHRPTR string, GENPTR member_ptr, INT4 type, CHRPTR format);
void asc2member_num(CHRPTR string, GENPTR struct_ptr, INT4 number);
CHRPTR member2asc(GENPTR member_ptr, INT4 type, INT4 verbose, CHRPTR format,
CHRPTR out_string);
CHRPTR member_num2asc(GENPTR struct_ptr, INT4 number, CHRPTR out_string);

FORTRAN_SYNOPSIS

call asc2member(string,member,type,format)
character*(*) string,format
integer*4 type
character*(*) member2asc
function member2asc(member,type,verbose,format,out_string)
integer*4 type,verbose
character*(*) format,out_string

DESCRIPTION

asc2member converts the ascii *string* to a variable of a specific *type* pointed to by *ptr*, which is typically an address of a member in a **suds** structure. **member2asc** converts the member pointed to by *ptr* to an ascii string. *type* is defined by the "Integer defines for standard variable types" in <**suds/suds.h**>. **asc2member_num** is a form of **asc2member** that only requires the number of the member in the structure counting from 0. **member_num2asc** is a form of **member2asc** that only requires the number of the member in the structure counting from 0. **out_string** must be declared large enough to hold the output string. **member2asc** and **member_num2asc** return pointers to **out_string**.

If **verbose** is not equal to 0, then members of type LONGIT or LATIT will be listed in degrees and minutes (verbose=1) and degrees, minutes, and seconds (verbose=2). For members of type MS_TIME and ST_TIME, verbose is set equal to form in **list_mstime**.

DIAGNOSTICS

member2asc returns a string containing an error message beginning with "ERROR:". **asc2member** reports errors by **st_error**.

EXAMPLE

```
INTV die(n) INTV n; { exit(n); }

main(argc,argv)
  INTV argc;
  CHAR **argv;
{
  CHAR c,temp[40];
  INT2 i2;
  INT4 i4,number;
  INT2TM itm;
  FLOAT8 l;
  FLOAT4 ll;
  SUDS_WAVEFORM wv;

  progname=argv[0];
  printf("\n\nCHECK member2asc and asc2member\n");
  c='c';
```

```
printf("CHAR %c is %s\n",c,member2asc(&c,CHR,0L,"%c",temp));
i2=321;
printf("INT2 %d is %s\n",i2,member2asc((GENPTR)&i2,IN2,0L,"%d",temp));
i2=NODATS;
strcpy(temp,"NODATS");
asc2member(temp,(GENPTR)&l,IN2,"%d");
printf("INT2 %s is %d\n",temp,i2);
i4=123;
printf("INT4 %ld is %s\n",i4,member2asc((GENPTR)&i4,IN4,0L,"%ld",temp));
i4=NODATL;
printf("INT4 %ld is %s\n",i4,member2asc((GENPTR)&i4,IN4,0L,"%ld",temp));
itm=1911*16+0xf;
printf("INT2TM %d or 0x%x or 1911:f is %s\n",
   itm,itm,member2asc((GENPTR)&itm,I2T,0L,"%d:%x",temp));
itm= -1911*16+0xf;
printf("INT2TM %d or 0x%x or -1911:f is %s\n",
   itm,itm,member2asc((GENPTR)&itm,I2T,0L,"%d:%x",temp));
l= -123.1234567;
printf("FLOAT8 %lf is %s\n",l,member2asc((GENPTR)&l,FL8,0L,"%lf",temp));
ll= -123.1234;
printf("FLOAT4 %lf is %s\n",ll,member2asc((GENPTR)&ll,FL4,0L,"%f",temp));

printf("LONGIT: verbose=0: %f is %s\n",
   l,member2asc((GENPTR)&l,LON,0L,"%lf",temp));
printf("LONGIT: verbose=1: %f is %s\n",
   l,member2asc((GENPTR)&l,LON,1L,"%lf",temp));
printf("LONGIT: verbose=2: %f is %s\n",
   l,member2asc((GENPTR)&l,LON,2L,"%lf",temp));
strcpy(temp,"123W  7 24.44412");
asc2member(temp,(GENPTR)&l,LON,"%lf");
printf("LONGIT: %s is %lf\n",temp,l);
strcpy(temp,"123W  7.40740");
asc2member(temp,(GENPTR)&l,LON,"%lf");
printf("LONGIT: %s is %lf\n",temp,l);
strcpy(temp,"-123.123457");
asc2member(temp,(GENPTR)&l,LON,"%lf");
printf("LONGIT: %s is %lf\n",temp,l);

l=get_mstime();
printf("MS_TIME %f verbose=0 is %s\n",
   l,member2asc((GENPTR)&l,MST,0L,"%lf",temp));
printf("MS_TIME %f verbose=1 is %s\n",
   l,member2asc((GENPTR)&l,MST,1L,"%lf",temp));
printf("MS_TIME %f verbose=2 is %s\n",
   l,member2asc((GENPTR)&l,MST,2L,"%lf",temp));
printf("MS_TIME %f verbose=3 is %s\n",
   l,member2asc((GENPTR)&l,MST,3L,"%lf",temp));
printf("MS_TIME %f verbose=4 is %s\n",
   l,member2asc((GENPTR)&l,MST,4L,"%lf",temp));
printf("MS_TIME %f verbose=5 is %s\n",
   l,member2asc((GENPTR)&l,MST,5L,"%lf",temp));
```

```
    printf("MS_TIME %f verbose=6 is %s\n",
      l,member2asc((GENPTR)&l,MST,6L,"%lf",temp));
    printf("MS_TIME %f verbose=7 is %s\n",
      l,member2asc((GENPTR)&l,MST,7L,"%lf",temp));
    printf("MS_TIME %f verbose=8 is %s\n",
      l,member2asc((GENPTR)&l,MST,8L,"%lf",temp));
    printf("MS_TIME %f verbose=9 is %s\n",
      l,member2asc((GENPTR)&l,MST,9L,"%lf",temp));
    l=MINTIME;
    printf("MS_TIME %f verbose=3 is %s\n",
      l,member2asc((GENPTR)&l,MST,3L,"%lf",temp));
    l=MAXTIME;
    printf("MS_TIME %f verbose=3 is %s\n",
      l,member2asc((GENPTR)&l,MST,3L,"%lf",temp));

    strcpy(temp,"686336312.710000");
    asc2member(temp,(GENPTR)&l,MST,"%lf");
    printf("MS_TIME: %s is %lf\n",temp,l);
    strcpy(temp,"911001165832.710");
    asc2member(temp,(GENPTR)&l,MST,"%lf");
    printf("MS_TIME: %s is %lf\n",temp,l);
    strcpy(temp,"911001165832");
    asc2member(temp,(GENPTR)&l,MST,"%lf");
    printf("MS_TIME: %s is %lf\n",temp,l);
    strcpy(temp,"91 10 01 16 58 32.710");
    asc2member(temp,(GENPTR)&l,MST,"%lf");
    printf("MS_TIME: %s is %lf\n",temp,l);
    strcpy(temp,"10/01/91 16:58 32.710");
    asc2member(temp,(GENPTR)&l,MST,"%lf");
    printf("MS_TIME: %s is %lf\n",temp,l);
    strcpy(temp,"Oct 1, 1991 16:58 32.710 GMT");
    asc2member(temp,(GENPTR)&l,MST,"%lf");
    printf("MS_TIME: %s is %lf\n",temp,l);

    wv.structure_type=WAVEFORMS;
    wv.waveform_id=12345;
    number=2;
    printf("Member 2 of waveform structure is %s\n",
      member_num2asc((GENPTR)&wv,number,temp));
    asc2member_num("99999",(GENPTR)&wv,number);
    printf("Member 2 of waveform structure is %s\n",
      member_num2asc((GENPTR)&wv,number,temp));
    exit(0);
}
```

This program produces the following output:

```
CHAR c is c
INT2 321 is 321
INT4 123 is 123
INT4 -2147483640 is NODATL
INT2TM 30591 or 0x777f or 1911:f is 1911:f
```

INT2TM -30561 or 0xffff889f or -1911:f is -1911:f
FLOAT8 -123.123457 is -123.123457
FLOAT4 -123.123398 is -123.123398
LONGIT: verbose=0: -123.123457 is -123.123457
LONGIT: verbose=1: -123.123457 is 123W 7.40740
LONGIT: verbose=2: -123.123457 is 123W 7 24.44412
LONGIT: 123W 7 24.44412 is -123.123457
LONGIT: 123W 7.40740 is -123.123457
LONGIT: -123.123457 is -123.123457
MS_TIME 742576103.760697 verbose=0 is 742576103.760697
MS_TIME 742576103.760697 verbose=1 is 930713150823.761
MS_TIME 742576103.760697 verbose=2 is 930713150823
MS_TIME 742576103.760697 verbose=3 is 93 07 13 15 08 23.761
MS_TIME 742576103.760697 verbose=4 is 93 07 13 15 08 23
MS_TIME 742576103.760697 verbose=5 is 07/13/93 15:08 23.761
MS_TIME 742576103.760697 verbose=6 is 07/13/93 15:08 23
MS_TIME 742576103.760697 verbose=7 is Jul 13, 1993 15:08 23.761 GMT
MS_TIME 742576103.760697 verbose=8 is Jul 13, 1993 15:08 23 GMT
MS_TIME 742576103.760697 verbose=9 is 1993/Jul/13/930713.150824
MS_TIME -2147472000.000000 verbose=3 is MINTIME
MS_TIME 2147472000.000000 verbose=3 is MAXTIME
MS_TIME: 686336312.710000 is 686336312.710000
MS_TIME: 911001165832.710 is 686336312.710000
MS_TIME: 911001165832 is 686336312.000000
MS_TIME: 91 10 01 16 58 32.710 is 686336312.710000
MS_TIME: 10 01 91 16 58 32.710 is 686336312.710000
MS_TIME: Oct 1, 1991 16:58 32.710 GMT is 665427512.710000
Member 2 of waveform structure is 12345
Member 2 of waveform structure is 99999

AUTHOR

Peter L. Ward, U.S. Geological Survey, Menlo Park, CA 94025

NAME

calc_pathname – calculate waveform pathname from data_group_id and data_group_dc

C_SYNOPSIS

#include <suds/suds.h>
CHRPTR calc_pathname(SUDS_WAVEFORM *waveform_ptr, CHRPTR path_buffer);

FORTRAN_SYNOPSIS

character*(*) calc_pathname
function calc_pathname(waveform_ptr,buffer)
character*(*) waveform_ptr,buffer

DESCRIPTION

Put a pathname in *path_buffer* and return a pointer to *path_buffer*. The waveform_ptr must point to a WAVEFORM structure. The pathname returned is **/waveform_path/year/month/day/yrmody.hrmnsc** where **waveform_path** is the default path for this domain specified in the file **suds.def** (see st_intro(2)) or if no default is specified, **waveform_path=/waveforms/domain**

When waveforms are written to a database, the waveform structure is typically stored in the database, but the waveforms are stored in a file with this pathname together with the waveform structures.

SEE ALSO

st_intro(2), st_time(2)

DIAGNOSTICS

Gives an error message and returns a blank or zero length string if data_group_id or data_group_dc are not defined or if the structure pointer does not point to a WAVEFORM structure.

EXAMPLE

```
INTV die(n) INTV n; { exit(n);}

main(argc,argv)
   INTV argc;
   CHAR **argv;
{
   SUDS_WAVEFORM wf;
   CHAR buf[100];
   SUDS_STREAM *ss;

   progname=argv[0];
   st_init(WAVEFORMS,(GENPTR)&wf);
   printf("The following ERROR is expected.\n");
   printf("pathname is (%s)\n",calc_pathname(&wf,buf));
   wf.data_group_dc=10000L;
   wf.data_group_id=(INT4)make_mstime(1992L,9L,25L,22L,45L,15.0);
   printf("data_group_id=%ld\n",wf.data_group_id);
   printf("pathname is (%s)\n",calc_pathname(&wf,buf));
   wf.data_group_dc=52000L;
   printf("pathname is (%s)\n",calc_pathname(&wf,buf));
   exit(0);
}
```

This program produces the following output:

```
ERROR in sun4/cal_path:
cannot calc_pathname when data_group_id undefined
   errno=2: No such file or directory
pathname is ()
data_group_id=717461115
```

pathname is (/suds/gsmen/1992/Sep/25/920925.224515)
pathname is (/waveforms/asro/1992/Sep/25/920925.224515)

AUTHOR

Peter L. Ward, U.S. Geological Survey, Menlo Park, CA 94025

NAME

copy_struct – copy a structure

C_SYNOPSIS

#include <suds/suds.h>
INT4 copy_struct(GENPTR *copy_ptr, GENPTR original_ptr, INT4 data_bytes_out);

DESCRIPTION

copy_struct makes a copy of a structure and any data following the structure. If this type of structure can have data after it, the *data_length* and *data_type* are read from the structure. In this case *data_bytes* is examined, and if it is greater than or equal to 0 and not equal to NODATL, the data associated with the new structure will contain *data_bytes* number of bytes. A pointer to the space created for the copy is returned through *copy_ptr*. Note that *copy_ptr* is the address of a pointer.

DIAGNOSTICS

copy_struct returns SUCCESSFUL or FAILED. Cases of failure include if sufficient space for a copy cannot be allocated or if a null pointer is passed as *original_ptr*.

EXAMPLE

If you want to convert data from INT2 to FLOAT4 and the number of words of INT2 data is 1000, then

```
INTV die(n) INTV n; { exit(n); }

main(argc,argv)
  INTV argc;
  CHAR **argv;
{
  SUDS_WAVEFORM *wv, *wv_new;
  FLOAT4 *new_data;
  INT2 *data;
  INTV i,j,k;
  SUDS_STREAM *out;

  progname=argv[0];
  i=st_create(WAVEFORMS,(GENPTR *)&wv,2000L);
  printf("st_create of waveforms returns %d\n",i);
  wv->data_length=1000;
  wv->data_type=IN2;
  data=(INT2 *)pointer_to_data((GENPTR)wv);
  for(i=0;i<1000;i++)data[i]=i;

  i=copy_struct((GENPTR *)&wv_new,(GENPTR)wv,4000L);
  printf("copy_struct returns %d\n",i);
  data=(INT2 *)pointer_to_data((GENPTR)wv);
  new_data=(FLOAT4 *)pointer_to_data((GENPTR)wv_new);
  wv_new->data_type=FL4;
  for(i=0;i<1000;i++) new_data[i]=data[i];
  printf("Write out short version to cp_short.out\n");
  out=st_open("cp_short.out","wb");
  st_put((GENPTR)wv,NULL,out);
  st_close(out);
  printf("Write out float version to cp_float.out\n");
  out=st_open("cp_float.out","wb");
  st_put((GENPTR)wv_new,NULL,out);
  st_close(out);
```

```
    printf("Compare these files with st2asc or\n");
    printf("    od -i cp_short.out ; od -f cp_float.out\n");
    printf("The 1000 data points after the structure should be the same\n");
    exit(0);
}
```

Outputs the following:

```
CHECK copy_str
st_create of waveforms returns 2184
copy_struct returns 4184
Write out short version to cp_short.out
Write out float version to cp_float.out
Compare these files with st2asc or
    od -i cp_short.out ; od -f cp_float.out
The 1000 data points after the structure should be the same
End test of copy_str
```

AUTHOR

Peter L. Ward, U.S. Geological Survey, Menlo Park, CA 94025

NAME

descr_trace – calculate minimum, maximum, average noise, and number of clipped values

C_SYNOPSIS

#include <suds/suds.h>

INTV descr_trace(FLOAT4 clip_value, SUDS_WAVEFORM *wv, GENPTR trace);

DESCRIPTION

descr_trace calculates and sets the values in **struct waveform** for **min_val_data** (minimum value of trace), **max_val_data** (maximum value of the trace), **average_noise** (average value of the first 200 samples), and **num_pos_clip** (number of positive clipped samples) and **num_neg_clip** (number of negative clipped samples). If the trace type is not short, long, or float, **descr_trace** has a return value of FAILED and no change was made. Otherwise the return value is SUCCESSFUL. If the pointer **trace** is 0, the trace is assumed to be contiguous to the end of **struct waveform.** The number of clipped samples is set only if the **clip_value** is not zero in **struct stationcomp**

AUTHOR

Peter L. Ward, U.S. Geological Survey, Menlo Park, CA 94025

BUGS

This routine needs to be improved.

NAME

get_code, list_code, list_authority– get or list codes used in suds structures

C_SYNOPSIS

#include <suds/suds.h>
INT4 get_code(CHAR *string, INT4 code_list_id);
CHAR *list_code(INT4 code, INT4 code_list_id, INT4 abbrev, CHAR *buffer, INT4 max_length);
INT4 list_authority(CHAR *seed, CHAR *place, INT4 max_len);
SUDS_CODE_LIST *get_code_list_ptr(INT4 type);

FORTRAN_SYNOPSIS

integer*4 get_code
character*(*) listcode
function get_code(string,list)
function list_code(code,list,abbrev,buffer,max_length)
  character*(*) list,buffer
  integer*4 code,abbrev,max_length

DESCRIPTION

Many fields in suds structures contain a **CODE1, CODE2, or CODE4** that represent a specific line of a list described in code_lists(5).

get_code returns the numeric code for a given string. The *list* is searched for a line beginning with all of the characters in *string.* If none is found, **NODATL** is returned. If more than one line is found that matches *string,* **0** is returned. However, if there is a perfect match to one of the lines and the length of the line is at least as long as the length of *string* then the numeric code is returned for that line.

list_code returns a pointer to the *buffer* that is filled with the string describing the *code.* If the code is not found, a string "**this code is undefined**" is returned.

code_list_id is a number assigned to a codelist specified in code_lists(6). It is typically specified as **member_info.code_list_id** but a define constant can also be used that is the name of the code_list in all capitol letters.

If **abbrev** is not equal to 0, then only the abbreviation or letters before the colon are returned. If the abbreviation has more letters than the value of **abbrev**, it is truncated. Thus if abbrev=6, the string returned will contain no more than 6 characters.

codelists(6) are compiled into suds structures and stored in <**suds/codelist.dat**>. They are loaded into memory when needed. The one exception is the codelist **authorities** which comes from the manual section **authorities(6)**. This list is large and changes often. Thus the authorities codelist is sorted by code and by list and indexed and put in the file <**suds/domains.dat**>. This file is then accessed by get_code and list_code as needed.

get_code_list_ptr returns a pointer to the **code_list** structure for a code_list of a given number. The **code_data** structures can then be accessed directly with **pointer_to_data(2)**.

list_authority gets a line from the <**suds/domains.dat**> file whose first letter begins with the first letter of *seed.* The format of the line is the *number* in spaces 0 through 11 and then the *meaning* (See **code_list**(3)). If *seed* begins with a number, **list_authority** returns the first line begining with that number. If *seed* is equal to **NULL**, **list_authority** returns the next line. **list_authority** returns FAILED at the end of the list or if *seed* begins with a character that is neither a number nor a letter. Upper and lower case letters in *seed* are treated the same.

AUTHOR

Peter L. Ward, U.S. Geological Survey, Menlo Park, CA 94025

EXAMPLE

INTV die(n) INTV n; {exit(n);}

```
CHAR seed[]="l";

main(argc,argv)
   INTV argc;
   CHAR **argv;
{
   INT4 i;
   CHAR line[256];
   SUDS_CODE_LIST *cl;
   SUDS_CODE_DATA *cd;

   progname=argv[0];
   printf("Phase p is number %ld in pick_types\n",
      get_code("p",PICK_TYPES));
   printf("Phase s is number %ld in pick_types\n",
      get_code("s",PICK_TYPES));
   printf("Phase f is number %ld in pick_types\n",
      get_code("f finis",PICK_TYPES));
   printf("\nNumber 10000 in code_list authorities is \n%s\n",
      list_code(10000L,AUTHORITIES,0L));
   printf("\n%s\nin code_list authorities is number %ld\n",
      list_code(10000L,authorities,0L),
      get_code(list_code(10000L,AUTHORITIES,0L),authorities));
   printf("\nAbbreviation for number 10000 is \"%s\"\n\n",
      list_code(10000L,AUTHORITIES,6L));
   list_authority(seed,line,256L);
   printf("%s0,line);
   list_authority(NULL,line,256L);
   printf("%s0,line);

   cl=get_code_list_ptr(COMPONENTS);
   cd=(SUDS_CODE_DATA *)pointer_to_data((GENPTR)cl);
   for(i=0;i<cl->number_members;i++)
      printf("'%c' = %s\n",cd[i].number,cd[i].meaning);

   exit(0);
}
```

Produces the following output:

```
Phase p is number 51 in pick_types
Phase s is number 101 in pick_types
Phase f is number 2 in pick_types

Number 10000 in code_list authorities is
gsmen: US Geological Survey, Menlo Park, CA

gsmen: US Geological Survey, Menlo Park, CA
in code_list authorities is number 10000

Abbreviation for number 10000 is "gsmen"

80000      lanl: Los Alamos National Labs, Los Alamos, NM
```

70000　lbl: Lawrence Berkeley Labs, U. C. Berkeley, CA

NAME

get_label – find next unique value for this label

C_SYNOPSIS

#**include <suds/suds.h>**
#**include <suds/00000.lab>**
INT4 get_label(DOMAIN domain, CHAR *name);

FORTRAN_SYNOPSIS

```
integer*4 get_label
function get_label(domain,name)
        integer*4 domain
        character*(*) name
```

DESCRIPTION

LABELS are unique integers that identify a particular instance of a structure within a given domain. These numbers are assigned in increasing order starting at 1. The largest value assigned to date is kept in a file **00000.lab** where the 00000 is the **domain_number** . This file is stored in the directory specified by the environmental variable **SUDS_INCLUDE**. If this variable does not exist, **get_label** tries to access the file in the directory **/usr/include/suds**. The format is one long integer in **XDR** binary format for each label in the order listed in the **code_list labels** (See **code_lists(5)**). This file is created with the command **make_lab** and accessed with the subroutine **get_label**.

get_label returns an integer one larger than the last integer used for a given **name** which must be the name of a label or primary key listed in the code_list labels (See code_lists(6)). A return value of **NODATL** designates an error.

In cases where **get_label** is being used continuously in a filter or similar program, it is much more efficient to leave the **00000.lab** file open. This should be done with care because if someone else accesses the file at the same time, the file would be corrupted, or if your program terminates abnormally, the updated values may not be written back to the disk. To leave the file open use the special call **get_label(domain,"OPEN");** and to close the file use the special call **get_label(domain,"CLOSE");**. These calls return **NODATL**. **Be sure to put the CLOSE call also in die**(st_error(2)).

When the value assigned to LABEL equals -NODATL, a very large positive number, then it is set equal to NODATL+1, a very large negative number.

SEE ALSO

make_lab(1), set_label(2)

AUTHOR

Peter L. Ward, U.S. Geological Survey, Menlo Park, CA 94025

EXAMPLE

```
static INT4 domain=10000;

#if defined (__STDC__) | defined (__cplusplus)
void die(INTV n)
#else
void die(n) INTV n;
#endif
{
   get_label(domain,"event_id");
   exit(n);
}

INTV main()
{
```

```
    INT4 ret;

    ret=get_label(domain,"source_id");
    printf("get_label(%ld,
    ret=get_label(domain,"OPEN");
    ret=get_label(domain,"source_id");
    printf("get_label(%ld,
    ret=get_label(domain,"event_id");
    printf("get_label(%ld,
    ret=get_label(domain,"waveform_id");
    printf("get_label(%ld,
    ret=get_label(domain,"CLOSE");
    return(0);
}
```

NAME

isnearf, isneard – test if a floating point variable is equal to NODATF

C_SYNOPSIS

#include <suds/suds.h>

YESNO isnearf(FLOAT4 x, FLOAT4 y);
YESNO isneard(FLOAT8 u, FLOAT8 v);

FORTRAN_SYNOPSIS

```
integer*4 isnearf,isneard
function isnearf(x,y)
        real*4 x,y;
function isneard(u,v)
        real*8 u,v;
```

DESCRIPTION

Use these functions to test whether one floating-point variable is nearly equal to another floating-point variable. Due to roundoff errors, a simple if(x==NODATF) may not work. Furthermore while floats are passed as doubles on many machines, this is not a portable assumption. These routines check to see if the first variable is within iff of the second variable where diff=second variable times 10 to the minus 7. They return **TRUE** or **FALSE**.

AUTHOR

Peter L. Ward, U.S. Geological Survey, Menlo Park, CA 94025

EXAMPLE

```
INTV die(n) INTV n; {exit(n);}

main(argc,argv)
  INTV argc;
  CHAR **argv;
{
  FLOAT4 x;
  FLOAT8 y;

  progname=argv[0];
  printf("\n\t\t\t\t\t1 means TRUE\n");
  printf("\t\t\t\t\t0 means FALSE\n");
  x=NODATF;
  printf(" %e is near %e\t%d\n",x,NODATF,isnearf(x,NODATF));
  y=NODATF;
  printf(" %e is near %e\t%d\n",y,NODATF,isneard(y,NODATF));
  x=MINTIME;
  printf("\n %e is near %e \t%d\n",x,(FLOAT8)MINTIME,isnearf(x,(FLOAT8)MINTIME));
  y=MAXTIME;
  printf("  %le is near  %le \t%d\n",y,(FLOAT8)MAXTIME,isneard(y,(FLOAT8)MAXTIME));
  x=NODATF;
  printf("\n %e ",x);
  if(x==NODATF)printf("is equal to");
  if(x!=NODATF)printf("is not equal to");
  printf(" %e\n",(FLOAT4)NODATF);
  return(0);
}
```

Produces the following output:

```
                                        1 means TRUE
```

```
                                    0 means FALSE
-1.700000e+36 is near -1.700000e+36     1
-1.700000e+36 is near -1.700000e+36     1

-2.147472e+09 is near -2.147472e+09     1
 2.147472e+09 is near  2.147472e+09     1

-1.700000e+36 is not equal to -1.700000e+36
```

NAME

is_suds_file – determine if file is in SUDS format

C_SYNOPSIS

#include <suds/suds.h>

INT4 is_suds_file(CHAR *file_name);

FORTRAN_SYNOPSIS

integer*4 is_suds_file
function is_suds_file(file_name)
character*(*) file_name;

DESCRIPTION

This function reads the first **structure_tag(3)**, reads the structure length and data length, skips to the second **structure_tag** and reads the first letter. If the first letter of both **structure_tags** is **'S**, then this is most likely a SUDS file. The following values are returned. All values less than or equal to 0 are not likely to be a SUDS or PC_SUDS file.

2 A PC_SUDS file.

1 A SUDS file.

0 Not a SUDS or PC_SUDS file.

-1 Cannot open file.

-2 Trouble reading file.

-3 Unknown computer type for this data.

-4 Unable to seek to second structure_tag.

-5 Unknown byte order for this machine.

AUTHOR

Peter L. Ward, U.S. Geological Survey, Menlo Park, CA 94025

NAME

make_comment, add_comment, replace_comment, get_comment – write and read suds comments

C_SYNOPSIS

#include <suds/suds.h>
SUDS_COMMENT *make_comment(CHAR *suds_structure_ptr, CHAR *member_name, CHAR *comment, DOMAIN domain);
INT4 add_comment(SUDS_COMMENT **comment_ptr_adr, CHAR *member_name, CHAR *comment);
INT4 replace_comment(SUDS_COMMENT **comment_ptr_adr, CHAR *member_name, CHAR *comment);
INT4 get_comment(SUDS_COMMENT *comment_ptr, CHAR *member_name, CHAR *comment, INT4 max_chars);

C_SYNOPSIS

integer*4 make_comment,add_comment, replace_comment, get_comment
function make_comment(suds_structure,member_name,comment,domain)
function add_comment(comment_ptr_adr,member_name,comment)
function replace_comment(comment_ptr_adr,member_name,comment)
function get_comment(comment_ptr,member_name,comment,max_chars)
integer*4 domain, max_chars
character*(*) member_name, comment

DESCRIPTION

Any **suds** structure may have a comment structure (**comment(5)**) associated with it that contains an ASCII string of any length up to 65535 bytes describing the structure or one or more members of the structure. Only one comment is available per instance of a structure. The same comment may refer to many instances of one type of structure. Comments may contain sub-comments that refer to specific members of the structure. Each must be prefaced with {**name**} where **name** is the name of the member. If a sub-comment refers to the whole structure then the preface is {}. Thus a comment about a **pick** structure might be as follows:
"{}analyst is feeling sick today{signal_2_noise}waveform very irregular, possible telemetry spikes{onset_type}quite debatable{gain_range}unable to tell if gain ranged"

These subroutines make it easy to create, modify and access comment structures.

make_comment creates a **comment** structure followed by data with the preface {**member_name**} and the character string **comment**. **make_comment** returns a pointer to this structure with data following. **comment_id** and **comment_dc** are set in the **suds_structure_ptr** to a unique number and to **domain** respectively. **type** specifies the type or define number of the structure. Returns **NULL** if the member_name is not recognized.

add_comment catenates a preface and comment string to an existing comment structure. **Note that the address of the comment_ptr_address is passed.** If the **member_name** already exists as a preface in the comment, the comment string is catenated to that substring without a new preface. The return value is the number of characters in the total comment string. 0 means there was no appropriate substring. **FAILED** means the **member_name** was not recognized.

replace_comment replaces the comment following a preface with the new comment string or, if the preface does not already occur, adds the new preface and comment string to the comment. The return value is the number of characters in the total comment string. **FAILED** means the **member_name** was not recognized.

get_comment fills the string **comment** with the substring for a specific **member_name**. If the substring is longer than **max_chars**, it is truncated to **max_chars**. The return value is the number of characters in the substring. 0 means there was no appropriate substring. **FAILED** means the **member_name** was not recognized. The special member_name "**all**" will fill the string **comment** with the complete comment string. A number for a member_name will return the appropriate substring. Thus "1" fills the comment string with the second substring, counting from 0, including preface in the comment data string.

A comment may contain any ASCII characters except { or } which are reserved for labeling. If either of these characters are included in a comment string passed to any of these routines, they will be converted to a **[** or **]** respectively. If they are put in a comment by other means, they may confuse these routines. Comments that refer to more than one structure member should be created using **make_comment** and several calls to **add_comment**.

add_comment and **replace_comment** create a duplicate comment structure but with a different **data_length**, make the necessary changes, delete the old **comment** structure and return the new pointer. Thus be sure to pass the address of a pointer to a **comment** structure.

SEE ALSO

comment(5)

AUTHOR

Peter L. Ward, U.S. Geological Survey, Menlo Park, CA 94025

EXAMPLE

```
CHAR test[]="{pick_id}not really good pick{observ_phase}may not really be p{time_picked}it was a bad day";

#define MAXCHARS 256L

INTV die(n) INTV n; {exit(n);}

main(argc,argv)
  INTV argc;
  CHAR **argv;
{
  SUDS_COMMENT *co,*com;
  SUDS_PICK *pic;
  INT4 i,begin,end,len,domain;
  CHAR comment[MAXCHARS];

  progname=argv[0];
  domain=10000;

  len=strlen(test)+1;
  if(len%8>0)len=(len/8 +1)*8;
  i=st_create(COMMENTS,(GENPTR *)&co,len);
  co->comment_id=0;
  co->comment_dc=domain;
  co->data_type=CHR;
  co->data_length=strlen(test);
  co->struct_number=PICKS;
  strncpy((CHAR *)pointer_to_data((CHAR *)co),test,(INTV)len);
  printf("Initial comment is   (%s)\n",pointer_to_data((CHAR *)co));
  printf("Gets parts of a comment\n");
  printf("%3ld %14s = (%s))\n",
    get_comment(co,"all",comment,MAXCHARS),"all",comment);
  printf("%3ld %14s = (%s)\n",
    get_comment(co,"pick_id",comment,MAXCHARS),"pick_id",comment);
  printf("%3ld %14s = (%s)\n",
    get_comment(co,"observ_phase",comment,MAXCHARS),"observ_phase",comment);
```

```
    printf("%3ld %14s = (%s)\n",
       get_comment(co,"time_picked",comment,MAXCHARS),"time_picked",comment);
    printf("\nNow make a new comment\n");
    st_create(PICKS,(GENPTR *)&pic,0L);
    com=make_comment((GENPTR)pic,"pick_id","not really good pick",domain);
    printf("%3ld (%s)\n",get_comment(com,"all",comment,MAXCHARS),comment);
    printf("comment structure made with data_length=%ld\n",com->data_length);
    printf("pick.comment_id = %ld pick.comment_dc = %ld\n",
       pic->comment_id,pic->comment_dc);
    printf("comment.comment_id = %ld comment.comment_dc = %ld\n",
       com->comment_id,com->comment_dc);

    printf("\nNow add a comment\n");
    i=add_comment(&com,"observ_phase","really this is junk");
    printf("i=%3ld %3ld %3ld\n\tall = (%s)\n",
       i,get_comment(com,"all",comment,MAXCHARS),
       com->data_length,comment);
    i=add_comment(&com,"pick_id","but certainly interesting");
    printf("i=%3ld %3ld %3ld\n\tall = (%s)\n",
       i,get_comment(com,"all",comment,MAXCHARS),
       com->data_length,comment);
    i=add_comment(&com,"authority","analyst is sick");
    printf("i=%3ld %3ld %3ld\n\tall = (%s)\n",
       i,get_comment(com,"all",comment,MAXCHARS),
       com->data_length,comment);
    printf("data_length=%ld\n",com->data_length);

    printf("\nNow replace a comment\n");
    replace_comment(&com,"pick_id","no, this is not thus");
    printf("%3ld %3ld (%s)\n",get_comment(com,"all",comment,MAXCHARS),
       com->data_length,comment);
    printf("data_length=%ld\n",com->data_length);

    printf("%3ld %3ld  0 = (%s)\n",get_comment(com,"0",comment,MAXCHARS),
       com->data_length,comment);
    printf("%3ld %3ld  1 = (%s)\n",get_comment(com,"1",comment,MAXCHARS),
       com->data_length,comment);
    printf("%3ld %3ld  2 = (%s)\n",get_comment(com,"2",comment,MAXCHARS),
       com->data_length,comment);
    printf("%3ld %3ld 11 = (%s)\n",get_comment(com,"11",comment,MAXCHARS),
       com->data_length,comment);
    printf("\nThe following ERROR is intentional:\n");
    printf("%3ld %3ld \"\" = (%s)\n",get_comment(com,"",comment,MAXCHARS),
       com->data_length,comment);
    return(0);
}
```

Produces the following output:

```
Initial comment is   ({pick_id}not really good pick{observ_phase}may not really be p{time_picked}it was a bad day)
Gets parts of a comment
 91            all = ({pick_id}not really good pick{observ_phase}may not really be p{time_picked}it was a bad day)
 20        pick_id = (not really good pick)
```

```
 19   observ_phase = (may not really be p)
 16    time_picked = (it was a bad day)

Now make a new comment
 29 ({pick_id}not really good pick)
comment structure made with data_length=32
pick.comment_id = 4 pick.comment_dc = 10000
comment.comment_id = 4 comment.comment_dc = 10000

Now add a comment
i= 62  62  64
        all = ({pick_id}not really good pick{observ_phase}really this is junk)
i= 88  88  96
        all = ({pick_id}not really good pick but certainly interesting{observ_phase}really this is junk)
i=114 114 120
        all = ({pick_id}not really good pick but certainly interesting{observ_phase}really this is junk{authority}analyst is si
data_length=120

Now replace a comment
 88  96 ({pick_id}no, this is not thus{observ_phase}really this is junk{authority}analyst is sick)
data_length=96
 29  96  0 = ({pick_id}no, this is not thus)
 33  96  1 = ({observ_phase}really this is junk)
 26  96  2 = ({authority}analyst is sick)
  0  96 11 = ()

The following ERROR is intentional:
ERROR in sun4/make_com:
get_member_info: member () in structure type 21 (pick) unknown
  0  96 () = ()
```

NAME

make_sig_name – make up the signal_name from its components

C_SYNOPSIS

```
#include <suds/suds.h>
CHAR *make_sig_name(SUDS_SIGNAL_PATH *signal_path);
```

FORTRAN_SYNOPSIS

```
character*(*) make_sig_name
function make_sig_name(sig_path)
        record /signal_path/ sig_path
```

DESCRIPTION

Name of a sensor component whose data are transmitted along a specific path and recorded on a particular recorder. Name is expected to be of the form network_station_CSBGP where the network is the abbreviation (part of the authority string preceeding the colon, 5 or less characters) for the **signal_path.network** code, station is **signal_path.station_name** (7 or less characters), C is the **signal_path.component_type** code (usually v, n, or e), S is the **signal_path.sensor_type** code, B is the **signal_path.band_type** code, G is the **signal_path.gain_type**, and P is the **signal_path.path_type** in only those stations where the same component may be recorded on two or more different recorders or transmitted over different paths.

make_sig_name catenates the components of **signal_name**, copies the string into **signal_path.signal_name** and returns a pointer to the string.

AUTHOR

Peter L. Ward, U.S. Geological Survey, Menlo Park, CA 94025

EXAMPLE

```
#include <string.h>
INTV die(n) INTV n; { exit(n);}

main(argc,argv)
  INTV argc;
  CHAR **argv;
{
  SUDS_SIGNAL_PATH sp;

  progname=argv[0];
  st_init(SIGNAL_PATHS,&sp);
  strcat(sp.station_name,"ksv");
  sp.network=10000;
  sp.component_type='e';
  sp.sensor_type='v';
  sp.path_type='s';
  printf("signal_name is (%s)\n",make_sig_name(&sp));
  exit(0);
}
```

Produces the output:

```
signal_name is (ksv.gsmen_evs)
```

NAME

set_label – assign label from previous values or return a unique new value

C_SYNOPSIS

```
#include <suds/suds.h>
INT4 load_label(CHAR **list, INT4 *list_len, GENPTR struct_ptr, CHAR *name,
  INT4 model,INT4 domain);
INT4 set_label(CHAR **list, INT4 *list_len, GENPTR struct_ptr, CHAR *name,
  INT4 model,INT4 domain);
```

DESCRIPTION

LABELS are unique integers that identify a particular instance of a structure within a given domain. These numbers are assigned in increasing order by the subroutine **get_label(2)**. Often when assigning labels for equipment, for example, a label might have already been assigned. These routines create a list of labels including their corresponding domain, an ASCII name, and a model. **load_label** simply loads these values into the list. **set_label** looks up the values in the list, and if no entry with name and model matching is found, **get_label(2)** is called and the a new entry made in the list.

set_label resets the value of **LABEL** and the corresponding **DOMAIN** in the structure. The **LABEL** is always the 3rd member of any structure that has a **LABEL**. Structures that do not have a **LABEL** should not be used with these routines. **load_label** will simply put their 3rd and 4th members in the list, but an error will be returned by **set_label** when it tries to call **get_label(2)** to get a new unique value for a non-existent label.

list points to space dynamically allocated to contain the list. A different pointer and associated **list_len** should be declared for each **LABEL**. **name** is, for example, signal_path.signal_name, sig_path_cmp.serial_number, site.site_name, source.source_name, etc. and is limited to 19 characters plus a null byte in length. If **model** equals **NODATL**, it is not searched. It might be set to sig_path_cmp.model, site.network, etc. **domain** is used by **set_label** when getting a new **LABEL** and assigning the corresponding **DOMAIN**.

set_label returns **FOUND** if the name, model, and domain were found in the list, **SUCCESSFUL** if added to the list, or **FAILED** if there are problems in allocating space, in which case an error message is also printed. **load_label** returns **SUCCESSFUL** or **FAILED**

SEE ALSO

make_lab(1), get_label(2)

EXAMPLE

```
main(argc,argv)
  INTV argc;
  CHAR **argv;
{
  SUDS_SIGNAL_PATH sp;
  CHAR *sp_list;
  INT4 list_len,domain,ret;

  progname=argv[0];
  list_len=argc;
  list_len=0;
  domain=10000;
  st_init(SIGNAL_PATHS,(GENPTR)&sp);
  sp.network=domain;

  strcpy(sp.signal_name,"gsmen_ABCD_very");
  sp.signal_path_id=0;
  sp.signal_path_dc=0;
  ret=set_label(&sp_list, &list_len, (GENPTR)&sp, sp.signal_name,
```

```
        sp.network,domain);
    printf("set_label for %-15s returns %ld label=%ld domain=%ld\n",
        sp.signal_name,ret,sp.signal_path_id,sp.signal_path_dc);

    strcpy(sp.signal_name,"gsmen_ABCD_bad");
    sp.signal_path_id=0;
    sp.signal_path_dc=0;
    ret=set_label(&sp_list, &list_len, (GENPTR)&sp, sp.signal_name,
        sp.network,domain);
    printf("set_label for %-15s returns %ld label=%ld domain=%ld\n",
        sp.signal_name,ret,sp.signal_path_id,sp.signal_path_dc);

    strcpy(sp.signal_name,"gsmen_ABCD_exam");
    sp.signal_path_id=0;
    sp.signal_path_dc=0;
    ret=set_label(&sp_list, &list_len, (GENPTR)&sp, sp.signal_name,
        sp.network,domain);
    printf("set_label for %-15s returns %ld label=%ld domain=%ld\n",
        sp.signal_name,ret,sp.signal_path_id,sp.signal_path_dc);

    strcpy(sp.signal_name,"gsmen_ABCD_ple");
    sp.signal_path_id=0;
    sp.signal_path_dc=0;
    ret=set_label(&sp_list, &list_len, (GENPTR)&sp, sp.signal_name,
        sp.network,domain);
    printf("set_label for %-15s returns %ld label=%ld domain=%ld\n",
        sp.signal_name,ret,sp.signal_path_id,sp.signal_path_dc);

    strcpy(sp.signal_name,"gsmen_ABCD_bad");
    sp.signal_path_id=0;
    sp.signal_path_dc=0;
    ret=set_label(&sp_list, &list_len, (GENPTR)&sp, sp.signal_name,
        sp.network,domain);
    printf("set_label for %-15s returns %ld label=%ld domain=%ld\n",
        sp.signal_name,ret,sp.signal_path_id,sp.signal_path_dc);
    return(0);
}
```

Produces the output:

```
set_label for gsmen_ABCD_very returns 0 label=3153 domain=10000
set_label for gsmen_ABCD_bad  returns 0 label=3154 domain=10000
set_label for gsmen_ABCD_exam returns 0 label=3155 domain=10000
set_label for gsmen_ABCD_ple  returns 0 label=3156 domain=10000
set_label for gsmen_ABCD_bad  returns 0 label=3154 domain=10000
```

AUTHOR

Peter L. Ward, U.S. Geological Survey, Menlo Park, CA 94025

NAME

st_abort_trans – cancel a pending transaction

C_SYNOPSIS

#include <suds/suds.h>

INT4 st_abort_trans(SUDS_STREAM *output_stream);

FORTRAN_SYNOPSIS

```
integer*4 st_abort_trans
function st_abort_trans(output_stream)
        integer*4 output_stream
```

DESCRIPTION

Any set of SUDS structures and accompanying data can be grouped into a *transaction* for entry into a database. The database management system (DBMS) guarantees that either all members of a transaction are stored, or none of them are stored. By this means, transactions can used to assure that data within a database remains logically consistent.

The **st_abort_trans** call is ignored, and SUCCESSFUL is returned, if *output_stream* is a file. If *output_stream* is a database the **st_abort_trans** call tells the DBMS to abort an incomplete transaction in database *output_stream*. This call would normally be used when an error occurs in the user's program and would cause all **st_puts** to *output_stream* since the beginning of the transaction to be deleted from or simply not added to *output_stream*.

SEE ALSO

st_begin_trans(2), st_put(2), st_end_trans(2)

DIAGNOSTICS

Returns SUCCESSFUL or FAILED. Errors are reported by st_error(2). A typical error is to issue an **st_abort_trans** call to a stream where no transaction is pending.

AUTHOR

Mark Anderson, Geophysical Institute, University of Alaska, Fairbanks, AK 99701 and Peter L. Ward, U.S. Geological Survey, Menlo Park, CA 94025

NAME

st_begin_trans – mark the beginning of a database transaction

C_SYNOPSIS

#include <suds/suds.h>

INT4 st_begin_trans(CHAR *trans_id, SUDS_STREAM *output_stream);

FORTRAN_SYNOPSIS

integer*4 st_begin_trans
function st_begin_trans(trans_id,output_stream)
integer*4 output_stream character*(*) trans_id

DESCRIPTION

Any set of SUDS structures and accompanying data can be grouped into a *transaction* for entry into a database. The database management system (DBMS) guarantees that either all members of a transaction are stored, or none of them are stored. By this means, transactions can used to assure that data within a database remains logically consistent.

The **st_begin_trans** call is ignored, and SUCCESSFUL is returned, if *output_stream* is a file. If *output_stream* is a database the **st_begin_transaction** call tells the DBMS that until further notice (see **st_end_trans**, **st_abort_trans**) all **st_put**s to *output_stream* are members of the transaction.

trans_id is a string that identifies this transaction. Some databases write this string in the transaction log file.

SEE ALSO

st_abort_trans(2), st_put(2), st_end_trans(2)

DIAGNOSTICS

Returns SUCCESSFUL or FAILED. Errors are reported by st_error(2).

AUTHOR

Mark Anderson, Geophysical Institute, University of Alaska, Fairbanks, AK 99701 and Peter L. Ward, U.S. Geological Survey, Menlo Park, CA 94025

NAME

st_close – close a stream of SUDS structures

C_SYNOPSIS

```
#include <stdio.h>
INT4 st_close(SUDS_STREAM *io_stream);
```

FORTRAN_SYNOPSIS

```
integer*4 st_close
function st_close(io_stream)
        integer*4 io_stream
```

DESCRIPTION

Close the connection between the calling program and *io_stream* as defined by **st_open**. If this stream is a file, any buffered data are written out before the file is closed and buffers allocated by the standard input/output system are freed. If *io_stream* is a database, any transaction in process will be ended and if this connection is the only one between the user program and the database, then all necessary database logout operations are automatically performed.

st_close is performed automatically for all open streams upon calling **st_die(3).** Any transaction in progress will be aborted.

SEE ALSO

fclose(3s), st_open(2), st_die(2), st_error(2), st_end_trans(2)

DIAGNOSTICS

Returns SUCCESSFUL or FAILED. Errors are reported by *st_error* and then a user supplied subroutine called **die(errno)** is called (See **st_error(2)**).

EXAMPLE

```
#include <suds/suds.h>

SUDS_STREAM *in;

in=st_open("myfile","r");
in=st_close(in);
```

By reassigning the SUDS_STREAM pointer to NULL with **st_close**, you can be sure the pointer can not be used until an **st_open** has been called.

AUTHOR

Mark Anderson, Geophysical Institute, University of Alaska, Fairbanks, AK 99701 and Peter L. Ward, U.S. Geological Survey, Menlo Park, CA 94025

NAME

st_command – Send a command to a database management system for execution

C_SYNOPSIS

#include <suds/suds.h>

INT4 st_command(CHAR *command, SUDS_STREAM *io_stream);

FORTRAN_SYNOPSIS

integer*4 st_command
function st_command(command,io_stream)
character*(*) command
integer*4 output_stream

DESCRIPTION

Send a command, typically expressed in SQL (Structured Query Language), to a database management system (DBMS) for execution. The command normally specifies some action to be taken upon *io_stream* but can be any command acceptable to the underlying DBMS.

An **st_get** call to a DBMS must normally be preceded by an **st_command** call sending an SQL SELECT command to the DBMS.

DIAGNOSTICS

Returns SUCCESSFUL or FAILED. Errors are reported by st_error(2).

If *io_stream* is a file, then *command* is ignored and SUCCESSFUL is returned. Errors are reported by **st_error(3s)**.

AUTHOR

Mark Anderson, Geophysical Institute, University of Alaska, Fairbanks, AK 99701 and Peter L. Ward, U.S. Geological Survey, Menlo Park, CA 94025

NAME

st_delete – delete a SUDS structure from a database SUDS stream

C_SYNOPSIS

#include <suds/suds.h>

INT4 st_delete(GENPTR suds_structure_ptr, SUDS_STREAM *suds_stream);

FORTRAN_SYNOPSIS

integer*4 st_delete
function st_delete(suds_stream)
integer*4 suds_stream

DESCRIPTION

Delete from a database the structure with the same primary key as that referenced by *suds_structure_ptr*. Any data associated with the target structure is also deleted.

A request to delete one structure from a database management system (DBMS) may cause zero, one, or many structures to be deleted. Structures related to the target structure may restrict the deletion of the target structure or they may be deleted along with it in order to preserve logical consistency within the DBMS. Relations are specified by keys where each structure has a unique **primary key** and related structures contain **foreign keys** that refer to a **primary key**. The nature of this relation is specified in the manual page in Chapter 5 describing each structure. Each **foreign key** has a deletion property specified as **db_delete=string** where the string is **restrict, nullify,** or **delete**.

Restrict means that the DBMS will not allow the target structure to be deleted if its specific **primary key** is referenced by at least one **foreign key** in the database. Nullify means that before the target structure is deleted, the DBMS nullifies all **foreign keys** pointing to the target structure throughout the database. Cascade means that if **foreign keys** in other structures point to the specific **primary key** of the target structure, the DBMS will also delete these other structures.

Because **st_delete** acts upon the underlying database rather than directly upon *suds_stream*, the effects of an **st_delete** will not be visible until the next **st_command(2)** to *suds_stream* is issued.

If *suds_stream* is a file, the **st_delete** is ignored and SUCCESSFUL is returned.

SEE ALSO

st_get(2), st_command(2)

DIAGNOSTICS

Returns number of structures deleted or 0 if structure exists but can not be deleted. Returns FAILED if structure does not exist or if the user has only read access to *suds_stream*. Errors are reported by st_error(2).

AUTHOR

Mark Anderson, Geophysical Institute, University of Alaska, Fairbanks, AK 99701 and Peter L. Ward, U.S. Geological Survey, Menlo Park, CA 94025

NAME

st_die – Close all opened SUDS_STREAMS gracefully when a program must terminate

C_SYNOPSIS

#**include <suds/suds.h>**

INT4 st_die();

FORTRAN_SYNOPSIS

integer*4 st_die, value
function st_die()

DESCRIPTION

st_die closes all opened SUDS streams, aborting any transactions in process. This routine should be called by the user supplied routine **die** (See **st_error**).

DIAGNOSTICS

Returns SUCCESSFUL or FAILED

AUTHOR

Mark Anderson, Geophysical Institute, University of Alaska, Fairbanks, AK 99701 and Peter L. Ward, U.S. Geological Survey, Menlo Park, CA 94025

NAME

st_end_trans, st_flush – mark the end of a transaction

C_SYNOPSIS

```
#include <suds/suds.h>

INT4 st_end_trans(SUDS_STREAM *output_stream);
INT4 st_flush(SUDS_STREAM *output_stream);
```

FORTRAN_SYNOPSIS

```
integer*4 st_end_trans,st_flush
function st_end_trans(output_stream)
function st_flush(output_stream)
        integer*4 output_stream
```

DESCRIPTION

Any set of SUDS structures and accompanying data can be grouped into a *transaction* for entry into a database. The database management system (DBMS) guarantees that either all members of a transaction are stored, or none of them are stored. By this means, transactions can used to assure that data within a database remains logically consistent.

If *output_stream* is a database, **st_end_trans** causes the DBMS to definitively store the pending transaction's SUDS structures and associated data in *output_stream*. All data passed by **st_put(2)** to *output_stream* after an **st_begin_trans(2)** call is only stored provisionally until the transaction is terminated normally by **st_end_trans** (or abnormally by **st_abort_trans(2)**).

If *output_stream* is a file, **st_end_trans** flushes any buffers. **st_flush** simply calls **st_end_trans** and is included for logical compatability with the standard IO library.

SEE ALSO

st_begin_trans(2), st_abort_trans(2), st_put(2)

DIAGNOSTICS

Returns SUCCESSFUL or FAILED. Errors are reported by st_error(2).

BUGS

Transactions may not be nested because this feature is not supported by most database management systems.

AUTHOR

Mark Anderson, Geophysical Institute, University of Alaska, Fairbanks, AK 99701 and Peter L. Ward, U.S. Geological Survey, Menlo Park, CA 94025

NAME

st_error, die – general purpose error reporting and handling

C_SYNOPSIS

#include <suds/suds.h>
void die(INTV error);

INTV st_error(void fcn(INTV error), CHAR *format, ...);
extern char *progname;
extern char *st_errout;

FORTRAN_SYNOPSIS

integer*4 die
subroutine st_error(function,format [,arg]...)
function die(value)
integer value
character*(*) format

DESCRIPTION

st_error reports errors on *stderr* and calls *function* before returning. *progname* should be set equal to *argv[0]* in the user's *main*. **st_error** outputs a message "ERROR in progname". The next line is the message given in the *printf(3)* type *format* and by *args*. The third line is the system error associated with *errno* (**INTRO(2)**) if *errno!=0*. *errno* is then reset to 0 and the *function* is called, if it is not equal to *NULL*, with *errnum* as an argument.

function can be **exit(2)**, any user defined function, typically **die**, or *NULL*.

Errors are normally output on **stderr**, however if **st_errout** is set to point at a file pathname before the first call to **st_error**, the errors will be put in that file.

Most st_routines call **st_error**. If the error should be fatal, the function passed is **die**. A user must define this function. A simple definition could be:

INTV die(err) INTV err; { exit(err); }

If the user has set special tty modes, these should be restored in **die**. To cause a core dump call **abort(3)** in **die**.

SEE ALSO

intro(2), st_intro(3)

BUGS

errno is not typically reset by standard UNIX routines and thus could have a spurious value. It is a good idea to set **errno=0** before calling any routines for which you plan to use **st_error** to report the errors.

AUTHORS

Peter Ward and Bruce Julian, U.S. Geological Survey, Menlo Park, CA 94025.

EXAMPLE

```
INTV die(n) INTV n; { exit(n); }
INTV err(n) INTV n; {printf("    function err called with value %d0,n);}
CHAR d[10]="Smile!";

main(argc,argv)
   INTV argc;
   CHAR **argv;
{
   MS_TIME clock;
   CHAR out_strg[48];
   INT2    a = 1;
   INT4    b = 2;
```

```
    FLOAT4 c = 3.0;

    progname=argv[0];
    printf("\nThe following errors are anticipated:\n");
    errno=2;
    st_error(err,"    testing: outputs are %d %ld %5.2f %s",a,b,c,d);
    st_error(NULL,"    Error of type: %d",45);
    return(0);
}
```

Produces the following output when run as "my_program" on a UNIX system:

```
The following errors are anticipated:
ERROR in my_program:
    testing: outputs are 1 2  3.00 Smile!
    errno=2: No such file or directory
    function err called with value 2
ERROR in my_program:
    Error of type: 45
End test of dep_unix
```

NAME

st_free – deallocate memory occupied by a SUDS structure

C_SYNOPSIS

```
#include <suds/suds.h>

INT4 st_free(GENPTR suds_struct_ptr);
```

FORTRAN_SYNOPSIS

```
integer*4 st_free
function st_free(suds_struct_ptr)
        integer*4 suds_struct_ptr
```

DESCRIPTION

Deallocate space in memory occupied by the structure referenced by *suds_struct_ptr*, together with memory occupied by any accompanying data.

SEE ALSO

st_get(2)

DIAGNOSTICS

Returns SUCCESSFUL or FAILED. Errors are reported by st_error(2).

AUTHOR

Mark Anderson, Geophysical Institute, University of Alaska, Fairbanks, AK 99701 and Peter L. Ward, U.S. Geological Survey, Menlo Park, CA 94025

NAME

st_get, st_load – get the next suds structure and data from a stream

C_SYNOPSIS

#include <suds/suds.h>

INT4 st_get(GENPTR *suds_struct_ptr, SUDS_STREAM *input_stream); INT4 st_load(GENPTR suds_struct_ptr, INT4 max_bytes, SUDS_STREAM *input_stream);

FORTRAN_SYNOPSIS

integer*4 st_get, st_bind, st_free
function st_get(struct_ptr,stream)
function st_bind(struct_ptr,record,data)
integer*4 struct_ptr,stream

DESCRIPTION

st_get returns through its arguments a pointer ***suds_struct_ptr** to the next SUDS structure in the stream. If this type of structure can be followed by data, then the data are read according to the structure members typically named **data_length** and **data_type** and specified in the manual as **sets_data_length=true** and **sets_data_type=true** and can be accessed through **ptr_to_data** (see **structure_properties(2)**).

st_load reads a structure into existing storage space containing *max_bytes* for the structure and any data.

If *input_stream* is a database, the **st_get** call must be preceded by an **st_command(2)** call that creates a list of structures to be read by **st_get**. Otherwise **st_get** will return **EOF**.

In FORTRAN, **st_get** returns an integer*4 pointing to the structure. This structure pointer can be used with **st_free** and the functions described in **structure_properties(3)** to determine the type of structure, the type and length of data, etc. Then **st_bind** should be used to put the values into records and data arrays that have been declared in FORTRAN. **st_bind** will cause the structure pointer to be deallocated or freed. Otherwise the function **st_free** should be used to free up memory allocated by the C routine **st_get**.

The properties of the structure read and of any data associated with the structure are best determined using the functions described in **structure_properties(2).**

All structures in a stream from a file are preceeded by a **structure_tag**(5) that tells what structure is coming, how long the structure is, how much data follows the structure, what machine format the data is in, and that contains a magic letter used to confirm that the structure was read correctly. When a stream is opened, the first **structure_tag** is read. When **st_get** is called, the structure is read as well as the next **structure_tag** to be sure that the structure was read correctly.

The return value of **st_get** is the total length in bytes of the structure and the total length in bytes of any data following the structure. When a structure is read from an older version of **suds** that does not have the same length as the current version, the new members are added to the old structure and initialized to their default values by **st_get**. Thus older versions are automatically updated when read. In all cases the length of the structure is defined as the return value minus **bytes_of_data(suds_struct_ptr)** or **length_of_structure(suds_struct_ptr)** (See **structure_properties(2)**).

st_get uses **malloc(3)** to allocate space for the structure. When the structure is no longer needed, the space previously reserved by **malloc(3)** should be freed using **st_free(2)**.

st_get does any necessary conversion of data from **xdr** or other binary format known to SUDS to the binary format for this machine based on the value of **structure_tag.computer_type**. **Suds** structures and data read by **st_get** must either be in **xdr** binary format (Big-endian, IEEE), 80x86 binary format (Little_endian, IEEE) or VAX binary format (Little-endian, VAX float).

Often when multiplexed data are written by an online detection program, the length of data to be written is not known when the STRUCTURE_TAG and MUX_DATA structures must be written. When a MUX_DATA structure is read, if **mux_waveform.length_data = NODATL** and

mux_waveform.data_type != NODATL, then the **SUDS** input routine **st_get** assumes that the data goes to the end of the file. The number of bytes from the end of the structure to the end of the file is determined. Four times **mux_waveform.numb_stations** rounded up to modulus 2 is subtracted to allow for one **signal_path_id** for each station. **structure_tag.len_data** is set equal to the remainder and **mux_waveform.data_length** is set equal to the remainder divided by the length of **mux_waveform.data_type**.

SEE ALSO

st_open(2), st_put(2), st_error(2), st_free(2), structure_properties(2)

DIAGNOSTICS

The most common error is likely to be a **Segmentation Violation** caused by not passing the addresses of **suds_struct_ptr**. Errors are reported by **st_error** and a user supplied subroutine called **die(errno)** is called (See **st_error(3s)**). **st_get** returns **EOF** on end of file and **FAILED** if an error occurred and **die** does not call **exit**.

EXAMPLE

Read a **suds** stream saving only **waveform** and **pick** structures.

```
#include <stdio.h>
#include <suds/suds.h>

FILE    *file_in;
INT4    numin,i,j;
CHAR  *suds_struct_ptr;
SUDS_WAVEFORM     *wv[20];
SUDS_PICK      *pk[100];

i=j=0;
while(numin=st_get((GENPTR *)&suds_struct_ptr,file_in)!=EOF) {
        switch(type_of_structure(suds_struct_ptr)) {
                case WAVEFORMS:     wv[i++]=(SUDS_WAVEFORM *)suds_struct_ptr;
                                    break;
                case PICK:          pk[j++]=(SUDS_PICK *)suds_struct_ptr;
                                    break;
                default: st_free(suds_struct_ptr);
        }
}
```

Note the use of the address of pointers **sp** in **st_get**!

AUTHOR

Mark Anderson, Geophysical Institute, University of Alaska, Fairbanks, AK 99701 and Peter L. Ward, U.S. Geological Survey, Menlo Park, CA 94025

NAME

st_index, st_position, st_tell, st_peek – index and reposition a stream of suds structures

C_SYNOPSIS

```
#include <stdio.h>
struct quick_index {
   INT4 tag_offset;
   INT4 structure_num;
   INT4 third_member;
   INT4 bytes_data;
} table[];
INT4 st_index(CHRPTR file_name, SUDS_STREAM *stream);
INT4 st_position(SUDS_STREAM *stream, INT4 number_of_structure, INT4 read_write);
INT4 st_tell(SUDS_STREAM *stream);
INT4 st_save_idx(CHRPTR file_name,SUDS_STREAM *stream);
INT4 search_index(INT4 structure_num, INT4 third_member, INT4 bytes_data,
    INT4 instance, SUDS_STREAM *suds_in_stream)
#define TO_READ      -827
#define TO_WRITE     -828
```

FORTRAN_SYNOPSIS

```
integer*4 st_index, st_position, st_tell, st_peek, st_save_idx, search_index, readwrite
function st_index(file_name,stream)
function st_position(stream, number_of_structure,readwrite)
function st_tell(stream)
function st_save_idx(file_name,stream)
function search_index(number_of_structure, third_member, bytes_data, instance, suds_in_stream)
        character*(*) file_name
        integer*4 stream, number_of_structure, third_member, bytes_data, instance
```

DESCRIPTION

These routines allow indexed access to a file of **SUDS** structures. Open the file with code such as:

stream=st_open(file_name,"rb")

st_index tries to open a file with the same file_name but with the suffix **st_index** expects this file to contain one **quick_index** structure per SUDS structure in the file opened with **st_open**. If no such file is found or if the modification time on such an index file is older than the modification time of the main file, **st_index** creates an index. In either case the table of **quick_index structures is pointed to by stream->file_index** and is of length **stream->length_index**.

st_position sets the position of the next input or output operation on the *stream*. The new position is just before the nth structure specified by *number_of_structure* counting the first structure in the stream as zero. **read_write** should be set to **TO_READ** to cause the positioning to be after the **structure_tag** or to **TO_WRITE** to cause the positioning to be before the **structure_tag**. **st_position** returns **SUCCESSFUL** or **FAILED**.

st_position undoes any effects of **st_unget (2).**

st_tell returns the offset, i.e. the number of the current structure relative to the beginning of the file associated with the named *stream.*

st_save_idx stores the index in the file *file_name.idx*. If **file_name** ends in **.st**, the **st** is replaced by **idx**. The suffix

The **quick_index** structure is initialized by reading each **structure_tag**(3), skipping 8 bytes and reading the 9th through 12th bytes of the structure, and then skipping to the next **structure_tag**. The 9th through 12th bytes of the structure are assumed to be the third member of the structure since the manual compiler requires the first two members to be *structure_type* and *structure_len* for all non data-only structures. The third member is typically the **LABEL** or primary key.

search_index returns the number of the structure in the file counting from zero, whose structure_number equals *structure_num*, whose third_member equals *third_member*, and whose bytes_of_data equals *bytes_data*. If *third_member* equals NODATL, it is ignored. If *bytes_data* equals 0 or NODATL, it is ignored and if *bytes_data* equals 1, the structure must have some data following it. *instance* should normally be set to 0. If multiple structures satisfy the search criteria, **search_index** returns minus the number of instances found, and individual instances can be found by setting *instance* to equal to 1 for the first instance, 2 for the second, etc. **search_index** returns **EOF** (defined as -1 in stdio.h) if no structure is found with required properties.

SEE ALSO

fseek(3), st_open(2), st_unget(2), stream(3)

DIAGNOSTICS

st_index returns SUCCESSFUL or FAILED. **st_position** returns FAILED for improper seeks, otherwise SUCCESSFUL. An improper seek can be, for example, an **st_position** done on a file that has not been opened via **st_open.**

These routines return an error if the file_name is a database.

AUTHOR

Peter L. Ward, U.S. Geological Survey, Menlo Park, CA 94025

EXAMPLE

```
#include <suds/suds.h>

INTV die(n) INTV n; {exit(n);}

main()
{
   SUDS_STREAM *in;
   INT4 i,struct_number,position;
   STRING indexed_name[24];
   struct quick_index  *index;
   INTV nargc;
   INT4 structure_num, third_member, bytes_data, instance;

   nargc=argc;
   progname=argv[0];
   in=st_open("test_idx.st","rb");
   position=st_index("test_idx.st",in);
   printf("st_index returns %ld\n",position);
   index=(struct quick_index *)in->index_to_file;
   printf("length index=%ld\n",in->length_index);
   printf("     File_offset  Struct_num  3rd_member  Bytes_data\n");
   for(i=0;i<in->length_index;i++)
      printf("%3d: %11ld %11ld %11ld %11ld\n",i,
         index[i].tag_offset,index[i].structure_num,
         index[i].third_member,index[i].bytes_data);
   struct_number=3;
   st_position(in,struct_number,TO_READ);
   printf("%2ld %10ld\n",struct_number,
      in->struct_number);
   struct_number=10;
   st_position(in,struct_number,TO_READ);
   printf("%2ld %10ld\n",struct_number,
      in->struct_number);
   structure_num=300;
```

```
    third_member=2;
    bytes_data=0;
    instance=0;
    i=search_index(structure_num, third_member, bytes_data, instance,in);
    printf("search_index(%ld,%ld,%ld,%ld) returns %ld\n",
       structure_num, third_member, bytes_data, instance,i);
    structure_num=324;
    third_member=1;
    instance=0;
    i=search_index(structure_num, third_member, bytes_data, instance,in);
    printf("search_index(%ld,%ld,%ld,%ld) returns %ld\n",
       structure_num, third_member, bytes_data, instance,i);
    third_member=1;
    instance=3;
    i=search_index(structure_num, third_member, bytes_data, instance,in);
    printf("search_index(%ld,%ld,%ld,%ld) returns %ld\n",
       structure_num, third_member, bytes_data, instance,i);
    third_member=1;
    instance=7;
    i=search_index(structure_num, third_member, bytes_data, instance,in);
    printf("search_index(%ld,%ld,%ld,%ld) returns %ld\n",
       structure_num, third_member, bytes_data, instance,i);
    third_member=150;
    instance=0;
    i=search_index(structure_num, third_member, bytes_data, instance,in);
    printf("search_index(%ld,%ld,%ld,%ld) returns %ld\n",
       structure_num, third_member, bytes_data, instance,i);
    st_save_idx("test_idx.st",in);
    st_close(in);
}
```

NAME

st_init, st_create – initialize or create and initialize a suds structure

C_SYNOPSIS

#include <suds/suds.h>
void st_init(INT4 type, GENPTR pointer);
INT4 st_create(INT4 type, GENPTR *pointer, INT4 data_bytes);

FORTRAN_SYNOPSIS

integer*4 st_create
subroutine st_init(type,structure)
function st_create(type,structure,data_bytes)
integer*4 type, data_bytes

DESCRIPTION

st_init initializes all fields of a **suds** structure of a given **type** to the default values defined in the **member_info(5)** stored in the file **suds_mem.h** This initialization is important since fields with no data should be initialized to one of the constants **NODATS, NODATL, NODATF, NOTIME, NOCHAR, NOSTRG,** (see variable_info(5)) and defined in the file **suds.h.**

st_create uses *malloc* to create space for a new structure and then initializes the structure and returns the number of bytes created.

data_bytes is the length of data in bytes that follow the structure when appropriate.

st_create causes the program to exit via a user supplied subroutine called **die** (st_error(3)) if the structure **type** is unknown or the program is unable to **malloc** space.

st_init causes the program to exit via a user supplied subroutine called **die** (st_error(3)) if the structure **type** is unknown.

NOTE that **pointer** in **st_create** is the address of a pointer. The most common problem in using this subroutine will be leaving the **&** out! Note you must use **(GENPTR *)** to keep ANSI C happy.

EXAMPLE

```
#include <suds/suds.h>
INT4 i;
SUDS_SIGNAL_PATH *sp;
        i=st_create(SIGNAL_PATHS,(GENPTR *)&sp,0);
```

AUTHOR

Peter L. Ward, U.S. Geological Survey, Menlo Park, CA 94025

NAME

st_open – open a stream containing suds structures

C_SYNOPSIS

#include <stdio.h>
SUDS_STREAM *st_open(CHAR *filename, CHAR *mode);

FORTRAN_SYNOPSIS

integer*4 st_open
function st_open(filename, type)
    character*(*) filename, type

DESCRIPTION

st_open opens the *filename* and associates a **SUDS_STREAM** handle with it. The *filename* is assumed to be a database name if it ends in ".db. The first **st_open** call to a database system performs all necessary login operations, using the operating system effective username as the database username. No password is supplied to or expected by the database system.

In FORTRAN the return value is an integer, which is the address of the SUDS_STREAM structure. This integer is simply for use in other functions such as **st_get, st_put, st_close** and should not be printed since its value is large and not particularly meaningful.

mode is a character string having one of the following values:

rb open for reading

wb open for writing; if *io_stream* is a file, truncate the file if it exists and create it if it does not exist.

ab append: open for writing if *filename* is a database (same as *w*); open for writing at end of file, or create for writing, if *filename* is a file.

r+b open for reading and writing at beginning of an existing file.

w+b open for reading and writing; if *io_stream* is a file, truncate the file if it exists or create the file if it does not exist.

a+b append: open for reading and writing if *filename* is a database (same as *w*); open for writing at end of file, or create for writing, if *filename* is a file.

IMPORTANT: Use of the update option is dangerous. If you write over an existing SUDS structure with a new structure that is longer or shorter, the file will become unreadable.

The second or third letter of the modestring must be a **b** for binary input or output. On many computers, the default mode is text, which means the end-of-line characters may be translated on input to **\n** and on output to whatever is standard on the machine. Thus if you are opening a file for reading or writing SUDS structures, you must specify the binary default. Actually to protect from human falability, the **b** is added by st_open if you leave it off.

st_open may be called with *filename* equal to "**stdin**", "**stdout**", or "**stderr**" to allow easy specification of stdio defaults for files. It is not necessary to specifically open stdio streams.

Defaults for input and output can be set in the file **suds.def**. When **st_open(3)** is called for the first time, it looks for a file with this name first in the present directory, then if not found it looks in the effective user's home directory, then in the directory specified by the environmental variable **SUDS_INCLUDE**, and if still not found it looks finally in **/usr/include/suds**. See **st_intro(3)**.

SEE ALSO

fopen(3s), st_close(2), st_put(2), st_get(2)

DIAGNOSTICS

st_open returns a **NULL** pointer on failure. Errors are reported by *st_error* and a user supplied subroutine called *die(errno)* is called (See *st_error(3s)*).

BUGS

In order to support the same number of open files as the system does, *st_open* must allocate additional memory for data structures using *malloc* when each file is opened. This might confuse some programs which use their own memory allocators.

AUTHOR

Mark Anderson, Geophysical Institute, University of Alaska, Fairbanks, AK 99701 and Peter L. Ward, U.S. Geological Survey, Menlo Park, CA 94025

NAME

st_peek – return structure type of next structure to be read by st_get

C_SYNOPSIS

#include <suds/suds.h>
INT4 st_peek(SUDS_STREAM *stream);

FORTRAN_SYNOPSIS

integer*4 st_peek
function st_peek(stream)
integer*4 stream, number_of_structure,index

DESCRIPTION

st_peek returns the structure_type member of the next structure to be read by **st_get.**

Returns EOF for no structure waiting to be read. Returns FAILED and prints error message if stream is NULL or not opened for reading.

Works for reading data from a file, database, stdio, etc.

SEE ALSO

st_get(2), st_index(2),stream(3)

AUTHOR

Peter L. Ward, U.S. Geological Survey, Menlo Park, CA 94025

NAME

st_put, st_put_mux – put a suds structure on a stream

C_SYNOPSIS

```
#include <stdio.h>
#include <suds/suds.h>
INT4 st_put(GENPTR suds_struct_ptr, GENPTR data_ptr, SUDS_STREAM *output_stream);

INT4 st_put_mux(GENPTR data_ptr, INT4 data_type, INT4 data_len, SUDS_STREAM *suds_out_stream);
```

FORTRAN_SYNOPSIS

```
integer*4 st_put
function st_put(record,data,output_stream)
        integer*4 output_stream
```

DESCRIPTION

st_put writes the structure pointed to by **suds_struct_ptr** onto the **output_stream**.

If **output_stream** is a file, **st_put** creates a **structure_tag(3)** and adds it to the file immediately ahead of the structure. Part or all of the structure may be buffered until **st_flush(2)** or **st_close(2)** is called.

If **output_stream** is a database and a transaction is not in effect (i.e. **st_begin_trans(2)** has not been called), the structure is written directly to the database. If a transaction is in effect, the structure will not be committed to the database until **st_end_trans** is called.

If this type of structure can be followed by data, then the data are written according to the structure members typically named **data_length** and **data_type**. These members must be specified in the manual as **sets_data_length=true** and **sets_data_type=true**. The data are assumed to begin in memory at the first byte after the end of the structure, but **data_ptr** may also point to an array located elsewhere in memory. If **data_ptr** is not equal to the address of the first byte after the structure and is not equal to **NULL**, then the data are read from the specified address. The data are padded with null bytes to end on an 8-byte boundary. **structure_tag.length_data** includes the pad characters but **data_length** in the structure does not.

st_put returns the total number of bytes written in the structure plus the data following the structure.

A special problem occurs with online earthquake detectors that write multiplexed data while an event is being detected. Often buffering limitations require that the **MUX_DATA** structure be written before an event has ended. Thus **st_put_mux** is provided to write additional data after a **MUX_DATA** structure. On reading these data later, **st_get** assumes the data extends to the end of the file if **mux_waveform.length_data = NODATL** and **mux_waveform.data_type != NODATL**.

SEE ALSO

st_open(2), st_close(2), st_get(2), structure_properties(2)

DIAGNOSTICS

Most errors cause **st_put** to call **die** (See **st_error(2)**). If **die** does not call **exit**, **st_put** returns **FAILED** when an error has occurred, but may not continue properly, depending on the error.

EXAMPLE

```
#include <stdio.h>
#include <suds/suds.h>

SUDS_WAVEFORM *wv;

INT2 wave[1000];
INT4 out_num

out_num=st_put(wv,wave,stdout);
```

AUTHOR

Mark Anderson, Geophysical Institute, University of Alaska, Fairbanks, AK 99701 and Peter L. Ward, U.S. Geological Survey, Menlo Park, CA 94025

NAME

make_mstime, scan_mstime, decode_mstime, list_mstime, get_mstime – suds time and date utilities

C_SYNOPSIS

#include <suds/suds.h>
MS_TIME make_mstime(INT4 year, INT4 month, INT4 day, INT4 hour, INT4 min, FLOAT8 second);
MS_TIME scan_mstime(CHRPTR field);
INT4 decode_mstime(MS_TIME time, INT4 *year, INT4 *month, INT4 *day, INT4 *hour, INT4 *min, FLOAT8 *second);
STRING *list_mstime(MS_TIME time, INT4 form, CHAR *out_string);
MS_TIME get_mstime();

FORTRAN_SYNOPSIS

double precision make_mstime, get_mstime
function make_mstime(year,month,day,hour,min,second)
function scan_mstime(field)
function decode_mstime(time,year,month,day,hour,min,second)
function list_mstime(time,form,out_string)
function get_mstime()
integer*4 year,month,day,hour,min,form
real*4 second
double precision time
character*(*) field
character*36 out_string

DESCRIPTION

All times and dates in **suds** are kept in terms of Greenwich Mean Time (GMT) either as **ms_time**, millisecond time, a double precision decimal number of seconds since 00:00:00 GMT Jan. 1,1970 (8 bytes of storage) or as **st_time**, stamp time, a long integer representation of the same value (4 bytes of storage). **ms_time** has a resolution of 5 microseconds between 1900 and 2040 AD.

These routines provide simple conversion to and from other forms of time. **year** is a four digit number such as 1988, **month** may be 1-12, **day** may be 1-31, **hour** may be 0-23, **min** may be 0-59, and **second** is a double precision number.

make_mstime returns an **ms_time** variable.

scan_mstime returns an **ms_time** variable from a string in one of the forms discussed under OPTIONS.

decode_mstime returns through pointers the components of the time variable.

get_mstime returns the computer's clock as an **ms_time** variable.

list_mstime fills the *out_string* with time in one of several options specified by *form* and returns a pointer to that string. The length of *out_string* should be 36.

OPTIONS

form may be an integer representing one of the following formats where yr=year (2 digits), mo=month (1-12), dy=day (1=31), hr=hour (0-23), mn=minute (0-59), and sc=second (0=59):

0 a floating-point number

1 yrmodyhrmnsc.000

2 yrmodyhrmnsc

3 yr mo dy hr mn sc.000

4 yr mo dy hr mn sc

5 mo/dy/yr hr:mn sc.000 GMT

6 mo/dy/yr hr:mn sc GMT

7 month_name dy, year hr:mn sc.000 GMT

8 month_name dy, year hr:mn sc GMT

9 year/month_name/day/yrmody.hrmnsc

10 yrmody.hrmnsc

If only a few digits are given, the remaining digits are assumed to be 0, except if month is not specified, it is assumed to be 1. Thus 79 is equivalent to 790100000000.000

When time is a nodata value of MINTIME or MAXTIME (see variable_info(5)), these words are printed with additional spaces to make up a string of the proper length for each of the above options.

Note when only 2 digits are used to represent the year, there is an ambiguity for years > 1999. Thus the negative of the value **NOTIME** (See variable_info(5)) is given as 01/19/38 which is January 19,2038.

SEE ALSO

gettimeofday(2), ctime(3), time(3C)

DIAGNOSTICS

Errors are reported by **st_error** and the routines return a zero value or NULL pointer.

EXAMPLES

```
INTV die(n) INTV n; { exit(n); }

main(argc,argv)
   INTV argc;
   CHAR **argv;
{
   FLOAT8 tim,tim1,sec;
   INT4 i,year,month,day,hour,min;
   STRING time_str[36];

   progname=argv[0];
   tim=get_mstime();
   printf("system time = %lf\n",tim);
   for(i=0;i<9;i++)printf("%s\n",list_mstime(tim,i,time_str));
   decode_mstime(tim,&year,&month,&day,&hour,&min,&sec);
   printf("year=%ld month=%ld day=%ld hour=%ld min=%ld sec=%f\n",
      year,month,day,hour,min,sec);
   tim1=make_mstime(year,month,day,hour,min,sec);
   printf("remade time is %lf which differs by %lf\n",tim1,tim1-tim);
   day=yrday(month,day,isleap(year,GREGORIAN));
   printf("day of year is %ld\n",day);
   tim1=make_mstime(year,0L,day,hour,min,sec);
   printf("remade time is %lf which differs by %lf\n",tim1,tim1-tim);
   printf("\n");

   year=1900;
   month=1;
   day=1;
   hour=0;
   min=0;
   sec=0.0;
   tim1=make_mstime(year,month,day,hour,min,sec);
   printf("time for %02ld/%02ld/%02ld %ld:%02ld %6.2lf is %lf\n",
      month,day,year-1900L,hour,min,sec,tim1);
   exit(0);
```

```
}
```

Which will produce output dependent on the date but looking like this:

```
system time = 742575595.424494
742575595.424494
930713145955.424
930713145955
93 07 13 14 59 55.424
93 07 13 14 59 55
07/13/93 14:59 55.424
07/13/93 14:59 55
Jul 13, 1993 14:59 55.424 GMT
Jul 13, 1993 14:59 55 GMT
year=1993 month=7 day=13 hour=14 min=59 sec=55.424494
remade time is 742575595.424494 which differs by 0.000000
day of year is 194
remade time is 742575595.424494 which differs by 0.000000
time for 01/01/00 0:00   0.00 is -2208988800.000000
```

AUTHORS

Bruce Julian and Peter L. Ward, U.S. Geological Survey, Menlo Park, CA 94025

NAME

st_unget – push a suds structure back into input stream

C_SYNOPSIS

#include <stdio.h>
INT4 st_unget(GENPTR st_ptr, SUDS_STREAM *stream);

FORTRAN_SYNOPSIS

integer*4 st_unget
funtion st_unget(st_ptr, stream)
character*(*) st_ptr;
integer*4 stream;

DESCRIPTION

st_unget pushes the structure pointed to by *st_ptr* back onto an input stream. That structure will be returned by the next *st_get* call on that stream. **st_unget** leaves the file *stream* unchanged.

Only one structure may be put back on the stream.

An **st_seek (2)** erases all memory of pushed back structure.

SEE ALSO

st_get(2), st_seek(2)

DIAGNOSTICS

Errors are reported by **st_error. st_unget** returns **EOF** if it can't push a structure back and *ST_ERR* (defined in st_error.h) if *stream* is not open.

AUTHOR

Peter L. Ward, U.S. Geological Survey, Menlo Park, CA 94025

BUGS

NOT PRESENTLY IMPLEMENTED

NAME

structure_properties – get information about the properties of structures and their members

C_SYNOPSIS

#include <suds/suds.h>
INT4 type_of_structure(GENPTR ptr_to_structure);
INT4 length_of_structure(GENPTR ptr_to_structure);
INT4 type_of_data(GENPTR ptr_to_structure);
INT4 words_of_data(GENPTR ptr_to_structure);
INT4 bytes_of_data(GENPTR ptr_to_structure);
CHAR *pointer_to_data(GENPTR ptr_to_structure);
CHAR *name_of_structure(GENPTR ptr_to_structure);
SUDS_STRUCTURE_INFO *get_structure_info(GENPTR ptr_to_structure);
SUDS_MEMBER_INFO *get_member_info(GENPTR ptr_to_structure, CHAR *member_name);
SUDS_VARIABLE_INFO *get_variable_info(INT4 type);
INT4 get_size_of(INT4 type);
INT4 get_type_from_name(CHAR *structure_name);
INT4 width_of_field(SUDS_MEMBER_INFO member_info);

FORTRAN_SYNOPSIS

integer*4 type_of_structure, length_of_structure, type_of_data
integer*4 words_of_data, bytes_of_data, get_size_of, type
integer*4 get_type_from_name,width_of_field
function type_of_structure(ptr_to_structure)
function length_of_structure(ptr_to_structure)
function type_of_data(ptr_to_structure)
function words_of_data(ptr_to_structure)
function bytes_of_data(ptr_to_structure)
character *pointer_to_data(ptr_to_structure)
  character *ptr_to_structure;
function get_size_of(type)
function get_type_from_name(CHAR *structure_name);
function width_of_field(ptr_to_member_info);
  character *member_info;

DESCRIPTION

These functions provide easy access to the properties of any structure. They also provide access to the central tables of **SUDS** (**member_info, structure_info, variable_info**). **These central tables should only be accessed through these functions** because these tables may be maintained on different computers in a database, a caching system, or some other computer-dependent form.

Given a pointer to a structure (**ptr_to_structure**), these functions return the following information:

type_of_structure returns the structure define number or **NODATL** if **ptr_to_structure** is **NULL**.

length_of_structure returns the total number of bytes in the structure, not including data or **NODATL** if **ptr_to_structure** is **NULL**.

type_of_data returns the type of data following the structure. NODATL means no data follow this structure or ptr_to_structure is **NULL**.

words_of_data returns the length of data following the structure in words, 0 if no data follow this structure, and **NODATL** if **ptr_to_structure** is **NULL** or structure type is unknown.

bytes_of_data returns the length of data following the structure in bytes, 0 if no data follow this structure, and **NODATL** if **ptr_to_structure** is **NULL** or structure type is unknown.

pointer_to_data returns a pointer to the first byte following this structure whether data exists or not. It returns **NULL** if **ptr_to_structure** is **NULL** or structure type is unknown.

name_of_structure returns the name of a structure. If the structure is not known or if **ptr_to_structure** is **NULL**, it returns NOSTRG.

get_type_from_name returns the type of a structure, given its name. If name is unrecognized, it returns **NODATL**.

get_structure_info returns a pointer to a **structure_info** structure with information about the structure pointed to by **ptr_to_structure**. This function may also be called with the address of an INT4 containing the structure_type. Errors are reported by **st_error** and in this case, this function returns **NULL**.

get_member_info returns a pointer to a **member_info** structure with information about the member with **member_name** in the structure pointed to by **ptr_to_structure**. This function may also be called with the address of an INT4 containing the structure_type in place of the **ptr_to_structure**. Errors are reported by **st_error** and in this case, this function returns **NULL**.

get_variable_info returns a pointer to a **variable_info** structure with information about the variable of a given **type** (which is variables[i].define_num). Errors are reported by **st_error** and in this case, this function returns **NULL**.

get_size_of returns the length of the variable or structure in bytes. If type is negative, it refers to the define types in **variable_info(3)**. If type is positive it refers to a structure define number and returns the struct_length (See structure_info(3)). IF the type is unknown, returns **NODATL**.

width_of_field returns the number of ASCII spaces needed to list this member.

EXAMPLE

```
INTV die(n) INTV n; { exit(n); }

main(argc,argv)
   INTV argc;
   CHAR **argv;
{
   SUDS_WAVEFORM *wf;
   INT4 i;
   SUDS_STRUCTURE_INFO *si;
   SUDS_MEMBER_INFO *mi;
   SUDS_VARIABLE_INFO *vi;

   progname=argv[0];
   i=st_create(WAVEFORMS,(GENPTR *)&wf,200L);
   wf->data_type=FL4;   /* FL4 = -13 */
   wf->data_length=50;
   printf("Type    of structure = %ld\n",type_of_structure((GENPTR)wf));
   printf("Length of structure = %ld\n",length_of_structure((GENPTR)wf));
   printf("Type    of data = %ld\n",type_of_data((GENPTR)wf));
   printf("Words  of data = %ld\n",words_of_data((GENPTR)wf));
   printf("Bytes of data = %ld\n",bytes_of_data((GENPTR)wf));
   printf("Pointer to structure = %ld pointer to data = %ld\n",
      wf,pointer_to_data((GENPTR)wf));
   printf("Pointer to data - Pointer to structure = %ld\n",
      (INT4)((INT4)pointer_to_data((GENPTR)wf)-(INT4)wf));
   si=get_structure_info((GENPTR)wf);
   if(si!=NULL) printf("The first 4 members of %s structure are:\n",
      si->struct_name);
   for(i=0;i<4;i++) printf("    %s\n",si->member_table[i].member_name);
   mi=get_member_info((GENPTR)wf,"num_spikes");
   printf("Member waveform.%s has the title \"%s\"\n",
      mi->member_name,mi->member_title);
```

```
    vi=get_variable_info(LON);
    printf("For variable type %s, the minimum value is %s\n",
       vi->variable_name,vi->min_value);
    exit(0);
}
```

The output of this program is:

```
Type   of structure = 13
Length of structure = 184
Type   of data = -13
Words  of data = 50
Bytes of data = 200
Pointer to structure = 0x46ea8 pointer to data = 0x46f60
Pointer to data - Pointer to structure = 184
The first 4 members of waveform structure are:
   structure_type
   structure_len
   waveform_id
   waveform_dc
Member waveform.num_spikes has the title "number of spikes"
For variable type LONGIT, the minimum value is -180.
```

AUTHOR

Peter L. Ward, U.S. Geological Survey, Menlo Park, CA 94025

SUDS

Chapter 3: System Structure Descriptions

This page left blank intentionally.

NAME

st_intro – introduction to SUDS system structures

OVERVIEW

The **S**eismic **U**nified **D**ata **S**ystem (**SUDS**) is based on organizing data into *structures or groups that provide all of the important information about a logical entity* such as an earthquake hypocentral solution, a seismic waveform, a phase pick, or the frequency response of a component or system. These structures or groups of data can then be accessed and utilized efficiently and stored in a machine-independent manner, in any order, on any type of storage device.

A structure consists of an ordered list of members. Multiple instances of a structure can be thought of as a table where there is one column for each member and one row for each instance of the structure. Some members of some structures refer to specific instances of other structures, providing a way to relate, for example, specific phase picks to a specific earthquake solution, a specific waveform to a specific recorder, etc. Thus **SUDS**, besides being a data format, is a model or plan for a relational database system.

Chapter 3 of the manual provides a description of structures used for system tasks. These include the central tables **variable_info.3, structure_info.3, member_info.3** and a structure **stream.3** defined whenever a **SUDS** file is opened.

See the introduction to chapter 4 for a detailed discussion of variable types, etc.

LIST OF STRUCTURES

Structures currently defined by the **SUDS** standard are as follows:

code_data	list relating number or letter codes to ASCII strings
code_list	code_list representation in a SUDS stream
comment	comment about structure contents
file_index	structure to index locations of structures in a file
gui_default	user-set defaults to control graphical user interface
member_info	system information about a members of a SUDS structure
stream	describes a SUDS input/output stream
structure_info	system information about a SUDS structure
structure_tag	tag that identifies the next structure in a stream
terminator	structure to end a group of structures in a stream
variable_info	system information about a variable used in SUDS structures

PROPERTIES OF THE STRUCTURES

NAME	NUMBER	BYTES	MEMBERS	
code_data	214	8	2	data only structure
code_list	205	64	12	data may follow
comment	211	32	8	data may follow
file_index	204	64	11	
gui_default	206	32	11	
member_info	202	208	34	
stream	203	88	25	
structure_info	201	128	21	
structure_tag	212	16	6	
terminator	213	24	6	
variable_info	200	192	22	

SEE ALSO

4. External Data Representation: Sun Technical Notes and 5. External Data Representation Standard: Protocol Specification in **Network Programming Guide**

st_intro(1), st_intro(2), st_intro(4)

FILES

/usr/include/suds/suds.h

/usr/include/suds/suds_mem.h

/usr/include/suds/suds_str.h

/usr/include/suds/suds_var.h

AUTHOR

Peter L. Ward, U.S. Geological Survey, Menlo Park, CA 94025

NAME

code_data – list relating number or letter codes to ASCII strings

C_SYNOPSIS

```
typedef struct {
        INT4        number;
        CHRPTR      meaning;
} SUDS_CODE_DATA;

#define CODE_DATAS  214L
```

DESCRIPTION

A list that follows a **code_list** structure and relates number or letter codes to ASCII strings. Any member of a SUDS structure of type **CODE1, CODE2,** or **CODE4** is a number or letter that references a codelist.

Codelists are used in SUDS to save space and to enforce standardization of ASCII strings for ease of searching. They allow ASCII strings of arbitrary length to be used, thus promoting clarity. In utility programs such as **st2asc** or **stedit**, the ASCII string can be printed next to the code number for clarity.

The SUDS library subroutines **asc2field** and **field2asc** provide a simple means to convert ASCII strings to codes and codes to ASCII strings.
[data_only=RESPONSES]

MEMBERS

number *number*

Number of the code.

meaning *ASCII string*

Pointer to the ASCII string corresponding to the code.

EXAMPLE

```
SUDS_CODE_DATA misc[] = {
        123,        "Acme M-4 geophone",
        'p',        "P or primary wave",
  };
```

SEE ALSO

asc2member(2), member2asc(2), code_lists(6)

NAME

code_list – code_list representation in a SUDS stream

SYNOPSIS

```
typedef struct {
        FIXED       structure_type;
        FIXED       structure_len;
        LABEL       code_list_id;
        DOMAIN      code_list_dc;
        FIXED       len_code_name;
        STRING      code_list_name[20];
        INT4        list_number;
        INT4        number_members;
        CODE4       data_type;
        INT4        data_length;
        REFERS2     comment_id;
        DOMAIN      comment_dc;
} SUDS_CODE_LIST, *LIST;

#define CODE_LISTS          205L
```

DESCRIPTION

Code lists are used throughout SUDS to relate characters or numbers that are stored in SUDS structures to descriptive strings. The **code_list** structure is the representation of code lists on a disk or in any SUDS stream. Code lists are loaded dynamically as needed. The **authorities** code list is very large, resides in a file, and is never loaded directly. Thus code lists should always be accessed through the subroutines **get_code** and **list_code**.

Data following the **code_list** structure consists of **number_members** instances of the **code_data** structure followed by all of the code strings concatenated together. **code_data.meaning** is then the offset in bytes from the beginning of the data to the beginning of a particular, null-terminated string. When a **code_list** structure is read into memory by **get_code(2)** or **list_code(2)**, the address of the first byte of data in memory is added to **code_data.meaning**. **data_type** is specified as **CHR** even though the **code_data** structures are in binary. **data_length** is the total number of bytes for the **code_data** structures plus the total number of bytes of concatented strings rounded up to an 8-byte boundary.

MEMBERS

structure_type *structure type*
Define number of this type of structure.

structure_len *structure length*
Length of this structure in bytes.

code_list_id *code_list id*
A number that uniquely identifies, within this **code_list_dc**, an instance of the **code_list** structure.
[key=part_primary, db_index=clustered]

code_list_dc *code_list domain*
Domain in which code_list_id is unique.
[codelist=authorities, key=part_primary]

len_code_name *len code_list name*
The maximum space reserved for the signal name, i.e. 20. Actual string can only contain 19 characters to allow for the NULL byte.

code_list_name *code_list name*
Name of the code_list.
[index_string=true]

list_number *code_list number*
Number if the code_list.

number_members *number members*
Number of members in this code_list.

data_type *data storage type*
An integer representing the type of data that follows this structure. If the integer is negative, it refers to **variable_tab.define_num**. If the integer is positive, it refers to **structure_tab.struct_number**. Normally type CHAR.
[sets_data_type=true, codelist=data_types]

data_length *number of samples*
Number of samples of type **data_type** in the waveform.
[sets_data_length=true, default_value=0]

comment_id *comment*
A number representing an institution or authority operating a network, calculating a solution, make an instrument calibration, etc. The authority is specified as a number that refers to an ASCII string in the authority codelist. Each institution has a base number such as 10000, 20000, etc. The institution may assign the 9999 numbers above their base number to individual people or groups. The individual number might be set to agree with the user number in /etc/passwd on UNIX systems.
[key=part_foreign(1,comment.comment_id), db_delete=nullify]

comment_dc *comment domain*
Domain in which comment_id is unique.
[codelist=authorities, key=part_foreign(1,comment.comment_dc), db_delete=nullify]

SEE ALSO

NAME

comment – comment about structure contents

C_SYNOPSIS

```
typedef struct {
        FIXED       structure_type;
        FIXED       structure_len;
        LABEL       comment_id;
        DOMAIN      comment_dc;
        CODE4       data_type;
        INT4        data_length;
        INT4        struct_number;
        INT4        spare;
} SUDS_COMMENT;

#define COMMENTS    211L
```

DESCRIPTION

Any **suds** structure may have a comment structure associated with it that contains an ASCII string of any length up to 65536 bytes describing the structure or one or more members of the structure. Only one comment is available per instance of a structure. The same comment may refer to many instances of one type of structure. Comments must refer to a specific member of the structure and thus must be prefaced with {**name**} where **name** is the name of the member. Thus a comment about a **pick** structure might be "{signal_2_noise} waveform very irregular, possible telemetry spikes {onset_type} quite debatable {gain_range} unable to tell if gain ranged".

Comments should be written and accessed through the subroutines described in **make_comment(2)**.

A comment may contain any ASCII characters except { or } which are reserved for labeling. If either of these characters are included in a comment string passed to any of the **make_comment(2)** routines, they will be converted to a **[** or **]** respectively. If they are put in a comment by other means, they may confuse the **make_comment(2)** routines.

There is no required format for the contents of comments. Thus it is recommended that they only be used for free-form descriptions and NOT as a place to encode important information to be read by a computer program.
[permissions="siud_siu_siu_s"]

MEMBERS

structure_type *structure type*
Define number of this type of structure.

structure_len *structure length*
Length of this structure in bytes.

comment_id *comment id*
A number that uniquely identifies, within this **domain**, an instance of the **comment** structure.
[key=part_primary, db_index=clustered]

comment_dc *uniqueness domain*
Domain in which comment_id is unique.
[key=part_primary, codelist=authorities]

data_type *data type*
Type of data following this structure. Must be defined as type CHR.
[sets_data_type=true, codelist=data_types]

data_length *number of points*
Number of chars that follow this structure. Comments are padded by the subroutines and by the input/output library to always have a length in storage evenly divisable by 8.
[sets_data_length=true, default_value=0]

struct_number *structure number*

Number of the structure type this comment is associated with.

spare *for future use*

SEE ALSO

make_comment(2)

AUTHOR

Peter L. Ward, U.S. Geological Survey, Menlo Park, CA 94025

NAME

gui_default – user-set defaults to control graphical user interface

SYNOPSIS

```
typedef struct {
        FIXED           structure_type;
        FIXED           structure_len;
        LABEL           default_id;
        DOMAIN          default_dc;
        CODE1           degree_rep;
        CHAR            show_fixed;
        CODE2           time_rep;
        INT2            spare_1;
        INT2            spare_2;
        REFERS2         comment_id;
        DOMAIN          comment_dc;
} SUDS_GUI_DEFAULT;

#define GUI_DEFAULTS    206L
```

DESCRIPTION

Default values for a given user that control graphical user interface and conversions to ASCII.

MEMBERS

structure_type *structure type*
Define number of this type of structure.

structure_len *structure length*
Length of this structure in bytes.

default_id *default id*
A number that uniquely identifies, within this **default_dc**, an instance of the **default** structure.
[key=part_primary, db_index=clustered]

default_dc *default domain*
Domain in which default_id is unique.
[codelist=authorities, key=part_primary]

degree_rep *degrees representation*
[codelist=degree_types]

show_fixed *show fixed*
If not NOCHAR, then display members of type FIXED.

time_rep *time representation*
[codelist=time_types, index_string=true]

spare_1 *for future use*

spare_2 *for future use*

comment_id *comment*
A number that uniquely refers, within this comment_dc, to an instance of the **comment** structure. Comments are generally not searchable because they are not of standard format. Thus it is recommended that comments not be heavily used.
[key=part_foreign(1,comment.comment_id), db_delete=nullify]

comment_dc *comment domain*
Domain in which comment_id is unique.
[codelist=authorities, key=part_foreign(1,comment.comment_dc), db_delete=nullify]

SEE ALSO

NAME

domain_def – list of maximum keys for domain 00000

C_SYNOPSIS

#**include <suds/labels_domain.00000>**

INT4 max_label[sizeof(labels)/sizeof(SUDS_CODE_LIST)];

DESCRIPTION

LABELS are unique integers that identify a particular instance of a structure within a given domain. These numbers are assigned in increasing order starting at 1. The largest value assigned to date is kept in a file **domain.def** where domain is the **domain** number from the authorities code_list. A domain number uniquely identifies which institution and possibly which machine, program, person, or group defined these LABELS. These files are stored in the directory **suds/include**, which is symbolically linked to **/usr/include/suds**. The format is one long integer in **XDR** binary format for each label in the order listed in the **code_list labels** (See **code_lists(6)**). This file is created with the command **make_lab(1)** and accessed with the subroutine **get_label(2)**.

SEE ALSO

make_lab(1), get_label(2)

NAME

file_index – structure to index locations of structures in a file

C_SYNOPSIS

```
typedef struct {
        FIXED           structure_type;
        FIXED           structure_len;
        INT4            struct_number;
        INT4            bytes_data;
        INT4            tag_offset;
        FIXED           len_indexed_n;
        STRING          indexed_name[24];
        INT4            label_id;
        INT4            label_dc;
        ST_TIME         from_time;
        ST_TIME         thru_time;
} SUDS_FILE_INDEX, *IDXPTR;

#define FILE_INDEXS     204L
```

DESCRIPTION

This structure is used to index a file so that individual structures in the file can be read in any order desired by the program. An index for a file is put in a file of the same name but with the suffix ’I’.

MEMBERS

structure_type *structure type*
: Define number of this type of structure.

structure_len *structure length*
: Length of this structure in bytes.

struct_number *structure number*
: Number of the structure.

bytes_data *bytes of data*
: Bytes of data following the structure.

tag_offset *offset to structure_tag*

len_indexed_n *length indexed name*
: The maximum space reserved for the indexed name, i.e. 24. Actual string can only contain 23 characters to allow for the NULL byte.

indexed_name *indexed name*
: If signal_name does not exist in a structure, this string would be the ASCII value of the member where the definition sets_index=true is found. If the field being indexed if longer than 23 bytes, this is the last 23 bytes.
[index_string=true]

label_id *primary key id*
: LABEL or primary key for this structure.

label_dc *primary key dc*
: Domain in which label_id is unique.

from_time *beginning time*
: Beginning time if specifed in this structure.

thru_time *ending time*
: Ending time if specifed in this structure.

SEE ALSO

st_index(2)

NAME

member_info – system information about a members of a SUDS structure

C_SYNOPSIS

```
typedef struct {
        FIXED           structure_type;
        FIXED           structure_len;
        INT4            struct_number;
        INT2            member_number;
        INT2            member_type;
        INT2            member_length;
        INT2            member_offset;
        INT2            pri_key_numb;
        INT2            for_key_numb;
        INT4            key_structure;
        INT4            key_member;
        CHAR            db_include;
        CHAR            db_must_be_in;
        CHAR            db_index_type;
        CHAR            db_delete_type;
        INT4            db_permission;
        INT2            editor_row;
        INT2            editor_column;
        FIXED           name_len;
        STRING          member_name[16];
        FIXED           title_len;
        STRING          member_title[24];
        INT4            code_list_id;
        FIXED           list_len;
        STRING          code_list_name[24];
        FIXED           default_len;
        STRING          default_values[24];
        FIXED           format_len;
        STRING          print_format[20];
        FIXED           allowed_len;
        STRING          allowed_chars[24];
        CODE1           checks_input;
        CHAR            file_idx_type;
        INT2            spare;
} SUDS_MEMBER_INFO, *MEMPTR;

#define MEMBER_INFOS    202L
```

DESCRIPTION

This system table explains all of the properties of a member of a structure in a manner used by **SUDS** utility programs. This table is created automatically from the manual pages by the program **compile_man(1)** and stored in the file **suds_mem.h**.

MEMBERS

structure_type *structure type*

Define number of this type of structure, i.e. 7.

structure_len *structure length*

Length of this structure in bytes.

struct_number *structure number*

Number of the structure.

member_number *member number*
Number of this member in the structure counting from 0.

member_type *member type*
Number of the type of this member referring to the three-letter defines in **variable_info(3)**.

member_length *member length*
If this member is an array, then this is the length of the array.

member_offset *member offset*
Offset of beginning of this member from the top of the structure or in other words the address of this member minus the address of the first member of the structure. The offset can be determined at compile time by the following macro:

```
#define oFFsetOF(type,memb) ((INT4)&((type *)0)->memb)
```

It may not be possible to cast NULL or 0 in this way on some machines. offsetof is an ANSI C feature. See page 263 of "C, A Reference Manual" by Samuel P. Harbison and Guy L. Steele Jr., Prentice Hall, 1991. This macro is needed since different machines will align structures in different ways and many of the SUDS programs refer to structure elements by pointer rather than name in order to be general.

pri_key_numb *primary key number*
If this is part of the primary key, set to 1, otherwise set to NODATS.

for_key_numb *foreign key number*
Number of composite foreign key >= 0 where the number of each composite key must be unique within the structure. If this is not a foreign key, set to NODATS.

key_structure *key structure*
If this member is not a key, then set key_structure to NODATL. If this member is a primary key, then set key_structure to 0. If this member is a foreign key,then set key_structure to the number of the structure containing the primary key.

key_member *key member*
If this member is not a key, then set key_member to NODATL. If this member is a primary key then set key_member to 0. If this member is a foreign key set key_member to the number of the member containing the primary key. If this member is a composite key, then set key_member to the number of the first member of the structure containing part of the composite key.

db_include *include in db*
Normally set to T. If this member will not be stored in the database, then set to F. FIXED variable types would normally not be stored in the database since they are always the same and can be redetermined on output to SUDS structures.
[allow_char="TF"]

db_must_be_in *must exist in db*
Normally set to F. If this member is a primary key or a foreign key that may not be null, i.e. the primary key must exist for insert or update of this structure, then set to T.
[allow_char="TF"]

db_index_type *database index type*
If this member is not indexed in the database then set=NOCHAR, if standard index then set=I, if unique index then set=U, if clustered index then set=C. A unique index means that only one instance of this value may exist. A clustered index means that the physical order of structures would be in the order of this index whenever possible.
[allow_char="IUC"]

db_delete_type *database delete type*
Normally set to NOCHAR. For foreign keys this member can be set to restricted (R), nullify (N), or cascade (C). Restricted means that as long as an instance of this foreign key exists, the

instance of the structure with the primary key referred to by this key may not be deleted. Nullify means this foreign key may be nullified so that the structure with the primary key may be deleted. Cascade means that if the structure with the primary key is deleted, delete this structure also.
[allow_char="RNC"]

db_permission *database permissions*
Database permissions in fields of 4 bits. The least significant bit of each field is select or read, bit 1 is insert, bit 2 is update, and bit 3 is delete. The least significant bit field is permissions for the everyone or public, the next field is for group, the next is for analyst, and the next is for the database manager. The 16 most significant bits are reserved for future use. The default permission is set from the default for this structure in **structure_info(3)**.

editor_row *editor row*
Row of this member in the forms editor counting from 0 at the top of the screen.

editor_column *editor column*
The column of this member in the forms editor counting from 0 at the left.

name_len *name length*
The maximum space reserved for the member name, i.e. 16. Actual string can only contain 15 characters to allow for the NULL byte.

member_name *member_name*
Name in C for this member.
[index_string=true]

title_len *title length*
The maximum space reserved for the title, i.e. 24. Actual string can only contain 23 characters to allow for the NULL byte.

member_title *member title*
Title for this member used, for example, in **stedit** or **st2asc -v**.

code_list_id *code_list id*
ID number of codelist. Domains of code_lists are all defined as 2.

list_len *codelist length*
The maximum space reserved for the code list name, i.e. 24. Actual string can only contain 23 characters to allow for the NULL byte.

code_list_name *code list name*
String containing the code list name for this member.

default_len *default length*
The maximum space reserved for the default_values, i.e. 24. Actual string can only contain 23 characters to allow for the NULL byte.

default_values *default value*
String containing the default value or undefined value for this member.

format_len *format length*
The maximum space reserved for the format, i.e. 20. Actual string can only contain 19 characters to allow for the NULL byte.

print_format *print format*
String containing the format to use when printing this member.

allowed_len *allowed length*
The maximum space reserved for the allowed-chars, i.e. 20. Actual string can only contain 19 characters to allow for the NULL byte.

allowed_chars *allowable characters*

String containing a list of allowable characters during input.

checks_input *input subroutine*

A letter code pointing to a subroutine that can check the input value of a member in such programs as **stedit**. The subroutines are described in **input_subroutines(2)**. The name of the subroutine is given in the code_list **input_subroutines** (See **code_lists(6)**). The manual compiler (**compile_man(1)**) uses the code_list to convert the statement **check_input=subroutine** to this letter. The input programs must select the proper routine using a **switch** statement based on this code.

[codelist=input_subroutines]

file_idx_type *index type*

The way in which this member is to be used in file_index. n=indexed_name, l=label_id, d=label_dc, f=from_time, t=thru_time

spare *for future use*

SEE ALSO

NAME

stream – describes a SUDS input/output stream

SYNOPSIS

```
typedef struct {
        FIXED           structure_type;
        FIXED           structure_len;
        CHAR            type;
        CHAR            mode;
        CODE1           endian_type;
        CODE1           float_type;
        INT4            byte_ptr;
        UINT4           pack;
        FIXED           len_io_name;
        STRING          io_name[24];
        CHRPTR          file_handle;
        CODE1           machine_type;
        CODE1           output_type;
        CODE1           endian_change;
        CODE1           float_change;
        IDXPTR          index_to_file;
        INT4            length_index;
        INT4            struct_to_read;
        INT4            bytes_in_file;
        CHAR            sync_char;
        CODE1           computer_type;
        INT2            suds_version;
        INT4            struct_number;
        INT4            struct_length;
        INT4            length_data;
} SUDS_STREAM;

#define STREAMS         203L
```

DESCRIPTION

SUDS input and output is based on a **stream** of structures, i.e. a linear sequence of structures being read into a program or being written out of a program. A stream may originate in a file, a pipe, a network interface, or a database management system. Whenever a **stream** is opened (see **st_open(2)**), a **STREAM** structure is dynamically allocated and maintained by the IO package until the stream is closed (see **st_close(2)**). **This structure is internal to SUDS. Because it contains pointers (CHRPTR and IDXPTR), it should never be stored in a file or passed in a stream.**

MEMBERS

structure_type *structure type*
: Define number of this type of structure.

structure_len *structure length*
: Length of this structure in bytes.

type *stream type*
: Type of stream. f=file or pipe, d=database.

mode *stream mode*
: Mode of stream. c=closed, r=opened at beginning to read, w=opened at beginning to write, a=append (opened at end to write).

endian_type *endian type*
: Type of byte order on this machine: big-endian (**b**) or little-endian (**l**).

[codelist=endian_types allow_char="bl"]

float_type *float type*

Type of floating-point representation on this machine: IEEE (**i**) or VAX (**v**).
[codelist=float_types allow_char="iv"]

byte_ptr *byte pointer*

Used for packing characters and shorts into longs for XDR conversion on this stream.

pack *pack*

The four-byte integer into which characters and shorts are packed for XDR conversion on this stream.

len_io_name *length of IO name*

The maximum space reserved for the signal name, i.e. 24. Actual string can only contain 23 characters to allow for the NULL byte.

io_name *input/output name*

Last 23 letters of the filename or database name. If this name ends in ".db", it is assumed by **st_open(2)** to be a database.
[index_string=true]

file_handle *FILE pointer*

If stream is a file, this points to the **FILE** structure (see **stdio.h**). If stream is a database, this points to a database structure.

machine_type *this computer type*

Type of this computer: x, i, or v.
[codelist=computer_types]

output_type *type of output computer*

Type of computer output is being written for: x, i, or v.
[codelist=computer_types]

endian_change *change byte order*

For this stream, byte-order change is necessary (**t**) or not necessary (**f**).
[codelist=endian_types allow_char="ft"]

float_change *change float type*

Floating-point numbers in this stream must to changed from VAX to IEEE (**f**), to VAX from IEEE (**t**), or not changed (**n**).
[codelist=float_types allow_char="ft"]

index_to_file *pointer to index*

Pointer to a table of **file_index** structures read or created by **st_index(2)**.

length_index *length index*

Length of index created by **st_index(2)**. Number of structures in this file.

struct_to_read *structure to be read*

Number of structure to be read or to be written in this file. The byte to be read can be determined using **ftell(2)** for example:

```
offset=ftell((FILE *)suds_in_stream->file_handle);
```

bytes_in_file *bytes in file*

Total number of bytes in this file from fstat(2). Equals NODATL for stdio or for a file opened for writing.

sync_char *synchronization char*

Structure_tag(3) for next structure in this stream.

computer_type *type of computer*

Structure_tag(3) for next structure in this stream.

[codelist=computer_types]

suds_version *version of suds*
Structure_tag(3) for next structure in this stream.

struct_number *structure code number*
Structure_tag(3) for next structure in this stream.

struct_length *length of the structure*
Structure_tag(3) for next structure in this stream.

length_data *length of data*
Structure_tag(3) for next structure in this stream.

SEE ALSO

NAME

structure_info – system information about a SUDS structure

C_SYNOPSIS

```
typedef struct {
        FIXED       structure_type;
        FIXED       structure_len;
        INT4        struct_number;
        INT4        struct_length;
        MEMPTR      member_table;
        INT4        num_members;
        FIXED       len_struct_n;
        STRING      struct_name[16];
        FIXED       len_typedef;
        STRING      typedef_name[24];
        FIXED       len_define;
        STRING      define_name[20];
        INT4        data_only_to;
        INT4        db_permission;
        INT4        data_type_off;
        INT4        data_len_off;
        INT4        data_off_off;
        INT4        xdr_struct_len;
        INT2        total_width;
        INT2        short_width;
        INT4        spare_a;
} SUDS_STRUCTURE_INFO;

#define STRUCTURE_INFOS     201L
```

DESCRIPTION

This structure explains all of the properties of a particular structure in a manner used by **SUDS** utility programs. This information is created automatically from the manual pages by **compile_man(1)** and stored in the file **suds_str.h**.

MEMBERS

structure_type *structure type*

Define number of this type of structure, i.e. 48.

structure_len *structure length*

Length of this structure_info structure in bytes. Calculated by **size_of** for this particular computer at compilation time.

struct_number *structure number*

Define number of the structure described by this instance.

struct_length *length of structure*

Number of bytes in the structure described by this instance. Calculated by **size_of** for this particular computer at compilation time.

member_table *member table pointer*

Pointer to member table for this structure.

num_members *number of members*

Number of members in the member_table.

len_struct_n *name length*

The maximum space reserved for the member name, i.e. 16. Actual string can only contain 15 characters to allow for the NULL byte.

struct_name *name of structure*
Name of this structure.
[index_string=true]

len_typedef *length typedef*
The maximum space reserved for the typedef name, i.e. 24. Actual string can only contain 23 characters to allow for the NULL byte.

typedef_name *typedef name*
Typedef name of this structure.

len_define *length of define*
The maximum space reserved for the define name, i.e. 20. Actual string can only contain 19 characters to allow for the NULL byte.

define_name *define name*
Define name of this structure.

data_only_to *data for struct*
This structure is intended only to be used as data following a structure with struct_number=data_only_to.

db_permission *database permissions*
Default database permissions for all members in this structure. See **member_info.db_permission** (**member_info(3)**). If a default is not specified in the manual then it is set by the following define statement to **PERMIS** = duis_uis_uis_s = 0xf771
#define PERMIS 0xf771

data_type_off *offset to data type*
If this structure is followed by data, this field tells the offset in bytes from the beginning of the structure to the member that contains the value of the INT4 data_type. Otherwise this field = NODATL.

data_len_off *offset to data length*
If this structure is followed by data, this field tells the offset in bytes from the beginning of the structure to the member that contains the value of the **INT4 data_length**, i.e. the number of data of unit data_type that follow this structure. Otherwise this field = NODATL.

data_off_off *offset to data length*
If this structure is followed by data, this field tells the offset in bytes from the beginning of the structure to the member that contains the value of the **INT4 data_offset**, i.e. offset in a **data_group** file to the beginning of the **structure_tag** before this waveform structure.

xdr_struct_len *structure len in XDR*
Length of this structure in XDR (eXternal Data Representation). For PC-SUDS structures, this is the length of the structure in the file system, i.e. without FIXED, PAD1, PAD2, and PAD4 members.

total_width *total width*
Number of ASCII spaces to list whole structure in one line. Calculated by manual compiler based on **width_of_field** (See **structure_properties(2)).**

short_width *short width*
Number of ASCII spaces to list whole structure less FIXED, PAD1, PAD2, and PAD4 members in one line. Calculated by manual compiler based on **width_of_field** (See **structure_properties(2)).**

spare_a *for future use*

SEE ALSO

NAME

structure_tag – tag that identifies the next structure in a stream

C_SYNOPSIS

```
typedef struct {
        CHAR        sync_char;
        CODE1       computer_type;
        INT2        suds_version;
        INT4        struct_number;
        INT4        struct_length;
        INT4        length_data;
} SUDS_STRUCTURE_TAG;

#define STRUCTURE_TAGS    212L

#define ST_MAGIC          'S'
#define LEN_ST_TAG        16
```

DESCRIPTION

All structures written in a stream such as on a disk, tape, and over the network, must be followed by a **structure_tag**. This tag is used for error detection and to explain what structure follows and how much data follow the structure. The **structure_tag** is the label used to identify structures.

MEMBERS

sync_char *synchronization char*

All **structure_tags** must begin with the letter S. When a structure and any data following the structure are read, the next structure_tag is also read, and if the first letter is not S, an error is declared. In this way when a structure is read, the computer knows that it has been read properly.

[default_value="S", allow_char="S"]

computer_type *type of computer*

Type of computer this structure was written on: x=xdr compatible computer such as a SUN-3 or SUN-4 SPARC, v=DEC VAX or similar computer, i=ibm PC or similar computer.

[codelist=computer_types]

suds_version *version of suds*

Version of suds software times 100. Thus version 2.0 is 200.

struct_number *structure code number*

An integer defining the type of structure that follows. The integers are defined on the manual pages defining the structures.

struct_length *length of the structure*

The length of the structure in bytes. **SUDS** allows for future extension of the lengths of structures. If a structure is read that is shorter than the version the program currently expects, the additional members are added to the structure being input and set to default values.

length_data *length of data*

Length of data in bytes that follows the structure. The type of data is defined within the structure.

SEE ALSO

NAME

terminator – structure to end a group of structures in a stream

C_SYNOPSIS

```
typedef struct {
        FIXED           structure_type;
        FIXED           structure_len;
        INT4            structure_num;
        INT4            spare;
        REFERS2         comment_id;
        DOMAIN          comment_dc;
} SUDS_TERMINATOR;

#define TERMINATORS     213L
```

DESCRIPTION

Structure to end a group of structures in a stream.

MEMBERS

structure_type *structure type*
Define number of this type of structure, i.e. 2.

structure_len *structure length*
Length of this structure in bytes.

structure_num *structure number*
Number of the structure type that began this sequence.
[index_string=true]

spare *for future use*

comment_id *comment*
A number that uniquely refers, within this comment_dc, to an instance of the **comment** structure. Comments are generally not searchable because they are not of standard format. Thus it is recommended that comments not be heavily used.
[key=part_foreign(1,comment.comment_id), db_delete=nullify]

comment_dc *comment domain*
Domain in which comment_id is unique.
[codelist=authorities, key=part_foreign(1,comment.comment_dc), db_delete=nullify]

SEE ALSO

NAME

variable_info – system information about a variable used in SUDS structures

C_SYNOPSIS

```
typedef struct {
        FIXED           structure_type;
        FIXED           structure_len;
        FIXED           name_len;
        STRING          variable_name[8];
        FIXED           define_len;
        STRING          define_name[4];
        INT2            define_num;
        INT2            xdr_num_bytes;
        FIXED           c_type_len;
        STRING          c_type[20];
        FIXED           default_len;
        STRING          default_values[24];
        FIXED           min_len;
        STRING          min_value[24];
        FIXED           max_len;
        STRING          max_value[24];
        FIXED           format_len;
        STRING          print_format[16];
        FIXED           allowed_len;
        STRING          allowed_chars[24];
        INT2            field_width;
        INT2            num_bytes;
} SUDS_VARIABLE_INFO;

#define VARIABLE_INFOS          200L
```

DESCRIPTION

This structure explains the properties of a particular variable type used by **SUDS** utility programs. The program **compile_man(1)** puts a copy of the **variables** table, which explains the properties of all variable types, in the file **suds_var.h**.

MEMBERS

structure_type *structure type*
: Define number of this type of structure.

structure_len *structure length*
: Length of this structure in bytes.

name_len *name length*
: The maximum space reserved for the variable name, i.e. 8. Actual string can only contain 7 characters to allow for the NULL byte.

variable_name *variable name*
: Name as a C define for this variable: INT4, FLOAT8, FIXED, etc.
[index_string=true]

define_len *define length*
: The maximum space reserved for the define name, i.e. 4. Actual string can only contain 3 characters to allow for the NULL byte.

define_name *define name*
: The 3 letter name defined as an integer for this variable: IN4, FL8, FIX, etc. The **suds.h** file will contain statements such as

define CHR 1

so that the 3-letter string can be used in C-programs for tests of member type, for example

if(member_info[i].member_type==CHR)

define_num *define number*
Number given for the 3 letter define for this variable type. Programmers should not use these numbers in their code since they may change.

xdr_num_bytes *number of xdr bytes*
Length of this variable in bytes in XDR (eXternal Data Representation).

c_type_len *c type length*
The maximum space reserved for the c_type, i.e. 16. Actual string can only contain 15 characters to allow for the NULL byte.

c_type *C type*
Type for this member in the C language.

default_len *default length*
The maximum space reserved for the default_values, i.e. 24. Actual string can only contain 23 characters to allow for the NULL byte.

default_values *default value*
String containing the default value or undefined value for this member.

min_len *minimum length*
The maximum space reserved for the min_value, i.e. 24. Actual string can only contain 23 characters to allow for the NULL byte.

min_value *minimum value*
String containing the minimum value for this member.

max_len *maximum length*
The maximum space reserved for the max_value, i.e. 24. Actual string can only contain 23 characters to allow for the NULL byte.

max_value *maximum value*
String containing the maximum value for this member.

format_len *format length*
The maximum space reserved for the format, i.e. 16. Actual string can only contain 15 characters to allow for the NULL byte.

print_format *print format*
String containing the format to use when printing this member.

allowed_len *allowed length*
The maximum space reserved for the allowed-chars, i.e. 24. Actual string can only contain 23 characters to allow for the NULL byte.

allowed_chars *allowable characters*
String containing a list of allowable characters during input.

field_width *field width*
Number of spaces needed to give ASCII representation of this variable type.

num_bytes *number of bytes*
Length of this variable in bytes of memory.

Special typedefs

In addition to the **typedefs** declared in the following table,
several are provided in **suds.h** for use in programming but may not be used within structures. They are **INTV** for **int**, which may vary in size on different machines, **UINTV** for **unsigned int**, **UINT4** for an **unsigned long**, and **SIZE_T** of a **size_t** which is the size of integer returned by the function **sizeof()**.

INITIAL VALUES

Testing whether a variable is equal to NODATF is tricky on different machines because of round-off errors. Use **isNODATF(3)** or **isNODATD(3)** to test equality with NODATF.

```
#define SUDS_VERSION 2.6
#define NODATS       (-32760)        /* NODATA short */
#define NODATL       (-2147483640L)  /* NODATA long */
#define NODATF       (-1.7e+36)      /* NODATA float */
#define NODATSPC     (-32767)        /* NODATA short or long PC_SUDS */
#define NODATFPC     (-32767.0)      /* NODATA float or double PC_SUDS */
#define MINTIME      (-2147472000L)  /* minimum NODATA time */
#define MAXTIME      (2147472000L)   /* maximum NODATA time */
#define NOCHAR       '_'             /* NODATA char */
#define NOSTRG       ""              /* NO STRING */
#define NOPTR        0L              /* NO POINTER */
#define NOLIST       (LIST)0L        /* NODATA lists */
#define ED_COL       25              /* Default column to start field in the editor */
#define PCSUDS_MAX   100

#define SNCHAR       "_"             /* NOCHAR string */
#define SNDATS       "-32760"        /* NODATA string short */
#define SMDATS       "32760"         /* NODATA string short */
#define SNDATL       "-2147483640"   /* NODATA string long */
#define SMDATL       "2147483640"    /* NODATA string long */
#define SNDATF       "-1.7e+36"      /* NODATA string float */
#define SMDATF       "1.7e+36"       /* NODATA string float */
#define SMNTIME      "-2147472000"   /* MINIMUM value string notime */
#define SMXTIME      "2147472000"    /* MAXIMUM value string notime */
#define SNLIST       " "             /* NODATA string lists */
#define SNPTR        " "             /* NODATA string pointer */

#define MAX_NAME_LEN  24             /* max characters in struct.member name */
#define MAX_MEMBERS 100L             /* max number of members in a structure */
#define MAX_STRUCT_LEN 256L          /* max bytes in a structure */

typedef struct {
        FLOAT4      cr;
        FLOAT4      ci;
} COMPLEX;

#define COMPLEXS     207L

typedef struct {
        FLOAT8      dr;
        FLOAT8      di;
} D_COMPLEX;

#define D_COMPLEXS   208L

typedef struct {
        FLOAT4      fx;
        FLOAT4      fy;
} VECTOR;

#define VECTORS      209L

typedef struct {
        FLOAT4      xx;
```

```
                FLOAT4      yy;
                FLOAT4      xy;
        } TENSOR;

        #define TENSORS     210L


SUDS_VARIABLE_INFO variables[] = {
    /*      Name                Define          N       C_type          Default         Minimum         MaximumFormat       Allo
    49,168,8,"CHAR",        4,"CHR",  -1,1,   20,"char",    48,SNCHAR,    48," ",       48,"~",
                            16,"%c",  24," ~~",1,sizeof(char),
    49,168,8,"STRING",      4,"STR",  -2,1,   20,"char",    48,NOSTRG,    48,NOSTRG,    48,NOSTRG,
                            16,"%s",  24," ~~",0,           sizeof(char),
    49,168,8,"INT2",        4,"IN2",  -3,2,   20,"short",   48,SNDATS,    48,SNDATS,    48,SMDATS,
                            16,"%d",  24,"0~9-",6,          sizeof(short),
    49,168,8,"INT4",        4,"IN4",  -4,4,   20,"long",    48,SNDATL,    48,SNDATL,    48,SMDATL,
                            16,"%ld", 24,"0~9-",11,         sizeof(long),
    49,168,8,"FIXED",       4,"FIX",  -5,4,   20,"long",    48,SNDATL,    48,SNDATL,    48,SMDATL,
                            16,"%ld", 24,"",11,             sizeof(long),
    49,168,8,"CODE1",       4,"CD1",  -6,1,   20,"char",    48,SNCHAR,    48,"0",       48,"z",
                            16,"%c",  24," ~~",1,           sizeof(char),
    49,168,8,"CODE2",       4,"CD2",  -7,2,   20,"short",   48,SNDATS,    48,SNDATS,    48,SMDATS,
                            16,"%d",  24,"0~9-",6,          sizeof(short),
    49,168,8,"CODE4",       4,"CD4",  -8,4,   20,"long",    48,SNDATL,    48,SNDATL,    48,SMDATL,
                            16,"%ld", 24,"0~9-",11,         sizeof(long),
    49,168,8,"LABEL",       4,"LAB",  -9,4,   20,"long",    48,SNDATL,    48,SNDATL,    48,SMDATL,
                            16,"%ld", 24,"0~9-",11,         sizeof(long),
    49,168,8,"REFERS2",     4,"REF",  -10,4,  20,"long",    48,SNDATL,    48,SNDATL,    48,SMDATL,
                            16,"%ld", 24,"0~9-",11,         sizeof(long),
    49,168,8,"DOMAIN",      4,"DOM",  -11,4,  20,"long",    48,SNDATL,    48,SNDATL,    48,SMDATL,
                            16,"%ld", 24,"0~9-",11,         sizeof(long),
    49,168,8,"AUTHOR",      4,"ATH",  -12,4,  20,"long",    48,SNDATL,    48,SNDATL,    48,SMDATL,
                            16,"%ld", 24,"0~9-",11,         sizeof(long),
    49,168,8,"FLOAT4",      4,"FL4",  -13,4,  20,"float",   48,SNDATF,    48,SNDATF,    48,SMDATF,
                            16,"%f",  24,"0~9.-",14,        sizeof(float),
    49,168,8,"FLOAT8",      4,"FL8",  -14,8,  20,"double",  48,SNDATF,    48,SNDATF,    48,SMDATF,
                            16,"%lf", 24,"0~9.-",20,        sizeof(double),
    49,168,8,"ST_TIME",     4,"STT",  -15,4,  20,"long",    48,SMNTIME,   48,SMNTIME,   48,SMXTIME,
                            16,"%ld", 24,"0~9.-",11,        sizeof(long),
    49,168,8,"MS_TIME",     4,"MST",  -16,8,  20,"double",  48,SMNTIME,   48,SMNTIME,   48,SMXTIME,
                            16,"%lf", 24,"0~9.-",20,        sizeof(double),
    49,168,8,"LONGIT",      4,"LON",  -17,8,  20,"double",  48,SNDATF,    48,"-180.",   48,"+180.",
                            16,"%lf", 24,"0~9.-",20,        sizeof(double),
    49,168,8,"LATIT",       4,"LAT",  -18,8,  20,"double",  48,SNDATF,    48,"-90.",    48,"+90.",
                            16,"%lf", 24,"0~9.-",20,        sizeof(double),
    49,168,8,"LIST",        4,"LST",  -19,4,  20,"SUDS_CODE_LIST *",48,SNLIST,48,SNLIST,        48,SNLIST,
                            16,"%lx", 24,"",11,             sizeof(char *),
    /* 12 bit data, 4 lsb BCD time code */
    49,168,8,"INT2TM",      4,"I2T",  -20,2,  20,"short",   48,SNDATS,    48,SNDATS,    48,SMDATS,
                            16,"%d:%x",       24,"0~9",6,   sizeof(short),
    49,168,8,"CODESTR",     4,"CDS",  -21,1,  20,"char",    48,NOSTRG,    48,NOSTRG,    48,NOSTRG,
                            16,"%s",  24," ~~",1,           sizeof(char),
    49,168,8,"GENPTR",      4,"GNP",  -22,4,  20,"char *",  48,SNPTR,     48,SNPTR,     48,SNPTR,
                            16,"%lx", 24,"0~9",11,          sizeof(char *),
```

```
    49,168,8,"CHRPTR",  4,"CHP", -23,4, 20,"char *",  48,SNPTR,   48,SNPTR,   48,SNPTR,
                        16,"%lx", 24,"0~9",11,        sizeof(char *),
    49,168,8,"MEMPTR",  4,"MEM",-24,4, 20,"SUDS_MEMBER_INFO *",         48,SNPTR,    48,SNPTR,48,SNPTR,
                        16,"%lx", 24,"",11,           sizeof(char *),
    49,168,8,"UINT4",   4,"UI4", -25,4, 20,"unsigned long",   48,SNDATL,  48,SNDATL,  48,SMDATL,
                        16,"0x%u",     24,"",11,      sizeof(unsigned long),
    49,168,8,"UINT2",   4,"UI2", -26,2, 20,"unsigned short",  48,SNDATL,  48,SNDATL,  48,SMDATL,
                        16,"0x%u",     24,"",6,       sizeof(unsigned short),
    49,168,8,"UCHAR",   4,"UCH", -27,1, 20,"unsigned char",   48,SNDATL,  48,SNDATL,  48,SMDATL,
                        16,"0x%u",     24,"",3,       sizeof(unsigned char),
    49,168,8,"YESNO",   4,"YNO", -28,4, 20,"long",    48,SNDATL,  48,SNDATL,  48,SMDATL,
                        16,"%ld", 24,"",11,           sizeof(long),
    49,168,8,"IDXPTR",  4,"IDX", -29,4, 20,"SUDS_FILE_INDEX *",  48,SNPTR,   48,SNPTR,   48,SNPTR,
                        16,"%lx", 24,"",11,           sizeof(char *),
    /* 24-bit integer data stored in 32-bit integers */
    49,168,8,"INT3",    4,"IN3", -30,4, 20,"long",    48,SNDATL,  48,SNDATL,  48,SMDATL,
                        16,"%ld", 24,"0~9-",11,       sizeof(long),
    49,168,8,"PAD1",    4,"PD1", -31,1, 20,"char",    48,SNCHAR,  48," ",     48,"~",
                        16,"%c",  24,"",1,sizeof(char),
    49,168,8,"PAD2",    4,"PD2", -32,2, 20,"short",   48,SNDATS,  48,SNDATS,  48,SMDATS,
                        16,"%d",  24,"",6,            sizeof(short),
    49,168,8,"PAD4",    4,"PD4", -33,4, 20,"long",    48,SNDATL,  48,SNDATL,  48,SMDATL,
                        16,"%ld", 24,"",11,           sizeof(long),
};
```

SEE ALSO

st_intro(4)

BUGS

SUDS

Chapter 4:
User Structure Descriptions

This page left blank intentionally.

NAME

st_intro – introduction to **SUDS** structures and codes

OVERVIEW

The **S**eismic **U**nified **D**ata **S**ystem (**SUDS**) is based on organizing data into *structures or groups that provide all of the important information about a logical entity* such as an earthquake hypocentral solution, a seismic waveform, a phase pick, or the frequency response of a component or system. These structures or groups of data can then be accessed and utilized efficiently and stored in a machine-independent manner, in any order, on any type of storage device.

A structure consists of an ordered list of members. Multiple instances of a structure can be thought of as a table where there is one column for each member and one row for each instance of the structure. Some members of some structures refer to specific instances of other structures, providing a way to relate, for example, specific phase picks to a specific earthquake solution, a specific waveform to a specific recorder, etc. Thus **SUDS**, besides being a data format, is a model or plan for a relational database system.

Chapter 4 of the manual provides a description of each structure and a definition for each member of each structure. The information in these manual pages provides a complete description of the **SUDS** structures in a format appropriate for the C-language. Use of **SUDS** with other programming languages such as FORTRAN that support structures, is easy and will be described elsewhere. All information in the manual required for computer programs to access and fully utilize **SUDS** structures is extracted automatically by a computer program (**comp_man(1)**) and compiled into four files that are typically included in source programs. These files contain the central tables used by utility programs to read, search, utilize, and write **SUDS** structures, files, streams, and databases. Thus the manual provides complete documentation of **SUDS** and is the ultimate authority for **SUDS**.

This introduction describes the conventions used in defining **SUDS** structures. It is important for all users of **SUDS** to understand these conventions.

VARIABLE TYPES

The types of variables used in **SUDS** are specified using **typedefs**, a C language facility for assigning new names to variable types. This is done to enhance portability, to clarify the purpose of each variable type, to extend some variable types to include an allowable range of values, and to facility use of other computer languages. The properties of all variable types are contained in **variable_info(2)**. Many variable types are similar to computer language types such as:

SUDS type	**C type**	**FORTRAN type**
INT2	short	INTEGER*2
INT4	long	INTEGER*4
FLOAT4	float	REAL*4
FLOAT8	double	REAL*8 or DOUBLE PRECISION
CHAR	char	CHARACTER
STRING	char x[len]	CHARACTER*LEN x

Other variable types carry additional information as follows:

LONGIT
Longitude as a FLOAT8 with a range limited from -180.0 to +180.0.

LATIT Latitude as a FLOAT8 with a range limited from -90.0 to +90.0.

ST_TIME
Stamp time or time in seconds since or before January 1, 1970, 00:00:00 Greenwich Mean Time. This is the same as **UNIX** time (see **time(3V)**) and is an **INT4**.

MS_TIME
Millisecond time is similar to ST_TIME but is a FLOAT8 in seconds with a precision of approximately 1 microsecond between 1900 and 2040 AD.

FIXED A constant fixed by definition. Each structure begins with 2 FIXED members specifying structure_type and structure_length. The eXternal Data Representation (**XDR**) used to store **SUDS** structures in machine independent binary format, requires that a character string must be preceded by its maximum length as an INT4 and that this length must be an integral multiple of 4. Thus FIXED members precede all strings. FIXED variable types may not be stored in some database implementations of **SUDS** since they do not change.

CODE A number or letter that refers to an ASCII string. Codes are used in **SUDS** to save space and to enforce standardization of ASCII strings for ease of searching. They allow standard ASCII strings of arbitrary length to be used, thus promoting clarity. Codes may be a character (CODE1), a 16-bit integer (CODE2), or a 32-bit integer (CODE4) depending on how many different codes are likely to be needed for a given application. Codes are related to the ASCII strings via the **code_list(4)** structure and are described in detail in **Appendix I: code_lists(6)**. Codes should be accessed only through the subroutines described in **find_code(2)** because some **code_lists** such as **authorities** will ultimately be stored in a disk file rather than in memory.

AUTHOR

A 32-bit integer that designates who analyzed the data, who is responsible for a given observation, who made a particular pick from a waveform, etc. The number is meant to be unique world-wide. In order to give each institution basic autonomy over its own needs, each institution is assigned a base number by a global authority, i.e. the organization managing the **SUDS** standard. The base number is an even multiple of 10,000. The institution can then assign the next 9,999 numbers above the base number in any way they choose and register these numbers with the global authority for periodic distribution to all **SUDS** users. The **SUDS** standard allows for more than 400,000 institutions to participate.

The **authority** can be used to signify an individual, a group for example responsible for producing an earthquake catalog, a computer program that calculates automatic solutions, etc. Often it is useful to display either the ASCII string or an abbreviation. The abbreviation is defined as the first 5 characters or less than 5 characters occurring before a colon. The abbreviation is used as the **network_name** in specifying the complete name of a seismic signal. Thus some care should be made to keep the abbreviations unique, at least among data that may ultimately be merged. Part of the **authorities** code list might be as follows:

```
SUDS CODE LIST authorities[] = {
        20000, "UOA: Seismology Lab, University of Atlantis (base number)",
        20100, "JS: Dr. John Smith, seismologist",
        20101, "CJcat: Charlotte Jones, as earthquake catalog production manager",
        20102, "CJ: Charlotte Jones, staff seismologist, for personal use",
        20103, "LR: Lisa Roberts, data analyst",
        20134, "SW: Sam Wong, graduate student",
        21000, "CPG: Central Atlantis processing group",
        22000, "PIC: auto picker program, version 1.1", };
```

LABEL

A unique INT4 assigned by an authority to label or identify a specific instance of a structure. Most structures have a label or primary key. A LABEL variable name normally ends in **_id** and must be followed in the structure by a DOMAIN member. Unique labels can be determined using the subroutine **find_label(2)**.

REFERS2

An INT4 that refers to a specific instance of a label or primary key. This member relates one structure to another structure and in database terminology is a foreign key. A REFERS2 variable name normally ends in **_id** and must be followed in the structure by a DOMAIN member.

DOMAIN

LABELs and REFERS2s are unique within a DOMAIN. The DOMAIN is the authority assigning the value to a LABEL. All LABEL and REFER2 members of structures must be followed by a DOMAIN member. The name of a DOMAIN member normally ends in **_dc**. In some database implementations, the DOMAIN may be omitted and all LABELs and REFERS2s are converted on input to a single DOMAIN.

The DOMAIN may apply to a network, a person, a group, etc. For example, the following entries might be added to the **authorities** code list:

SUDS CODE LIST authorities[] = {
  29001, "CAS: Central Atlantis Seismic Networking",
  29002, "WIN: Wingding Volcano Seismic Network",
  29010, "LOMA: Loma Prieta Aftershock Temporary Network",
  29901, "SUNB: SUN3/60 earthquake detection and recording system at WIN", }

Thus the value of DOMAIN members defined by the SUNB processor would be 29901. The **domain_code** for the private database of John Smith would be 20100, and those for the official earthquake catalog at the University of Atlantis might be 21000 or 29001.

It is generally advisable to use new values for DOMAIN members only when truly required. For example an earthquake recording device is not likely to know about any other domain than its own. When the data are demultiplexed and added to a data set for catalog processing, however, it is typically best to adopt the DOMAIN value for the new data set and to change the relevant LABEL and REFER2 members to be unique in the new domain.

MISSING DATA

In the real world of data collection, it is often important to differentiate between data that has a value of 0 and data that is missing typically because of hardware failure. **SUDS** uses special numbers near the maximum or minimum range of numbers to designate missing data. The different missing number symbols for different data types are defined in **variable_info(3)** and include **NODATS**, **NODATL**, **NODATF**, **NOTIME**, **NOCHAR**, **NOSTRG**, and **NOLIST**. The **SUDS** utility programs print these symbols and accept input of these symbols. Programmers should refer to missing data using the defines given in **variable_info(3)**.

LIST OF STRUCTURES

Structures currently defined by the **SUDS** standard are as follows:

beam_data	component of a beam of waveforms
calibration	information about a calibration signal
clock_rate	rate of change of a clock error
coordinate_sys	information to define a local coordinate system
data_group	information about storage of a collection of waveforms
event	information about processing of an event
filter	specifies filtering applied to waveforms
focal_mech	information about the focal mechanism and moment for a solution of an event
lsa_detection	a specific long-term, short-term average event detection
lsa_set_data	signal_paths and subnets for event and cross triggers
lsa_setting	settings of long-term, short-term average event detection program
magnitude	magnitude calculated for a solution
map_element	lines and points to be plotted on a map
mux_waveform	information about waveforms that are multiplexed
pick	information about a phase pick or any other picked feature of a waveform
pick_residual	residual for one pick in a solution and association of the pick with the solution
polarity	evidence for reversed polarity for a signal_path
processing	a processing command or error message
recorder	information about a recorder of signals
recorder_ass	associates recorders with signal paths
resp_cfs_data	response values for corner frequency and slope
resp_fap_data	response values for frequency, amplitude, and phase
resp_fir_data	response values for finite impulse response filters
resp_pz_data	response values for infinite impulse response filters
resp_sen_data	response sensitivity/gain

response	information about the frequency response of a sensor, component, or total system
seismo_ass	associates seismometers with signal paths
seismometer	information about a seismometer
service	record of service to a signal_path
sig_cmp_data	the wiring of one sig_path_cmp to another
sig_path_ass	associates signal path components with signal paths
sig_path_cmp	information about an individual component in a signal path
sig_path_data	List of signal_paths to follow the focal_mechanism structure
signal_path	information about a data path from a single sensor to a recorder
signif_event	information about a major earthquake that complements the solution
site	geographical location and other information about a site containing equipment, source, etc.
solution	information about a particular solution of an event
solution_err	error for an earthquake solution
source	description of a man-made seismic event such as an explosion
spectra	spectra of a waveform
ssam_band_data	passband for the Seismic Spectral Amplitude Monitor
ssam_output	data from Seismic Spectral Amplitude Monitor
ssam_setup	parameters to setup Seismic Spectral Amplitude Monitor
user_vars	user defined variables
vel_layer_data	information about a horizontal layer in a crustal velocity model
vel_model	information about a horizontally flat-layered crustal velocity model
waveform	information about a waveform for a single station component

PROPERTIES OF THE STRUCTURES

NAME	NUMBER	BYTES	MEMBERS	
beam_data	318	16	4	data only structure
calibration	320	72	21	
clock_rate	130	72	19	
coordinate_sys	326	120	27	
data_group	108	224	23	
event	112	88	20	
filter	315	56	17	
focal_mech	116	272	65	data may follow
lsa_detection	125	96	28	
lsa_set_data	304	16	5	data only structure
lsa_setting	126	112	30	data may follow
magnitude	307	56	17	
map_element	312	88	21	data may follow
mux_waveform	106	192	32	data may follow
pick	110	144	38	
pick_residual	111	80	25	
polarity	309	48	14	
processing	308	40	12	data may follow
recorder	131	104	24	
recorder_ass	324	72	20	
resp_cfs_data	305	8	2	data only structure
resp_fap_data	303	24	6	data only structure
resp_fir_data	314	24	6	data only structure
resp_pz_data	123	32	8	data only structure
resp_sen_data	319	16	4	data only structure
response	109	104	26	data may follow
seismo_ass	325	64	16	
seismometer	313	96	28	
service	323	96	16	

sig_cmp_data	302	16	7	data only structure
sig_path_ass	316	72	19	
sig_path_cmp	104	88	23	data may follow
sig_path_data	306	8	2	data only structure
signal_path	105	112	28	data may follow
signif_event	113	152	34	
site	300	152	34	
solution	114	168	47	
solution_err	115	112	28	
source	321	96	26	
spectra	301	80	23	data may follow
ssam_band_data	317	8	2	data only structure
ssam_output	311	40	10	data may follow
ssam_setup	310	40	11	data may follow
user_vars	322	88	30	
vel_layer_data	119	40	11	data only structure
vel_model	118	96	19	data may follow
waveform	107	184	48	data may follow

DEFINING NEW STRUCTURES

In principal, new **SUDS** structures can be defined to meet any need. Any person wishing to define a new structure, however, needs to consider whether an existing structure can possibly fit the need. When at all possible, existing structures should be used in order to minimize programming complexity. While many utility programs can work with any structure, most work-horse programs will only utilize a small set of structures. Thus, for example, if there were several structures that described earthquake phase readings, each phase-processing program would need to know about all of these structures or some of these programs would not know how to utilize some of these structures and the data would grow incompatible. THUS TRY TO USE EXISTING STRUCTURES WHENEVER POSSIBLE. New members can be added to the ends of existing structures to improve their utility in your case and spare variables exist in some structures to meet the same need. Additions and modifications should not be taken lightly, however, and should be viewed as a last resort.

SUDS structures are defined so that on machines with Big-endian byte order and IEEE floating point representation, they are already in **XDR** (eXternal Data Representation) format and can be read and written without any conversion or bit manipulation. This imposes several restrictions that are enforced by the manual compiler. Each member must begin on a byte boundary evenly divisible by its length. Thus a FLOAT8 must begin on a byte boundary divisible by 8. This implies that CHAR and INT2 members must be grouped as 4 CHARs, 2 CHARS and an INT2, an INT2 and 2 CHARS, or 2 INT2s. In fact XDR does not recognize these types and they are packed and unpacked by **SUDS** input-output routines. Structures must have a total length evenly divisible by 8. XDR requires that all strings have a maximum length evenly divisible by 4 and that they be preceded by an INT4 or FIXED containing the maximum length.

The first member of a structure must be **FIXED structure_type** and the second member must be **FIXED structure_len**. The **structure_type** is used by many routines to identify the structure. The **structure_len** is to provide an error check in the future when members may be added to existing structures.

LABEL and **REFERS2** members must be followed by corresponding **DOMAIN** members.

Names in **SUDS** are limited to the following lengths to be compatible with some computer compilers and database systems. These lengths are enforced by the lengths of strings in the variable_info, member_info, and structure_info:

Structure names	15	preferably <= 12
Member names	15	preferably <= 12
Variable names	7	
Structure typedef names	23	

Structure define names 16

Structure members are uniquely defined as **structure_name.member_name**. The total length of such names should be less than 27. The **define_name** for each structure should be the structure name capitalized followed by the letter **S** The typedef name of structures should be the structure name capitalized preceded by **SUDS_**

Structures are groups of data that define logical entities. Often this entity includes data that follows the structure. A **waveform** structure is followed by waveform data and a **vel_model** structure is followed by data in the format of **vel_layer** structures. In order for the data to be read and written properly with **XDR** routines, these header structures must contain the members **CODE4 data_type** and **INT4 data_length** and the definitions of these members must contain the fields **sets_data_type=true** and **sets_data_length=true**.

NORMALIZATION

The relational model requires that each member of a structure must be "atomic", i.e. it may not contain multiple values of the same meaning or type. Multiple values should typically be put in another table referred to by the first table.

For purposes of searching, from experience, and from relational algebra, relational tables should be normalized typically in what is known as Third Normal Form. There is a lot written about normalization and there are many approaches. Primary goals of normalization are to find and isolate time-independent properties of relationships, remove redundant information, and provide unique identification of individual records.

> A table is in FIRST NORMAL FORM if a primary key exists that is not NULL, is unique, and does not contain a submember that is a primary key itself. All attributes are atomic, i.e. not repeating.
>
> A table is in SECOND NORMAL FORM if it is in first normal form and all non-key attributes are fully functionally dependent on the primary key or in other words the non-key attributes must be uniquely identifiable from a subset of the primary key.
>
> A table is in THIRD NORMAL FORM if it is in second normal form and all non-key attributes are non-transitively dependent on the primary key or in other words a non-key attribute must be solely dependent on (determined by) the primary key, not by anything else.

Normalization is a guide but not a strict requirement because sometimes it is far more efficient given the way particular data are used to allow some redundancy. A typical tradeoff in SUDS is whether all event related structures should contain an event_id. If they do, it is easier within or without a database, to search everything concerning an event.

Denormalization has been introducted for user efficiency in several cases. Care needs to be taken to be sure these fields do not become inconsistent. **signal_name** is duplicated in **signal_path, waveform,** and **pick.** **solution.pref_magnitude** duplicates the magnitude value when **magnitude.preferred** equals 'P'.

MANUAL FORMAT

Because the manual is compiled to create the **SUDS** include files, certain requirements must be met when creating new sections. Each manual directory contains a **TEMPLATE** that should be used as a starting point. The original files are in standard **troff(1)** format using the **man** macro package. The manual compiler extracts all lines with the token #**define** but ignores lines with a space between the # and **define**. The compiler looks for the tokens **typedef struct** and copies until the end of the line containing the right brace. The compiler decodes the first line after the line **.SH NAME** and after each line **.TP** and their format should be kept standard. The compiler collects all information between square brackets which in the original files are designated by lines **.BB** and **.EB** signifying begin bracket and end bracket. These macros are defined in the file **../man_macros/suds_man_macros** which is included in the **troff** file with the line

All fields between the brackets must be of the form **key_word = string** and be separated by commas. Allowable key_words are as follows:

allow_char=string

A string of characters allowed on input to specify this member. The characters may include a range in ASCII order specified by a ˜. To allow negative numbers, the minus sign must be included. Thus for a positive integer, the string would be "0˜9", for a positive or negative float it is "0˜9.-", for a string with only letters it is "a˜zA˜Z", for all ascii characters it is the string " ˜˜". To accept input of the character '˜', make '˜' the last character in the string. The **allow_char** specification is needed only if different from the default for this data type. For members of type **CODE1**, the allow_char string is determined by the manual compiler from the appropriate **codelist** if the **codelist** has 23 or less members. To override this feature, specify the **allow_char** after specifying the **codelist**.

check_input=subroutine

name of subroutine to be used to check input to this member (see **check_input(2)**).

codelist=name

if this member is a code, give the name of code_list as given in **code_lists(6)**.

db_delete=string

where the string is restrict, nullify, or cascade. Restrict means that if a request is made to delete a primary key, do not allow deletion until all foreign references have been deleted. Nullify means that if a request is made to delete a primary key, nullify this foreign reference. Cascade means that if a request is made to delete a primary key referred to by this structure, delete this structure also.

db_index=string

where string is true, unique, or clustered. Include if this member is to be indexed in the database by a standard index that allow duplicates, a unique index that requires uniqueness, or a clustered index that requires uniqueness and makes storage logically contiguous by index order. Clustered defaults to unique if not supported by a particular database management system.

db_must_exist=true

Most members of structures may be NULL but primary keys and some foreign keys must exist for the structure to have meaning. If a structure is being inserted in the database, db_must_exist members must be specified. ASSUMED TRUE FOR ALL PRIMARY KEYS.

default_value=string

default is only included if this value is different than the standard default for this variable type. The default value is a string read using the format.

ed_col=number

column in the forms editor that this member begins to be displayed at. The default value is the define **ED_COL** in **variable_info(3)**.

ed_row=number

row in the forms editor that this member is to be displayed on. The default value is (2 ∗ member_number) + 1. When a structure contains to many members the default becomes simply member_number + 1.

format=string

format is the C format string used to read the default_value string and to write the value in utility programs such as the screen editor and st2asc. The default is determined for the member type from **variable_info(3)**.

in_db=false

Specifies that the member is to be omitted from the database. In some database implementations this specification may be automatic for members of FIXED type because the values are known and for DOMAIN variables when the database is set up for a single DOMAIN. This specification is optional for other members, but should normally not be used since those members will be set to the default_value when written out by the database.

index_string=true

This member is used to create an ASCII string used in **st_index(2) stdescr(1)**, etc.

key=part_primary
key=part_foreign(composite_key_#,table_name.member_name)

There may be only one group of part_primary keys per structure. Structures are related to each other by keys. The existence of keys means that on insertion, update, or deletion of structures with keys, other structures may be affected. In the case of an associative table where the primary_key is made up of two or more composite foreign keys the specification should be

key=part_primary, key=part_foreign(1,table_name.member_name)

permissions=string

permissions is a string giving permissions to select or read a member or table(s), insert (i) a member or table, update (u) a member or table, and delete (d) a member or table for the manager, analyst, group, and general public. Each permission field may contain up to the following 4 letters to grant permission such as "siud_siu_siu_s" where the manager has all permissions and the public may only select. The default permission is given by the define **PERMIS** in **member_info(3)**. The permissions are encoded in the member_infole in four 4-bit blocks contained in the db_permission variable. The most significant 4 bits (bits 15 to 12) refer to the manager and the least significant 4 bits refer to public. Bits 31 to 16 are reserved for future categories of permissions. In each 4-bit block, bit0(lsb)=d, bit1=u, bit2=i, bit3=s. Thus a default_value of "siud_siu_siu_s" becomes 0xf771. Permissions can be set for all members of a table by putting the permissions = statement at the end of the **DESCRIPTION** section. This global permission will apply if no specific permission is assigned to a member.

sets_data_type=true

If this structure is followed by data, one of these statements must be included for the member that specifies the type of the data.

sets_data_length=true

If this structure is followed by data, one of these statements must be included for the member that specifies the length of the data.

sets_data_offset=true

If this structure is followed by large amounts of data that will not be stored directly in a database system, one of these statements can be included for the member that specifies the file_offset of the data. See **waveform(4)**.

SEE ALSO

4. External Data Representation: Sun Technical Notes and 5. External Data Representation Standard: Protocol Specification in **Network Programming Guide**

st_intro(1), st_intro(2)

FILES

/usr/include/suds/suds.h

/usr/include/suds/suds_cod.h

/usr/include/suds/suds_mem.h

/usr/include/suds/suds_str.h

/usr/include/suds/suds_var.h

AUTHOR

Peter L. Ward, U.S. Geological Survey, Menlo Park, CA 94025

NAME

beam_data – component of a beam of waveforms

C_SYNOPSIS

```
typedef struct {
        REFERS2     signal_path_id;
        DOMAIN      signal_path_dc;
        FLOAT4      delay;
        FLOAT4      weight;
} SUDS_BEAM_DATA;

#define BEAM_DATAS  318L
```

DESCRIPTION

When a **waveform** is formed by the addition of several waveforms, a separate **signal_path** structure should be created with at least **component** and **path_type** members reset and followed by a number of these structures. The beam azimuth and dip (a function of slowness) should be put in the **sensor_azimuth** and **sensor_dip** members. This method also applies to specifying radial and transverse components formed by summing signals from two horizontal sensors.

There is no way to restrict deletion of a **signal_path** from a database simply because it exists in data following a **signal_path or mux_waveform** structure. Thus it is conceivable to have **signal_path_ids** that are orphans pointing nowhere. Generally **signal_path structures** should not be deleted from a database.

[data_only=SIGNAL_PATHS]

MEMBERS

signal_path_id *signal path id*

A number that uniquely refers, within this **domain_code**, to one instance of the **signal_path** structure representing a total signal path from a particular sensor to its recorder. In some cases the same sensor and recorder may be connected by separate paths.

[key=part_foreign(1,signal_path.signal_path_id)]

signal_path_dc *signal path domain*

Domain in which signal_path_id is unique.

[codelist=authorities, key=part_foreign(1,signal_path.signal_path_dc)]

delay *time delay*

Time in seconds added to this signal_path waveform before summing to find array.

[index_string=true]

weight *weight*

Value this signal_path waveform was multiplied by before summing.

SEE ALSO

NAME

calibration – information about a calibration signal

C_SYNOPSIS

```
typedef struct {
        FIXED         structure_type;
        FIXED         structure_len;
        LABEL         calibration_id;
        DOMAIN        calibration_dc;
        REFERS2       waveform_id;
        DOMAIN        waveform_dc;
        MS_TIME       begin_time;
        MS_TIME       end_time;
        FLOAT4        amplitude;
        FLOAT4        frequency;
        CODE1         event_type;
        CODE1         ampl_units;
        CODE1         amplitude_type;
        CODE1         cause;
        CODE1         first_motion;
        CHAR          continuation;
        INT2          number;
        INT4          spare;
        AUTHOR        authority;
        REFERS2       comment_id;
        DOMAIN        comment_dc;
} SUDS_CALIBRATION;

#define CALIBRATIONS         320L
```

DESCRIPTION

When a **waveform** contains the output of a calibration signal, this structure describes the calibration signal input to the seismometer.

MEMBERS

structure_type *structure type*

Define number of this type of structure.

structure_len *structure length*

Length of this structure in bytes.

calibration_id *calibration id*

A number that uniquely identifies, within this **calibration_dc**, an instance of the **calibration** structure.

[key=part_primary, db_index=clustered]

calibration_dc *calibration domain*

Domain in which calibration_id is unique.

[codelist=authorities, key=part_primary]

waveform_id *waveform id*

Unique identification number of a waveform input to a system to make this calibration.

[key=part_foreign(1,waveform.waveform_id), db_delete=restrict, db_must_exist=true, index_string=true]

waveform_dc *waveform domain*

Domain in which waveform_id is unique.

[codelist=authorities, key=part_foreign(1,waveform.waveform_dc), db_delete=restrict, db_must_exist=true]

begin_time *Beginning time*

GMT time of the beginning of the calibration including all clock corrections.

end_time *ending time*
GMT time of the ending of the calibration including all clock corrections.

amplitude *amplitude*
Amplitude of calibration signal.

frequency *frequency*
Frequency of the calibration signal in hertz.

event_type *type of calibration*
[codelist=event_types]

ampl_units *units of amplitude*
Units used for amplitude.
[codelist=units_types]

amplitude_type *type of amplitude*
[codelist=amplitude_types]

cause *cause*
Manual or automatic.
[codelist=causes]

first_motion *first motion*
U=up, D=down, +=probable up, -=probable down
[codelist=first_motions]

continuation *continuation*
If this is a continuation of a previous calibration, set to c.

number *number of calibrations*
Number of applications of the calibration signal during this session.

spare *for future use*

authority *authority*
A number representing an institution or authority operating a network, calculating a solution, make an instrument calibration, etc. The authority is specified as a number that refers to an ASCII string in the authority codelist. Each institution has a base number such as 10000, 20000, etc. The institution may assign the 9999 numbers above their base number to individual people or groups. The individual number might be set to agree with the user number in /etc/passwd on UNIX systems.
[codelist=authorities]

comment_id *comment id*
A number that uniquely refers, within this comment_dc, to an instance of the **comment** structure. Comments are generally not searchable because they are not of standard format. Thus it is recommended that comments not be heavily used.
[key=part_foreign(2,comment.comment_id), db_delete=nullify]

comment_dc *comment domain*
Domain in which comment_id is unique.
[codelist=authorities, key=part_foreign(2,comment.comment_dc), db_delete=nullify]

SEE ALSO

NAME

clock_rate – rate of change of a clock error

C_SYNOPSIS

```
typedef struct {
        FIXED       structure_type;
        FIXED       structure_len;
        LABEL       clock_rate_id;
        DOMAIN      clock_rate_dc;
        REFERS2     signal_path_id;
        DOMAIN      signal_path_dc;
        MS_TIME     from_correct;
        MS_TIME     thru_correct;
        ST_TIME     from_time;
        ST_TIME     thru_time;
        FLOAT4      drift_rate;
        CHAR        obsolete;
        CHAR        satellite_hops;
        CODE1       sync_cd_type;
        CODE1       program_type;
        ST_TIME     time_done;
        AUTHOR      authority;
        REFERS2     comment_id;
        DOMAIN      comment_dc;
} SUDS_CLOCK_RATE;

#define CLOCK_RATES     130L
```

DESCRIPTION

Information about the rate that a clock drifts from true Greenwich Mean Time for a given recorder. A number of instances of this structure make a table of clock corrections that are searched to determine the time correction to be applied at any specific time. More than one instance of this structure may apply to a **signal_path** at the same time. For example a signal_path transmitted through an earth-orbiting satellite may have one correction for the satellite delay, and another for the clock at the recorder. A time tear is represented by two instances of this structure where the **thru_timeR of one equals the from_time** of the other.

Structures that contain time-critical values typically include a time member and a nominal-time member. Nominal time is the observed time without any clock corrections being applied. Time is the corrected time. In this way, if a timing error is discovered at a later date, a program can recalculate all corrected times based on a new table of time corrections.

MEMBERS

structure_type *structure type*
    Define number of this type of structure.

structure_len *structure length*
    Length of this structure in bytes.

clock_rate_id *clock rate id*
    Number of the clock correction that is unique within this domain.
    [key=part_primary, db_index=clustered]

clock_rate_dc *clock rate domain*
    Domain in which clock_rate_id is unique.
    [codelist=authorities, key=part_primary]

signal_path_id *signal path id*
    A number that uniquely refers, within this **domain_code**, to one instance of the **signal_path**

structure representing a total signal path from a particular sensor to its recorder. In some cases the same sensor and recorder may be connected by separate paths.
[key=part_foreign(1,signal_path.signal_path_id), db_delete=restrict, db_must_exist=true, db_index=true, index_string=true]

signal_path_dc *signal path domain*
Domain in which signal_path_id is unique.
[codelist=authorities, key=part_foreign(1,signal_path.signal_path_dc), db_delete=restrict, db_must_exist=true]

from_correct *from time correction*
True time minus the clock time in seconds at from_time.

thru_correct *thru time correction*
True time minus the clock time in seconds at thru_time.

from_time *valid from time*
Time this clock correction became valid.

thru_time *valid thru time*
Time this clock correction became no longer valid.

drift_rate *clock drift rate*
Drift rate of clock between **from_time** and **thru_time**.

obsolete *obsolete*
If this time clock-rate record has been replaced by another, set this field to the letter t. This allows keeping incorrect clock corrections as an audit trail to help decipher data corrected with a correction that was later changed.

satellite_hops *satellite hops*
Number of times this signal on a telephone line is transmitted through an earth-orbiting satellite. Each hop causes a delay of 0.27 seconds. If a number is given for this member, then the **from_correct** and **thru_correct** should be 0.27 times the number of hops. This instance of this structure may be in addition to other instances for a given **signal_path**. This field is an ASCII character 1 thru 9, not an integer.

sync_cd_type *syncronization code*
Method used to determine time correction used.
[codelist=synchronization_types]

program_type *program type*
Type of program used.
[codelist=clock_programs]

time_done *time correction done*
Time that this correction was determined.

authority *pick authority*
Who determined this correction.
[codelist=authorities]

comment_id *comment id*
A number that uniquely refers, within this comment_dc, to an instance of the **comment** structure. Comments are generally not searchable because they are not of standard format. Thus it is recommended that comments not be heavily used.
[key=part_foreign(2,comment.comment_id), db_delete=nullify]

comment_dc *comment domain*
Domain in which comment_id is unique.
[codelist=authorities, key=part_foreign(2,comment.comment_dc), db_delete=nullify]

SEE ALSO

NAME

coordinate_sys – information to define a local coordinate system

C_SYNOPSIS

```
typedef struct {
        FIXED       structure_type;
        FIXED       structure_len;
        LABEL       coordinate_id;
        DOMAIN      coordinate_dc;
        FIXED       len_grid_name;
        STRING      grid_name[20];
        FLOAT4      a_trans_coeff;
        FLOAT4      b_trans_coeff;
        FLOAT4      c_trans_coeff;
        FLOAT4      d_trans_coeff;
        FLOAT4      e_trans_coeff;
        FLOAT4      f_trans_coeff;
        CODE1       map_projection;
        CODE1       horizontal_ref;
        CODE1       spheroid;
        CODE1       prime_merid;
        FLOAT4      central_merid;
        FLOAT4      scale_factor;
        FLOAT4      northing;
        FLOAT4      easting;
        FLOAT4      semi_major;
        FLOAT4      flattening;
        FLOAT4      primary_merid;
        FLOAT8      origin_lat;
        FLOAT8      origin_long;
        REFERS2     comment_id;
        DOMAIN      comment_dc;
} SUDS_COORDINATE_SYS;

#define COORDINATE_SYSS     326L
```

DESCRIPTION

The absolute location and orientation of a local coordinate system is specified in terms of the complete geodetic coordinates, the map projection transformation to rectangular coordinates, and the affine transformation to the local coordinates. Since the earth's shape is too irregular to describe locations conveniently and mathematically, common practice is to assume the earth is a spheroid with a specific semi_major axis and flattening. Then a map projection is used to transform this spheroid onto a planar coordinate system. Finally an affine transformation is used to apply a linear translation in X and Y, a clockwise rotation, and a scale change.

MEMBERS

structure_type *structure type*

Define number of this type of structure.

structure_len *structure length*

Length of this structure in bytes.

coordinate_id *coordinate id*

A number that uniquely refers, within this **coordinate_dc**, to an instance of the **coordinate_sys** structure.

[key=part_primary, db_index=clustered]

coordinate_dc *coordinate domain*

Domain in which coordinate_id is unique.
[codelist=authorities, key=part_primary]

len_grid_name *length signal name*
The maximum space reserved for the grid name, i.e. 20. Actual string can only contain 19 characters to allow for the NULL byte.

grid_name *grid name*
Name assigned to this particular local grid.
[index_string=true]

a_trans_coeff *affine coefficient a*
a = scale_change times the cosine of the angle of rotation.

b_trans_coeff *affine coefficient b*
b = minus scale_change times the sine of the angle of rotation.

c_trans_coeff *affine coefficient c*
c = map_projection easting minus bin_grid_X.

d_trans_coeff *affine coefficient d*
d = scale_change times the sine of the angle of rotation.

e_trans_coeff *affine coefficient e*
e = scale_change times cosine of the angle of rotation.

f_trans_coeff *affine coefficient f*
f = map_projection northing minus bin_grid_Y.

map_projection *map projection*
Map projection used.
[codelist=map_projections]

horizontal_ref *name of horiz. datum*
Name of the horizontal reference surface used.
[codelist=horiz_datums]

spheroid *name of spheroid*
Spheroid to which coordinates are referenced.
[codelist=spheroids]

prime_merid *prime meridian*
Name of a meridian from which longitudes are reckoned. Normally the meridian through Greenwich, England, is defined as the prime meridian.
[codelist=prime_meridians]

central_merid *central meridian*
Central meridian for this map projection.

scale_factor *projection scale factor*
Scale factor for this map projection.

northing *projection northing*
Northing of origin for this projection.

easting*projection easting*
Easting of origin for this projection.

semi_major *semi major axis*
Semi-major axis of this spheroid.

flattening *flattening of spheroid*
Flattening of this spheroid.

primary_merid *primary meridian*

Primary meridian of this spheroid.

origin_lat *projection origin lat*
Latitude of the origin of this projection.

origin_long *projection origin long*
Longitude of the origin of this projection.

comment_id *comment id*
A number that uniquely refers, within this comment_dc, to an instance of the **comment** structure. Comments are generally not searchable because they are not of standard format. Thus it is recommended that comments not be heavily used.
[key=part_foreign(1,comment.comment_id), db_delete=nullify]

comment_dc *comment domain*
Domain in which comment_id is unique.
[codelist=authorities, key=part_foreign(1,comment.comment_dc), db_delete=nullify]

SEE ALSO

BUGS

NAME

data_group – information about storage of a collection of waveforms

C_SYNOPSIS

```
typedef struct {
        FIXED           structure_type;
        FIXED           structure_len;
        LABEL           data_group_id;
        DOMAIN          data_group_dc;
        REFERS2         source_id;
        DOMAIN          source_dc;
        MS_TIME         from_time;
        MS_TIME         thru_time;
        CODE1           media_type;
        CODE1           group_type;
        INT2            spare;
        FIXED           len_media_l;
        STRING          media_label[16];
        FIXED           len_media_p;
        STRING          media_path[64];
        INT4            media_block;
        INT4            job_number;
        INT4            line_number;
        INT4            reel_number;
        FIXED           len_online_p;
        STRING          online_path[64];
        REFERS2         comment_id;
        DOMAIN          comment_dc;
} SUDS_DATA_GROUP;

#define DATA_GROUPS     108L
```

DESCRIPTION

data_group identifies a collection of **waveform** structures and their corresponding waveforms, and specifies where this collection is archived, and (if applicable) its disk location on-line. For refraction or reflection type data, a **data_group** is a equivalent to a shot_gather.

In an institution where the majority of event-detected data is recorded on a single machine, the beginning time of a recorded event is used as an identifier of a **data_group**. **waveform** structures and waveforms from other sources, possibly with differing start times but covering the same event, may later also be associated with a **data_group**. Normally, only one event is present in the **data_group** collection, but during aftershock sequences and volcanic eruptions, a single **data_group** may contain several events of interest.

[permissions="siu_siu_s_s"]

MEMBERS

structure_type *structure type*

Define number of this type of structure.

structure_len *structure length*

Length of this structure in bytes.

data_group_id *data group id*

A number identifying a collection of waveform data. The number is assigned by an authority when many waveforms are associated into a group that normally contains all the waveforms for one earthquake. The value must be unique within a domain and is assumed to be of type **ST_TIME** (i.e. seconds since the beginning of Jan, 1970) representing a time at or near the time of the first samples of much of the data. In practice this number would typically be

assigned when the data from the primary network detector are demultiplexed. Then as data from other detectors are added, they are assigned this data_group_id. **waveform** structures and their associated waveforms for all station components within a data group will usually be stored together either in a file or in a directory with the name based on the ASCII representation of this time. The ascii string is of the form: YYMMDD.HHMMSS, where YY is the year (00-99), MM is the month (01-12), DD is the day(01-31), HH is the hour (00-23), MM is the minute (00-59), and SS the second (00-59), in universal (GMT) time. For example 910824.123600
[key=part_primary, db_index=clustered]

data_group_dc *data group domain*
Domain in which data_group_id is unique.
[key=part_primary, codelist=authorities]

source_id *source id*
A number that uniquely identifies, within this **source_dc**, an instance of the **source** structure.
[key=part_foreign(1,source.source_id), db_delete=nullify]

source_dc *source domain*
Domain in which source_id is unique.
[codelist=authorities, key=part_foreign(1,source.source_dc)]

from_time *beginning time*
GMT time of the first sample in the waveforms including all clock corrections.

thru_time *ending time*
GMT time of the last sample in the waveform including all clock corrections.

media_type *type of media*
Specifies the media on which this data group is archived.
[codelist=media]

group_type *data group type*
Type of this data_group.
[codelist=data_group_types]

spare *for future use*

len_media_l *length of media label*
The maximum space reserved for the label written on the media, i.e. 12. Actual string can only contain 11 characters to allow for the NULL byte.

media_label *media label*
Specifies the label written on the optical disk or tape containing the data. In other words the name that is physically written on the paper label stuck to the disk or tape, for example: May-1990.

len_media_p *len media pathname*
The maximum space reserved for the media_path string, i.e. 64. Actual string can only contain 63 characters to allow for the NULL byte.

media_path *pathname on media*
Name of the file on the media containing the data.
[ed_col=18]

media_block *block on media*
Block in which this **data_group** occurs on the media.

job_number *job number*
Primarily for reflection/refraction jobs.

line_number *line number*
Primarily for reflection/refraction jobs.

reel_number *reel number*
Primarily for reflection/refraction jobs.

len_online_p *len online pathname*
The maximum space reserved for the path string, i.e. 64. Actual string can only contain 63 characters to allow for the NULL byte.

online_path *pathname online*
Specifies where to find the data if it is mounted on the computer or network. This field may be valid for only a short period of time, while the data is actively being processed.
[ed_col=18, index_string=true]

comment_id *comment id*
A number that uniquely refers, within this comment_dc, to an instance of the **comment** structure. Comments are generally not searchable because they are not of standard format. Thus it is recommended that comments not be heavily used.
[key=part_foreign(2,comment.comment_id), db_delete=nullify]

comment_dc *comment domain*
Domain in which comment_id is unique.
[codelist=authorities, key=part_foreign(2,comment.comment_dc), db_delete=nullify]

SEE ALSO

NAME

event – information about processing of an event

C_SYNOPSIS

```
typedef struct {
        FIXED           structure_type;
        FIXED           structure_len;
        LABEL           event_id;
        DOMAIN          event_dc;
        REFERS2         data_group_id;
        DOMAIN          data_group_dc;
        FIXED           geo_name_len;
        STRING          geographic_name[36];
        FLOAT4          distance;
        FLOAT4          azimuth;
        CODE1           event_type;
        CHAR            local_1_cd;
        CHAR            local_2_cd;
        CHAR            local_3_cd;
        CHAR            local_4_cd;
        CHAR            local_5_cd;
        CHAR            local_6_cd;
        CHAR            local_7_cd;
        REFERS2         comment_id;
        DOMAIN          comment_dc;
} SUDS_EVENT;

#define EVENTS          112L
```

DESCRIPTION

The event table allows many events to be associated with one data group, and many solutions to be associated with each event. It also provides for seven processing or quality control status codes whose meaning may be defined locally by each institution.
[permissions="siud_siu_s_s"]

MEMBERS

structure_type *structure type*
Define number of this type of structure.

structure_len *structure length*
Length of this structure in bytes.

event_id *event id*
A number that uniquely refers, within this event_dc, to an instance of the **event** structure.
[key=part_primary, db_index=clustered]

event_dc *event domain*
Domain in which event_id is unique.
[codelist=authorities, key=part_primary]

data_group_id *data group id*
A number identifying a collection of waveform data. The number is assigned by an authority when many waveforms are associated into a group that normally contains all the waveforms for one earthquake. The value must be unique within a domain and is assumed to be of type **ST_TIME** (i.e. seconds since the beginning of Jan, 1970) representing a time at or near the time of the first samples of much of the data. In practice this number would typically be assigned when the data from the primary network detector are demultiplexed. Then as data from other detectors are added, they are assigned this data_group_id. **waveform** structures and

their associated waveforms for all station components within a data group will usually be stored together either in a file or in a directory with the name based on the ASCII representation of this time. The ascii string is of the form: YYMMDD.HHMMSS, where YY is the year (00-99), MM is the month (01-12), DD is the day(01-31), HH is the hour (00-23), MM is the minute (00-59), and SS the second (00-59), in universal (GMT) time. For example 910824.123600

[key=part_foreign(1,data_group.data_group_id), db_delete=restrict, db_must_exist=true, index_string=true]

data_group_dc *data group domain*

Domain in which data_group_id is unique.

[codelist=authorities, key=part_foreign(1,data_group.data_group_dc), db_delete=restrict, db_must_exist=true]

geo_name_len

The maximum space reserved for the earthquake name, i.e. 36. Actual string can only contain 35 characters to allow for the NULL byte.

geographic_name

Geographic name of a feature near this event, to which event can be referenced by distance and azimuth. e.g. 10 km NE of San Francisco.

distance

Distance in kilometers of event from geographic named location.

azimuth

Azimuth in degrees clockwise from north of event from geographic named location.

event_type *type of event*

A character designating the type of event.

[codelist=event_types]

local_1_cd *local code 1*

Locally defined code for processing status information.

local_2_cd *local code 2*

Locally defined code for processing status information.

local_3_cd *local code 3*

Locally defined code for processing status information.

local_4_cd *local code 4*

Locally defined code for processing status information.

local_5_cd *local code 5*

Locally defined code for processing status information.

local_6_cd *local code 6*

Locally defined code for processing status information.

local_7_cd *local code 7*

Locally defined code for processing status information.

comment_id *comment id*

A number that uniquely refers, within this comment_dc, to an instance of the **comment** structure. Comments are generally not searchable because they are not of standard format. Thus it is recommended that comments not be heavily used.

[key=part_foreign(2,comment.comment_id), db_delete=nullify]

comment_dc *comment domain*

Domain in which comment_id is unique.

[codelist=authorities, key=part_foreign(2,comment.comment_dc), db_delete=nullify]

SEE ALSO

NAME

filter – specifies filtering applied to waveforms

C_SYNOPSIS

```
typedef struct {
        FIXED           structure_type;
        FIXED           structure_len;
        REFERS2         waveform_id;
        DOMAIN          waveform_dc;
        REFERS2         response_id;
        DOMAIN          response_dc;
        REFERS2         prev_wave_id;
        DOMAIN          prev_wave_dc;
        AUTHOR          authority;
        INT2            position;
        CODE1           decim_type;
        CHAR            decim_points;
        INT2            decim_interv;
        INT2            decim_index;
        INT4            spare;
        REFERS2         comment_id;
        DOMAIN          comment_dc;
} SUDS_FILTER;

#define FILTERS         315L
```

DESCRIPTION

Associates a **waveform** with the response of a filter that has been applied to a waveform. Any waveform that has been filtered should indicate so in its **waveform** structure using the **filter_code** member. One **filter** structure should exist for every filter that has been applied to a waveform. The intermediate waveforms resulting from several filtering operations may or may not be saved. This structure is also used to specify decimation.

[permissions="siud_siu_s_s"]

MEMBERS

structure_type *structure type*

Define number of this type of structure.

structure_len *structure length*

Length of this structure in bytes.

waveform_id *waveform id*

A number that uniquely refers, within this **waveform_dc**, to an instance of the **waveform** structure and waveform after the specified filtering has been applied.

[key=part_primary, key=part_foreign(1,waveform.waveform_id), db_delete=cascade, db_must_exist=true, index_string=true]

waveform_dc *waveform domain*

Domain in which waveform_id is unique.

[codelist=authorities, key=part_primary, key=part_foreign(1,waveform.waveform_dc), db_delete=cascade, db_must_exist=true]

response_id *response id*

A number that uniquely refers, within this **response_dc**, to an instance of the **response** structure that specifies the filter response.

[key=part_primary, key=part_foreign(2,response.response_id), db_delete=restrict, db_must_exist=true]

response_dc *response domain*

Domain in which response_id is unique.

[codelist=authorities, key=part_primary, key=part_foreign(2,response.response_dc), db_delete=restrict, db_must_exist=true]

prev_wave_id *previous waveform id*

A number that uniquely refers, within this **waveform_dc**, to an instance of the **waveform** structure and waveform before the specified filtering has been applied.

[key=part_foreign(3,waveform.waveform_id), db_delete=cascade, db_must_exist=true]

prev_wave_dc *waveform domain*

Domain in which prev_wave_id is unique.

[codelist=authorities, key=part_foreign(3,waveform.waveform_dc), db_delete=cascade, db_must_exist=true]

authority *authority for filter*

Who designed this filter.

[codelist=authorities]

position *position in sequence*

The position of this filter in a sequence of filters. The first filter applied is position 1, the second is position 2, etc.

decim_type *decimation type*

Type of decimation done: s=simple, a=average, e=envelope

[codelist=decimation_types]

decim_points *decimation res pts*

Number of resulting points from **decim_interv**. For simple decimation, this equals 1. For envelope decimation this equals 2.

[default_value="1"]

decim_interv *decimation interval*

Number of original sample taken to produce result. If every tenth sample is taken in simple decimation where **decim_points** is 1, the **decim_interv** *is 10.*

decim_index *decimation index*

Index of the first decimated sample in the waveform. If not equal to 0, then **decim_index**-*1 samples were discarded before the first sample.*

spare *for future use*

comment_id *comment id*

A number that uniquely refers, within this comment_dc, to an instance of the **comment** structure. Comments are generally not searchable because they are not of standard format. Thus it is recommended that comments not be heavily used.

[key=part_foreign(4,comment.comment_id), db_delete=nullify]

comment_dc *comment domain*

Domain in which comment_id is unique.

[codelist=authorities, key=part_foreign(4,comment.comment_dc), db_delete=nullify]

SEE ALSO

NAME

focal_mech – information about the focal mechanism and moment for a solution of an event

C_SYNOPSIS

```
typedef struct {
        FIXED       structure_type;
        FIXED       structure_len;
        LABEL       focal_mech_id;
        DOMAIN      focal_mech_dc;
        REFERS2     solution_id;
        DOMAIN      solution_dc;
        MS_TIME     origin_time;
        LATIT       origin_lat;
        LONGIT      origin_long;
        FLOAT4      origin_depth;
        CHAR        prefer_plane;
        CODE1       mechanism_type;
        CODE1       time_func_type;
        CHAR        spare;
        REFERS2     vel_model_id;
        DOMAIN      vel_model_dc;
        MS_TIME     centroid_time;
        LATIT       centroid_lat;
        LONGIT      centroid_long;
        FLOAT4      centroid_depth;
        FLOAT4      cent_time_err;
        FLOAT4      cent_lat_err;
        FLOAT4      cent_long_err;
        FLOAT4      cent_depth_err;
        FLOAT4      time_func_dur;
        FLOAT4      scalar_moment;
        FLOAT4      scalar_mom_err;
        FLOAT4      moment_magnit;
        FLOAT4      stress_drop;
        FLOAT4      a_strike;
        FLOAT4      a_strike_err;
        FLOAT4      a_dip;
        FLOAT4      a_dip_err;
        FLOAT4      a_rake;
        FLOAT4      a_rake_err;
        FLOAT4      b_strike;
        FLOAT4      b_strike_err;
        FLOAT4      b_dip;
        FLOAT4      b_dip_err;
        FLOAT4      b_rake;
        FLOAT4      b_rake_err;
        FLOAT4      moment_xx;
        FLOAT4      moment_yy;
        FLOAT4      moment_zz;
        FLOAT4      moment_xy;
        FLOAT4      moment_xz;
        FLOAT4      moment_yz;
        FLOAT4      eigen_pressure;
        FLOAT4      plunge_pressure;
```

```
        FLOAT4          strike_pressure;
        FLOAT4          eigen_null;
        FLOAT4          plunge_null;
        FLOAT4          strike_null;
        FLOAT4          eigen_tension;
        FLOAT4          plunge_tension;
        FLOAT4          strike_tension;
        FLOAT4          percent_dc;
        FLOAT4          percent_clvd;
        FLOAT4          percent_iso;
        INT4            authority;
        ST_TIME         from_time;
        CODE4           data_type;
        INT4            data_length;
        REFERS2         comment_id;
        DOMAIN          comment_dc;
} SUDS_FOCAL_MECH;

#define FOCAL_MECHS     116L
```

DESCRIPTION

Focal mechanism of an earthquake including focal planes, moment, and stress-axes. May be followed by structures of type **sig_path_data** to list stations used in this moment of focal mechanism.
[permissions="siud_siu_s_s"]

MEMBERS

structure_type *structure type*
Define number of this type of structure.

structure_len *structure length*
Length of this structure in bytes.

focal_mech_id *focal mechanism id*
Unique number of this focal mechanism.
[key=part_primary, db_index=clustered]

focal_mech_dc *focal mechanism domain*
Domain in which focal_mech_id is unique.
[codelist=authorities, key=part_primary]

solution_id *solution id*
A number that uniquely refers, within this solution_dc, to an instance of the **solution** structure.
[key=part_foreign(1,solution.solution_id), db_delete=nullify, index_string=true]

solution_dc *solution domain*
Domain in which solution_id is unique.
[codelist=authorities, key=part_foreign(1,solution.solution_dc), db_delete=nullify]

origin_time *origin time*
Origin time. These origin parameters were those used in the solution refered to by **solution_id**. They are included because often a moment that is not very sensitive to the starting solution may be calculated based on a soution that will later be discarded as too preliminary. If the **solution_id** is defined, parameters from that structure should be used in preference to the values given in this structure.

origin_lat *origin latitude*
Latitude, south is negative.

origin_long *origin longitude*
Longitude, west is negative.

origin_depth *origin depth*
Depth of hypocenter in kilometers below the ground surface.

prefer_plane *preferred plane*
Preferred slip plane, either a or b.
[allow_char="ab"]

mechanism_type *mechanism type*
[codelist=mechanism_types]

time_func_type *time function type*
[codelist=time_func_types]

spare *for future use*

vel_model_id *velocity model id*
Unique identifier of velocity model used to calculate this residual.
[key=part_foreign(2,vel_model.vel_model_id), db_delete=cascade, db_must_exist=true]

vel_model_dc *velocity model domain*
Domain in which vel_model_id is unique.
[codelist=authorities, key=part_foreign(2,vel_model.vel_model_dc), db_delete=cascade, db_must_exist=true]

centroid_time *centroid time*

centroid_lat *centroid latitude*

centroid_long *centroid longitude*

centroid_depth *centroid depth*

cent_time_err *centroid time error*

cent_lat_err *centroid latitude error*

cent_long_err *centroid longit error*

cent_depth_err *centroid depth error*

time_func_dur *time function duration*

scalar_moment *scalar moment*

scalar_mom_err *scalar moment error*

moment_magnit *moment magnitude*

stress_drop *stress drop*

a_strike *strike of a plane*
Strike of the **a** plane.

a_strike_err *strike a plane error*
Error in strike of the **a** plane.

a_dip *dip of the a plane*
Dip of the **a** plane.

a_dip_err *dip of the a plane*
Dip of the **a** plane.

a_rake *rake of a plane*
Rake of the **a** plane.

a_rake_err *rake a plane error*
Error in rake of the **a** plane.

b_strike *strike of b plane*
Strike of the **b** plane.

b_strike_err *strike b plane error*
Error in strike of the **b** plane.

b_dip *dip of b plane*
Dip of the **b** plane.

b_dip_err *dip b plane error*
Error in dip of the **b** plane.

b_rake *rake of b plane*
Rake of the **b** plane.

b_rake_err *rake b plane error*
Error in rake of the **b** plane.

moment_xx *moment xx*

moment_yy *moment yy*

moment_zz *moment zz*

moment_xy *moment xy*

moment_xz *moment xz*

moment_yz *moment yz*

eigen_pressure *eigenval pressure axis*

plunge_pressure *plunge pressure axis*

strike_pressure *strike pressure axis*

eigen_null *eigenvalue null axis*

plunge_null *plunge null axis*

strike_null *strike null axis*

eigen_tension *eigenvalue tension axis*

plunge_tension *plunge tension axis*

strike_tension *strike tension axis*

percent_dc *percent dc*

percent_clvd *percent clvd*

percent_iso *percent iso*

authority *authority*

from_time *time of solution*

data_type *data storage type*
An integer representing the type of data that follows this structure. If the integer is negative, it refers to **variable_tab.define_num**. If the integer is positive, it refers to **structure_tab.struct_number**. This structure should be followed by structures of type **sig_path_data**.
[sets_data_type=true, codelist=data_types]

data_length *number of samples*
Number of samples of type **data_type** in the waveform.
[sets_data_length=true, default_value=0]

comment_id *comment id*
A number that uniquely refers, within this comment_dc, to an instance of the **comment** structure. Comments are generally not searchable because they are not of standard format. Thus it is recommended that comments not be heavily used.

[key=part_foreign(3,comment.comment_id), db_delete=nullify]

comment_dc *comment domain*

Domain in which comment_id is unique.

[codelist=authorities, key=part_foreign(3,comment.comment_dc), db_delete=nullify]

SEE ALSO

AUTHOR

Peter Ward, U. S. Geological Survey and Lind Gee, University of California, Berkeley.

NAME

lsa_detection – a specific long-term, short-term average event detection

C_SYNOPSIS

```
typedef struct {
        FIXED        structure_type;
        FIXED        structure_len;
        LABEL        detection_id;
        DOMAIN       detection_dc;
        REFERS2      signal_path_id;
        DOMAIN       signal_path_dc;
        REFERS2      lsa_setting_id;
        DOMAIN       lsa_setting_dc;
        MS_TIME      lsa_onset_time;
        FLOAT4       amplitude;
        FLOAT4       frequency;
        FLOAT4       signal_2_noise;
        FLOAT4       longterm_ave;
        FLOAT4       shortterm_ave;
        FLOAT4       other_ave;
        FLOAT4       level;
        INT2         local_1;
        INT2         local_2;
        INT2         local_3;
        INT2         local_4;
        INT2         local_5;
        INT2         local_6;
        CODE1        event_type;
        CODE1        first_motion;
        INT2         num_detections;
        AUTHOR       authority;
        REFERS2      comment_id;
        DOMAIN       comment_dc;
} SUDS_LSA_DETECTION;

#define LSA_DETECTIONS        125L
```

DESCRIPTION

Values at the time of a specific lsa_detection in an event detection program using the long-term, short-term average technique. The values used to define the lsa_detection are specified in **lsa_setting(5)**.

MEMBERS

structure_type *structure type*

Define number of this type of structure.

structure_len *structure length*

Length of this structure in bytes.

detection_id *detection id*

A number that uniquely refers, within this solution_dc, to an instance of the **solution** structure.
[key=part_primary, db_index=clustered]

detection_dc *detection domain*

Domain in which detection_id is unique.
[codelist=authorities, key=part_primary]

signal_path_id *signal path id*

A number that uniquely refers, within this **domain_code**, to one instance of the **signal_path**

structure representing a total signal path from a particular sensor to its recorder. In some cases the same sensor and recorder may be connected by separate paths.
[key=part_foreign(1,signal_path.signal_path_id), db_delete=restrict, db_must_exist=true, index_string=true]

signal_path_dc *signal path domain*
Domain in which signal_path_id is unique.
[codelist=authorities, key=part_foreign(1,signal_path.signal_path_dc), db_delete=restrict, db_must_exist=true]

lsa_setting_id *lsa setting id*
A number that uniquely identifies a particular **lsa_setting(5)** structure.
[key=part_foreign(2,lsa_setting.lsa_setting_id), db_delete=restrict, db_must_exist=true]

lsa_setting_dc *lsa setting domain*
Domain in which lsa_setting_id is unique.
[codelist=authorities, key=part_foreign(2,lsa_setting.lsa_setting_dc), db_delete=restrict, db_must_exist=true]

lsa_onset_time *lsa onset time*
Time the lsa_detection was issued.

amplitude *amplitude*
Amplitude of the triggering signal.

frequency *frequency*
Frequency in hertz of the triggering signal.

signal_2_noise *signal to noise*
Ratio of signal to noise.

longterm_ave *longterm average*
Long term average at the time of triggering.

shortterm_ave *shortterm average*
Short term average at the time of triggering.

other_ave *other average*
Other average at the time of triggering, dependent on algorithm.

level *level*
Level at the time of the trigger.

local_1 *local const 1*
Variable defined for this specific algorithm.

local_2 *local const 2*
Variable defined for this specific algorithm.

local_3 *local const 3*
Variable defined for this specific algorithm.

local_4 *local const 4*
Variable defined for this specific algorithm.

local_5 *local const 5*
Variable defined for this specific algorithm.

local_6 *local const 6*
Variable defined for this specific algorithm.

event_type *event type*
A character designating the type of event.
[codelist=event_types]

first_motion *first motion*
U=up, D=down, +=probable up, -=probable down
[codelist=first_motions]

num_detections *number of detections*
Number of station_components that triggered.

authority *authority*
Which machine made this trigger.
[codelist=authorities]

comment_id *comment id*
A number that uniquely refers, within this comment_dc, to an instance of the **comment** structure. Comments are generally not searchable because they are not of standard format. Thus it is recommended that comments not be heavily used.
[key=part_foreign(3,comment.comment_id), db_delete=nullify]

comment_dc *comment domain*
Domain in which comment_id is unique.
[codelist=authorities, key=part_foreign(3,comment.comment_dc), db_delete=nullify]

SEE ALSO

lsa_setting(5)

NAME

lsa_setting – settings of long-term, short-term average event detection program

C_SYNOPSIS

```
typedef struct {
        FIXED       structure_type;
        FIXED       structure_len;
        LABEL       lsa_setting_id;
        DOMAIN      lsa_setting_dc;
        CODE1       algorithm;
        CHAR        spare;
        INT2        decimation;
        INT2        num_intervals;
        INT2        spare_a;
        MS_TIME     start;
        FLOAT4      before_secs;
        FLOAT4      after_secs;
        FLOAT4      begin_level;
        FLOAT4      end_level;
        FLOAT4      sweep;
        FLOAT4      aperture;
        FLOAT4      constant_1;
        FLOAT4      constant_2;
        FLOAT4      constant_3;
        FLOAT4      constant_4;
        FLOAT4      constant_5;
        FLOAT4      constant_6;
        FLOAT4      constant_7;
        AUTHOR      authority;
        ST_TIME     from_time;
        ST_TIME     thru_time;
        CODE4       data_type;
        INT4        data_length;
        REFERS2     comment_id;
        DOMAIN      comment_dc;
} SUDS_LSA_SETTING;

#define LSA_SETTINGS        126L
```

DESCRIPTION

Settings of the trigger for an event detection program. The values at the time of a specific trigger are specified in **lsa_detection(4)**. The structure should not only contains settings for a trigger, but be able to provide input to setup the trigger.

MEMBERS

structure_type *structure type*

Define number of this type of structure.

structure_len *structure length*

Length of this structure in bytes.

lsa_setting_id *signal path id*

A number that uniquely refers, within this **lsa_setting_dc**, to an instance of the **lsa_setting** structure.

[key=part_primary, db_index=clustered]

lsa_setting_dc *signal path domain*

Domain in which lsa_setting_id is unique.

[codelist=authorities, key=part_primary]

algorithm *detection algorithm*
Type of detection or trigger algorithm used.
[codelist=detector_types]

spare *for future use*

decimation *decimation*
Number of samples to decimate.

num_intervals *number of intervals*
For a time trigger, number of times a new trigger should start **aperture** number of seconds after the previous trigger.

spare_a *for future use*

start *start time*
Time a time-trigger should start.

before_secs *seconds before*
Save data starting this many seconds before the onset of the trigger.

after_secs *seconds after*
Save data ending this many seconds after the trigger shuts off.

begin_level *begin trigger*
Level above which a signal must be in order to start a trigger.

end_level *end trigger*
Level below which a signal must be in order to end a trigger.

sweep *sweep*
Time in seconds over which a short-term average is calculated.

aperture *aperture*
Time in seconds during which triggers at different stations must occur to declare that an event has occurred.

constant_1 *constant 1*
Constant whose meaning depends on the algorithm.

constant_2 *constant 2*
Constant whose meaning depends on the algorithm.

constant_3 *constant 3*
Constant whose meaning depends on the algorithm.

constant_4 *constant 4*
Constant whose meaning depends on the algorithm.

constant_5 *constant 5*
Constant whose meaning depends on the algorithm.

constant_6 *constant 6*
Constant whose meaning depends on the algorithm.

constant_7 *constant 7*
Constant whose meaning depends on the algorithm.

authority *authority*
Who set these values.
[codelist=authorities, index_string=true]

from_time *valid from time*
Time these settings became valid.

thru_time *valid thru time*
Time these settings became no longer valid.

data_type *data type*
An integer representing the type of data that follows this structure. If the integer is negative, it refers to **variable_tab.define_num**. If the integer is positive, it refers to **structure_tab.struct_number**. Should be of type **LSA_SET_DATAS** only.
[sets_data_type=true, codelist=data_types]

data_length *data length*
Total number of samples of **data_type** following this structure. Number of samples per station = **data_length**/**numb_stations**.
[sets_data_length=true, default_value=0]

comment_id *comment id*
A number that uniquely refers, within this comment_dc, to an instance of the **comment** structure. Comments are generally not searchable because they are not of standard format. Thus it is recommended that comments not be heavily used.
[key=part_foreign(1,comment.comment_id), db_delete=nullify]

comment_dc *comment domain*
Domain in which comment_id is unique.
[codelist=authorities, key=part_foreign(1,comment.comment_dc), db_delete=nullify]

SHORT-TERM/LONG-TERM AVERAGE TRIGGER

A common algorithm for triggering based on long-term and short-term averages is the following:

1) For each signal_path (s) of interest, sum each sample (sum[s]) and sum abs_sum[s]) the absolute value of each sample minus the long-term average (lta[s]) during **sweep** seconds.

2) Calculate **eta** which is the short-term absolute average minus the weighted long-term average minus the absolute value of the difference of the long-term and short-term averages (which corrects for DC offset). Where **ns_sweep** is the number of samples in a **sweep**, then
eta = abs_sum[s]/ns_sweep-(constant_1*abs_ltas[s])/constant_2 - abs(ltas[s]-(sum[s]/ns_sweep))

3) If **eta** is > **begin_level**, then set
sta_trigger[s] = aperture/sweep

4) Recalculate long-term averages
ltas[s] = ((sum[s]/ns_sweep)+constant_3*ltas[s])/constant_4
if **ltas[s]** does not change, increment lta by 1 if (sta[s] > lta[s]) or -1 if (sta[s] < lta[s])
abs_ltas[s]= ((abs_sum[s]/ns_sweep)+constant_3*abs_ltas[s])/constant_4
if **abs_ltas[s]** does not change, increment abs_lta by constant_5 if (abs_sta[s] > abs_lta[s]) or -constant_5 if (abs_sta[s] > abs_lta[s])

5) Age trigger
if(sta_trigger[s]>0) sta_trigger[s]--

6) Scan each subnet, if **min_channels** are triggered during **aperature**, then declare an event.

7) Process next sweep.

Typical values for a regional-network trigger are **before_secs = 20.0, after_secs = 60.0, aperture = 20.0, sweep = 3.0, constant_1 = 2.0, constant_2 = 1.0, constant_3 = 7.0, constant_4 = 8.0, constant_5 = 1.0, begin_level = 5.0, decimation = 1**

TIME TRIGGER

Specify the time of the trigger in **start**, the length in **before_secs** and **after_secs**, the number of subsequent triggers in **num_intervals**, to occur one after the other separated by **aperture** seconds.

LEVEL TRIGGER

Specify **begin_level**, **end_level**, **before_secs**, and **after_secs**, which is the maximum number of seconds of data after the trigger allowed in a single trigger.

CROSS TRIGGER

SEE ALSO

lsa_detection(4), lsa_set_data(4)

NAME

lsa_set_data – signal_paths and subnets for event and cross triggers

C_SYNOPSIS

```
typedef struct {
        REFERS2         signal_path_id;
        DOMAIN          signal_path_dc;
        INT2            subnet;
        INT2            min_channels;
        INT4            spare;
} SUDS_LSA_SET_DATA;

#define LSA_SET_DATAS   304L
```

DESCRIPTION

A number of these structures normally follow an **lsa_setting** structure specifying which **signal_paths** should be processed tp detect events, which subgroup they occur in, and the minimum number of detections that must occur in that subgroup.

[permissions="siu_s_siu_s", data_only=LSA_SETTINGS]

MEMBERS

signal_path_id *signal path id*

A number that uniquely refers, within this **domain_code**, to one instance of the **signal_path** structure representing a total signal path from a particular sensor to its recorder. In some cases the same sensor and recorder may be connected by separate paths.

[key=part_foreign(1,signal_path.signal_path_id)]

signal_path_dc *signal path domain*

Domain in which signal_path_id is unique.

[codelist=authorities, key=part_foreign(1,signal_path.signal_path_dc)]

subnet *subnet number*

Number of the subnet counting up from zero.

min_channels *minimum channels*

Minimum number of channels that must trigger within this subnet to declare an event. It is possible that different **min_channels** could be given for the same subgroup. In this case it is the algorithm's responsibility to specify an error or make a choice.

spare *for future use*

SEE ALSO

lsa_setting(4), lsa_detection(4)

NAME

magnitude – magnitude calculated for a solution

C_SYNOPSIS

```
typedef struct {
        FIXED       structure_type;
        FIXED       structure_len;
        LABEL       magnitude_id;
        DOMAIN      magnitude_dc;
        REFERS2     solution_id;
        DOMAIN      solution_dc;
        FLOAT4      mag_value;
        FLOAT4      mag_error;
        CODE1       mag_type;
        CHAR        preferred;
        INT2        num_reports;
        INT2        num_used;
        INT2        spare_a;
        FLOAT4      rms_of_mag;
        AUTHOR      authority;
        REFERS2     comment_id;
        DOMAIN      comment_dc;
} SUDS_MAGNITUDE;

#define MAGNITUDES          307L
```

DESCRIPTION

MEMBERS

structure_type *structure type*

Define number of this type of structure.

structure_len *structure length*

Length of this structure in bytes.

magnitude_id *magnitude id*

A number that uniquely identifies, within this **magnitude_dc**, an instance of the **magnitude** structure.

[key=part_primary, db_index=clustered]

magnitude_dc *magnitude domain*

Domain in which magnitude_id is unique.

[codelist=authorities, key=part_primary]

solution_id *solution id*

A number that uniquely refers, within this solution_dc, to an instance of the **solution** structure.

[key=part_foreign(1,solution.solution_id), db_delete=cascade, db_must_exist=true]

solution_dc *solution domain*

Domain in which solution_id is unique.

[codelist=authorities, key=part_foreign(1,solution.solution_dc), db_delete=cascade, db_must_exist=true]

mag_value *value of magnitude*

[index_string=true]

mag_error *error in magnitude*

mag_type *type of magnitude*

Type of magnitude.

[codelist=magnitude_types]

preferred *preferred*

If this is the preferred magnitude for this solution, set to the capitol letter ’P’.

num_reports *number of reports*
Number of readings reported.

num_used *number used*
Number of readings used in the calculation of magnitude.

spare_a *for future use*

rms_of_mag *rms of magnitudes*
Root mean square of the individual magnitudes averaged.

authority *authority*
A number representing an institution or authority operating a network, calculating a solution, make an instrument calibration, etc. The authority is specified as a number that refers to an ASCII string in the authority codelist. Each institution has a base number such as 10000, 20000, etc. The institution may assign the 9999 numbers above their base number to individual people or groups. The individual number might be set to agree with the user number in /etc/passwd on UNIX systems.
[codelist=authorities]

comment_id *comment id*
A number that uniquely refers, within this comment_dc, to an instance of the **comment** structure. Comments are generally not searchable because they are not of standard format. Thus it is recommended that comments not be heavily used.
[key=part_foreign(2,comment.comment_id), db_delete=nullify]

comment_dc *comment domain*
Domain in which comment_id is unique.
[codelist=authorities, key=part_foreign(2,comment.comment_dc), db_delete=nullify]

SEE ALSO

NAME

map_element – lines and points to be plotted on a map

C_SYNOPSIS

```
typedef struct {
        FIXED       structure_type;
        FIXED       structure_len;
        LABEL       map_element_id;
        DOMAIN      map_element_dc;
        LATIT       latitude;
        LONGIT      longitude;
        FLOAT4      elevation;
        CODE4       element;
        CODE4       map_source;
        INT4        map_scale;
        FLOAT8      line_value;
        ST_TIME     time_mapped;
        ST_TIME     time_encoded;
        AUTHOR      authority;
        INT2        importance;
        CODE1       compression;
        CODE1       units;
        CODE4       data_type;
        INT4        data_length;
        REFERS2     comment_id;
        DOMAIN      comment_dc;
} SUDS_MAP_ELEMENT;

#define MAP_ELEMENTS    312L
```

DESCRIPTION

Information about lines or points to be plotted on maps. Virtually anything that could be plotted on a map could be included in the **code_list map_items** refered to by the member **element**. The data that follows this structure would typically be of type **CHAR** for a name or of type **map_data** for series of locations such as those specifying contour lines.

MEMBERS

structure_type *structure type*
Define number of this type of structure.

structure_len *structure length*
Length of this structure in bytes.

map_element_id *map element id*
A number that uniquely identifies, within this **map_element_dc**, an instance of the **map_element** structure.
[key=part_primary, db_index=clustered]

map_element_dc *map element domain*
Domain in which map_element_id is unique.
[codelist=authorities, key=part_primary]

latitude *latitude*
Latitude, south is negative.
[index_string=true]

longitude *longitude*
Longitude, west is negative.

elevation *elevation*

Elevation in kilometers, above sea level is positive.

element *map element code*
What type of line or point these data represent.
[codelist=map_items]

map_source *map source code*
Source of these data.
[codelist=authorities]

map_scale *map scale*
Scale of map digitized. If 1:20,000 then set map_scale=20000.

line_value *line value*
If this element is a contour line, then this is the value of the contour.

time_mapped *time mapped*
Time this map was made.

time_encoded *time map digitized*
Time this map was digitized.

authority *who digitized map*
Who digitized this map.
[codelist=authorities]

importance *importance*
Importance of this feature.

compression *compression algorithm*
Type of algorithm used to compress the data following this structure. NOCHAR means the data is not compressed.
[codelist=compression_types]

units *units*
Type of units represented by the contour line.
[codelist=units_types]

data_type *data type*
Type of data that follows this structure.
[sets_data_type=true, codelist=data_types]

data_length *number of points*
Number of data points of type **data_type** that follow this structure.
[sets_data_length=true, default_value=0]

comment_id *comment id*
A number that uniquely refers, within this comment_dc, to an instance of the **comment** structure. Comments are generally not searchable because they are not of standard format. Thus it is recommended that comments not be heavily used.
[key=part_foreign(1,comment.comment_id), db_delete=nullify]

comment_dc *comment domain*
Domain in which comment_id is unique.
[codelist=authorities, key=part_foreign(1,comment.comment_dc), db_delete=nullify]

SEE ALSO

NAME

mux_waveform – information about waveforms that are multiplexed

C_SYNOPSIS

```
typedef struct {
        FIXED       structure_type;
        FIXED       structure_len;
        LABEL       mux_waveform_id;
        DOMAIN      mux_waveform_dc;
        REFERS2     recorder_id;
        DOMAIN      recorder_dc;
        FIXED       len_contr_f;
        STRING      name_contr_f[12];
        REFERS2     clock_rate_id;
        DOMAIN      clock_rate_dc;
        MS_TIME     from_time;
        MS_TIME     thru_time;
        FIXED       len_media_l;
        STRING      media_label[16];
        FIXED       len_media_p;
        STRING      media_path[64];
        CODE1       media;
        CODE1       detector_type;
        CODE1       trigger_type;
        CODE1       event_type;
        CODE1       compression;
        CODE1       data_units;
        CHAR        spare_charA;
        CODE1       clock_type;
        INT4        dc_offset;
        FLOAT4      nom_dig_rate;
        INT4        numb_stations;
        INT4        block_size;
        CODE4       data_type;
        INT4        data_length;
        REFERS2     comment_id;
        DOMAIN      comment_dc;
} SUDS_MUX_WAVEFORM;

#define MUX_WAVEFORMS    106L
```

DESCRIPTION

Header for data detected and saved by a waveform recorder. Usually the data are multiplexed together, but they may be all or partially demultiplexed. This header should contain the information needed by a demultiplexing program to produce individual **waveform** structures for the waveforms for each station. The association of individual channels to **signal_path_ids** is done with the **recorder_ass** structures. This structure is followed by the multiplexed data.

In SUDS, most programs utilize data associated with **waveform** structures. Thus the primary use for the **mux_waveform** structure is temporary, when transferring data from a recorder to a demultiplexing program. Some network operators may also chose to archive the original data in multiplexed format. GENERALLY MUX_WAVEFORM STRUCTURES SHOULD BE SHORT LIVED.

Often when multiplexed data are written by an online detection program, the length of data to be written is not known when the structure_tag and structure must be written. If **structure_tag.length_data** equals **NODATL**, and the member **data_type** is set to some value other than **NODATL**, then the

SUDS input routine **st_get(2)** assumes that the data goes to the end of the file. The number of bytes from the end of the structure to the end of the file is determined to set **structure_tag.length_data** and divided by the length of the **mux_waveform.data_type** type to set **length_data**. When writing multiplexed data of unknown length, call **st_put** with **data_len** equal to the length of the array of **signal_path_ids**. Use **st_put_mux(2)** to write the rest of the data.
[permissions="siu_si_s_s"]

MEMBERS

structure_type *structure type*
Define number of this type of structure.

structure_len *structure length*
Length of this structure in bytes.

mux_waveform_id *multiplexed waveform id*
A number that uniquely refers, within this mux_data_dc, to an instance of the **mux_data** structure. This number is typically assigned by the detecting machine for this event trigger. This number must be unique for a detector, specified by a **recorder_id** that is unique within a recorder_dc. For consistency, this number should, if possible, represent the approximate time of writing the data represented as **ST_TIME** (i.e. seconds since beginning of Jan 1, 1970).
[key=part_primary, db_index=clustered]

mux_waveform_dc *mux data domain*
Domain in which mux_waveform_id is unique.
[codelist=authorities, key=part_primary]

recorder_id *recorder id*
A number that uniquely refers, within this **recorder_id**, to an instance of the **recorder** structure.
[key=part_foreign(1,recorder.recorder_id), db_delete=restrict, db_must_exist=true, index_string=true]

recorder_dc *recorder domain*
Domain in which recorder_id is unique.
[codelist=authorities, key=part_foreign(1,recorder.recorder_dc), db_delete=restrict, db_must_exist=true]

len_contr_f *len control file name*
Length of string for the name of the control file for the detection program, 12. True length may only be 11 to allow for the NULL byte.

name_contr_f *name of control file*
Name of the control file for the program detecting earthquakes on this recorder.

clock_rate_id *clock rate id*
A number that uniquely refers, within this **clock_rate_id**, to an instance of the **clock_rate** structure that has been applied to the data.
[key=part_foreign(2,clock_rate.clock_rate_id), db_delete=restrict]

clock_rate_dc *clock correction domain*
Domain in which clock_rate_id is unique.
[codelist=authorities, key=part_foreign(2,clock_rate.clock_rate_dc), db_delete=restrict]

from_time *nominal start time*
Time of the beginning of the traces.

thru_time *nominal end time*
Time of the ending of the traces.

len_media_l *length of media label*
The maximum space reserved for the media label, i.e. 16. Actual string can only contain 15 characters to allow for the NULL byte.

media_label *media label*

Label written on the storage medium.

len_media_p *length of media path*
The maximum space reserved for the media path name, i.e. 64. Actual string can only contain 63 characters to allow for the NULL byte.

media_path *media pathname*
Pathname for file containing this data on the media. Typically stored on an archival media such as an optical disk or a tape.
[ed_col=16]

media *type of storage media*
Type of media that the data is stored on.
[codelist=media]

detector_type *type of detector*
Type of detector.
[codelist=recorder_types]

trigger_type *type of trigger*
Type of trigger: l=longterm versus shortterm average, t=teleseismic
[codelist=trigger_types]

event_type *event type*
A character designating the type of event.
[codelist=event_types]

compression *compression algorithm*
Type of algorithm used to compress the data following this structure. NOCHAR means the data is not compressed.
[codelist=compression_types]

data_units *data units type*
Type of data units: d=digital counts, g=ground motion in nanometers, n=nanometers/sec, N=nanometers/sec/sec), v=millivolts, V=volts.
[codelist=units_types]

spare_charA *for future use*

clock_type *clock type*
Type of clock giving time.
[codelist=clock_types]

dc_offset *dc offset*
dc offset in volts.

nom_dig_rate *samples per second*
Nominal rate of digitization in samples per second.

numb_stations *number of stations*
Number of stations whose data are multiplexed together. Structure is followed by one **signal_path_id** for each station and then by **data_length** samples of a given **data_type**.

block_size *block size*
If data is partially demultiplexed, this is the total number of samples in each block. Number of samples per station in each block = **block_size** / **numb_stations**.

data_type *data type*
An integer representing the type of data that follows this structure. If the integer is negative, it refers to **variable_tab.define_num**. If the integer is positive, it refers to **structure_tab.struct_number**.
[sets_data_type=true, codelist=data_types]

data_length *data length*

Total number of samples of **data_type** following this structure. Number of samples per station = **data_length/numb_stations**.

[sets_data_length=true, default_value=0]

comment_id *comment id*

A number that uniquely refers, within this comment_dc, to an instance of the **comment** structure. Comments are generally not searchable because they are not of standard format. Thus it is recommended that comments not be heavily used.

[key=part_foreign(3,comment.comment_id), db_delete=nullify]

comment_dc *comment domain*

Domain in which comment_id is unique.

[codelist=authorities, key=part_foreign(3,comment.comment_dc), db_delete=nullify]

SEE ALSO

NAME

pick – information about a phase pick or any other picked feature of a waveform

C_SYNOPSIS

```
typedef struct {
        FIXED           structure_type;
        FIXED           structure_len;
        LABEL           pick_id;
        DOMAIN          pick_dc;
        REFERS2         event_id;
        DOMAIN          event_dc;
        REFERS2         waveform_id;
        DOMAIN          waveform_dc;
        FIXED           len_signal_n;
        STRING          signal_name[20];
        MS_TIME         pick_time;
        MS_TIME         nominal_time;
        FLOAT4          error_minus;
        FLOAT4          error_plus;
        FLOAT4          signal_2_noise;
        FLOAT4          spare;
        CODE2           observ_phase;
        CODE1           obs_time_qual;
        CODE1           onset_type;
        CODE1           orig_first_mot;
        CODE1           first_motion;
        CHAR            omit_from_sol;
        CODE1           pick_method;
        CODE1           record_media;
        CODE1           obs_ampl_qual;
        CODE1           amplitude_type;
        CODE1           ampl_units;
        FLOAT4          nom_amplitude;
        FLOAT4          amp_gain_range;
        FLOAT4          media_gain;
        FLOAT4          period;
        FLOAT4          obs_azimuth;
        FLOAT4          obs_slowness;
        FLOAT4          rectilinearity;
        ST_TIME         time_picked;
        AUTHOR          authority;
        REFERS2         comment_id;
        DOMAIN          comment_dc;
} SUDS_PICK;

#define PICKS           110L
```

DESCRIPTION

Basic information about any type of feature picked from a waveform that is associated with a **waveform** structure. Features are listed in the codelist **pick_types** and include different seismic phases, types of amplitude measurements, specification of a time or amplitude window, etc.
[permissions="siud_siud_s_s"]

MEMBERS

structure_type *structure type*

Define number of this type of structure.

structure_len *structure length*
Length of this structure in bytes.

pick_id *pick id*
A number that uniquely refers, within this **pick_dc**, to an instance of the **pick** structure.
[key=part_primary, db_index=clustered]

pick_dc *pick domain*
Domain in which pick_id is unique.
[codelist=authorities, key=part_primary]

event_id *event id*
A number that uniquely refers, within this **event_dc**, to an instance of the **event** structure.
[key=part_foreign(1,event.event_id), db_delete=cascade, db_must_exist=true]

event_dc *event domain*
Domain in which event_id is unique.
[codelist=authorities, key=part_foreign(1,event.event_dc), db_delete=cascade, db_must_exist=true]

waveform_id *waveform id*
A number that uniquely refers, within this **waveform_dc**, to an instance of the **waveform** structure.
[key=part_foreign(2,waveform.waveform_id), db_delete=nullify, index_string=true]

waveform_dc *waveform domain*
Domain in which waveform_id is unique.
[codelist=authorities, key=part_foreign(2,waveform.waveform_dc), db_delete=nullify]

len_signal_n *length signal name*
The maximum space reserved for the signal name, i.e. 20. Actual string can only contain 19 characters to allow for the NULL byte.

signal_name *signal name*
Name of a sensor component whose data are transmitted along a specific path and recorded on a particular recorder. Name is expected to be of the form network_station_CSBGP where the network is the abbreviation (part of the authority string preceeding the colon, 5 or less characters) for the **signal_path.network** code, station is **signal_path.station_name** (7 or less characters), C is the **signal_path.component_type** code (usually v, n, or e), S is the **signal_path.sensor_type** code, B is the **signal_path.band_type** code, G is the **signal_path.gain_type**, and P is the **signal_path.path_type** in only those stations where the same component may be recorded on two or more different recorders or transmitted over different paths.

pick_time *time of pick*
Time of pick.

nominal_time *nominal time of pick*
Nominal time of pick. This is the time based on **waveform.nominal_time**, which is the base time associated with a waveform before any clock corrections have been applied. A picking program should calculate both time and nominal_time. The reason for keeping both values is to allow correction of time-code corrections by always maintaining the uncorrected time and a separate table of time-code corrections.

error_minus *pick error minus*
Preferred time of pick minus earliest likely time of pick. Errors in timing have traditionally be represented by **obs_time_qual** with a scale from 0 to 4. This member and the next member allow a more quantitative measure of error in picking.

error_plus *pick error plus*
Latest likely time of pick minus preferred time of pick.

signal_2_noise *ratio signal to noise*

Ratio of signal to noise. Absolute amplitude of first half cycle after the pick to the average absolute amplitude of 100 samples of the waveform prior to any arrivals for this event.

spare *for future use*

observ_phase *observed phase code*
Code for observed phase from **pick_types** codelist.
[codelist=pick_types]

obs_time_qual *quality of timing*
Quality of timing. 0 equals best, 4 equals worst. This is an estimation of pick time accuracy made by the picker.
[codelist=timing_qualities]

onset_type *type of onset*
e=emersio, i=impulsive.
[codelist=onset_types]

orig_first_mot *original first motion*
Original first motion before any correction for polarity. The reason for keeping both values is to allow correction of polarity corrections by always maintaining the uncorrected first motion and a separate table of polarity corrections.
[codelist=first_motions]

first_motion *first motion*
U=up, D=down, +=probable up, -=probable down
[codelist=first_motions]

omit_from_sol *omit from solution*
To omit this phase from a solution, set to the small letter 'o'.
[allow_char="o"]

pick_method *type of picking method*
i=interactive,a=automatic,r=rtp, or user code
[codelist=pick_methods]

record_media *recording media*
Media on which waveform was analyzed if not in digital form.
[codelist=recording_medias]

obs_ampl_qual *quality of amplitude*
Quality of amplitude. 0 equals best, 4 equals worst. This is an estimation of amplitude pick accuracy made by the picker.
[codelist=timing_qualities]

amplitude_type *type of amplitude*
[codelist=amplitude_types]

ampl_units *units of amplitude*
Units used for amplitude.
[codelist=units_types]

nom_amplitude *nominal amplitude*
Amplitude picked as a signed variable equal to amplitude of later sample minus amplitude of earlier sample. The reason for keeping the sign is to make it possible to redraw amplitude lines on top of a waveform.

amp_gain_range *gain range*
Factor by which amplitude should be multiplied to correct for automatic gain-ranging in the field amplifier.

media_gain *media gain*
Factor by which amplitude should be multiplied to correct for gain of the media if not digital.

period *period*

Period in seconds of the amplitude picked. This is the time from peak to trough times 2.

obs_azimuth *observed azimuth*

Azimuth to event source as observed from waveforms.

obs_slowness *observed slowness*

Slowness of the waveform travelling by station.

rectilinearity *rectilinearity*

Rectilinearity of the waveform.

time_picked *time pick made*

Time this pick was made.

authority *pick authority*

Who made this pick.
[codelist=authorities]

comment_id *comment id*

A number that uniquely refers, within this comment_dc, to an instance of the **comment** structure. Comments are generally not searchable because they are not of standard format. Thus it is recommended that comments not be heavily used.
[key=part_foreign(3,comment.comment_id), db_delete=nullify]

comment_dc *comment domain*

Domain in which comment_id is unique.
[codelist=authorities, key=part_foreign(3,comment.comment_dc), db_delete=nullify]

SEE ALSO

BUGS

Is CUSP coda information taken care of?

NAME

pick_residual – residual for one pick in a solution and association of the pick with the solution

C_SYNOPSIS

```
typedef struct {
        FIXED       structure_type;
        FIXED       structure_len;
        REFERS2     pick_id;
        DOMAIN      pick_dc;
        REFERS2     solution_id;
        DOMAIN      solution_dc;
        REFERS2     vel_model_id;
        DOMAIN      vel_model_dc;
        CODE1       cal_time_qual;
        CODE1       cal_ampl_qual;
        CODE1       mag_type;
        CHAR        omit_from_sol;
        CODE1       weighted_out;
        CHAR        spare;
        INT2        spare_a;
        FLOAT4      pick_magnitude;
        FLOAT4      residual_val;
        FLOAT4      weight_used;
        FLOAT4      site_delay;
        FLOAT4      elevation_delay;
        FLOAT4      azm_2_stat;
        FLOAT4      dist_2_stat;
        FLOAT4      angle_emerg;
        REFERS2     comment_id;
        DOMAIN      comment_dc;
} SUDS_PICK_RESIDUAL;

#define PICK_RESIDUALS    111L
```

DESCRIPTION

The **pick_residual** structure serves a dual purpose: to associate picks with solutions (accomplished by the foreign key pair **pick_id** and **solution_id**), and to store information about the residual calculated by the location program for the associated pick.
[permissions="siu_siu_s_s"]

MEMBERS

structure_type *structure type*
Define number of this type of structure.

structure_len *structure length*
Length of this structure in bytes.

pick_id *pick id*
A number that uniquely refers, within this **pick_dc**, to an instance of the **pick** structure.
[key=part_primary, key=part_foreign(1,pick.pick_id), db_delete=cascade, db_must_exist=true, index_string=true]

pick_dc *pick domain*
Domain in which pick_id is unique.
[codelist=authorities, key=part_primary, key=part_foreign(1,pick.pick_dc), db_delete=cascade, db_must_exist=true]

solution_id *solution id*

A number that uniquely refers, within this solution_dc, to an instance of the **solution** structure.
[key=part_primary, key=part_foreign(2,solution.solution_id), db_delete=cascade, db_must_exist=true]

solution_dc *solution domain*
Domain in which solution_id is unique.
[codelist=authorities, key=part_primary, key=part_foreign(2,solution.solution_dc), db_delete=cascade, db_must_exist=true]

vel_model_id *velocity model id*
Unique identifier of velocity model used to calculate this residual.
[key=part_foreign(3,vel_model.vel_model_id), db_delete=cascade, db_must_exist=true]

vel_model_dc *velocity model domain*
Domain in which vel_model_id is unique.
[codelist=authorities, key=part_foreign(3,vel_model.vel_model_dc), db_delete=cascade, db_must_exist=true]

cal_time_qual *quality of timing* Quality of timing: 0 equals best, 4 equals worst. Calculated by the location program.
[codelist=timing_qualities]

cal_ampl_qual *quality of amplitude* Quality of amplitude: 0 equals best, 4 equals worst. Calculated by the location program.
[codelist=timing_qualities]

mag_type *magnitude code*
Type of magnitude calculated.
[codelist=magnitude_types]

omit_from_sol *omit from solution*
If this phase was omitted from the solution, this field is set to the small letter 'o'.
[allow_char="o"]

weighted_out *reason weighted out*
Reason this phase was weighted out of the solution.
[codelist=zero_weights]

spare *for future use*

spare_a *for future use*

pick_magnitude *pick magnitude*
Magnitude calculated from this pick at this station.

residual_val *residual value*
Residual in seconds defined as the observed arrival time minus the origin time of the solution minus the calculated traveltime minus the site_delay minus the elevation delay.

weight_used *weight used*
Weight used in the solution.

site_delay *site delay used*
Station delay used in the solution. Sum of all delays related to this site except for the elevation delay.

elevation_delay *elevation delay*
Elevation delay used in the solution. Typically elevation of site divided by some velocity.

azm_2_stat *azimuth to station*
Azimuth from earthquake to station. 0 is north.

dist_2_stat *distance to station*
Distance from epicenter to station in kilometers.

angle_emerg *angle of emergence*

Angle of emergence of wave from the hypocenter. 0 is vertical.

comment_id *comment id*

A number that uniquely refers, within this comment_dc, to an instance of the **comment** structure. Comments are generally not searchable because they are not of standard format. Thus it is recommended that comments not be heavily used.

[key=part_foreign(4,comment.comment_id), db_delete=nullify]

comment_dc *comment domain*

Domain in which comment_id is unique.

[codelist=authorities, key=part_foreign(4,comment.comment_dc), db_delete=nullify]

SEE ALSO

BUGS

Should a magnitude structure be available for each pick_residual?

NAME

polarity – evidence for reversed polarity for a signal_path

C_SYNOPSIS

```
typedef struct {
        FIXED       structure_type;
        FIXED       structure_len;
        LABEL       polarity_id;
        DOMAIN      polarity_dc;
        REFERS2     signal_path_id;
        DOMAIN      signal_path_dc;
        CODE1       evidence;
        CODE1       clarity;
        INT2        spare;
        AUTHOR      authority;
        ST_TIME     from_time;
        ST_TIME     thru_time;
        REFERS2     comment_id;
        DOMAIN      comment_dc;
} SUDS_POLARITY;

#define POLARITYS           309L
```

DESCRIPTION

The **polarity** structure describes evidence that a **signal_path** had a polarity reversal over a specific period of time. It is assumed that the polarity is correct unless a **polarity** structure exists.

[permissions="s_s_siu_s"]

MEMBERS

structure_type *structure type*

Define number of this type of structure.

structure_len *structure length*

Length of this structure in bytes.

polarity_id *polarity*

A number that uniquely refers, within this **polarity_dc**, to an instance of the **polarity** structure.

[key=part_primary, db_index=clustered]

polarity_dc *polarity domain*

Domain in which polarity_id is unique.

[codelist=authorities, key=part_primary]

signal_path_id *signal path number*

A number that uniquely refers, within this **domain_code**, to one instance of the **signal_path** structure representing a total signal path from a particular sensor to its recorder. In some cases the same sensor and recorder may be connected by separate paths.

[key=part_foreign(1,signal_path.signal_path_id), db_delete=restrict, db_must_exist=true, db_index=true]

signal_path_dc *signal path domain*

Domain in which signal_path_id is unique.

[codelist=authorities, key=part_foreign(1,signal_path.signal_path_dc), db_delete=restrict, db_must_exist=true]

evidence *evidence*

Evidence for polarity reversal.

[codelist=rev_evidence]

clarity *clarity*

Clarity of the evidence for this polarity reversal.

[codelist=clarities]

spare *for future use*

authority *authority*

Who set up this association.

[codelist=authorities, index_string=true]

from_time *valid from time*

Time this polarity reversal became valid.

thru_time *valid thru time*

Time this polarity reversal became no longer valid.

comment_id *comment id*

A number that uniquely refers, within this comment_dc, to an instance of the **comment** structure. Comments are generally not searchable because they are not of standard format. Thus it is recommended that comments not be heavily used.

[key=part_foreign(2,comment.comment_id), db_delete=nullify]

comment_dc *comment domain*

Domain in which comment_id is unique.

[codelist=authorities, key=part_foreign(2,comment.comment_dc), db_delete=nullify]

SEE ALSO

NAME

processing – a processing command or error message

C_SYNOPSIS

```
typedef struct {
        FIXED       structure_type;
        FIXED       structure_len;
        LABEL       processing_id;
        DOMAIN      processing_dc;
        CODE1       process_type;
        CHAR        spare;
        INT2        spare_a;
        AUTHOR      authority;
        CODE4       data_type;
        INT4        data_length;
        REFERS2     comment_id;
        DOMAIN      comment_dc;
} SUDS_PROCESSING;

#define PROCESSINGS     308L
```

DESCRIPTION

This structure is followed by an ASCII string containing commands that would duplicate the processing done on this waveform. The command language has not been fully developed. These could be Unix shell type commands, but we also would like to have a way that the actions taken within the SUDS graphical user interface ciould be specified.

In other words this structure is a proposal, not a full implementation as of this date.

MEMBERS

structure_type *structure type*

Define number of this type of structure.

structure_len *structure length*

Length of this structure in bytes.

processing_id *processing id*

A number that uniquely identifies, within this **processing_dc**, an instance of the **processing** structure.

[key=part_primary, db_index=clustered]

processing_dc *processing domain*

Domain in which processing_id is unique.

[codelist=authorities, key=part_primary]

process_type *type of processing*

[codelist=process_types]

spare *for future use*

spare_a *for future use*

authority *authority*

A number representing an institution or authority operating a network, calculating a solution, make an instrument calibration, etc. The authority is specified as a number that refers to an ASCII string in the authority codelist. Each institution has a base number such as 10000, 20000, etc. The institution may assign the 9999 numbers above their base number to individual people or groups. The individual number might be set to agree with the user number in /etc/passwd on UNIX systems.

[codelist=authorities, index_string=true]

data_type *data storage type*

An integer representing the type of data that follows this structure. If the integer is negative, it refers to **variable_tab.define_num**. If the integer is positive, it refers to **structure_tab.struct_number**. Typically type **CHAR**.

[sets_data_type=true, codelist=data_types]

data_length *number of samples*

Number of characters that follow this structure.

[sets_data_length=true, default_value=0]

comment_id *comment id*

A number that uniquely refers, within this comment_dc, to an instance of the **comment** structure. Comments are generally not searchable because they are not of standard format. Thus it is recommended that comments not be heavily used.

[key=part_foreign(1,comment.comment_id), db_delete=nullify]

comment_dc *comment domain*

Domain in which comment_id is unique.

[codelist=authorities, key=part_foreign(1,comment.comment_dc), db_delete=nullify]

SEE ALSO

NAME

recorder – information about a recorder of signals

C_SYNOPSIS

```
typedef struct {
        FIXED       structure_type;
        FIXED       structure_len;
        LABEL       recorder_id;
        DOMAIN      recorder_dc;
        FIXED       len_name;
        STRING      recorder_name[12];
        FIXED       len_serial_n;
        STRING      serial_number[12];
        CODE4       model;
        FLOAT4      speed;
        CODE1       speed_units;
        CODE1       data_units;
        CHAR        spare_a;
        CODE1       recorder_type;
        FLOAT4      conv_2_mvolts;
        FLOAT4      gain;
        FLOAT4      clip_value;
        FIXED       len_detect_p;
        STRING      name_detect_p[12];
        INT2        ver_detect_p;
        INT2        spare;
        CODE4       storage_type;
        REFERS2     comment_id;
        DOMAIN      comment_dc;
} SUDS_RECORDER;

#define RECORDERS           131L
```

DESCRIPTION

Information about a signal recorder (data acquisition hardware and software.)

[permissions="s_s_siu_s"]

MEMBERS

structure_type *structure type*

Define number of this type of structure.

structure_len *structure length*

Length of this structure in bytes.

recorder_id *recorder id*

A number that uniquely refers, within this **recorder_dc** domain code, to an instance of the **recorder** structure.

[key=part_primary, db_index=clustered]

recorder_dc *recorder domain code*

Domain in which recorder_id is unique.

[codelist=authorities, key=part_primary]

len_name *length of name*

The maximum space reserved for the name of this recorder, i.e. 12. Actual string can only contain 11 characters.

recorder_name *recorder name*

Name of this recorder.

[index_string=true]

len_serial_n *length serial number*

The maximum space reserved for the serial number, i.e. 12. Actual string can only contain 11 characters.

serial_number *serial number*

Serial number of the piece of equipment. Should be unique in the world for this model.

model *model code*

Number that is unique in the world designating the model of this piece of equipment. This number is associated with an ASCII string in codelist **equip_models**.

[codelist=equip_models]

speed *speed of recording*

Nominal rate of digitization in samples per second for digital recorders. Inches per second for analog recorders.

speed_units *speed units type*

Type of data units: d=digital counts, g=ground motion in nanometers, n=nanometers/sec, N=nanometers/sec/sec), v=millivolts, V=volts.

[codelist=units_types]

data_units *data units type*

Type of data units: d=digital counts, g=ground motion in nanometers, n=nanometers/sec, N=nanometers/sec/sec), v=millivolts, V=volts.

[codelist=units_types]

spare_a *for future use*

recorder_type *recorder type*

Recorder type: a=analog j=jade, w=willie-pc, s=sun-cdd, p=pdas

[codelist=recorder_types]

conv_2_mvolts *conv to mvolts*

For digital recorders, the conversion factor to millivolts: mv per digital count. This number should include the a_2_d_gain. This is the single number that when multiplied times the digital counts, gives the millivolts of output of the discriminator, which should be approximately equal to the output of the seismic amplifier before input to the VCO.
max_ground_motion=digital_sample*conv_2_mvolts*max_gain

gain *gain of recorder input*

Gain of analog to digital converter.

clip_value *clip value*

+-value of data where clipping begins in whatever units the data are in. This is the value before any DC shift is made in the data. While the cause of clipping and the precise value where clipping begins may vary for each station, this number should be a conservative value that applies to most stations. It is used in such subroutines as **descr_trace**(2) to calculate the number of clipped data points.

len_detect_p *len detect prog name*

Length of string for name of the detection program, 12. True length may only be 11 to allow for the NULL byte.

name_detect_p *name of detect program*

Name of the program detecting earthquakes.

ver_detect_p *version detect program*

Version of the detection program. 10 means version 1.0

spare *for future use*

storage_type *data type*

Type of data generated by this recorder. Typically 16 bit integer such as INT2 (variable_info(3)).

[codelist=data_types]

comment_id *comment id*

A number that uniquely refers, within this comment_dc, to an instance of the **comment** structure. Comments are generally not searchable because they are not of standard format. Thus it is recommended that comments not be heavily used.

[key=part_foreign(1,comment.comment_id), db_delete=nullify]

comment_dc *comment domain*

Domain in which comment_id is unique.

[codelist=authorities, key=part_foreign(1,comment.comment_dc), db_delete=nullify]

SEE ALSO

BUGS

What about AtoDs that digitize 0 to +volts rather than -volts to +volts?

NAME

recorder_ass – associates recorders with signal paths

C_SYNOPSIS

```
typedef struct {
        FIXED       structure_type;
        FIXED       structure_len;
        REFERS2     signal_path_id;
        DOMAIN      signal_path_dc;
        REFERS2     recorder_id;
        DOMAIN      recorder_dc;
        REFERS2     site_id;
        DOMAIN      site_dc;
        INT4        spare;
        AUTHOR      authority;
        INT2        pos_in_path;
        INT2        channel_number;
        INT2        rack_slot;
        INT2        mux_order;
        FLOAT4      frequency;
        FLOAT4      attenuation;
        ST_TIME     from_time;
        ST_TIME     thru_time;
        REFERS2     comment_id;
        DOMAIN      comment_dc;
} SUDS_RECORDER_ASS;

#define RECORDER_ASSS       324L
```

DESCRIPTION

The **recorder_ass** structure associates **recorders** with a **signal_path** for a specific period of time. A given **signal_path** may be recorded by one or more computers, analog-tape systems, etc.
[permissions="s_s_siu_s"]

MEMBERS

structure_type *structure type*
Define number of this type of structure.

structure_len *structure length*
Length of this structure in bytes.

signal_path_id *signal path id*
A number that uniquely refers, within this **domain_code**, to one instance of the **signal_path** structure representing a total signal path from a particular sensor to its recorder. In some cases the same sensor and recorder may be connected by separate paths.
[key=part_primary, key=part_foreign(1,signal_path.signal_path_id), db_delete=restrict, db_must_exist=true, index_string=true]

signal_path_dc *signal path domain*
Domain in which signal_path_id is unique.
[codelist=authorities, key=part_primary, key=part_foreign(1,signal_path.signal_path_dc), db_delete=restrict, db_must_exist=true]

recorder_id *recorder id*
A number that uniquely refers within this **recorder_dc**, to an instance of the **recorder** structure.
[key=part_primary, key=part_foreign(2,recorder.recorder_id), db_delete=restrict, db_must_exist=true]

recorder_dc *component domain*

Domain in which recorder_id is unique.
[codelist=authorities, key=part_primary, key=part_foreign(2,recorder.recorder_dc), db_delete=restrict, db_must_exist=true]

site_id *site id*
A number that uniquely refers to an instance of the **site** structure.
[key=part_foreign(3, site.site_id), db_delete=restrict, db_must_exist=true]

site_dc *site site domain*
Domain in which site_id is unique.
[codelist=authorities, key=part_foreign(3,site.site_dc), db_delete=restrict, db_must_exist=true]

spare *for future use*

authority *authority*
Who set up this association.
[codelist=authorities]

pos_in_path *position in path*
Used to determine the proper ordering of component structures such as **seismometer, sig_path_cmp, recorder** associated with the same **signal_path_id**. The sensor should be number 0, the on-site calibrator 10, the amp/vco 20, the computer atod 1000, the analog tape recorder 900, and the discriminator 950. Other components such as transmitters, receivers, antennas, summing amplifiers should be numbered in between or after these numbers as appropriate. Where multiple signal_paths go through the same component, the **pos_in_path** may not be identical for the same piece of hardware in different signal_paths.

channel_number *channel number*
Number of the physical channel ("pin number") on which this signal is recorded. This is not necessarily the same as the position in the sampling order, since some systems may sample input channels in non-sequential order. On analog systems, this is the tape-head number.

rack_slot *slot in rack*
Slot in rack connected to this channel_number. Rack typically contains discriminators.

mux_order *multiplexer order*
Order that multiplexer samples this station counting from zero. When the data exists in a **mux_waveform** structure, this number is used by the demultiplexing program to associate a channel of data to a **signal_path_id**.

frequency *frequency*
Frequency associated with this particular **component** and **signal_path**.

attenuation *attenuation*
Attenuation associated with this particular **component** and **signal_path**. Negative number means gain.

from_time *valid from time*
Time this association became valid.

thru_time *valid thru time*
Time this association became no longer valid.

comment_id *comment id*
A number that uniquely refers, within this comment_dc, to an instance of the **comment** structure. Comments are generally not searchable because they are not of standard format. Thus it is recommended that comments not be heavily used.
[key=part_foreign(4,comment.comment_id), db_delete=nullify]

comment_dc *comment domain*
Domain in which comment_id is unique.
[codelist=authorities, key=part_foreign(4,comment.comment_dc), db_delete=nullify]

SEE ALSO

NAME

response – information about the frequency response of a sensor, component, or total system

C_SYNOPSIS

```
typedef struct {
        FIXED         structure_type;
        FIXED         structure_len;
        LABEL         response_id;
        DOMAIN        response_dc;
        FIXED         len_name_resp;
        STRING        name_response[20];
        CODE1         response_type;
        CODE1         input_units;
        CODE1         output_units;
        CHAR          spare_char;
        FLOAT4        maximum_gain;
        FLOAT4        normalization;
        FLOAT4        frequency_max;
        FLOAT4        inp_samp_rate;
        INT2          decim_factor;
        INT2          decim_offset;
        FLOAT4        estim_delay;
        FLOAT4        used_delay;
        ST_TIME       from_time;
        ST_TIME       thru_time;
        AUTHOR        authority;
        INT4          spare;
        CODE4         data_type;
        INT4          data_length;
        REFERS2       comment_id;
        DOMAIN        comment_dc;
} SUDS_RESPONSE;

#define RESPONSES             109L
```

DESCRIPTION

Frequency response information for a particular sensor, channel, recorder, or total system. The **response** structure is followed by a number of structures (**data_length**) specifying a response curve of corner frequency and slope (**resp_cfs_data**), frequency, amplitude, and phase (**resp_fap_data**), a finite impulse response (**resp_fir_data**), complex poles and zeros (**resp_pz_data**), a calibration (**resp_sen_data**), or sensitivity/gain (**resp_sen_data**). This structure can also be followed by a series of **response_id**s for this **response_dc** specifying a sequence of filters in order from that applied first to that applied last.

[permissions="siud_s_siud_s"]

MEMBERS

structure_type *structure type*

Define number of this type of structure.

structure_len *structure length*

Length of this structure in bytes.

response_id *response id*

A number that uniquely identifies, within this **response_dc**, an instance of the **response** structure.

[key=part_primary, db_index=clustered]

response_dc *response domain*

Domain in which response_id is unique.

[codelist=authorities, key=part_primary]

len_name_resp *len response name*
Length of string for response name, 20. True length may only be 19 to allow for the NULL byte.

name_response *response name*
Name of this response. Typically used if this response applies to a generic instrument type.
[index_string=true]

response_type *type of response*
Type of response structure that follows the **response** structure.
[codelist=response_types]

input_units *type of input units*
Type of data units on input.
[codelist=units_types]

output_units *type of output units*
Type of data units on output.
[codelist=units_types]

spare_char *for future use*

maximum_gain *maximum gain*
Value of the maximum value on the calibration curve. This is the factor by which the curve values are multiplied to get the total gain.

normalization *normalization factor*
Value by which to multiply the calibration curve to cause the peak gain to be 1. In other words this is 1 divided by the peak value of the calibration curve not including the factor **maximum_gain**.

frequency_max *frequency at max*
Frequency at the point of maximum gain on the response curve.

inp_samp_rate *input sample rate*
For finite impulse response (FIR) filters, this is the sample rate of the input signal.

decim_factor *decimation factor*
For finite impulse response (FIR) filters, this is the amount by which the input signal is decimated.

decim_offset *decimation offset*
For finite impulse response (FIR) filters, this is which sample is chosen when decimation is used. Count from zero to any number less that the **decim_factor**.

estim_delay *estimated delay*
For finite impulse response (FIR) filters, this is the estimated time delay of the system in seconds.

used_delay *delay used*
For finite impulse response (FIR) filters, this is the time delay of the system in seconds that was used.

from_time *valid from time*
Time this calibration became valid.

thru_time *valid thru time*
Time this calibration became no longer valid.

authority *authority*
Who specified this response.
[codelist=authorities]

spare *for future use*

data_type *data type*

Type of structure containing response data that follows this structure. This is the number specified in the define statement for each filter type: RESP_CFS_DATAS, RESP_FAP_DATAS, RESP_FIR_DATAS, RESP_PZ_DATAS, and RESP_SEN_DATAS.

[sets_data_type=true, codelist=data_types]

data_length *number of points*

Number of points in the data curve of type **data_type** that follow this structure.

[sets_data_length=true, default_value=0]

comment_id *comment id*

A number that uniquely refers, within this comment_dc, to an instance of the **comment** structure. Comments are generally not searchable because they are not of standard format. Thus it is recommended that comments not be heavily used.

[key=part_foreign(1,comment.comment_id), db_delete=nullify]

comment_dc *comment domain*

Domain in which comment_id is unique.

[codelist=authorities, key=part_foreign(1,comment.comment_dc), db_delete=nullify]

SEE ALSO

resp_cfs_data(4), resp_fap_data(4), resp_fir_data(4), resp_pz_data(4), resp_sen_data(4)

NAME

resp_cfs_data – response values for corner frequency and slope

C_SYNOPSIS

```
typedef struct {
        FLOAT4          corner_freq;
        FLOAT4          db_per_decade;
} SUDS_RESP_CFS_DATA;

#define RESP_CFS_DATAS  305L
```

DESCRIPTION

Response information when specified as a sequence of corner frequencies and slopes (db per decade). A number of these structures (**response.data_length**) follow structure **response** as data.
[data_only=RESPONSES]

MEMBERS

corner_freq *corner frequency*
Corner frequency point on the amplitude versus frequency curve in hertz.

db_per_decade *slope (db/decade)*
Slope of the line after (higher frequency) the corner in db/decade.

SEE ALSO

response(4), resp_fap_data(4), resp_fir_data(4), resp_pz_data(4), resp_sen_data(4)

NAME

resp_fap_data – response values for frequency, amplitude, and phase

C_SYNOPSIS

```
typedef struct {
        FLOAT4      frequency;
        FLOAT4      amplitude;
        FLOAT4      amplitude_err;
        FLOAT4      phase;
        FLOAT4      phase_error;
        INT4        spare;
} SUDS_RESP_FAP_DATA;

#define RESP_FAP_DATAS     303L
```

DESCRIPTION

Response information when specified as a sequence of triplets specifying frequency, amplitude, and phase. A number of these structures (**response.data_length**) follow structure **response** as data.
[data_only=RESPONSES]

MEMBERS

frequency *frequency*
Frequency point on the amplitude versus frequency curve in hertz.

amplitude *amplitude*
Amplitude point on the amplitude versus frequency curve.

amplitude_err *amplitude error*
Error in the amplitude point on the amplitude versus frequency curve.

phase *phase*
Phase angle in degrees for a point on the amplitude versus frequency curve.

phase_error *phase error*
Error in the phase angle in degrees for a point on the amplitude versus frequency curve.

spare *for future use*

SEE ALSO

response(4), resp_cfs_data(4), resp_fir_data(4), resp_pz_data(4), resp_sen_data(4)

NAME

resp_fir_data – response values for finite impulse response filters

C_SYNOPSIS

```
typedef struct {
        INT4        position;
        INT4        spare;
        FLOAT4      numer_coef;
        FLOAT4      numer_coef_err;
        FLOAT4      denom_coef;
        FLOAT4      denom_coef_err;
} SUDS_RESP_FIR_DATA;

#define RESP_FIR_DATAS    314L
```

DESCRIPTION

Response information when specified as a sequence of finite impulse response coefficients. A number of these structures (**response.data_length**) follow structure **response** as data.
[data_only=RESPONSES]

MEMBERS

position *position*
Position of this coefficient in the sequence, counting from 0.

spare *for future use*

numer_coef *numerator coefficient*
Numerator of the finite impulse response coefficient.

numer_coef_err *numerator coef error*
Error in the numerator of the finite impulse response coefficient.

denom_coef *denominator coefficient*
Denominator of the finite impulse response coefficient.

denom_coef_err *denominator coef error*
Error in the denominator of the finite impulse response coefficient.

SEE ALSO

response(4), resp_cfs_data(4), resp_fap_data(4), resp_pz_data(4), resp_sen_data(4)

NAME

resp_pz_data – response values for infinite impulse response filters

C_SYNOPSIS

```
typedef struct {
        FLOAT4          pole_r;
        FLOAT4          pole_i;
        FLOAT4          pole_err_r;
        FLOAT4          pole_err_i;
        FLOAT4          zero_r;
        FLOAT4          zero_i;
        FLOAT4          zero_err_r;
        FLOAT4          zero_err_i;
} SUDS_RESP_PZ_DATA;

#define RESP_PZ_DATAS   123L
```

DESCRIPTION

Response information when specified as a sequence of poles and zeroes. A number of these structures (**response.data_length**) follow structure **response** as data. If there are more poles than zeros or visa versa, set unknowns to NODATF.
[data_only=RESPONSES]

MEMBERS

pole_r *pole real*
Real part response pole value.

pole_i *pole imaginary*
Imaginary part response pole value.

pole_err_r *pole error real*
Real part response pole errors.

pole_err_i *pole error imaginary*
Imaginary part response pole errors.

zero_r *zero real*
Real part response zero value.

zero_i *zero imaginary*
Imaginary part response zero value.

zero_err_r *zero error real*
Real part response zero errors.

zero_err_i *zero error imaginary*
Imaginary part response zero errors.

SEE ALSO

response(4), resp_cfs_data(4), resp_fap_data(4), resp_fir_data(4), resp_sen_data(4)

NAME

resp_sen_data – response sensitivity/gain

C_SYNOPSIS

```
typedef struct {
        FLOAT4      sensitivity;
        FLOAT4      frequency;
        ST_TIME     cal_time;
        INT4        spare;
} SUDS_RESP_SEN_DATA;

#define RESP_SEN_DATAS      319L
```

DESCRIPTION

Response information when specified as sensitivity or gain at a series of frequencies resulting from a calibration. A number of these structures (**response.data_length**) follow structure **response** as data.
[data_only=RESPONSES]

MEMBERS

sensitivity *sensitivity/gain*
Sensitivity/gain at a given frequency.

frequency *frequency*
Frequency point on the amplitude versus frequency curve in hertz.

cal_time *time of calibration*
Time this point was calibrated.

spare *for future use*

SEE ALSO

response(4), resp_cfs_data(4), resp_fap_data(4), resp_fir_data(4), resp_pz_data(4)

NAME

seismometer – information about a seismometer

C_SYNOPSIS

```
typedef struct {
        FIXED       structure_type;
        FIXED       structure_len;
        LABEL       seismometer_id;
        DOMAIN      seismometer_dc;
        REFERS2     response_id;
        DOMAIN      response_dc;
        CODE4       model;
        FIXED       len_serial_n;
        STRING      serial_number[12];
        FLOAT4      free_frequency;
        FLOAT4      motor_const;
        FLOAT4      eff_mo_const;
        FLOAT4      mass;
        CODE1       seis_type;
        CHAR        pad_type;
        CODE1       component_type;
        CHAR        spare;
        INT2        r_coil;
        INT2        r_crit_damp;
        FLOAT4      eff_damping;
        INT2        r_lpad;
        INT2        r_tpad;
        INT2        r_shunt;
        INT2        r_cal_coil;
        FLOAT4      cal_mo_const;
        AUTHOR      authority;
        REFERS2     comment_id;
        DOMAIN      comment_dc;
} SUDS_SEISMOMETER;

#define SEISMOMETERS        313L
```

DESCRIPTION

Information about a seismometer. This structure is used to record both bookkeeping information (model, serial_number) and some of the information necessary to calculate a transfer function for the seismometer itself, according to how the instrument is set up.

[permissions="s_s_siu_s"]

MEMBERS

structure_type *structure type*

Define number of this type of structure.

structure_len *structure length*

Length of this structure in bytes.

seismometer_id *seismometer id*

A number that uniquely refers, within this **seismometer_dc**, to an instance of the **seismometer** structure.

[key=part_primary, db_index=clustered]

seismometer_dc *seismometer domain* Domain in which seismometer_id is unique.

[codelist=authorities, key=part_primary]

response_id *response id*
A number that uniquely identifies, within this **response_dc**, to an instance of the **response** structure.
[key=part_foreign(1,response.response_id), db_delete=nullify]

response_dc *response domain*
Domain in which response_id is unique.
[codelist=authorities, key=part_foreign(1,response.response_dc), db_delete=nullify]

model *model code*
Number that is unique in the world designating the model of this piece of equipment.
[codelist=equip_models]

len_serial_n *length serial number*
The maximum space reserved for the serial number, i.e. 12. Actual string can only contain 11 characters.

serial_number *serial number*
Serial number of the piece of equipment. Should be unique in the world for this model.
[index_string=true]

free_frequency *free frequency*
The free-frequency in hertz or the inverse of the free period in seconds of this seismometer.

motor_const *motor constant*
Motor constant of the seismometer coil and magnet. UNITS in MKS??

eff_mo_const *effective motor const*
Effective motor constant of the seismometer coil and magnet. UNITS in MKS??

mass *seismometer mass*
Mass of the moving element in kilograms.

seis_type *seismometer type code*
Type of seismometer.
[codelist=sensor_types]

pad_type *pad type*
L or T resistor pad between seismometer and amplifier. A designates pad for 24 db attenuation.

component_type *component type code*
Type of component. Vertical, horizontal, or other.
[codelist=components]

spare *for future use*

r_coil *coil resistance*
Resistance of the seismometer coil in Ohms.

r_crit_damp *crit damp resistance*
Critical damping resistance in Ohms.

eff_damping *effective damping*
Effective damping of the seismometer.

r_lpad *L pad resistor*
Resistance in Ohms of the damping resistor in series with the seismometer coil and the shunt resistor.

r_tpad *T pad resistor*
Resistance in Ohms of the damping resistor in series with the amplifier and the shunt resistor.

r_shunt *shunt resistor*
Resistance in Ohms of the shunt damping resistor.

r_cal_coil *cal coil resistance*
Resistance of the calibration coil in Ohms.

cal_mo_const *cal motor constant*
Calibration coil motor constant.

authority *authority*
Who specified this information.
[codelist=authorities]

comment_id *comment id*
A number that uniquely refers, within this comment_dc, to an instance of the **comment** structure. Comments are generally not searchable because they are not of standard format. Thus it is recommended that comments not be heavily used.
[key=part_foreign(2,comment.comment_id), db_delete=nullify]

comment_dc *comment domain*
Domain in which comment_id is unique.
[codelist=authorities, key=part_foreign(2,comment.comment_dc), db_delete=nullify]

SEE ALSO

NAME

seismo_ass – associates seismometers with signal paths

C_SYNOPSIS

```
typedef struct {
        FIXED       structure_type;
        FIXED       structure_len;
        REFERS2     signal_path_id;
        DOMAIN      signal_path_dc;
        REFERS2     seismometer_id;
        DOMAIN      seismometer_dc;
        REFERS2     site_id;
        DOMAIN      site_dc;
        INT4        spare;
        AUTHOR      authority;
        FLOAT4      frequency;
        FLOAT4      attenuation;
        ST_TIME     from_time;
        ST_TIME     thru_time;
        REFERS2     comment_id;
        DOMAIN      comment_dc;
} SUDS_SEISMO_ASS;

#define SEISMO_ASSS        325L
```

DESCRIPTION

The structure associates a **seismometer** with a **signal_path** for a specific period of time. A given **signal_path** may be recorded by one or more computers, analog-tape systems, etc.

[permissions="s_s_siu_s"]

MEMBERS

structure_type *structure type*

Define number of this type of structure.

structure_len *structure length*

Length of this structure in bytes.

signal_path_id *signal path id*

A number that uniquely refers, within this **domain_code**, to one instance of the **signal_path** structure representing a total signal path from a particular sensor to its recorder. In some cases the same sensor and recorder may be connected by separate paths.

[key=part_primary, key=part_foreign(1,signal_path.signal_path_id), db_delete=restrict, db_must_exist=true, index_string=true]

signal_path_dc *signal path domain*

Domain in which signal_path_id is unique.

[codelist=authorities, key=part_primary, key=part_foreign(1,signal_path.signal_path_dc), db_delete=restrict, db_must_exist=true]

seismometer_id *seismometer id*

A number that uniquely refers, within this **component_type** and this **seismometer_dc**, to an instance of the **seismometer** structure.

[key=part_primary, key=part_foreign(2,seismometer.seismometer_id), db_delete=restrict, db_must_exist=true]

seismometer_dc *component domain*

Domain in which seismometer_id is unique.

[codelist=authorities, key=part_primary, key=part_foreign(2,seismometer.seismometer_dc), db_delete=restrict, db_must_exist=true]

site_id *site id*

A number that uniquely refers to an instance of the **site** structure.

[key=part_foreign(3, site.site_id), db_delete=restrict, db_must_exist=true]

site_dc *site site domain*

Domain in which site_id is unique.

[codelist=authorities, key=part_foreign(3,site.site_dc), db_delete=restrict, db_must_exist=true]

spare *for future use*

authority *authority*

Who set up this association.

[codelist=authorities]

frequency *frequency*

Frequency associated with this particular **component** and **signal_path**.

attenuation *attenuation*

Attenuation associated with this particular **component** and **signal_path**. Negative number means gain.

from_time *valid from time*

Time this association became valid.

thru_time *valid thru time*

Time this association became no longer valid.

comment_id *comment id*

A number that uniquely refers, within this comment_dc, to an instance of the **comment** structure. Comments are generally not searchable because they are not of standard format. Thus it is recommended that comments not be heavily used.

[key=part_foreign(4,comment.comment_id), db_delete=nullify]

comment_dc *comment domain*

Domain in which comment_id is unique.

[codelist=authorities, key=part_foreign(4,comment.comment_dc), db_delete=nullify]

SEE ALSO

NAME

service – record of service to a signal_path

C_SYNOPSIS

```
typedef struct {
        FIXED       structure_type;
        FIXED       structure_len;
        LABEL       service_id;
        DOMAIN      service_dc;
        REFERS2     signal_path_id;
        DOMAIN      signal_path_dc;
        ST_TIME     from_time;
        ST_TIME     thru_time;
        CODE4       authority;
        INT4        spare;
        FIXED       len_reasons;
        CODESTR     reasons[20];
        FIXED       len_actions;
        CODESTR     actions[20];
        REFERS2     comment_id;
        DOMAIN      comment_dc;
} SUDS_SERVICE;

#define SERVICES    323L
```

DESCRIPTION

Description of a service or maintenance visit related to a signal_path.

MEMBERS

structure_type *structure type*
Define number of this type of structure.

structure_len *structure length*
Length of this structure in bytes.

service_id *service identifier*
A number that uniquely refers, within this **service_dc**, to an instance of the **service** structure.
[key=part_primary, db_index=clustered]

service_dc *service domain*
Domain in which service_id is unique.
[codelist=authorities, key=part_primary]

signal_path_id *signal path id*
A number that uniquely refers, within this **signal_path_dc**, to a **signal_path** where the equipment that was serviced is physically located. If, for example, the repair is to a radio relay, the signal_path name would be different from the signal_path given as part of **signal_name**.
[key=part_foreign(1, signal_path.signal_path_id), db_delete=restrict, db_must_exist=true, index_string=true]

signal_path_dc *signal path domain*
Domain in which signal_path_id is unique.
[codelist=authorities, key=part_foreign(1,signal_path.signal_path_dc), db_delete=restrict, db_must_exist=true]

from_time *visit began*
Time repair was started.
[db_index=true]

thru_time *visit ended*

Time repair was completed.

authority *authority*

A number representing an institution or authority operating a network, calculating a solution, make an instrument calibration, etc. The authority is specified as a number that refers to an ASCII string in the authority codelist. Each institution has a base number such as 10000, 20000, etc. The institution may assign the 9999 numbers above their base number to individual people or groups. The individual number might be set to agree with the user number in /etc/passwd on UNIX systems.

[codelist=authorities]

spare *for future use*

len_reasons *length reasons*

The maximum space reserved for the reasons, i.e. 20. Actual string can only contain 19 characters to allow for the NULL byte.

reasons *reasons for visit*

A string of characters that refer to the codelist **equip_reasons** listing all of the reasons this service was made.

[codelist=equip_reasons]

len_actions *length actions*

The maximum space reserved for the actions, i.e. 20. Actual string can only contain 19 characters to allow for the NULL byte.

actions *actions taken*

A string of characters that refer to the codelist **equip_actions** listing all of the actions taken during this service visit.

[codelist=equip_actions]

comment_id *comment id*

A number that uniquely refers, within this comment_dc, to an instance of the **comment** structure. Comments are generally not searchable because they are not of standard format. Thus it is recommended that comments not be heavily used.

[key=part_foreign(2,comment.comment_id), db_delete=nullify]

comment_dc *comment domain*

Domain in which comment_id is unique.

[codelist=authorities, key=part_foreign(2,comment.comment_dc), db_delete=nullify]

SEE ALSO

NAME

signal_path – information about a data path from a single sensor to a recorder

C_SYNOPSIS

```
typedef struct {
        FIXED       structure_type;
        FIXED       structure_len;
        LABEL       signal_path_id;
        DOMAIN      signal_path_dc;
        REFERS2     site_id;
        DOMAIN      site_dc;
        FIXED       len_signal_n;
        STRING      signal_name[20];
        FIXED       len_site_n;
        STRING      site_name[8];
        AUTHOR      network;
        CODE1       component_type;
        CODE1       sensor_type;
        CODE1       band_type;
        CODE1       gain_type;
        CHAR        path_type;
        CODE1       amp_response;
        INT2        sensor_depth;
        FLOAT4      sensor_azimuth;
        FLOAT4      sensor_dip;
        FLOAT4      time_delay;
        FLOAT4      seismic_delay;
        ST_TIME     from_time;
        ST_TIME     thru_time;
        CODE4       data_type;
        INT4        data_length;
        REFERS2     comment_id;
        DOMAIN      comment_dc;
} SUDS_SIGNAL_PATH;

#define SIGNAL_PATHS        105L
```

DESCRIPTION

The **signal_path** structure is the primary link between data recorded and information about the hardware used to detect, transmit, and record the data. This structure contains information about a signal that is being recorded including the sensor type and sensor location as well as the path by which the signal is transmitted to the recorder. There should be at least one **signal_path** structure for each sensor. If a signal from the same sensor is transmitted by several paths to the same recorder, then a **signal_path** structure should exist for each path. If a signal from the same sensor is recorded on different recorders, then a **signal_path** structure should exist for each recorder. Once created, signal_path structures should not be modified, unless an error is discovered. If the path between a sensor and recorder changes, a new signal_path structure should be created, and the old one left alone so that it accurately describes the signal_path on which old data was transmitted.

Signal paths are labeled by the member **signal_name** which includes information about the site site name, the network name, the **component_type** (vertical, NS, EW), the **sensor_type**, the **band_type**, the **gain_type**, and the **path_type**. This is an extension of the old site names, and would typically be kept the same as the signal_path changes, as long as the pieces making up the name remain the same. In other words, the signal_name is not an identifier for the signal_path; use signal_path_id for unique identification.

When a **waveform** is formed by the addition of several waveforms, a separate **signal_path** structure should be created with at least **component** and **path_type** members reset and followed by a number of **beam_data** structures. The beam azimuth and dip (a function of slowness) should be put in the **sensor_azimuth** and **sensor_dip** members. This method also applies to specifying radial and transverse components formed by summing signals from two horizontal sensors.
[permissions="s_s_siu_s"]

MEMBERS

structure_type *structure type*
Define number of this type of structure.

structure_len *structure length*
Length of this structure in bytes.

signal_path_id *signal path id*
A number that uniquely refers, within this **domain_code**, to one instance of the **signal_path** structure representing a total signal path from a particular sensor to its recorder. In some cases the same sensor and recorder may be connected by separate paths.
[key=part_primary, db_index=clustered]

signal_path_dc *signal path domain*
Domain in which signal_path_id is unique.
[codelist=authorities, key=part_primary]

site_id *site id*
A number that uniquely refers to an instance of the **site** structure.
[key=part_foreign(1, site.site_id), db_delete=restrict, db_must_exist=true]

site_dc *site site domain*
Domain in which site_id is unique.
[codelist=authorities, key=part_foreign(1,site.site_dc), db_delete=restrict, db_must_exist=true]

len_signal_n *length signal name*
The maximum space reserved for the signal name, i.e. 20. Actual string can only contain 19 characters to allow for the NULL byte.

signal_name *signal name*
Name of a sensor component whose data are transmitted along a specific path and recorded on a particular recorder. Name is expected to be of the form network_station_CSBGP where the network is the abbreviation (part of the authority string preceeding the colon, 5 or less characters) for the **signal_path.network** code, station is **signal_path.station_name** (7 or less characters), C is the **signal_path.component_type** code (usually v, n, or e), S is the **signal_path.sensor_type** code, B is the **signal_path.band_type** code, G is the **signal_path.gain_type**, and P is the **signal_path.path_type** in only those stations where the same component may be recorded on two or more different recorders or transmitted over different paths.
[index_string=true]

len_site_n *length site name*
The maximum space reserved for the site name, i.e. 8. Actual string can only contain 7 characters to allow for the NULL byte.

site_name *site name*
Name of the site. Must be unique within this network. The site name is concatenated into the **signal_name.**

network *network*
The network containing this site. Typically this is an authority code assigned by the organization that installed and maintains the physical site. If a site is part of two or more networks, users may decide to have only one site structure with the network code for the primary

operator or duplicate site structures for each network with different network codes.
[codelist=authorities]

component_type *component type*
Sensor component: v=vertical, e=east-west, n=north-south, o=other (specified by sensor_azimuth and sensor_dip), etc. The **component_type** is concatenated into the **signal_name.**
[codelist=components]

sensor_type *sensor type*
Type of sensor. The **sensor type** is concatenated into the **signal_name.**
[codelist=sensor_types]

band_type *bandpass type*
Passband and general sampling rate of the sensor and signal_path. The **band_type** is concatenated into the **signal_name.**
[codelist=band_types]

gain_type *gain type*
Gain type for sites with several different outputs at several different gains from one amplifier or sensor.
[codelist=gain_types]

path_type *path type*
Type of path: a character locally defined for a network to differentiate between different signal paths between a specific sensor and a recorder. If this character is defined, it is concatenated into the **signal_name**.

amp_response *amplitude response type*
Type of amplitude response: n=normal, g=gain ranged.
[codelist=amplitude_types]

sensor_depth *depth to sensor, meters*
If sensor is in a borehole below a site, this is the depth in meters.

sensor_azimuth *seismometer azimuth*
Azimuth of sensor in degrees from north. Set =0 for vertical component.

sensor_dip *seismometer incidence*
Angle sensor makes with the horizontal in degrees. Thus for a vertical seismometer, the dip is 90 and for a horizontal seismometer the dip is 0.

time_delay *total time delay*
Total time delay of the path.

seismic_delay *seismic delay*
A time delay in seconds assigned to this **path** based on traveltime residuals determined when locating earthquakes. This delay will be subtracted from arrival times when locating an earthquake.

from_time *valid from time*
Time this calibration became valid.

thru_time *valid thru time*
Time this calibration became no longer valid.

data_type *data storage type*
An integer representing the type of data that follows this structure. If the integer is negative, it refers to **variable_tab.define_num**. If the integer is positive, it refers to **structure_tab.struct_number**. Data should only be BEAM_COMP_DATAS
[sets_data_type=true, codelist=data_types]

data_length *number of samples*

Number of structures of type **data_type** (beam_data(4)) that follow this structure.
[sets_data_length=true, default_value=0]

comment_id *comment id*
A number that uniquely refers, within this comment_dc, to an instance of the **comment** structure. Comments are generally not searchable because they are not of standard format. Thus it is recommended that comments not be heavily used.
[key=part_foreign(2,comment.comment_id), db_delete=nullify]

comment_dc *comment domain*
Domain in which comment_id is unique.
[codelist=authorities, key=part_foreign(2,comment.comment_dc), db_delete=nullify]

SEE ALSO

make_signal_name(2)

NAME

sig_cmp_data – the wiring of one sig_path_cmp to another

C_SYNOPSIS

```
typedef struct {
        REFERS2         connected_id;
        DOMAIN          connected_dc;
        INT2            pin_plus;
        INT2            pin_minus;
        INT2            pin_ground;
        CHAR            in_or_out;
        CHAR            spare_char;
} SUDS_SIG_CMP_DATA;

#define SIG_CMP_DATAS   302L
```

DESCRIPTION

Describes wiring one piece of equipment to another. The **sig_path_cmp** structure may be followed by **sig_path_cmp.data_length** structures of type **sig_cmp_data**.

[data_only=SIG_PATH_CMPS]

MEMBERS

connected_id *connected equipment id*

A number that uniquely identifies, within this domain, an instance of the **sig_path_cmp** structure for the piece of equipment connected to this piece of equipment.

[key=part_foreign(2,sig_path_cmp.sig_path_cmp_id)]

connected_dc *uniqueness domain*

Domain in which connected_id is unique.

[codelist=authorities, key=part_foreign(2,sig_path_cmp.sig_path_cmp_dc)]

pin_plus *plus pin number*

Number of the pin connected to the plus signal.

pin_minus *minus pin number*

Number of the pin connected to the minus signal.

pin_ground *ground pin number*

Number of the pin connected to ground.

in_or_out *direction*

Direction of this connection: i=input, o=output.

spare_char *for future use*

SEE ALSO

sig_path_cmp(4)

NAME

sig_path_ass – associates signal path components with signal paths

C_SYNOPSIS

```
typedef struct {
        FIXED           structure_type;
        FIXED           structure_len;
        REFERS2         signal_path_id;
        DOMAIN          signal_path_dc;
        REFERS2         sig_path_cmp_id;
        DOMAIN          sig_path_cmp_dc;
        REFERS2         site_id;
        DOMAIN          site_dc;
        INT4            spare;
        AUTHOR          authority;
        INT2            pos_in_path;
        INT2            channel_number;
        FLOAT4          frequency;
        FLOAT4          gain;
        FLOAT4          attenuation;
        ST_TIME         from_time;
        ST_TIME         thru_time;
        REFERS2         comment_id;
        DOMAIN          comment_dc;
} SUDS_SIG_PATH_ASS;

#define SIG_PATH_ASSS           316L
```

DESCRIPTION

The **sig_path_ass** structure associates **signal_paths** with components. For each **signal_path**, there typically exist several **sig_path_ass** structures, each with the same **signal_path_id** but with different **sig_path_ass_ids**. This is an associative table that implements a many-to-many relationship: a **signal_path** has many components and a component can be part of many **signal_paths**.

[permissions="s_s_siu_s"]

MEMBERS

structure_type *structure type*

Define number of this type of structure.

structure_len *structure length*

Length of this structure in bytes.

signal_path_id *signal path id*

A number that uniquely refers, within this **domain_code**, to one instance of the **signal_path** structure representing a total signal path from a particular sensor to its recorder. In some cases the same sensor and recorder may be connected by separate paths.

[key=part_primary, key=part_foreign(1,signal_path.signal_path_id), db_delete=restrict, db_must_exist=true, index_string=true]

signal_path_dc *signal path domain*

Domain in which signal_path_id is unique.

[codelist=authorities, key=part_primary, key=part_foreign(1,signal_path.signal_path_dc), db_delete=restrict, db_must_exist=true]

sig_path_cmp_id *sig path component id*

A number that uniquely refers, within this **component_type** and this **sig_path_cmp_dc**, to an instance of the **component** structure.

[key=part_primary, key=part_foreign(2,sig_path_cmp.sig_path_cmp_id), db_delete=restrict, db_must_exist=true]

sig_path_cmp_dc *component domain*

Domain in which sig_path_cmp_id is unique.
[codelist=authorities, key=part_primary, key=part_foreign(2,sig_path_cmp.sig_path_cmp_dc), db_delete=restrict, db_must_exist=true]

site_id *site id*

A number that uniquely refers to an instance of the **site** structure.
[key=part_foreign(3, site.site_id), db_delete=restrict, db_must_exist=true]

site_dc *site site domain*

Domain in which site_id is unique.
[codelist=authorities, key=part_foreign(3,site.site_dc), db_delete=restrict, db_must_exist=true]

spare *for future use*

authority *authority*

Who set up this association.
[codelist=authorities]

pos_in_path *position in path*

Used to determine the proper ordering of component structures such as **seismometer, sig_path_cmp, recorder** associated with the same **signal_path_id**. The sensor should be number 0, the on-site calibrator 10, the amp/vco 20, the computer atod 1000, the analog tape recorder 900, and the discriminator 950. Other components such as transmitters, receivers, antennas, summing amplifiers should be numbered in between or after these numbers as appropriate. Where multiple signal_paths go through the same component, the **pos_in_path** may not be identical for the same piece of hardware in different signal_paths.

channel_number *channel number*

Number of the physical channel ("pin number") on which this signal is recorded. This is not necessarily the same as the position in the sampling order, since some systems may sample input channels in non-sequential order. On analog systems, this is the tape-head number.

frequency *frequency*

Frequency associated with this particular **component** and **signal_path**.

gain *gain*

Gain as a scalar quanitity associated with this particular **sig_path_cmp** and **signal_path**.

attenuation *attenuation*

Attenuation associated with this particular **component** and **signal_path**. Negative number means gain.

from_time *valid from time*

Time this association became valid.

thru_time *valid thru time*

Time this association became no longer valid.

comment_id *comment id*

A number that uniquely refers, within this comment_dc, to an instance of the **comment** structure. Comments are generally not searchable because they are not of standard format. Thus it is recommended that comments not be heavily used.
[key=part_foreign(4,comment.comment_id), db_delete=nullify]

comment_dc *comment domain*

Domain in which comment_id is unique.
[codelist=authorities, key=part_foreign(4,comment.comment_dc), db_delete=nullify]

SEE ALSO

NAME

sig_path_cmp – information about an individual component in a signal path

C_SYNOPSIS

```
typedef struct {
        FIXED           structure_type;
        FIXED           structure_len;
        LABEL           sig_path_cmp_id;
        DOMAIN          sig_path_cmp_dc;
        REFERS2         response_id;
        DOMAIN          response_dc;
        AUTHOR          authority;
        CODE4           model;
        FIXED           len_serial_n;
        STRING          serial_number[12];
        FLOAT4          maximum_gain;
        CODE1           gain_units;
        CHAR            spare;
        INT2            setting1;
        INT2            setting2;
        INT2            setting3;
        FLOAT4          spare_a;
        FLOAT4          frequency;
        ST_TIME         new_battery;
        CODE4           data_type;
        INT4            data_length;
        REFERS2         comment_id;
        DOMAIN          comment_dc;
} SUDS_SIG_PATH_CMP;

#define SIG_PATH_CMPS   104L
```

DESCRIPTION

Information about an individual component in a **sig_path**.
[permissions="s_s_siu_s"]

MEMBERS

structure_type *structure type*
Define number of this type of structure.

structure_len *structure length*
Length of this structure in bytes.

sig_path_cmp_id *signal path comp id*
A number that uniquely refers, within this **sig_path_cmp_dc**, to an instance of the **sig_path_cmp** structure.
[key=part_primary, db_index=clustered]

sig_path_cmp_dc *signal path domain*
Domain in which sig_path_cmp_id is unique.
[codelist=authorities, key=part_primary]

response_id *response id*
A number that uniquely identifies, within this **response_dc**, an instance of the **response** structure.
[key=part_foreign(1,response.response_id), db_delete=nullify]

response_dc *response domain*
Domain in which response_id is unique.

[codelist=authorities, key=part_foreign(1,response.response_dc), db_delete=nullify]

authority *authority*
: Who specified this component.
[codelist=authorities]

model *model code*
: Number that is unique in the world designating the model of this piece of equipment. This number is associated with an ASCII string in codelist **equip_models**.
[codelist=equip_models]

len_serial_n *length serial number*
: The maximum space reserved for the serial number, i.e. 12. Actual string can only contain 11 characters.

serial_number *serial number*
: Serial number of the piece of equipment. Should be unique in the world for this model.
[index_string=true]

maximum_gain *maximum gain*
: Maximum gain of this component.

gain_units *gain units code*
: Units of **maximum gain**: d=decibels, p=pure scale multiplier.
[codelist=gain_unit_types]

spare *for future use*

setting1 *setting 1*
: Setting1, which should be attenuation or gain.

setting2 *setting 2*
: Setting 2, if it exists.

setting3 *setting 3*
: Setting 3, if it exists.

spare_a *for future use*

frequency *frequency*
: Natural frequency or center frequency as appropriate.

new_battery *date of new battery*
: Date new battery was installed.

data_type *data type*
: Type of structure that follows this structure. Normally SIG_PATH_DATAS.
[sets_data_type=true, codelist=data_types]

data_length *number of structures*
: Number of structures of type **sig_path_data** that follow this structure.
[sets_data_length=true, default_value=0]

comment_id *comment id*
: A number that uniquely refers, within this comment_dc, to an instance of the **comment** structure. Comments are generally not searchable because they are not of standard format. Thus it is recommended that comments not be heavily used.
[key=part_foreign(2,comment.comment_id), db_delete=nullify]

comment_dc *comment domain*
: Domain in which comment_id is unique.
[codelist=authorities, key=part_foreign(2,comment.comment_dc), db_delete=nullify]

SEE ALSO

sig_path_data(4)

NAME

sig_path_data – List of signal_paths to follow the focal_mechanism structure

SYNOPSIS

```
typedef struct {
        REFERS2         signal_path_id;
        DOMAIN          signal_path_dc;
} SUDS_SIG_PATH_DATA;

#define SIG_PATH_DATAS  306L
```

DESCRIPTION

List of signal_paths to follow the **focal_mechanism** structure.
[data_only=FOCAL_MECHS]

MEMBERS

signal_path_id *signal path id*

A number that uniquely refers, within this **domain_code**, to one instance of the **signal_path** structure representing a total signal path from a particular sensor to its recorder. In some cases the same sensor and recorder may be connected by separate paths.
[key=part_foreign(1,signal_path.signal_path_id)]

signal_path_dc *signal path domain*

Domain in which signal_path_id is unique.
[codelist=authorities, key=part_foreign(1,signal_path.signal_path_dc)]

SEE ALSO

NAME

signif_event – information about a major earthquake that complements the solution

C_SYNOPSIS

```
typedef struct {
        FIXED       structure_type;
        FIXED       structure_len;
        REFERS2     event_id;
        DOMAIN      event_dc;
        FIXED       len_eq_name;
        STRING      eq_name[20];
        FIXED       len_country;
        STRING      country[16];
        FIXED       len_state;
        STRING      state[16];
        INT2        local_time;
        INT2        num_felt_rep;
        AUTHOR      felt_authority;
        FLOAT4      event_magnitude;
        AUTHOR      mag_authority;
        AUTHOR      mm_authority;
        INT2        mm_intensity;
        CHAR        event_type;
        CHAR        spare_code;
        CHAR        tectonism;
        CHAR        waterwave;
        CHAR        mechanism;
        CHAR        medium;
        AUTHOR      tect_auth;
        AUTHOR      water_auth;
        AUTHOR      mech_auth;
        AUTHOR      medium_auth;
        FLOAT4      len_aftersh;
        FLOAT4      dip_aftersh;
        FLOAT4      strike_aftersh;
        FLOAT4      peak_accel;
        AUTHOR      accel_auth;
        REFERS2     comment_id;
        DOMAIN      comment_dc;
} SUDS_SIGNIF_EVENT;

#define SIGNIF_EVENTS       113L
```

DESCRIPTION

Information about a significant or major earthquake. This is information that is additional to an **event** structure that is typically of interest for very large earthquakes only.
[permissions="siud_s_s_s"]

MEMBERS

structure_type *structure type*
Define number of this type of structure.

structure_len *structure length*
Length of this structure in bytes.

event_id *event id*
A number that uniquely refers, within this event_dc, to an instance of the **event** structure.

[key=part_foreign(1,event.event_id), db_delete=cascade, db_must_exist=true, index_string=true]

event_dc *event domain*
Domain in which event_id is unique.
[codelist=authorities, key=part_foreign(1,event.event_dc), db_delete=cascade, db_must_exist=true]

len_eq_name *length eq name*
The maximum space reserved for the earthquake name, i.e. 20. Actual string can only contain 19 characters to allow for the NULL byte.

eq_name *earthquake name*
Name of this earthquake.

len_country *length country name*
The maximum space reserved for the country name, i.e. 16. Actual string can only contain 15 characters to allow for the NULL byte.

country *country name*
Name of country earthquake is in or off the coast of.

len_state *length of state name*
The maximum space reserved for the state name, i.e. 16. Actual string can only contain 15 characters to allow for the NULL byte.

state *state name*
Name of state containing the earthquake.

local_time *local time*
Difference of local time minus Greenwich mean time in minutes.

num_felt_rep *number of felt reports*
Number of felt reports.

felt_authority *felt authority*
Who collected the felt reports.
[codelist=authorities]

event_magnitude *event magnitude*
Summary magnitude of this event.

mag_authority *magnitude authority*
Who calculated the magnitude.
[codelist=authorities]

mm_authority *mercali authority*
Who determined the Modified Mercalli Intensity.
[codelist=authorities]

mm_intensity *mm intensity*
The Modified Mercalli Intensity.

event_type *event type*
A character designating the type of event.

spare_code *for future use*

tectonism *tectonism observed*
Observed u=uplift, s=subsidence, S=strikeslip faulting, N=normal faulting, T=thrust.

waterwave *waterwave observed*
s=seiche, t=tsunami.

mechanism *focal mechanism*
t=thrust, s=strike-slip, n=normal, e=explosive.

medium *medium*

Medium around the earthquake or explosion if known.

tect_auth *tectonism authority*
Who reported tectonism.
[codelist=authorities]

water_auth *water authority*
Who reported water waves.
[codelist=authorities]

mech_auth *mechanism authority*
Who reported focal mechanism.
[codelist=authorities]

medium_auth *medium authority*
Who reported the medium.
[codelist=authorities]

len_aftersh *length aftershocks*
Length of aftershock zone in kilometers.

dip_aftersh *dip aftershocks*
Dip of aftershock zone in degrees from horizontal.

strike_aftersh *strike aftershocks*
Strike of aftershock zone in degrees clockwise from north.

peak_accel *peak acceleration*
Largest peak acceleration observed by all reliable accelerographs during this event.

accel_auth *medium authority*
Who reported the peak acceleration.
[codelist=authorities]

comment_id *comment id*
A number that uniquely refers, within this comment_dc, to an instance of the **comment** structure. Comments are generally not searchable because they are not of standard format. Thus it is recommended that comments not be heavily used.
[key=part_foreign(2,comment.comment_id), db_delete=nullify]

comment_dc *comment domain*
Domain in which comment_id is unique.
[codelist=authorities, key=part_foreign(2,comment.comment_dc), db_delete=nullify]

SEE ALSO

NAME

site – geographical location and other information about a site containing equipment, source, etc.

C_SYNOPSIS

```
typedef struct {
        FIXED       structure_type;
        FIXED       structure_len;
        LABEL       site_id;
        DOMAIN      site_dc;
        LATIT       site_lat;
        LONGIT      site_long;
        FLOAT4      site_elev;
        CODE1       coordinates;
        CODE1       distance_units;
        CODE1       depth_units;
        CODE1       site_type;
        REFERS2     coordinate_id;
        DOMAIN      coordinate_dc;
        FIXED       len_site_n;
        STRING      site_name[8];
        FIXED       len_old_name;
        STRING      old_name[8];
        CODE4       network;
        CODE1       site_precision;
        CODE1       elev_precision;
        CODE1       survey_method;
        CHAR        spare;
        CODE1       status;
        CODE1       region_type;
        INT2        region;
        CODE1       site_cond;
        CODE1       enclosure;
        CODE2       rock_type;
        FIXED       len_site_d;
        STRING      site_descrip[44];
        ST_TIME     from_time;
        ST_TIME     thru_time;
        REFERS2     comment_id;
        DOMAIN      comment_dc;
} SUDS_SITE;

#define SITES       300L
```

DESCRIPTION

A site is a geographical location where sensors, other pieces of equipment, a seismic source, etc. are located.

[permissions="s_s_siu_s"]

MEMBERS

structure_type *structure type*

Define number of this type of structure.

structure_len *structure length*

Length of this structure in bytes.

site_id *site id*

A number that uniquely refers, within this **site_dc**, to an instance of the **site** structure.

[key=part_primary, db_index=clustered]

site_dc *site domain*
Domain in which site_id is unique.
[key=part_primary, codelist=authorities]

site_lat *site latitude*
Latitude, south is negative. May be north-south coordinate on a local coordinate system, or distance on a radial coordinate system.

site_long *site longitude*
Longitude, west is negative. May be east-west coordinate on a local coordinate system, or azimuth in degrees on a radial coordinate system.

site_elev *site elevation*
Elevation in kilometers, above sea level is positive.

coordinates *coordinate system*
[codelist=coordinate_types]

distance_units *distance units*
[codelist=units_types]

depth_units *depth units*
[codelist=units_types]

site_type *type of site*
[codelist=site_types]

coordinate_id *coordinate id*
If coordinates are on a local grid, this points to a coordinate_sys structure describing the grid in earth coordinates.
[key=part_foreign(1,coordinate_sys.coordinate_id), db_delete=nullify]

coordinate_dc *coordinate system dc*
Domain in which coordinate_id is unique.
[codelist=authorities, key=part_foreign(1,coordinate_sys.coordinate_dc), db_delete=nullify]

len_site_n *len site name*
The maximum space reserved for the site name, i.e. 8. Actual string can only contain 7 characters to allow for the NULL byte.

site_name *site name*
Name of the site. Must be unique within this network. The site name is concatenated into the **signal_name.**
[index_string=true]

len_old_name *len old site name*
The maximum space reserved for the old site name, i.e. 8. Actual string can only contain 7 characters to allow for the NULL byte.

old_name *old site name*
Former name of this site. For use in networks where names have been changed to protect the innocent. This name acts as a cross-reference to another **site** structure whose **site.thru_time** has passed.

network *network*
The network containing this site. Typically this is an authority code assigned by the organization that installed and maintains the physical site. If a site is part of two or more networks, users may decide to have only one site structure with the network code for the primary operator or duplicate site structures for each network with different network codes.
[codelist=authorities]

site_precision *site precision*
Precision with which site is located.
[codelist=precision_codes]

elev_precision *elevation precision*
Precision with which elevation is determined.
[codelist=precision_codes]

survey_method *survey method*
Method used to survey location of site.
[codelist=survey_methods]

spare *for future use*

status *site status*
Status of this site.
[codelist=status]

region_type *type of region*
[codelist=region_types]

region *region in network*
Number of the region within this network. These numbers are assigned by the authority for this network and apply only within this network.

site_cond *site condition type*
Condition of site: p=permafrost
[codelist=site_conditions]

enclosure *enclosure code*
Type of structure that encloses the sensor: d=dam, n=nuclear power plant, v=underground vault, b=buried, s=on surface, etc.
[codelist=enclosure_types]

rock_type *rock code*
Code for type of rock.
[codelist=rock_types]

len_site_d *length site description*
The maximum space reserved for the site description, i.e. 44. Actual string can only contain 43 characters to allow for the NULL byte.

site_descrip *site description*
Description of the site.

from_time *beginning time*
Time any data first began to be collected from this site.
[db_index=true]

thru_time *ending time*
Time last data was collected from this site.
[db_index=true]

comment_id *comment id*
A number that uniquely refers, within this comment_dc, to an instance of the **comment** structure. Comments are generally not searchable because they are not of standard format. Thus it is recommended that comments not be heavily used.
[key=part_foreign(2,comment.comment_id), db_delete=nullify]

comment_dc *comment domain*
Domain in which comment_id is unique.
[codelist=authorities, key=part_foreign(2,comment.comment_dc), db_delete=nullify]

SEE ALSO

NAME

solution – information about a particular solution of an event

C_SYNOPSIS

```
typedef struct {
        FIXED       structure_type;
        FIXED       structure_len;
        LABEL       solution_id;
        DOMAIN      solution_dc;
        REFERS2     event_id;
        DOMAIN      event_dc;
        ST_TIME     time_sol_done;
        AUTHOR      authority;
        MS_TIME     origin_time;
        LATIT       origin_lat;
        LONGIT      origin_long;
        FLOAT4      origin_depth;
        CODE1       solution_type;
        CODE1       depth_control;
        CODE1       time_control;
        CODE1       epi_control;
        CODE4       region;
        CODE1       region_type;
        CHAR        spare_a;
        CODE1       quality;
        CODE1       hypo_program;
        INT2        hypo_prog_vers;
        CODE1       convergence;
        CODE1       pref_mag_type;
        FLOAT4      pref_magnitude;
        AUTHOR      pref_mag_auth;
        INT4        spare;
        FIXED       len_contr_n;
        STRING      control_name[20];
        INT2        num_iterations;
        INT2        gap_of_stations;
        FLOAT4      rms_of_resids;
        FLOAT4      horiz_error;
        FLOAT4      depth_error;
        FLOAT4      depth_err_up;
        FLOAT4      depth_err_down;
        FLOAT4      dist_near_stat;
        FLOAT4      near_s_p_time;
        FLOAT4      p2s_vel_ratio;
        INT2        num_stat_good;
        INT2        num_p_rep_good;
        INT2        num_p_used;
        INT2        num_s_rep_good;
        INT2        num_s_used;
        INT2        num_resid_disc;
        REFERS2     comment_id;
        DOMAIN      comment_dc;
} SUDS_SOLUTION;

#define SOLUTIONS           114L
```

DESCRIPTION

Information about a particular solution for the hypocenter of an earthquake.
[permissions="siud_siud_s_s"]

MEMBERS

structure_type *structure type*
Define number of this type of structure.

structure_len *structure length*
Length of this structure in bytes.

solution_id *solution id*
A number that uniquely refers, within this solution_dc, to an instance of the **solution** structure.
[key=part_primary, db_index=clustered]

solution_dc *solution domain*
Domain in which solution_id is unique.
[codelist=authorities, key=part_primary]

event_id *event id*
A number that uniquely refers, within this **event_dc**, to an instance of the **event** structure.
[key=part_foreign(1,event.event_id), db_delete=cascade, db_must_exist=true, index_string=true]

event_dc *event domain*
Domain in which event_id is unique.
[codelist=authorities, key=part_foreign(1,event.event_dc), db_delete=cascade, db_must_exist=true]

time_sol_done *time solution done*
Time solution done.

authority *authority for solution*
Who did this solution.
[codelist=authorities]

origin_time *origin time*
Origin time.

origin_lat *origin latitude*
Latitude, south is negative.

origin_long *origin longitude*
Longitude, west is negative.

origin_depth *origin depth*
Depth of hypocenter in kilometers below the ground surface.

solution_type *type of solution*
Type of solution such as automatic, catalog or final, preliminary, etc.
[codelist=solution_types]

depth_control *depth control*
Whether depth was held fixed and by what criteria.
[codelist=depth_controls]

time_control *time control code*
Whether time was held fixed and by what criteria.
[codelist=hypo_controls]

epi_control *epicenter control*
Whether the epicenter was held fixed and by what criteria.
[codelist=hypo_controls]

region *region code*
Code number of the region containing the earthquake.

[codelist=regions]

region_type *type of region*
Type of region code being used: k=Klein system in California, f=Flynn-Engdahl region in the world
[codelist=region_types]

spare_a *for future use*

quality *quality of solution*
An estimate of quality of the solution. 0 equals best, 4 equals worst,. or A equals best, D equals worst.
[codelist=solution_qualities]

hypo_program *location program type*
Type of location program used.
[codelist=hypo_programs]

hypo_prog_vers *hypo prog vers type*
Version of hypocenter program. 10 means version 1.0

convergence *convergence type*
Type of convergence in the solution.
[codelist=convergences]

pref_mag_type *pref magnitude type*
Type of preferred magnitude.
[codelist=magnitude_types]

pref_magnitude *preferred magnitude*
Magnitude preferred for this solution.

pref_mag_auth *authority of pref mag*
Authority who determined preferred magnitude.
[codelist=authorities]

spare *for future use*

len_contr_n *len control name*
Length of string for control file name, 20. True length may only be 19 to allow for the NULL byte.

control_name *control file name*
File name of control file for hypocenter program.

num_iterations *number of iterations*
Number of iterations to calculate this solutions. Gives some relative indication of rate of convergence.

gap_of_stations *gap of stations*
Maximum gap between azimuths to stations in degrees.
[ed_col=62, ed_row=26]

rms_of_resids *rms of residuals*
Root mean square of the residuals.

horiz_error *horizontal error*
Horizontal error in kilometers.
[ed_col=62, ed_row=27]

depth_error *depth error*
Depth error in kilometers.

depth_err_up *depth error up*
Possible error in depth in the upward direction in kilometers.

[ed_col=62, ed_row=28]

depth_err_down *depth error down*
Possible error in depth in the downward direction in kilometers.

dist_near_stat *distance nearest stat*
Distance from epicenter to the nearest station in kilometers.
[ed_col=62, ed_row=29]

near_s_p_time *nearest s-p time*
Difference in seconds of s arrival time minus p arrival time for the nearest station.

p2s_vel_ratio *p2s vel ratio*
Ratio of P-wave velocity to S-wave velocity.
[ed_col=62, ed_row=30]

num_stat_good *num stats reporting*
Number of stations reporting good p or s phases.

num_p_rep_good *num good p reported*
Number of stations reporting p phases.
[ed_col=62, ed_row=31]

num_p_used *num p used*
Number of p phases used in the solution.

num_s_rep_good *num good s reported*
Number of stations reporting good s phases.
[ed_col=62, ed_row=32]

num_s_used *num s used*
Number of s phases used in the solution.

num_resid_disc *num residuals discarded*
Number of residuals discarded from the solution by the program.
[ed_col=62, ed_row=33]

comment_id *comment id*
A number that uniquely refers, within this comment_dc, to an instance of the **comment** structure. Comments are generally not searchable because they are not of standard format. Thus it is recommended that comments not be heavily used.
[key=part_foreign(2,comment.comment_id), db_delete=nullify]

comment_dc *comment domain*
Domain in which comment_id is unique.
[codelist=authorities, key=part_foreign(2,comment.comment_dc), db_delete=nullify]

SEE ALSO

BUGS

Error in lat, lon, time instead of horizontal error?

NAME

solution_err – error for an earthquake solution

C_SYNOPSIS

```
typedef struct {
        FIXED           structure_type;
        FIXED           structure_len;
        REFERS2         solution_id;
        DOMAIN          solution_dc;
        FLOAT4          covar_xx;
        FLOAT4          covar_yy;
        FLOAT4          covar_zz;
        FLOAT4          covar_tt;
        FLOAT4          covar_xy;
        FLOAT4          covar_xz;
        FLOAT4          covar_yz;
        FLOAT4          covar_tx;
        FLOAT4          covar_ty;
        FLOAT4          covar_tz;
        FLOAT4          std_error;
        FLOAT4          major_azimuth;
        FLOAT4          major_dip;
        FLOAT4          major_length;
        FLOAT4          inter_azimuth;
        FLOAT4          inter_dip;
        FLOAT4          inter_length;
        FLOAT4          minor_length;
        FLOAT4          depth_error;
        FLOAT4          time_error;
        FLOAT4          confidence;
        FLOAT4          spare;
        REFERS2         comment_id;
        DOMAIN          comment_dc;
} SUDS_SOLUTION_ERR;

#define SOLUTION_ERRS           115L
```

DESCRIPTION

Error of solution express as the covariance matrix.
[permissions="siu_siu_s_s"]

MEMBERS

structure_type *structure type*
Define number of this type of structure.

structure_len *structure length*
Length of this structure in bytes.

solution_id *solution id*
A number that uniquely refers, within this solution_dc, to an instance of the **solution** structure.
[key=part_foreign(1,solution.solution_id), db_delete=cascade, db_must_exist=true, index_string=true]

solution_dc *solution domain*
Domain in which solution_id is unique.
[codelist=authorities, key=part_foreign(1,solution.solution_dc), db_delete=cascade, db_must_exist=true]

covar_xx *covariance matrix 1*
Covariance matrix. Element 1.

covar_yy *covariance matrix 2*
Covariance matrix. Element 2.

covar_zz *covariance matrix 3*
Covariance matrix. Element 3.

covar_tt *covariance matrix 4*
Covariance matrix. Element 4.

covar_xy *covariance matrix 5*
Covariance matrix. Element 5.

covar_xz *covariance matrix 6*
Covariance matrix. Element 6.

covar_yz *covariance matrix 7*
Covariance matrix. Element 7.

covar_tx *covariance matrix 8*
Covariance matrix. Element 8.

covar_ty *covariance matrix 9*
Covariance matrix. Element 9.

covar_tz *covariance matrix 10*
Covariance matrix. Element 10.

std_error *standard error*

major_azimuth *azimuth major axis*
Azimuth of the semi-major axis of the error ellipse.

major_dip *dip major axis*
Dip of the semi-major axis of the error ellipse.

major_length *length major axis*
Length of the semi-major axis of the error ellipse in kilometers.

inter_azimuth *azimuth inter axis*
Azimuth of the intermediate axis of the error ellipse.

inter_dip *dip inter axis*
Dip of the intermediate axis of the error ellipse.

inter_length *length inter axis*
Length of the intermediate axis of the error ellipse in kilometers.

minor_length *length minor axis*
Length of the semi-minor axis of the error ellipse in kilometers.

depth_error *depth error*

time_error *origin time error*

confidence *confidence*

spare *for future use*

comment_id *comment id*
A number that uniquely refers, within this comment_dc, to an instance of the **comment** structure. Comments are generally not searchable because they are not of standard format. Thus it is recommended that comments not be heavily used.
[key=part_foreign(2,comment.comment_id), db_delete=nullify]

comment_dc *comment domain*
Domain in which comment_id is unique.
[codelist=authorities, key=part_foreign(2,comment.comment_dc), db_delete=nullify]

SEE ALSO

NAME

source – description of a man-made seismic event such as an explosion

C_SYNOPSIS

```
typedef struct {
        FIXED       structure_type;
        FIXED       structure_len;
        LABEL       source_id;
        DOMAIN      source_dc;
        REFERS2     site_id;
        DOMAIN      site_dc;
        FIXED       len_src_name;
        STRING      source_name[12];
        MS_TIME     origin_time;
        MS_TIME     nominal_time;
        FLOAT4      water_depth;
        FLOAT4      yield;
        CODE1       coordinates;
        CODE1       event_type;
        CODE1       sweep_type;
        CODE1       taper_type;
        INT2        begin_freq;
        INT2        end_freq;
        INT2        sweep_length;
        INT2        begin_taper;
        INT2        end_taper;
        INT2        signal_lag;
        FLOAT4      source_static;
        AUTHOR      authority;
        REFERS2     comment_id;
        DOMAIN      comment_dc;
} SUDS_SOURCE;

#define SOURCES     321L
```

DESCRIPTION

Information about a man-made seismic event such as an explosion or a Vibroseis sweep.

MEMBERS

structure_type *structure type*

Define number of this type of structure.

structure_len *structure length*

Length of this structure in bytes.

source_id *source id*

A number that uniquely identifies, within this **source_dc**, an instance of the **source** structure.
[key=part_primary, db_index=clustered, index_string=true]

source_dc *source domain*

Domain in which source_id is unique.
[codelist=authorities, key=part_primary]

site_id *site id*

A number that uniquely refers to an instance of the **site** structure.
[key=part_foreign(1, site.site_id), db_delete=restrict, db_must_exist=true]

site_dc *site site domain*

Domain in which site_id is unique.

[codelist=authorities, key=part_foreign(1,site.site_dc), db_delete=restrict, db_must_exist=true]

len_src_name *length source name*
The maximum space reserved for the event name, i.e. 20. Actual string can only contain 19 characters to allow for the NULL byte.

source_name *event name*
Name of this event.
[index_string=true]

origin_time *origin time*
Origin time.

nominal_time *nominal origin time*
Nominal origin time of the explosion with no time corrections.

water_depth *water depth*
Depth of water in kilometers below the **origin_elev**. If shot is in the water, **origin_depth** is less than **water_depth**. Plus is down.

yield *source yield*
Kilograms of explosive or equivalent for this type of source.

coordinates *coordinates*
Units of the latitude and longitude if not in degrees.
[codelist=units_types]

event_type *type of event*
A character designating the type of event.
[codelist=event_types]

sweep_type *sweep function*
Type of sweep.
[codelist=functions]

taper_type *type of taper*
Type of taper applied to the sweep signal.
[codelist=functions]

begin_freq *frequency begin sweep*
For a vibroseis type signal source, beginning frequency in hertz.

end_freq *frequency end sweep*
For a vibroseis type signal source, ending frequency in hertz.

sweep_length *sweep length*
For a vibroseis type signal source, length of the frequency sweep in milliseconds.

begin_taper *beginning taper*
For a vibroseis type signal source, length of the beginning taper in milliseconds.

end_taper *ending taper*
For a vibroseis type signal source, length of the ending taper in milliseconds.

signal_lag *lag of signal*
Amount in degrees that seismic signal lags pilot signal to vibrator.

source_static *source static*
Static time correction at the source.

authority *authority*
A number representing an institution or authority operating a network, calculating a solution, make an instrument calibration, etc. The authority is specified as a number that refers to an ASCII string in the authority codelist. Each institution has a base number such as 10000, 20000, etc. The institution may assign the 9999 numbers above their base number to

individual people or groups. The individual number might be set to agree with the user number in /etc/passwd on UNIX systems.
[codelist=authorities]

comment_id *comment id*
A number that uniquely refers, within this comment_dc, to an instance of the **comment** structure. Comments are generally not searchable because they are not of standard format. Thus it is recommended that comments not be heavily used.
[key=part_foreign(2,comment.comment_id), db_delete=nullify]

comment_dc *comment domain*
Domain in which comment_id is unique.
[codelist=authorities, key=part_foreign(2,comment.comment_dc), db_delete=nullify]

SEE ALSO

NAME

spectra – spectra of a waveform

C_SYNOPSIS

```
typedef struct {
        FIXED       structure_type;
        FIXED       structure_len;
        LABEL       spectra_id;
        DOMAIN      spectra_dc;
        REFERS2     waveform_id;
        DOMAIN      waveform_dc;
        CODE1       spectra_type;
        CODE1       x_units;
        CODE1       y_units;
        CODE1       taper_type;
        FLOAT4      low_taper_from;
        FLOAT4      low_taper_to;
        FLOAT4      high_taper_from;
        FLOAT4      high_taper_to;
        FLOAT4      damping;
        FLOAT4      corner_freq;
        FLOAT4      prec_dig_rate;
        FLOAT4      spare;
        AUTHOR      authority;
        CODE4       data_type;
        INT4        data_length;
        REFERS2     comment_id;
        DOMAIN      comment_dc;
} SUDS_SPECTRA;

#define SPECTRAS    301L
```

DESCRIPTION

X and Y points along a spectral curve determined from a **waveform**.

[permissions="siud_siu_s_s"]

MEMBERS

structure_type *structure type*

Define number of this type of structure.

structure_len *structure length*

Length of this structure in bytes.

spectra_id *spectra id*

A number that uniquely refers, within this **spectra_dc**, to an instance of the **spectra** structure.

[key=part_primary, db_index=clustered]

spectra_dc *spectra domain*

Domain in which spectra_id is unique.

[codelist=authorities, key=part_primary]

waveform_id *waveform id*

A number that uniquely refers, within this **waveform_dc**, to an instance of the **waveform** structure and waveform after the specified filtering has been applied.

[key=part_foreign(1,waveform.waveform_id), db_delete=cascade, db_must_exist=true, index_string=true]

waveform_dc *waveform domain*

Domain in which waveform_id is unique.

[codelist=authorities key=part_foreign(1,waveform.waveform_dc), db_delete=cascade, db_must_exist=true

]

spectra_type *spectra type*
[codelist=spectra_types]

x_units *x units*
Units on x-axis. Typically seconds.
[codelist=units_types]

y_units *y units*
Units on y-axis.
[codelist=units_types]

taper_type *taper type*
Type of taper used at the lower and upper bounds of the window defining the part of the waveform used to caluculate this spectra.
[codelist=taper_types]

low_taper_from *low taper from*
Lowest period of lower taper.

low_taper_to *low taper to*
Highest period of lower taper.

high_taper_from *high taper from*
Lowest period of higher taper.

high_taper_to *high taper to*
Highest period of higher taper.

damping *damping used*
Percent damping used.

corner_freq *corner frequency*

prec_dig_rate *calculated samples/sec*
Rate of digitization in samples per second used to calculate the spectra. This should be less than or equal to the **prec_dig_rate** given in the waveform structure used.

spare *for future use*

authority *authority for spectra*
Who calculated this spectra.
[codelist=authorities]

data_type *data storage type*
An integer representing the type of data that follows this structure. If the integer is negative, it refers to **variable_tab.define_num**. If the integer is positive, it refers to **structure_tab.struct_number**. The data may be of types such as FLOAT4, FLOAT8, etc. and will be in order of X-Y or period vs spectral-amplitude pairs. Thus the data_length equals twice the number of points.
[sets_data_type=true, codelist=data_types]

data_length *number of samples*
Number of samples of type **data_type** in the spectra.
[sets_data_length=true, default_value=0]

comment_id *comment id*
A number that uniquely refers, within this comment_dc, to an instance of the **comment** structure. Comments are generally not searchable because they are not of standard format. Thus it is recommended that comments not be heavily used.
[key=part_foreign(2,comment.comment_id), db_delete=nullify]

comment_dc *comment domain*

Domain in which comment_id is unique.
[codelist=authorities, key=part_foreign(2,comment.comment_dc), db_delete=nullify]

SEE ALSO

NAME

ssam_output – data from Seismic Spectral Amplitude Monitor

C_SYNOPSIS

```
typedef struct {
        FIXED           structure_type;
        FIXED           structure_len;
        REFERS2         ssam_setup_id;
        DOMAIN          ssam_setup_dc;
        INT4            num_band_chan;
        INT4            spare;
        CODE4           data_type;
        INT4            data_length;
        REFERS2         comment_id;
        DOMAIN          comment_dc;
} SUDS_SSAM_OUTPUT;

#define SSAM_OUTPUTS    311L
```

DESCRIPTION

Data header from the SSAM (Seismic Spectral Amplitude Monitor). Typically followed by data of type **FLOAT4** which is the average absolute spectral amplitude within each frequency passband described in the **ssam_band_data** following an ssam_setup structure.

MEMBERS

structure_type *structure type*

Define number of this type of structure.

structure_len *structure length*

Length of this structure in bytes.

ssam_setup_id *ssam setup id*

A number that uniquely refers, within this **ssam_setup_dc**, to an instance of the **ssam_setup** structure.

[key=part_foreign(1,ssam_setup.ssam_setup_id), db_delete=cascade, db_must_exist=true, index_string=true]

ssam_setup_dc *ssam setup domain*

Domain in which ssam_setup_id is unique.

[codelist=authorities, key=part_foreign(1,ssam_setup.ssam_setup_dc), db_delete=cascade, db_must_exist=true]

num_band_chan *num bandpass channels*

Number of FLOAT4 data points following this header structure. Each data point is the average absolute spectral amplitude within each frequency passband specified by a **ssam_setup**.

spare *for future use*

data_type *data type*

Type of data that follows this structure. Typically of type **FLOAT4**.

[sets_data_type=true, codelist=data_types]

data_length *number of points*

Number of structures of type **data_type** that follow this structure.

[sets_data_length=true, default_value=0]

comment_id *comment id*

A number that uniquely refers, within this comment_dc, to an instance of the **comment** structure. Comments are generally not searchable because they are not of standard format. Thus it is recommended that comments not be heavily used.

[key=part_foreign(2,comment.comment_id), db_delete=nullify]

comment_dc *comment domain*

Domain in which comment_id is unique.

[codelist=authorities, key=part_foreign(2,comment.comment_dc), db_delete=nullify]

SEE ALSO

ssam_band_data(4), ssam_setup(4)

NAME

ssam_band_data – passband for the Seismic Spectral Amplitude Monitor

C_SYNOPSIS

```
typedef struct {
        FLOAT4          upper_freq;
        FLOAT4          lower_freq;
} SUDS_SSAM_BAND_DATA;

#define SSAM_BAND_DATAS  317L
```

DESCRIPTION

Passband for the SSAM (Seismic Spectral Amplitude Monitor). A number of these structures follow an **ssam_setup** specifying the different spectral passbands.
[data_only=SSAM_SETUPS]

MEMBERS

upper_freq *upper frequency*
Frequency of the upper end of the frequency passband.

lower_freq *lower frequency*
Frequency of the lower end of the frequency passband.

SEE ALSO

ssam_data(4), ssam_setup(4)

NAME

ssam_setup – parameters to setup Seismic Spectral Amplitude Monitor

C_SYNOPSIS

```
typedef struct {
        FIXED        structure_type;
        FIXED        structure_len;
        LABEL        ssam_setup_id;
        DOMAIN       ssam_setup_dc;
        FLOAT4       nom_dig_rate;
        INT2         num_band_chan;
        INT2         samp_per_fft;
        CODE4        data_type;
        INT4         data_length;
        REFERS2      comment_id;
        DOMAIN       comment_dc;
} SUDS_SSAM_SETUP;

#define SSAM_SETUPS          310L
```

DESCRIPTION

Setup parameters for the SSAM (Seismic Spectral Amplitude Monitor). SSAM digitizes data from a sensor at the **nom_dig_rate**. A fast fourier transform (fft) is then applied to a series of samples (**samp_per_fft**). The average absolute spectral amplitude within each frequency passband is then output following an **ssam_data**. This ssam_setup structure is typically followed by a number of **ssam_band_data** structures.

MEMBERS

structure_type *structure type*
Define number of this type of structure.

structure_len *structure length*
Length of this structure in bytes.

ssam_setup_id *ssam setup id*
A number that uniquely refers, within this **ssam_setup_dc**, to an instance of the **ssam_setup** structure.
[key=part_primary, db_index=clustered]

ssam_setup_dc *ssam setup domain*
Domain in which ssam_setup_id is unique.
[codelist=authorities, key=part_primary]

nom_dig_rate *samples per second*
Nominal rate of digitization in samples per second.

num_band_chan *num bandpass channels*
Number of spectral bandpass channels. This structure is followed by an **ssam_passband** structure for each channel.
[index_string=true]

samp_per_fft *samples per fft*
Number of samples over which the fft is performed.

data_type *data type*
Type of structure that follows this structure. Normally SSAM_BAND_DATAS.
[sets_data_type=true, codelist=data_types]

data_length *number of points*
Number of structures of type **ssam_band_data** that follow this structure.
[sets_data_length=true, default_value=0]

comment_id *comment id*

A number that uniquely refers, within this comment_dc, to an instance of the **comment** structure. Comments are generally not searchable because they are not of standard format. Thus it is recommended that comments not be heavily used.

[key=part_foreign(2,comment.comment_id), db_delete=nullify]

comment_dc *comment domain*

Domain in which comment_id is unique.

[codelist=authorities, key=part_foreign(2,comment.comment_dc), db_delete=nullify]

SEE ALSO

ssam_data(4), ssam_band_data(4)

NAME

user_vars – user defined variables

C_SYNOPSIS

```
typedef struct {
        FIXED       structure_type;
        FIXED       structure_len;
        LABEL       user_vars_id;
        DOMAIN      user_vars_dc;
        REFERS2     waveform_id;
        DOMAIN      waveform_dc;
        INT4        waveform_type;
        INT2        spare;
        CHAR        type_zero;
        CHAR        type_one;
        CHAR        type_two;
        CHAR        type_three;
        CHAR        type_four;
        CHAR        type_five;
        CHAR        type_six;
        CHAR        type_seven;
        CHAR        type_eight;
        CHAR        type_nine;
        FLOAT4      zero;
        FLOAT4      one;
        FLOAT4      two;
        FLOAT4      three;
        FLOAT4      four;
        FLOAT4      five;
        FLOAT4      six;
        FLOAT4      seven;
        FLOAT4      eight;
        FLOAT4      nine;
        REFERS2     comment_id;
        DOMAIN      comment_dc;
} SUDS_USER_VARS;

#define USER_VARSS          322L
```

DESCRIPTION

A list of 10 variables that can be defined by a user and referenced to a particular instance of a waveform or any other structure through the primary key of any other structure type.

MEMBERS

structure_type *structure type*
Define number of this type of structure.

structure_len *structure length*
Length of this structure in bytes.

user_vars_id *user variables id*
A number that uniquely identifies, within this **user_vars_dc**, an instance of the **user_vars** structure.
[key=part_primary, db_index=clustered]

user_vars_dc *user variables domain*
Domain in which user_vars_id is unique.
[codelist=authorities, key=part_primary]

waveform_id *waveform id*

A number that uniquely refers, within this **waveform_dc**, to an instance of the waveform structure.

[key=part_foreign(1,waveform.waveform_id), db_delete=cascade, index_string=true, db_must_exist=true]

waveform_dc *waveform domain*

Domain in which waveform_id is unique.

[codelist=authorities, key=part_foreign(1,waveform.waveform_dc), db_delete=cascade, db_must_exist=true]

waveform_type *waveform structure*

Number of the structure type whose primary key the waveform_id refers to.

spare *for future use*

type_zero *type zero*

type_one *type one*

type_two *type two*

type_three *type three*

type_four *type four*

type_five *type five*

type_six *type six*

type_seven *type seven*

type_eight *type eight*

type_nine *type nine*

zero *variable zero*

one *variable one*

two *variable two*

three *variable three*

four *variable four*

five *variable five*

six *variable six*

seven *variable seven*

eight *variable eight*

nine *variable nine*

comment_id *comment id*

A number that uniquely refers, within this comment_dc, to an instance of the **comment** structure. Comments are generally not searchable because they are not of standard format. Thus it is recommended that comments not be heavily used.

[key=part_foreign(2,comment.comment_id), db_delete=nullify]

comment_dc *comment domain*

Domain in which comment_id is unique.

[codelist=authorities, key=part_foreign(2,comment.comment_dc), db_delete=nullify]

SEE ALSO

NAME

vel_layer_data – information about a horizontal layer in a crustal velocity model

C_SYNOPSIS

```
typedef struct {
        FLOAT4          depth_2_top;
        FLOAT4          p_vel_top;
        FLOAT4          s_vel_top;
        FLOAT4          depth_2_base;
        FLOAT4          p_vel_base;
        FLOAT4          s_vel_base;
        CODE2           vel_function;
        CODE2           dens_function;
        FLOAT4          density;
        FLOAT4          attenuation;
        INT4            spare;
} SUDS_VEL_LAYER_DATA;

#define VEL_LAYER_DATAS     119L
```

DESCRIPTION

Description of a horizontal layer in a crustal velocity model. A number (**vel_model.data_length**) of these structures follow the **vel_model** structure.
[data_only=VEL_MODELS]

MEMBERS

depth_2_top *depth to top*
Depth in kilometers to the top of this layer.

p_vel_top *p velocity top*
P wave velocity at the top of this layer.

s_vel_top *s velocity _top*
S wave velocity at the top of this layer.

depth_2_base *depth to base*
Depth in kilometers to the top of this layer.

p_vel_base *p velocity base*
P wave velocity at the base of this layer.

s_vel_base *s velocity base*
S wave velocity at the base of this layer.

vel_function *velocity function*
Velocity function within this layer, such as linear increase, exponential increase, etc.
[codelist=functions]

dens_function *density function*
Density function within this layer, such as linear increase, exponential increase, etc.
[codelist=functions]

density *density*
Density at the top of the layer.

attenuation *attenuation*
Attenuation of **Q** of the layer.

spare *for future use*

SEE ALSO

vel_model(4)

NAME

vel_model – information about a horizontally flat-layered crustal velocity model

C_SYNOPSIS

```
typedef struct {
        FIXED       structure_type;
        FIXED       structure_len;
        LABEL       vel_model_id;
        DOMAIN      vel_model_dc;
        LATIT       A_latitude;
        LONGIT      A_longitude;
        LATIT       B_latitude;
        LONGIT      B_longitude;
        CODE1       model_type;
        CHAR        spare_char;
        INT2        spare;
        FIXED       len_model_n;
        STRING      model_name[16];
        ST_TIME     from_time;
        AUTHOR      authority;
        CODE4       data_type;
        INT4        data_length;
        REFERS2     comment_id;
        DOMAIN      comment_dc;
} SUDS_VEL_MODEL;

#define VEL_MODELS          118L
```

DESCRIPTION

Description of a horizontally flat-layered crustal velocity model. If this model is based on a profile, then the profile extends from point A to point B. If this model applies to a rectangular area, points A and B are opposite corners of the rectangle (see **model_type**). This structure must be followed by a number of **vel_layer_data** structures in increasing order of depth and with all layers specified.

[permissions="siu_si_s_s"]

MEMBERS

structure_type *structure type*

Define number of this type of structure.

structure_len *structure length*

Length of this structure in bytes.

vel_model_id *velocity model id*

A number that uniquely refers, within this **vel_model_dc**, to an instance of the **vel_model** structure.

[key=part_primary, db_index=clustered]

vel_model_dc *velocity model domain*

Domain in which vel_model_id is unique.

[codelist=authorities, key=part_primary]

A_latitude *latitude point A*

Latitude of point A, south is negative.

A_longitude *longitude point A*

Longitude of point A, west is negative.

B_latitude *latitude point B*

Latitude of point B, south is negative.

B_longitude *longitude point B*
Longitude of point B, west is negative.

model_type *model type*
p=profile between points A and B, a=rectangular area with opposite corners at A and B.
[codelist=model_types]

spare_char *for future use*

spare *for future use*

len_model_n *length model name*
The maximum space reserved for the model name, i.e. 16. Actual string can only contain 15 characters to allow for the NULL byte.

model_name *model name*
Name of this model.
[index_string=true]

from_time *from time*
Time this model was created.

authority *model authority*
Who determined this crustal structure.
[codelist=authorities]

data_type *data type*
Type of structure that follows this structure. Normally VEL_LAYER_DATAS.
[sets_data_type=true, codelist=data_types]

data_length *number of points*
Number of structures of type **vel_layer_data** that follow this structure.
[sets_data_length=true, default_value=0]

comment_id *comment id*
A number that uniquely refers, within this comment_dc, to an instance of the **comment** structure. Comments are generally not searchable because they are not of standard format. Thus it is recommended that comments not be heavily used.
[key=part_foreign(1,comment.comment_id), db_delete=nullify]

comment_dc *comment domain*
Domain in which comment_id is unique.
[codelist=authorities, key=part_foreign(1,comment.comment_dc), db_delete=nullify]

SEE ALSO

vel_layer_data(4)

NAME

waveform – information about a waveform for a single station component

C_SYNOPSIS

```
typedef struct {
        FIXED          structure_type;
        FIXED          structure_len;
        LABEL          waveform_id;
        DOMAIN         waveform_dc;
        REFERS2        signal_path_id;
        DOMAIN         signal_path_dc;
        REFERS2        mux_waveform_id;
        DOMAIN         mux_waveform_dc;
        REFERS2        data_group_id;
        DOMAIN         data_group_dc;
        REFERS2        calibration_id;
        DOMAIN         calibration_dc;
        FIXED          len_signal_n;
        STRING         signal_name[20];
        MS_TIME        from_time;
        MS_TIME        thru_time;
        MS_TIME        nominal_time;
        INT2           local_time;
        CODE1          resolution;
        CODE1          data_units;
        AUTHOR         digitized_by;
        INT4           spare_a;
        FLOAT4         nom_dig_rate;
        FLOAT4         prec_dig_rate;
        FLOAT4         min_val_data;
        FLOAT4         max_val_data;
        FLOAT4         average_noise;
        FLOAT4         dc_removed;
        INT4           num_pos_clip;
        INT4           num_neg_clip;
        INT2           num_spikes;
        INT2           num_glitches;
        FLOAT4         weight;
        CODE1          time_source;
        CODE1          gain_ranged;
        CODE1          signal_type;
        CODE1          filter_code;
        CODE1          compression;
        CODE1          time_status;
        CHAR           spare_b;
        CHAR           spare_c;
        INT4           file_offset;
        CODE4          data_type;
        INT4           data_length;
        REFERS2        processing_id;
        DOMAIN         processing_dc;
        REFERS2        comment_id;
        DOMAIN         comment_dc;
} SUDS_WAVEFORM;
```

#define WAVEFORMS 107L

DESCRIPTION

Descriptive information about a seismic waveform. Normally followed by the waveform. The waveform is an array of numbers of binary storage type **data_type** in units of **data_units** and length sizeof(data_type) times **data_length**.

[permissions="siud_siu_s_s"]

MEMBERS

structure_type *structure type*

Define number of this type of structure.

structure_len *structure length*

Length of this structure in bytes.

waveform_id *waveform id*

A number that uniquely refers, within this **waveform_dc**, to an instance of the **waveform** structure.

[key=part_primary, db_index=clustered]

waveform_dc *waveform domain*

Domain in which waveform_id is unique.

[codelist=authorities, key=part_primary]

signal_path_id *signal path id*

A number that uniquely refers, within this **domain_code**, to one instance of the **signal_path** structure representing a total signal path from a particular sensor to its recorder. In some cases the same sensor and recorder may be connected by separate paths.

[key=part_foreign(1,signal_path.signal_path_id), db_delete=restrict, db_must_exist=true]

signal_path_dc *signal path domain*

Domain in which signal_path_id is unique.

[codelist=authorities, key=part_foreign(1,signal_path.signal_path_dc), db_delete=restrict, db_must_exist=true]

mux_waveform_id *mux data id*

A number that uniquely refers, within this mux_data_dc, to an instance of the **mux_data** structure. This number is typically assigned by the detecting machine for this event trigger. This number must be unique for a detector, specified by a **recorder_id** that is unique within a recorder_dc. For consistency, this number should, if possible, represent the approximate time of writing the data represented as **ST_TIME** (i.e. seconds since beginning of Jan 1, 1970).

[key=part_foreign(2,mux_waveform.mux_waveform_id), db_delete=nullify]

mux_waveform_dc *mux waveform domain*

Domain in which mux_waveform_id is unique.

[codelist=authorities, key=part_foreign(2,mux_waveform.mux_waveform_dc), db_delete=nullify]

data_group_id *data group id*

A number identifying a collection of waveform data. The number is assigned by an authority when many waveforms are associated into a group that normally contains all the waveforms for one earthquake. The value must be unique within a domain and is assumed to be of type **ST_TIME** (i.e. seconds since the beginning of Jan, 1970) representing a time at or near the time of the first samples of much of the data. In practice this number would typically be assigned when the data from the primary network detector are demultiplexed. Then as data from other detectors are added, they are assigned this data_group_id. **waveform** structures and their associated waveforms for all station components within a data group will usually be stored together either in a file or in a directory with the name based on the ASCII representation of this time. The ascii string is of the form: YYMMDD.HHMMSS, where YY is the year (00-99), MM is the month (01-12), DD is the day(01-31), HH is the hour (00-23), MM is the minute (00-59), and SS the second (00-59), in universal (GMT) time. For example

910824.123600
[key=part_foreign(3,data_group.data_group_id), db_delete=cascade]

data_group_dc *data group domain*
Domain in which data_group_id is unique.
[codelist=authorities, key=part_foreign(3,data_group.data_group_dc), db_delete=cascade]

calibration_id *calibration id*
Key to description of a calibration signal contained in this waveform.
[key=part_foreign(4,calibration.calibration_id), db_delete=nullify]

calibration_dc *calibration domain*
Domain in which calibration_id is unique.
[codelist=authorities, key=part_foreign(4,calibration.calibration_dc)]

len_signal_n *len signal name*
The maximum space reserved for the signal name, i.e. 20. Actual string can only contain 19 characters to allow for the NULL byte.

signal_name *signal name*
Name of a sensor component whose data are transmitted along a specific path and recorded on a particular recorder. Name is expected to be of the form network_station_CSBGP where the network is the abbreviation (part of the authority string preceeding the colon, 5 or less characters) for the **signal_path.network** code, station is **signal_path.station_name** (7 or less characters), C is the **signal_path.component_type** code (usually v, n, or e), S is the **signal_path.sensor_type** code, B is the **signal_path.band_type** code, G is the **signal_path.gain_type**, and P is the **signal_path.path_type** in only those stations where the same component may be recorded on two or more different recorders or transmitted over different paths.
[index_string=true]

from_time *beginning time*
GMT time of the first sample in the waveform including all clock corrections.

thru_time *ending time*
GMT time of the last sample in the waveform including all clock corrections.

nominal_time *nominal time*
GMT time of the first sample in the waveform without any clock corrections. This value is set when the time code is associated with the waveform usually in the demultiplexing program and should never be changed. The reason for keeping this time is to allow checking and changing all time corrections applied to it.

local_time *local time*
Minutes to add to GMT to get local time.

resolution *bits of resolution*
Number of bits of resolution of the waveform when it is originally digitized.
[codelist=resolutions]

data_units *data units type*
Type of data units: d=digital counts, g=ground motion in nanometers, n=nanometers/sec, N=nanometers/sec/sec), v=millivolts, V=volts.
[codelist=units_types]

digitized_by *digitized by*
Group or person collecting this data.
[codelist=authorities]

spare_a *for future use*

nom_dig_rate *samples per second*

Nominal rate of digitization in samples per second.

prec_dig_rate *calculated samples/sec*
Calculated rate of digitization in samples per second.

min_val_data *minimum value*
Smallest data value in the waveform (datatype s,l,f only).

max_val_data *maximum value*
Largest data value in the waveform (datatype s,l,f only).

average_noise *average noise*
Average value of the first 200 samples of the waveform (datatype s,l,f only).

dc_removed *dc removed*
DC offset that has been removed from the data.

num_pos_clip *num + clipped samples*

num_neg_clip *num - clipped samples*

num_spikes *number of spikes*

num_glitches *number of glitches*

weight *weight*
Weight used to sum this waveform.

time_source *source of time*
i=IRIG local clock, I=IRIG radio time, s=IRIG satellite time, S=BCD encoded on waveform satellite time, l=local clock, r=WWV, WWVH, b=WWVB, o=other.
[codelist=time_codes]

gain_ranged *gain ranging*
g=waveform may include gain ranging changes, c=gain ranging has been corrected for, and NOCHAR=gain ranging is never used for this component.
[codelist=amplitude_types]

signal_type *type of signal*
Type of triggered signal or ’C’ for continuous data.
[codelist=event_types]

filter_code *filter code*
Indicates that waveform has been filtered or decimated since creation and implies the existence of one or more **filter** structures associating the appropriate **response** structures that specify the filter response. f=filtered
[codelist=filter_types]

compression *compression algorithm*
Type of algorithm used to compress the data following this structure. NOCHAR means the data is not compressed.
[codelist=compression_types]

time_status *time correction status*
Status of the time correction. Time corrections are typically not a problem now, but when they are, a description of what corrections have been applied should be put in the comment associated with this structure.
[codelist=time_corrects]

spare_b *for future use*

spare_c *for future use*

file_offset *offset in file*
Number of bytes from beginning of file to the beginning of the **structure_tag** announcing this

instance of this structure. This offset is for use in indexing waveform files and will be valid only in limited circumstances. In most implementations of databases for **SUDS**, the **waveform** structures will be stored in the database without the data and the **waveform** structures and data will be stored in files often located on read-only mass storage systems. The file name and path are based on the **data_group_id** and **data_group_dc** and the location in the file is this offset. Thus the database software is able to access the waveform directly and return it to the user as if it existed within the database.

[sets_data_offset=true]

data_type *data storage type*

An integer representing the type of data that follows this structure. If the integer is negative, it refers to **variable_tab.define_num**. If the integer is positive, it refers to **structure_tab.struct_number**.

[sets_data_type=true, codelist=data_types]

data_length *number of samples*

Number of samples of type **data_type** in the waveform.

[sets_data_length=true, default_value=0]

processing_id *processing id*

[key=part_foreign(5,processing.processing_id), db_delete=nullify]

processing_dc *processing domain*

Domain in which processing_id is unique.

[codelist=authorities, key=part_foreign(5,processing.processing_dc), db_delete=nullify]

comment_id *comment id*

A number that uniquely refers, within this comment_dc, to an instance of the **comment** structure. Comments are generally not searchable because they are not of standard format. Thus it is recommended that comments not be heavily used.

[key=part_foreign(6,comment.comment_id), db_delete=nullify]

comment_dc *comment domain*

Domain in which comment_id is unique.

[codelist=authorities, key=part_foreign(6,comment.comment_dc), db_delete=nullify]

SEE ALSO

BUGS

Detector uptime?

Flag for triggered on?

SUDS

Chapter 5: PC-SUDS Structure Descriptions

This page left blank intentionally.

NAME

st_intro_pc – introduction to PC_SUDS structures and codes

OVERVIEW

The initial version of SUDS was adopted for use in the IASPEI Software Library edited by W. H. K. Lee for use on IBM-compatible personal computers and was first released in 1989. This software is available to thousands of people and is used throughout the world. This early version of SUDS has come to be known as PC_SUDS. PC_SUDS is substantially different from the new SUDS described in this manual. It is not machine independent, is not a database model, and does not handle as many types of seismological data. Many more advanced tools will be available for the new SUDS.

In order to make a direct and simple migration path from PC_SUDS to SUDS and in order to allow use of many new utility programs with existing PC_SUDS data, the SUDS input/output library has been programmed to recognize PC_SUDS data and to read it in a machine independent manner into structures in memory that have the same members as PC_SUDS structures, but that also have the members properly aligned for machine independence and include the **structure_type** and **structure_len** members that start all new SUDS structures. This allows the PC_SUDS structures to work with most SUDS subroutines and utilities. The conversion is done on input and on output so that on recording media and disks PC_SUDS data is still in PC_SUDS format. This allows PC_SUDS and SUDS data to exist together. **However, we strongly encourage you not to mix SUDS structures with PC_SUDS structures. This will lead to many problems.** If you have a great deal of data in PC_SUDS format, you may wish to use some of the new tools with that data. You should consider moving to the new SUDS whenever the tools you need are available and when you are working with 80386 and more advanced computers. A filter **pc2suds(1)** is provided to convert data from PC_SUDS to SUDS when needed.

VERSION

The descriptions on the following pages are based on the PC_SUDS documentation and include files for version 1.44 (23-Aug-1993).

INTERNAL FORMAT

The input library recognizes PC_SUDS structures when reading the **structtag** for PC_SUDS and the **structure_tag** for SUDS. Both start with the letter **S**, but the second character for PC_SUDS is a letter **6** while for SUDS the second character is typically the letter **x** or other letters listed in the code_list **computer_types**. In PC_SUDS the letter **6** is required by the PC_SUDS utilities. If a different letter is used, that data will not be correctly read by the PC_SUDS or the SUDS utilities.

The internal format of PC_SUDS structures is described in the following manual pages. The members **structure_type** and **structure_len** have been added since these are used by many utilities when handling SUDS structures. Members have also been added to assure alignment of members on a byte that is evenly divisible by their length, or in the case of strings, on a 4-byte boundary. These members that do not exist in the PC_SUDS structures all have names starting with **pad_**.

SUDS does not allow for nested structures or members that are arrays in order to be compatible with most database systems. The **statident** structure is contained within many PC_SUDS structures and is thus included member by member in the equivalent SUDS structures. Array members are given explicit names. These changes mean that existing PC_SUDS utilities can not use the new input/output library unless the references to these members are changed. However, existing PC_SUDS utilities and other programs will still continue to work when using PC_SUDS data written by the new SUDS library routines.

PROBLEMS

Association of structures that are related, for example, to a single event, is done in PC_SUDS by physical organization into files or directories. In SUDS association is done by relational primary and foreign keys (LABEL and REFERS2 member types). This adds some compexity to converting PC_SUDS to SUDS but has the benefit of being more explicit and being database compatible.

In C, ASCII strings are terminated by a null byte ( ). This byte is used whenever printing, copying, scanning a string, etc. In PC-SUDS network names are typically of length 4 and station names are of length 5. This means they can only have 3 and 4 characters respectively. However, much PC-SUDS data exists where someone has not so cleverly put 4 and 5 characters respectively in these names. Thus if after a name you get garbage characters, it means the program has to add the null byte or print the string character by character.

NODATA in PC_SUDS is defined as -32767. for all variable types. The IO library converts these values to suitable values in SUDS depending on type: short integer (NODATS=-32760), long integer (NODATL=-2147483640L), and floating point (NODATF=-1.7e+36). NOTIME in PC_SUDS is the same as MINTIME in SUDS.

LIST OF STRUCTURES

Structures defined in the **PC_SUDS** standard are as follows:

atodinfo	information on the A to D converter
calib	poles and zeros of calibration information
chanset	associate station/components into sets
chansetentry	associate station/components into sets
descriptrace	descriptive information about a seismic trace
detector	information on detector program being used
equipment	equipment making up a station/component
error	error matrix
evdescr	descriptive information about an event
eventsetting	settings for earthquake trigger system
feature	observed phase arrival time, amplitude, and period
focalmech	general information about a focal mechanism
instrument	instrument hardware settings, mainly PADS related
layers	velocity layers
loctrace	location of trace
moment	moment tensor information
muxdata	header for multiplexed data
origin	information about a specific solution for a given event
pc_calibration	calibration information for a station component
pc_comment	comment tag to be followed by the bytes of comment
pc_event	general information about an event
pc_terminator	structure to end a sequence of related structures
residual	calculated residuals for arrival times, magnitudes, etc.
stationcomp	generic station component information
structtag	structure to identify structures when archived together
timecorrection	time correction information
triggers	earthquake detector trigger statistics
trigsetting	settings for earthquake trigger system
velmodel	velocity model

The structures calib, calibration, equipment, error, evdescr, event, eventsetting, focalmech, layers, loctrace, moment, residual, and velmodel are not commonly used in PC_SUDS and we suggest that you do not use them.

PROPERTIES OF THE STRUCTURES

NAME	NUMBER	BYTES	MEMBERS	
atodinfo	29	24	10	
calib	23	16	4	data only structure
chanset	32	40	12	
chansetentry	33	32	10	data only structure
descriptrace	7	88	24	data may follow
detector	28	40	11	
equipment	4	96	28	
error	15	48	12	

evdescr	13	80	11	
eventsetting	27	48	15	
feature	10	64	23	
focalmech	16	40	12	
instrument	31	104	31	
layers	19	32	9	
loctrace	8	40	11	
moment	17	40	12	
muxdata	6	48	14	data may follow
origin	14	112	34	
pc_calibration	9	56	15	data may follow
pc_comment	20	16	6	
pc_event	12	32	14	
pc_terminator	3	16	5	
residual	11	64	19	
stationcomp	5	96	33	
structtag	2	16	6	data only structure
timecorrection	30	56	17	
triggers	25	48	15	
trigsetting	26	48	16	
velmodel	18	72	15	

REFERENCES

The following materials describe PC_SUDS and are in chronological order. Volumes in the IASPEI Software Library may be ordered from the Seismological Society of America, 201 Plaza Professional Building, El Cerrito, CA 94530, 510/525-5474, fax 510/525-7204.

Ward, Peter L., 1989, **SUDS: Seismic Unified Data System,** U.S. Geological Survey Open-File Report 89-188, 123 pages.

Lee, W. H. K., editor, 1989, **Toolbox for Seismic Data Acquisition, Processing, and Analysis**: IASPEI Software Library, Volume 1, 284 pages.

Lee, W. H. K., editor, 1990, **Toolbox for Plotting and Displaying Seismic and Other Data**: IASPEI Software Library, Volume 2, 207 pages.

Banfill, Robert, 1992, **SUDS, Seismic Unified Data System, Version 1.31,** Small Systems Support, Big Water, Utah, available from the IASPEI PC Working Group, send a request by FAX to 415/858-2599, 27 pages.

Lee, W. H. K., editor, 1993, **Digital Seismogram Analysis and Waveform Inversion**: IASPEI Software Library, Volume 3, 166 pages.

Scherbaum, Frank, and Johnson, James, 1992, **Programmable Interactive Toolbox for Seismological Analysis (PITSA)**: IASPEI Software Library, Volume 5, 269 pages.

Banfill, Robert, 1 December 1993, **PC-SUDS Utilities**, A collection of programs for routine processing of seismic data stored in the Seismic Unified Data System for DOS (PC-SUDS): IASPEI Software Library Supplement #1, 91 pages.

NAME

atodinfo – information on the A to D converter

SYNOPSIS

```
typedef    struct {
           FIXED        structure_type;
           FIXED        structure_len;
           INT2         base_address;
           INT2         device_id;
           UINT2        device_flags;
           INT2         extended_bufs;
           INT2         external_mux;
           CODE1        timing_source;
           CODE1        trigger_source;
           PAD4         pad_A;
} PCSUDS_ATODINFO;

#define PC_ATODINFO            29L
```

DESCRIPTION

Commonly used in PC-SUDS. Information about the analog to digital converter used on PC computers.

MEMBERS

structure_type *structure type*
Define number of this type of structure.

structure_len *structure length*
Length of this structure in bytes.

base_address *base address*
Base I/O address of this device.

device_id *device identifier*
[index_string=true]

device_flags *device flags*

extended_bufs # *of extended bufs*
Number of extended buffers used.

external_mux *external mux*
AtoD external mux control word.

timing_source *AtoD timing source*
AtoD timing source: i=internal, e=external.
[codelist=timings]

trigger_source *AtoD trigger source*
AtoD trigger source: i=internal, e=external.
[codelist=timings]

pad_A *padding*

SEE ALSO

NAME

pc_calibration – calibration information for a station component

SYNOPSIS

```
typedef struct {
        FIXED       structure_type;
        FIXED       structure_len;
        FIXED       len_netn_A;
        STRING      network[4];
        FIXED       len_st_nam_A;
        STRING      st_name[5];
        CODE1       component;
        CODE2       inst_type;
        FLOAT4      maxgain;
        FLOAT4      normaliz;
        ST_TIME     begint;
        ST_TIME     endt;
        PAD4        pad_G;
        CODE4       data_type;
        INT4        data_length;
} PCSUDS_CALIBRATION;

#define PC_CALIBRATION      9L
```

DESCRIPTION

Not commonly used in PC-SUDS. Calibration information about a seismometer. Initially designed to be compatible with the calibration information contained within the **AH** format developed at Lamont-Doherty Earth observatory. The PC-SUDS version contains the poles and zeros in an array of fixed length within the structure called **SUDS_CALIBR call[NOCALPTS]** where **NOCALPTS** is 30. This is not the way it is done in SUDS. Thus the IO library puts these data in **calib**(5) structures following this structure and and the **structure_properties**(2) routines return the correct type and length. The number of **calib** structures is almost always less than 30, with the remainder being filled with zeros or NODATA.

MEMBERS

structure_type *structure type*
    Define number of this type of structure.

structure_len *structure length*
    Length of this structure in bytes.

len_netn_A *len network name*

network *network name*

len_st_nam_A *len station name*

st_name *station name*
    [index_string=true]

component *component*
    [codelist=comps]

inst_type *instrument type*
    [codelist=inst_type]

maxgain *maximum gain*
    Maximum gain of the calibration curve.

normaliz *normalization factor*
    Factor to multiply standard calib by to make peak at given frequency=1.

begint *time effective*
Time this calibration becomes effective.

endt *end time effective*
Time this calibration is no longer effective.

pad_G *padding*

data_type *data storage type*
An integer representing the type of data that follows this structure. If the integer is negative, it refers to **variable_tab.define_num**. If the integer is positive, it refers to **structure_tab.struct_number**.
[sets_data_type=true, codelist=data_types]

data_length *number of samples*
Number of samples of type **data_type** in the waveform.
[sets_data_length=true, default_value=0]

SEE ALSO

NAME

calib – poles and zeros of calibration information

SYNOPSIS

```
typedef   struct {
          FLOAT4         pole_cr;
          FLOAT4         pole_ci;
          FLOAT4         zero_cr;
          FLOAT4         zero_ci;
} PCSUDS_CALIBR;

#define PC_CALIB            23L
#define   PC_NUMCALPTS      30L
```

DESCRIPTION

Not commonly used in PC-SUDS. **Data only** following the PC_SUDS **calibration** structure.

MEMBERS

pole_cr *pole real*

pole_ci *pole imaginary*

zero_cr *zero real*

zero_ci *zero imaginary*

SEE ALSO

NAME

chanset – associate station/components into sets

SYNOPSIS

```
typedef    struct {
           FIXED         structure_type;
           FIXED         structure_len;
           INT2          type;
           INT2          entries;
           FIXED         len_netn_B;
           STRING        network[4];
           FIXED         len_set_nam_B;
           STRING        name[5];
           CHAR          pc_pad;
           PAD2          pad_C;
           ST_TIME       active;
           ST_TIME       inactive;
} PCSUDS_CHANSET;

#define PC_CHANSET                    32L
```

DESCRIPTION

Commonly used in PC-SUDS. Associates station/components into sets. This structure is followed by data consisting of **entries** instances of **chansetentry(pc)** structures.

MEMBERS

structure_type *structure type*
Define number of this type of structure.

structure_len *structure length*
Length of this structure in bytes.

type *set type*
Set type; 0=single channel(s), 1=orthogonal vector.

entries *entries in set*
Number of entries in set (these follow as data).
[sets_data_length=true, default_value=0]

len_netn_B *len network name*

network *network name*

len_set_nam_B *len set name*

name *set name*
[index_string=true]

pc_pad *pc padding*
Padding inserted by 80x86 computers to align integers on an even byte boundary.

pad_C *padding*

active *active time*
Set is defined after this time.

inactive *inactive time*
Set is not defined after this time.

SEE ALSO

chansetentry(pc)

AUTHOR

Robert Banfill, August 23, 1993.

NAME

chansetentry – associate station/components into sets

SYNOPSIS

```
typedef   struct {
          INT4          inst_num;
          INT2          stream_num;
          INT2          chan_num;
          FIXED         len_netn_C;
          STRING        network[4];
          FIXED         len_st_n_C;
          STRING        st_name[5];
          CODE1         component;
          CODE2         inst_type;
          PAD4          pad_U;
} PCSUDS_CHANSETENTRY;

#define PC_CHANSETENTRY          33L
```

DESCRIPTION

Commonly used in PC-SUDS. **Data only** following the CHANSET structure.

MEMBERS

inst_num *serial number*
Instrument serial number.

stream_num *stream number*
Stream of instrument.

chan_num *channel number*
Channel of stream.

len_netn_C *len network name*

network *network*

len_st_n_C *len station name*

st_name *station name*

component *component*
[codelist=comps]

inst_type *instrument type*
[codelist=inst_type]

pad_U *padding*

SEE ALSO

chansetentry(pc)

AUTHOR

Robert Banfill, August 23, 1993.

NAME

pc_comment – comment tag to be followed by the bytes of comment

SYNOPSIS

```
typedef   struct {
          FIXED         structure_type;
          FIXED         structure_len;
          INT2          refer;
          INT2          item;
          INT2          data_length;
          INT2          unused;
} PCSUDS_COMMENT;

#define PC_COMMENT              20L
```

DESCRIPTION

Commonly used in PC-SUDS. Comment about a structure. Followed by the comment as data in ASCII.

MEMBERS

structure_type *structure type*

Define number of this type of structure.

structure_len *structure length*

Length of this structure in bytes.

refer *structure identifier*

Type of structure that this comment refers to.
[index_string=true]

item *member number*

Number of member, counting from 0 after structure_len, that this comment refers to.

data_length *comment length*

Length of comment in bytes.
[sets_data_length=true]

unused *for future use*

SEE ALSO

NAME

descriptrace – descriptive information about a seismic trace

SYNOPSIS

```
typedef    struct {
           FIXED          structure_type;
           FIXED          structure_len;
           FIXED          len_netn_D;
           STRING         network[4];
           FIXED          len_st_nam_D;
           STRING         st_name[5];
           CODE1          component;
           CODE2          inst_type;
           PAD4           pad_D;
           MS_TIME        begintime;
           INT2           localtime;
           CODE1          datatype;
           CODE1          descriptor;
           INT2           digi_by;
           INT2           processed;
           INT4           data_length;
           FLOAT4         rate;
           FLOAT4         mindata;
           FLOAT4         maxdata;
           FLOAT4         avenoise;
           INT4           numclip;
           MS_TIME        time_correct;
           FLOAT4         rate_correct;
           PAD4           pad_V;
} PCSUDS_DESCRIPTRACE;

#define PC_DESCRIPTRACE        7L
```

DESCRIPTION

Very commonly used in PC-SUDS. Description of the properties of a seismic trace or waveform that follows this structure in the form of **datatype** and the length of **length**.

MEMBERS

structure_type *structure type*
Define number of this type of structure.

structure_len *structure length*
Length of this structure in bytes.

len_netn_D *len network name*

network *network name*

len_st_nam_D *len station name*

st_name *station name*
[index_string=true]

component *component*
[codelist=comps]

inst_type *instrument type*
[codelist=inst_type]

pad_D *padding*

begintime *initial sample time*
Time of first data sample.

localtime *local time diff*
Minutes to add to GMT to get local time.

datatype *data type*
[sets_data_type=true, codelist=datatyp]

descriptor *data descriptor*
[codelist=descript]

digi_by *digitized by*
Agency code who digitized record.

processed *processed by*
Processing done on this waveform.

data_length *number of samples*
Number of samples in trace.
[sets_data_length=true]

rate *samples per second*

mindata *minimum data value*

maxdata *maximum data value*

avenoise *average noise*
Average value of first 200 samples.

numclip *num clipped samples*
Number of clipped datapoints.

time_correct *time correction*
Time correction to be added to begintime.

rate_correct *rate correction*
Rate correction to be added to rate.

pad_V *padding*

SEE ALSO

NAME

detector – information on detector program being used

SYNOPSIS

```
typedef    struct {
           FIXED        structure_type;
           FIXED        structure_len;
           CODE1        dalgorithm;
           CODE1        event_type;
           PAD2         pad_E;
           FIXED        len_net_node;
           STRING       net_node_id[10];
           PAD2         pad_F;
           FLOAT4       versionnum;
           INT4         event_number;
           INT4         spareL;
} PCSUDS_DETECTOR;

#define PC_DETECTOR             28L
```

DESCRIPTION

Commonly used in PC-SUDS. Information about the online dection program being used.

MEMBERS

structure_type *structure type*
Define number of this type of structure.

structure_len *structure length*
Length of this structure in bytes.

dalgorithm *algorithm*
Triggering algorithm: x=xdetect, m=mdetect.
[codelist=algorith]

event_type *event type*
[codelist=eventtyp]

pad_E *padding*

len_net_node *length net_node*

net_node_id *network node id*
Network node identification.
[index_string=true]

pad_F *padding*

versionnum *version number*
Software version number.

event_number *event number*
Unique event number assigned locally.

spareL *for future use*

SEE ALSO

NAME

equipment – equipment making up a station/component

SYNOPSIS

```
typedef   struct {
          FIXED        structure_type;
          FIXED        structure_len;
          FIXED        len_t_netw;
          STRING       t_network[4];
          FIXED        len_st_nam;
          STRING       t_st_name[5];
          CODE1        t_component;
          CODE2        t_inst_type;
          FIXED        len_p_netw;
          STRING       p_network[4];
          FIXED        len_p_st_nam;
          STRING       p_st_name[5];
          CODE1        p_component;
          CODE2        p_inst_type;
          FIXED        len_n_netw;
          STRING       n_network[4];
          FIXED        len_n_st_nam;
          STRING       n_st_name[5];
          CODE1        n_component;
          CODE2        n_inst_type;
          FIXED        len_serial;
          STRING       serial[8];
          CODE2        model;
          INT2         knob1;
          INT2         knob2;
          CODE2        reason;
          FLOAT4       frequency;
          ST_TIME      effective;
} PCSUDS_EQUIPMENT;

#define PC_EQUIPMENT           4L
```

DESCRIPTION

Not commonly used in PC-SUDS. Information about a piece of equipment and what it is connected to.

MEMBERS

structure_type *structure type*
: Define number of this type of structure.

structure_len *structure length*
: Length of this structure in bytes.

len_t_netw *len network name*

t_network *network name*
: Network name for this station component.

len_st_nam *len station name*

t_st_name *station name*
: Station name for this station component.
[index_string=true]

t_component *component*
: Component for this station component.

[codelist=comps]

t_inst_type *instrument type*
Instrument type for this station component.
[codelist=inst_type]

len_p_netw *len network name*

p_network *network name*
Network name for the previous station component.

len_p_st_nam *len station name*

p_st_name *station name*
Station name for the previous station component.

p_component *component*
Component for the previous station component.
[codelist=comps]

p_inst_type *instrument type*
Instrument type for the previous station component.
[codelist=inst_type]

len_n_netw *len network name*

n_network *network name*
Network name for the next station component.

len_n_st_nam *len station name*

n_st_name *station name*
Station name for the next station component.

n_component *component*
Component for the next station component.
[codelist=comps]

n_inst_type *instrument type*
Instrument type for the next station component.
[codelist=inst_type]

len_serial *len serial*

serial *serial number*

model *equipment model*
[codelist=equip_model]

knob1 *knob 1 setting*
Knob setting or series resistor value of Lpad.

knob2 *knob 2 setting*
Knob setting or shunt resistor value of Lpad.

reason *repair reason*
Reason change was made.
[codelist=equip_reason]

frequency *frequency*
Sensor corner frequency, vco frequency, transmitter frequency, etc.

effective *date effective*
Date/time these values became effective.

SEE ALSO

NAME

error – error matrix

SYNOPSIS

```
typedef   struct {
          FIXED          structure_type;
          FIXED          structure_len;
          FLOAT4         covarr0;
          FLOAT4         covarr1;
          FLOAT4         covarr2;
          FLOAT4         covarr3;
          FLOAT4         covarr4;
          FLOAT4         covarr5;
          FLOAT4         covarr6;
          FLOAT4         covarr7;
          FLOAT4         covarr8;
          FLOAT4         covarr9;
} PCSUDS_ERROR;

#define PC_ERROR                 15L
```

DESCRIPTION

Not commonly used in PC-SUDS. Covariance error matrix for an origin.

MEMBERS

structure_type *structure type*

Define number of this type of structure.

structure_len *structure length*

Length of this structure in bytes.

covarr0 *covariance 0*

[index_string=true]

covarr1 *covariance 1*

covarr2 *covariance 2*

covarr3 *covariance 3*

covarr4 *covariance 4*

covarr5 *covariance 5*

covarr6 *covariance 6*

covarr7 *covariance 7*

covarr8 *covariance 8*

covarr9 *covariance 9*

SEE ALSO

origin(pc)

NAME

evdescr – descriptive information about an event

SYNOPSIS

```
typedef struct {
        FIXED       structure_type;
        FIXED       structure_len;
        FIXED       len_eqname;
        STRING      eqname[20];
        FIXED       len_country;
        STRING      country[16];
        FIXED       len_state;
        STRING      state[16];
        INT2        localtime;
        INT2        spareB;
        PAD4        pad_T;
} PCSUDS_EVDESCR;

#define PC_EVDESCRIPT       13L
```

DESCRIPTION

Not commonly used in PC-SUDS. Name and location of an event.

MEMBERS

structure_type *structure type*
Define number of this type of structure.

structure_len *structure length*
Length of this structure in bytes.

len_eqname *len eq name*

eqname *earthquake name*
Popular name used to refer to this earthquake.
[index_string=true]

len_country *len county*

country *county*
Country of earthquake.

len_state *len state*

state *state*
State, province or other political subdivision.

localtime *local time diff*
Hours to add to GMT to get local time.

spareB *for future use*

pad_T *padding*

SEE ALSO

NAME

pc_event – general information about an event

SYNOPSIS

```
typedef    struct {
           FIXED           structure_type;
           FIXED           structure_len;
           CODE2           authority;
           PAD2            pad_H;
           INT4            number;
           INT2            felt;
           CHAR            mintensity;
           CODE1           ev_type;
           CODE1           tectonism;
           CODE1           waterwave;
           CODE1           mechanism;
           CODE1           medium;
           FLOAT4          size;
           PAD4            pad_I;
} PCSUDS_EVENT;

#define PC_EVENT                        12L
```

DESCRIPTION

Not commonly used in PC-SUDS. Information about an event.

MEMBERS

structure_type *structure type*
Define number of this type of structure.

structure_len *structure length*
Length of this structure in bytes.

authority *organization*
Organization processing the data.
[codelist=authority]

pad_H *padding*

number *event number*
Unique event number assigned by organization.
[index_string=true]

felt *number felt reps*
Number of felt reports.

mintensity *MM intensity*
Maximum Modified Mercali Intensity.
[allow_char="0-9abc"]

ev_type *event type*
Type of event.
[codelist=eventtyp]

tectonism *tectonism*
Observed tectonism: uplift, subsidence, etc.
[codelist=tecton]

waterwave *waterwaves*
Observed waterwaves: seiche, tsunami, etc.
[codelist=waterwave]

mechanism *mechanism type*
Type of focal mechanism.
[codelist=mechan]

medium *explosive medium*
Medium containing explosion or event.
[codelist=medium]

size *magnitude or lbs TNT*
Magnitude or pounds TNT for explosions.

pad_I *padding*

SEE ALSO

NAME

eventsetting – settings for earthquake trigger system

SYNOPSIS

```
typedef   struct {
          FIXED        structure_type;
          FIXED        structure_len;
          FIXED        len_netnw;
          STRING       netwname[4];
          MS_TIME      beginttime;
          INT2         const1;
          INT2         const2;
          INT2         threshold;
          INT2         const3;
          FLOAT4       minduration;
          FLOAT4       maxduration;
          CODE1        algorithm;
          CHAR         spareK;
          INT2         spareI;
          PAD4         pad_J;
} PCSUDS_EVENTSETTING;

#define PC_EVENTSETTING        27L
```

DESCRIPTION

Not commonly used in PC-SUDS. Settings for an earthquake detection program.

MEMBERS

structure_type *structure type*
Define number of this type of structure.

structure_len *structure length*
Length of this structure in bytes.

len_netnw *len network name*

netwname *network name*
[index_string=true]

beginttime *begin time*
Time these values in effect.

const1 *constant 1*

const2 *constant 2*

threshold *threshold*

const3 *constant 3*

minduration *min duration*
Minimum duration for event.

maxduration *max duration*
Maximum duration for event.

algorithm *algorithm*
Triggering algorithm: x=xdetect, m=mdetect, e=eqdetect.
[codelist=algorith]

spareK *for future use*

spareI *for future use*

pad_J *padding*

SEE ALSO

NAME

feature – observed phase arrival time, amplitude, and period

SYNOPSIS

```
typedef struct {
        FIXED      structure_type;
        FIXED      structure_len;
        FIXED      len_netn_E;
        STRING     network[4];
        FIXED      len_st_nam_E;
        STRING     st_name[5];
        CODE1      component;
        CODE2      inst_type;
        CODE2      obs_phase;
        CODE1      onset;
        CODE1      direction;
        INT2       sig_noise;
        CODE1      data_source;
        CODE1      tim_qual;
        CODE1      amp_qual;
        CODE1      ampunits;
        INT2       gain_range;
        MS_TIME    pick_time;
        FLOAT4     amplitude;
        FLOAT4     period;
        ST_TIME    time_of_pick;
        CODE2      pick_authority;
        INT2       pick_reader;
} PCSUDS_FEATURE;

#define PC_FEATURE            10L
```

DESCRIPTION

Commonly used in PC-SUDS. A feature such as a phase picked from a waveform.

MEMBERS

structure_type *structure type*
Define number of this type of structure.

structure_len *structure length*
Length of this structure in bytes.

len_netn_E *len network name*

network *network name*

len_st_nam_E *len stat name*

st_name *station name*
[index_string=true]

component *component*
[codelist=comps]

inst_type *instrument type*
[codelist=inst_type]

obs_phase *observed phase*
Observed phase code.
[codelist=feat_phase]

onset *onset type*
Wave onset descriptor, i or e.
[codelist=onsetz]

direction *first motion*
[codelist=firstm]

sig_noise *signal to noise*
Ratio of amplitude of the first peak or trough to the noise.

data_source *data source*
[codelist=datasrc]

tim_qual *timing quality*
Timing quality assigned by the analyst.
[codelist=timequal]

amp_qual *amplitude quality*
Amplitude quality assigned by the analyst.
[codelist=timequal]

ampunits *amplitude units*
Units amplitude measured in.
[codelist=ampunits]

gain_range *gain range*
1 or gain multiplier if gain range in effect.

pick_time *phase time*
Phase time, x value where pick was made.

amplitude *amplitude*
Peak-to-peak amplitude of phase.

period *period*
Period of waveform measured.

time_of_pick *time picked*
Time this pick was made.

pick_authority *organization picking*
Organization processing the data.
[codelist=authority]

pick_reader *person picking*
Person processing the data.

SEE ALSO

NAME

focalmech – general information about a focal mechanism

SYNOPSIS

```
typedef   struct {
          FIXED           structure_type;
          FIXED           structure_len;
          FLOAT4          astrike;
          FLOAT4          adip;
          FLOAT4          arake;
          FLOAT4          bstrike;
          FLOAT4          bdip;
          FLOAT4          brake;
          CHAR            prefplane;
          CHAR            spareC;
          INT2            spareD;
          PAD4            pad_K;
} PCSUDS_FOCALMECH;

#define PC_FOCALMECH      16L
```

DESCRIPTION

Not commonly used in PC-SUDS.

MEMBERS

structure_type *structure type*

Define number of this type of structure.

structure_len *structure length*

Length of this structure in bytes.

astrike *strike of a*

[index_string=true]

adip *dip of a*

arake *rake of a*

bstrike *strike of b*

bdip *dip of b*

brake *rake of b*

prefplane *preferred plane*

Preferred plane a or b.

[allow_char="ab"]

spareC *for future use*

spareD *for future use*

pad_K *padding*

SEE ALSO

NAME

instrument – instrument hardware settings, mainly PADS related

SYNOPSIS

```
typedef   struct {
          FIXED        structure_type;
          FIXED        structure_len;
          FIXED        len_netn_F;
          STRING       network[4];
          FIXED        len_st_nam_F;
          STRING       st_name[5];
          CODE1        component;
          CODE2        inst_type;
          INT2         in_serial;
          INT2         comps;
          INT2         channel_num;
          CODE1        sens_type;
          CODE1        datatype;
          INT4         void_samp;
          FLOAT4       dig_con;
          FLOAT4       aa_corner;
          FLOAT4       aa_poles;
          FLOAT4       nat_freq;
          FLOAT4       damping;
          FLOAT4       mot_con;
          FLOAT4       gain;
          FLOAT4       local_x;
          FLOAT4       local_y;
          FLOAT4       local_z;
          ST_TIME      effective;
          FLOAT4       pre_event;
          INT2         trig_num;
          PAD2         pad_X;
          FIXED        len_study;
          STRING       study[6];
          INT2         sn_serial;
} PCSUDS_INSTRUMENT;

#define PC_INSTRUMENT          31L
```

DESCRIPTION

Very commonly used in PC-SUDS.

MEMBERS

structure_type *structure type*

Define number of this type of structure.

structure_len *structure length*

Length of this structure in bytes.

len_netn_F *len net name*

network *network name*

len_st_nam_F *len stat name*

st_name *station name*

[index_string=true]

component *component*

[codelist=comps]

inst_type *instrument type*
[codelist=inst_type]

in_serial *serial number*
Instrument serial number.

comps *num components*
Number of components recorded by instrument.

channel_num *channel number*
Actual channel number on recorder.

sens_type *sensor type*
[codelist=sensor]

datatype *data type*
[codelist=datatyp]

void_samp *void value*
Invalid or void sample value.

dig_con *digital counts*
Digitizing constant (counts / volt).

aa_corner *alias corner*
Anti-alias filter corner frequency (Hz).

aa_poles *alias poles*
Anti-alias filter poles.

nat_freq *natural freq*
Transducer natural frequency (Hz).

damping *damping*
Transducer damping coefficient.

mot_con *motor constant*
Transducer motion constant (volts / GMU).

gain *gain*
Amplifier gain (dB).

local_x *local X*
Local coordinate X (meters).

local_y *local Y*
Local coordinate Y (meters).

local_z *local Z*
Local coordinate Z (meters).

effective *time effective*
Time these setting took effect.

pre_event *pre event time*
Pre-event length (IST+pre_event=trigger time) in seconds.

trig_num *trigger number*
Trigger number on instrument.

pad_X *padding*

len_study *len study name*

study *name of study*
Study name, used to insure unique station name.

sn_serial *serial number*
Sensor serial number.

SEE ALSO

AUTHOR

Robert Banfill, January 1991

NAME

layers – velocity layers

SYNOPSIS

```
typedef struct {
        FIXED           structure_type;
        FIXED           structure_len;
        FLOAT4          thickness;
        FLOAT4          pveltop;
        FLOAT4          pvelbase;
        FLOAT4          sveltop;
        FLOAT4          svelbase;
        CODE2           function;
        INT2            spareF;
} PCSUDS_LAYERS;

#define PC_LAYERS                19L
```

DESCRIPTION

Not commonly used in PC-SUDS. Thickness and P-wave and S-wave velocities for flat-layered velocity models. A number of these structures must follow a **velmodel** structure.

MEMBERS

structure_type *structure type*
Define number of this type of structure.

structure_len *structure length*
Length of this structure in bytes.

thickness *thickness*
Thickness of layer in kilometers.
[index_string=true]

pveltop *P at top*
P velocity at top of layer.

pvelbase *P at base*
P velocity at base of layer.

sveltop *S at top*
S velocity at top of layer.

svelbase *S at base*
S velocity at base of layer.

function *velocity function*
[codelist=velfunc]

spareF *for future use*

SEE ALSO

NAME

loctrace – location of trace

SYNOPSIS

```
typedef  struct {
         FIXED          structure_type;
         FIXED          structure_len;
         FIXED          len_netn_G;
         STRING         network[4];
         FIXED          len_st_nam_G;
         STRING         st_name[5];
         CODE1          component;
         CODE2          inst_type;
         CHRPTR         fileloc;
         CHRPTR         tapeloc;
         INT4           beginloc;
} PCSUDS_LOCTRACE;

#define PC_LOCTRACE     8L
```

DESCRIPTION

Not commonly used in PC-SUDS.

MEMBERS

structure_type *structure type*

Define number of this type of structure.

structure_len *structure length*

Length of this structure in bytes.

len_netn_G *len net name*

network *network name*

len_st_nam_G *len stat name*

st_name *station name*

[index_string=true]

component *component*

[codelist=comps]

inst_type *instrument type*

[codelist=inst_type]

fileloc *file name*

Pointer to pathname in file system.

tapeloc *tape name*

Pointer to name of tape or offline storage.

beginloc *offset in file*

Bytes from begining of file to trace.

SEE ALSO

AUTHOR

Robert Banfill, January 1991

NAME

moment – moment tensor information

SYNOPSIS

```
typedef  struct {
         FIXED        structure_type;
         FIXED        structure_len;
         CODE1        datatypes;
         CODE1        constraints;
         INT2         spareD;
         FLOAT4       sc_moment;
         FLOAT4       norm_ten0;
         FLOAT4       norm_ten1;
         FLOAT4       norm_ten2;
         FLOAT4       norm_ten3;
         FLOAT4       norm_ten4;
         FLOAT4       norm_ten5;
} PCSUDS_MOMENT;

#define PC_MOMENT                 17L
```

DESCRIPTION

Not commonly used in PC-SUDS.

MEMBERS

structure_type *structure type*
Define number of this type of structure.

structure_len *structure length*
Length of this structure in bytes.

datatypes *data types*
Types of data used.
[codelist=moment_dtype]

constraints *constraints*
Ways solution is constrained.
[codelist=moment_constr]

spareD *for future use*

sc_moment *scalar moment*
[index_string=true]

norm_ten0 *tensor 0*

norm_ten1 *tensor 1*

norm_ten2 *tensor 2*

norm_ten3 *tensor 3*

norm_ten4 *tensor 4*

norm_ten5 *tensor 5*

SEE ALSO

NAME

muxdata – header for multiplexed data

SYNOPSIS

```
typedef  struct {
         FIXED      structure_type;
         FIXED      structure_len;
         FIXED      len_netname;
         STRING     netname[4];
         MS_TIME    begintime;
         INT2       loctime;
         INT2       numchans;
         FLOAT4     dig_rate;
         CODE1      typedata;
         CODE1      descript;
         INT2       spareG;
         INT4       numsamps;
         INT4       blocksize;
         PAD4       pad_L;
} PCSUDS_MUXDATA;

#define PC_MUXDATA                 6L
```

DESCRIPTION

Commonly used in PC-SUDS. Description of multiplexed data from an online detector. The multiplexed data follows this structure in the form of **typedata** and the length of **numsamps**.

MEMBERS

structure_type *structure type*
Define number of this type of structure.

structure_len *structure length*
Length of this structure in bytes.

len_netname *len net name*

netname *network name*
[index_string=true]

begintime *begin time*

loctime *local time*

numchans *num channels*

dig_rate *digitization rate*

typedata *type of data*
[sets_data_type=true, codelist=datatyp]

descript *description*
Description of event.
[codelist=descript]

spareG *for future use*

numsamps *num samples*
Number of sample sweeps. Typically not known when header is written, but can be added later.
[sets_data_length=true]

blocksize *block size*
Number of demultiplexed samples per channel if data is partially demultiplexed, otherwise=0.

pad_L *padding*

SEE ALSO

NAME

origin – information about a specific solution for a given event

SYNOPSIS

```
typedef struct {
        FIXED       structure_type;
        FIXED       structure_len;
        INT4        number;
        INT2        authority;
        CHAR        version;
        CHAR        or_status;
        CHAR        preferred;
        CHAR        program;
        CHAR        depcontrl;
        CHAR        convergence;
        INT4        region;
        MS_TIME     orgtime;
        LATIT       or_lat;
        LONGIT      or_long;
        FLOAT4      depth;
        FLOAT4      err_horiz;
        FLOAT4      err_depth;
        FLOAT4      res_rms;
        FIXED       len_crustmod;
        STRING      crustmodel[6];
        INT2        gap;
        FLOAT4      nearstat;
        INT2        num_stats;
        INT2        rep_p;
        INT2        used_p;
        INT2        rep_s;
        INT2        used_s;
        INT2        mag_type;
        INT2        rep_m;
        INT2        used_m;
        FLOAT4      magnitude_val;
        FLOAT4      weight;
        FLOAT4      mag_rms;
        ST_TIME     effective;
} PCSUDS_ORIGIN;

#define PC_ORIGIN                 14L
```

DESCRIPTION

Very commonly used in PC-SUDS.

MEMBERS

structure_type *structure type*
Define number of this type of structure.

structure_len *structure length*
Length of this structure in bytes.

number *event number*
Unique event number assigned by organization.

authority *authority*
Organization processing the data.

version *version*
Version of solution within organization.

or_status *processing status*
Processing status.

preferred *preferred sol*

program *processing program*
Name of processing program.

depcontrl *depth control*
Depth control.

convergence *convergence*
hypocentral convergence.

region *region*
Geographic region code assigned locally.

orgtime *origin time*
Origin time in GMT.
[index_string=true]

or_lat *latitude*
Origin latitude, north is plus.

or_long *longitude*
Origin longitude, east is plus.

depth *depth*
Origin depth in kilometers, + down.

err_horiz *horizontal error*
Horizontal error in km.

err_depth *depth error*
Vertical error in km.

res_rms *rms of residuals*
Root mean square of the residuals.

len_crustmod *len crustmodel*

crustmodel *name of crustmodel*

gap *gap*
Azimuthal gap in degrees.

nearstat *nearest station*
Distance in km to nearest station.

num_stats *number of stations*
Number of stations reporting phases.

rep_p *P reported*
Number of p phases reported.

used_p *P used*
Number of p times used in the solution.

rep_s *S reported*
Number of s phases reported.

used_s *S used*
Number of s times used in the solution.

mag_type *magnitude type*

Magnitude type: coda,tau,xmag ml,mb,ms,mw.

rep_m *mags reported*
Number of magnitude readings reported.

used_m *mags used*
Number of magnitude readings used.

magnitude_val *magnitude*

weight *weight*
Average magnitude weight.

mag_rms *rms of magnitudes*
Rms of magnitudes.

effective *date calculated*
Time this solution was calculated.

SEE ALSO

NAME

residual – calculated residuals for arrival times, magnitudes, etc.

SYNOPSIS

```
typedef struct {
        FIXED     structure_type;
        FIXED     structure_len;
        INT4      event_num;
        FIXED     len_netn_H;
        STRING    network[4];
        FIXED     len_st_nam_H;
        STRING    st_name[5];
        CODE1     component;
        CODE2     inst_type;
        CODE2     set_phase;
        CODE1     set_tim_qual;
        CODE1     set_amp_qual;
        FLOAT4    residual_val;
        FLOAT4    weight_used;
        FLOAT4    delay;
        FLOAT4    azimuth;
        FLOAT4    distance;
        FLOAT4    emergence;
        PAD4      pad_M;
} PCSUDS_RESIDUAL;

#define PC_RESIDUAL        11L
```

DESCRIPTION

Not commonly used in PC-SUDS.

MEMBERS

structure_type *structure type*
Define number of this type of structure.

structure_len *structure length*
Length of this structure in bytes.

event_num *event number*
Unique event number.

len_netn_H *len net name*

network *network name*

len_st_nam_H *len stat name*

st_name *station name*
[index_string=true]

component *component*
[codelist=comps]

inst_type *instrument type*
[codelist=inst_type]

set_phase *observed phase*
Phase code set for this solution.
[codelist=feat_phase]

set_tim_qual *timing quality*
Timing quality assigned for this solution.

[codelist=timequal]

set_amp_qual *amplitude quality*
Amplitude quality assigned for this solution.
[codelist=timequal]

residual_val *residual*
Traveltime residual or phase magnitude.

weight_used *weight used*
Weight used in this solution.

delay *delay*
Delay time or station correction used.

azimuth *azimuth*
Azimuth event to station, 0 north.

distance *distance*
Distance in km event to station.

emergence *angle emergence*
Angle of emergence from source, 0=down,180=up.

pad_M *padding*

SEE ALSO

NAME

stationcomp – generic station component information

SYNOPSIS

```
typedef     struct {
            FIXED           structure_type;
            FIXED           structure_len;
            FIXED           len_netn_I;
            STRING          network[4];
            FIXED           len_st_nam_I;
            STRING          st_name[5];
            CODE1           component;
            CODE2           inst_type;
            INT2            azim;
            INT2            incid;
            LATIT           st_lat;
            LONGIT          st_long;
            FLOAT4          elev;
            CODE1           enclosure;
            CODE1           annotation;
            CODE1           recorder_type;
            CODE1           rockclass;
            CODE2           rocktype;
            CODE1           sitecondition;
            CODE1           sensor_type;
            CODE1           datatyp;
            CODE1           data_units;
            CODE1           polarity_type;
            CODE1           st_status;
            FLOAT4          max_gain;
            FLOAT4          clip_value;
            FLOAT4          con_mvolts;
            INT2            channel_num;
            INT2            atod_gain;
            ST_TIME         effective;
            FLOAT4          clock_correct;
            FLOAT4          station_delay;
            PAD4            pad_N;
} PCSUDS_STATIONCOMP;

#define PC_STATIONCOMP      5L
```

DESCRIPTION

Very commonly used in PC-SUDS.

MEMBERS

structure_type *structure type*

Define number of this type of structure.

structure_len *structure length*

Length of this structure in bytes.

len_netn_I *len net name*

network *network name*

len_st_nam_I *len stat name*

st_name *station name*

[index_string=true]

component *component*
[codelist=comps]

inst_type *instrument type*
[codelist=inst_type]

azim *azimuth*
Component azimuth clockwise from north.

incid *angle incidence*
Component angle of incidence from vertical. 0 is vertical, 90 is horizontal.

st_lat *latitude*
Station latitude, north is plus.

st_long *longitude*
Station longitude, east is plus.

elev *elevation*
Station elevation in meters.

enclosure *enclosure*
[codelist=enclosur]

annotation *annotation*
Annotated comment code.
[codelist=ann_com]

recorder_type *type recorder*
Type device data recorded on.
[codelist=reco_type]

rockclass *class of rock*
[codelist=rock_class]

rocktype *type rock*
[codelist=rock_type]

sitecondition *site condition*
[codelist=site_condit]

sensor_type *sensor type*
[codelist=sensor]

datatyp *data type*
[codelist=datatyp]

data_units *data units*
[codelist=ampunits]

polarity_type *polarity*
[codelist=polar]

st_status *station status*
[codelist=stat_stat]

max_gain *maximum gain*
Maximum gain of the amplifier.

clip_value *clipping value*
+-value of data where clipping begins.

con_mvolts *conversion to mv*
Conversion factor to millivolts: mv per count. 0 means not defined or not appropriate. Max_ground_motion = digital_sample∗con_mvolts.

channel_num *channel number*
Analog to digital converter channel number.

atod_gain *atod gain*
Gain of analog to digital converter.

effective *time effective*
Date/time these values became effective.

clock_correct *clock correction*
Clock correction in seconds.

station_delay *station delay*
Seismological station delay.

pad_N *padding*

SEE ALSO

NAME

structtag – structure to identify structures when archived together

SYNOPSIS

```
typedef   struct {
          CHAR        sync;
          CODE1       machine;
          INT2        id_struct;
          INT4        len_struct;
          INT4        len_data;
          PAD4        pad_S;
} PCSUDS_STRUCTTAG;

#define PC_STRUCTTAG   2L
```

DESCRIPTION

All structures written in a stream such as on a disk, tape, and over the network, must be followed by a **structtag**. This tag is used for error detection and to explain what structure follows and how much data follow the structure. The **structtag** is the label used to identify structures.

Required in a stream before each structure.

MEMBERS

sync *synchronization char*

All **structtags** must begin with the letter S. When a structure and any data following the structure are read, the next structure_tag is also read, and if the first letter is not S, an error is declared. In this way when a structure is read, the computer knows that it has been read properly.

[default_value="S", allow_char="S"]

machine *type of computer*

Type of computer this structure was written on. For PC_SUDS this must be a '6' respresenting an 80x86 computer in PC_SUDS format. The input/output library routines identify PC_SUDS data by this character. Anything other than a '6' will be interpreted as SUDS data and will cause program termination for PC_SUDS data.

[codelist=computer_types]

id_struct *structure id*

An integer defining the type of structure that follows. The integers are defined on the manual pages defining the structures.

len_struct *len structure*

The length of the structure in bytes.

len_data *len data*

Length of data in bytes that follows the structure. The type of data is defined within the structure.

pad_S *padding*

SEE ALSO

NAME

pc_terminator – structure to end a sequence of related structures

SYNOPSIS

```
typedef   struct {
          FIXED          structure_type;
          FIXED          structure_len;
          INT2           structid;
          INT2           spareA;
          PAD4           pad_O;
} PCSUDS_TERMINATOR;

#define PC_TERMINATOR    3L
```

DESCRIPTION

Not commonly used in PC-SUDS.

MEMBERS

structure_type *structure type*
Define number of this type of structure.

structure_len *structure length*
Length of this structure in bytes.

structid *structure id*
Id for structure at beginning of this sequence.
[index_string=true]

spareA *for future use*

pad_O *padding*

SEE ALSO

NAME

timecorrection – time correction information

SYNOPSIS

```
typedef    struct {
           FIXED        structure_type;
           FIXED        structure_len;
           FIXED        len_netn_J;
           STRING       network[4];
           FIXED        len_st_nam_J;
           STRING       st_name[5];
           CODE1        component;
           CODE2        inst_type;
           PAD4         pad_P;
           FLOAT8       time_correct;
           FLOAT4       rate_correct;
           CODE1        sync_code;
           CHAR         program;
           PAD2         pad_Q;
           ST_TIME      effective_time;
           INT2         spareM;
           PAD2         pad_B;
} PCSUDS_TIMECORRECTION;

#define PC_TIMECORRECTION 30L
```

DESCRIPTION

Commonly used in PC-SUDS.

MEMBERS

structure_type *structure type*

Define number of this type of structure.

structure_len *structure length*

Length of this structure in bytes.

len_netn_J *len net name*

network *network name*

len_st_nam_J *len stat name*

st_name *station name*

[index_string=true]

component *component*

[codelist=comps]

inst_type *instrument type*

[codelist=inst_type]

pad_P *padding*

time_correct *time correction*

Time correction to be added to begintime.

rate_correct *rate correction*

Rate correction to be added to rate.

sync_code *synchonization*

Synchronization code.

[codelist=syncs]

program *program*

pad_Q *padding*

effective_time *time effective*
Time this correction was calculated.

spareM *for future use*

pad_B *padding*

SEE ALSO

NAME

triggers – earthquake detector trigger statistics

SYNOPSIS

```
typedef  struct {
         FIXED          structure_type;
         FIXED          structure_len;
         FIXED          len_netn_K;
         STRING         network[4];
         FIXED          len_st_nam_K;
         STRING         st_name[5];
         CODE1          component;
         CODE2          inst_type;
         INT2           sta;
         INT2           lta;
         INT2           abs_sta;
         INT2           abs_lta;
         INT2           trig_value;
         INT2           num_triggers;
         MS_TIME        trig_time;
} PCSUDS_TRIGGERS;

#define PC_TRIGGERS            25L
```

DESCRIPTION

Commonly used in PC-SUDS.

MEMBERS

structure_type *structure type*
Define number of this type of structure.

structure_len *structure length*
Length of this structure in bytes.

len_netn_K *len net name*

network *network name*

len_st_nam_K *len stat name*

st_name *station name*
[index_string=true]

component *component*
[codelist=comps]

inst_type *instrument type*
[codelist=inst_type]

sta *sta* Short term average.

lta *long term average*
Long term average; pre_lta for xdetect.

abs_sta *abs sta*
short term absolute average.

abs_lta *abs lta*
Long term absolute average.

trig_value *trigger level*
Value of trigger level (eta).

num_triggers *num triggers*

Number of times triggered during this event.

trig_time *trigger time*
Time of first trigger.

SEE ALSO

NAME

trigsetting – settings for earthquake trigger system

SYNOPSIS

```
typedef struct {
        FIXED      structure_type;
        FIXED      structure_len;
        FIXED      len_netwn;
        STRING     netwname[4];
        MS_TIME    beginttime;
        INT2       const1;
        INT2       const2;
        INT2       threshold;
        INT2       const3;
        INT2       const4;
        INT2       wav_inc;
        FLOAT4     sweep;
        FLOAT4     aperture;
        CODE1      algorithm;
        CHAR       spareJ;
        INT2       spareI;
} PCSUDS_TRIGSETTING;

#define PC_TRIGSETTING     26L
```

DESCRIPTION

Commonly used in PC-SUDS.

MEMBERS

structure_type *structure type*
Define number of this type of structure.

structure_len *structure length*
Length of this structure in bytes.

len_netwn *len net name*

netwname *network name*
[index_string=true]

beginttime *time effective*

const1 *constant 1*
Trigger constant 1.

const2 *constant 2*
Trigger constant 2.

threshold *threshold*
Trigger threshold.

const3 *constant 3*
Trigger constant 3.

const4 *constant 4*
Trigger constant 4.

wav_inc *increment*
Weighted average increment.

sweep *sweep*
Trigger sweep time in seconds.

aperture *aperture*
Seconds for coincident station triggers.

algorithm *algoritm*
[codelist=algorith]

spareJ *for future use*

spareI *for future use*

SEE ALSO

NAME

velmodel – velocity model

SYNOPSIS

```
typedef     struct {
            FIXED       structure_type;
            FIXED       structure_len;
            FIXED       len_netnam;
            STRING      netname[4];
            PAD4        pad_R;
            FIXED       len_modelname;
            STRING      modelname[6];
            CHAR        spareE;
            CHAR        modeltype;
            LATIT       latA;
            LONGIT      longA;
            LATIT       latB;
            LONGIT      longB;
            ST_TIME     time_effective;
            PAD4        pad_W;
} PCSUDS_VELMODEL;

#define PC_VELMODEL             18L
```

DESCRIPTION

Not commonly used in PC-SUDS. Definition of a flat-layered velocity model. This structure should be followed by a number of **layer** structures.

MEMBERS

structure_type *structure type*
: Define number of this type of structure.

structure_len *structure length*
: Length of this structure in bytes.

len_netnam *len net name*

netname *network name*

pad_R *padding*

len_modelname *len model name*

modelname *model name*
: [index_string=true]

spareE *for future use*

modeltype *model type*
: p=profile A to B, a=area within corners A B.

latA *latitude A*
: Latitude of point A, north is plus.

longA *longitude A*
: Longitude of point A, north is plus.

latB *latitude B*
: Latitude of point B, north is plus.

longB *longitude B*
: Longitude of point B, east is plus.

time_effective *time created*

Time this model was created.

pad_W *padding*

SEE ALSO

This page left blank intentionally.

SUDS

Chapter 6: Code Lists

This page left blank intentionally.

NAME

authorities – codelist identifying network or who done it

C_SYNOPSIS

```
SUDS_CODE_LIST authorities[]={
        0,          "none: not given",
        1,          "temp: Temporary, for testing purposes",
        2,          "suds: Internal to SUDS",
    10000,          "gsmen: US Geological Survey, Menlo Park, CA",
    10001,          "suds: testing of suds at the USGS, Menlo Park, CA",
    10002,          "calnt: network porcessing group, USGS, Menlo Park, CA",
    10005,          "5day: 5 day recorders US Geological Survey, Menlo Park, CA",
    10006,          "geos: GEOS recorders US Geological Survey, Menlo Park, CA",
    10007,          "cent: centipede recorders US Geological Survey, Menlo Park, CA",
    10008,          "citgs: CIT stations maintained by USGS, Menlo Park, CA",
    10009,          "lllgs: LLL stations maintained by USGS, Menlo Park, CA",
    10010,          "dwrgs: LLL stations maintained by USGS, Menlo Park, CA",
    10011,          "unrgs: UNR stations maintained by USGS, Menlo Park, CA",
    10012,          "yel: Yellowstone Park, Wyoming, maintained by USGS, Menlo Park, CA",
    10500, "RTP: main rtp, USGS, Menlo Park",
    10501, "PRTP: prototype rtp, USGS, Menlo Park",
    10502, "MRTP: motorola rtp, USGS, Menlo Park",
    10503, "TUST1: CUSP Tustin A/D #1",
    10504, "TUST2: CUSP Tustin A/D #2",
    10505, "ECLIP: CUSP Eclipse digitizer",
    10506, "CVAX: CUSP-VAX/750 digitizer",
    10507, "HPARK: Haliburton digital, Parkfield",
  10520, "CITT1: Tustin #1, Pasadena",
  10521, "CITT2: Tustin #2, Pasadena",
  10522, "CITN3: 11/34 online, Pasadena",
  10523, "CITS3: 11/34 online, Pasadena",
  10524, "CITD1: Nova/Eclipse, Pasadena",
  10525, "CITF: VAX, Pasadena",
  10526, "CITH: hand timed in Pasadena",
  11000,            "Daiss, Charles, USGS, Menlo Park, CA",
    11001,          "Oppenheimer, Dave, USGS, Menlo Park, CA",
    11002,          "Eaton, Jerry, USGS, Menlo Park, CA",
    15000,          "gspas: US Geological Survey, Pasadena, CA",
    15001,          "tergp: TERRAscope, US Geological Survey, Pasadena, CA",
    20000,          "uofa: Geophysical Institute, University of Alaska, College, AK",
    30000,          "uofw: Geophysics, University of Washington, WA",
    40000,          "ldgo: Lamont Doherty Geological Observatory, Palisades, NY",
    50000,          "iris: IRIS Consortium, Seattle Data Center, WA",
    51000,          "gsn: Global Seismographic Network, USGS, Albuquerque, NM",
    52000,          "asro: Abbreviated Seismic Research Observatories",
    53000,          "passc: PASSCAL Program, IRIS",
    60000,          "lll: Lawrence Livermore Labs, Livermore, CA",
    70000,          "lbl: Lawrence Berkeley Labs, U. C. Berkeley, CA",
    80000,          "lanl: Los Alamos National Labs, Los Alamos, NM",
    90000,          "stl: St. Louis University, St. Louis, MO",
   100000,          "ucsd: University of California, San Diego and SCRIPPS",
   110000,          "ucb: University of California, Berkeley, CA",
   120000,          "ucsb: University of California, Santa Barbara, CA",
   130000,          "ucsc: University of California, Santa Cruz, CA",
```

140000,	"usc: University of Southern California, Los Angeles, CA",
150000,	"cit: California Institute of Technology, Pasadena, CA",
150001,	"terct: TERRAscope network, California Institute of Technology, Pasadena, CA",
160000,	"nnunr: Northern Nevada net, University of Nevada, Reno, NV",
160001,	"snunr: Southern Nevada net, University of Nevada, Reno, NV",
170000,	"utah: University of Utah, Salt Lake City, UT",
180000,	"msu: Memphis State University, Memphis, TN",
180000	"msu: Memphis State University, Memphis, TN",
181010	"sanju: PANDA experiment in SAN JUAN, Argentina",
181011	"jujuy: PANDA experiment in JUJUY, Argentina",
181020	"newma: PANDA experiment in NEW MADRID, TN",
181030	"arken: PANDA experiment in AK",
181040	"hawii: PANDA experiment in HAWII",
181050	"palmn: PANDA experiment in PALMERSTON NORTH, New Zealand",
181051	"taran: PANDA2 experiment in mountain TARAMAKI, New Zealand",
181060	"taiwa: PANDA2 experiment in Taiwan",
187000	"archj: ARCH Johnston, professor, director of research",
187001	"jmch: Jer-Ming CHiu, professor",
187002	"wych: Wai-Ying CHung, associate research professor",
187003	"hjdo: H.James DOrman, executive director",
187004	"mell: Michael ELLis, associate professor",
187005	"Josep: JOSE Pujol, associate professor",
187006	"paulr: PAUL Rydelek, assistant research professor",
187007	"robsm: ROBert SMalley, assistant research professor",
187008	"paulb: PAUL Bodin, assistant professor",
187009	"eusc: EUgene SChweig, adjunct professor, USGS geologist",
187010	"johng: JOHN Geomberg, adjunct professor, USGS geophysicist",
187011	"scda: SCott DAvis, USGS guest researcher",
187500	"jimbo: JIM BOllwerk, seismic networks engineer",
187501	"stepb: STEPhen Brewer, ceri seismic networks director",
187502	"cchiu: Christy CHIU, research associate II",
187503	"michf: MICHael Frohme, director of computing",
188000	"zrli: ZhaoRen LI, graduate research assistant",
188001	"kcch: Kou-Cheng Chen, graduate research assistant",
189000	"group: data processing GROUP in ceri",
190000,	"aftac: AFTAC Center for Seismic Studies, Alexandria, VA",
200000,	"uhhil: University of Hawaii, Hilo, HA",
210000,	"uhhon: University of Hawaii, Honolulu, HA",
220000,	"mit: Massachusetts Institute of Technology, Cambridge, MA",
230000,	"dtm: Department of Terrestrial Magnetism, Washington, DC",
240000,	"vpi: Virginia Polytechnic Institute, Blacksburg, VA",
250000,	"anu: Australian National University",
260000,	"gsgol: US Geological Survey, Golden, CO",
260001,	"nngsg: Northern Nevada network, US Geological Survey, Golden, CO",
260002,	"sngsg: Southern Nevada network, US Geological Survey, Golden, CO",
270000,	"bmr: Bureau of Mineral Resources",
280000,	"cands: Canadian Digital Seismic Network",
290000,	"cdsn: China Digital Seismic Network",
300000,	"cdmg: California Division Mines-Geology, Sacramento, CA",
310000,	"pge: Pacific Gas and Electric/Woodward-Clyde, CA",
315001,	"unoiv: Union Oil, Imperial Valley, CA",
315002,	"unoml: Union Oil, Medicine Lake",
320000,	"terra: Terra Corporation, Mendocino, CA",

```
        330000,        "cadwr: California Division of Water Resources",
        340000,        "gikar: Geophysical Institute, Karlsruhe, Germany",
        350000,        "gfz: GeoForschungsZentrum, Potsdam, Germany",
        360000,        "cnrir: CNR-IRS, Milan, Italy",
        370000,        "gsc: Geological Survey of Canada, Ottawa, Canada",
        380000,        "ind: industry",
        385000,        "geot: Geotech, Garland, Texas",
        390000,        "nano: Nanometrics, Kanata, Ontario, Canada",
        395000,        "lenn: Lennartz Electronic, Tubingen, Germany",
        400000,        "kine: Kinemetrics, Pasadena, CA",
        405000,        "snl: Sandia National Laboratories, Albuquerque, NM",
        410000,        "cices: CICESE, Ensenada, Mexico",
        415000,        "nmt: New Mexico Inst Mining and Tech, Soccorro, NM",
        0,             0,
};
```

NAME

codelists – initialization of codelists used in SUDS

C_SYNOPSIS

101:amplitude_types

’M’ = "peak-to-median"
’c’ = "gain range corrected for"
’g’ = "gain ranged"
’m’ = "peak-to-mean"
’n’ = "normal"
’p’ = "peak-to-peak"
’r’ = "root mean square"
’s’ = "step, 0 to peak"
’z’ = "peak-to-zero"

102:authorities /* see authorities(6) */

103:band_types

’E’ = "extremely long period, ˜0.001 s/s"
’b’ = "broad band, 10<s/s<80"
’e’ = "extremely short period, 80<s/s"
’h’ = "high broad band, 80<s/s"
’l’ = "long period, ˜1 s/s"
’m’ = "mid period, 1<s/s<10"
’s’ = "short period, 10<s/s<80"
’u’ = "ultra long period, ˜0.01 s/s"
’v’ = "very long period, ˜0.1 s/s"
’w’ = "Wood-Anderson"
’x’ = "experimental"

104:cal_data_srcs

’1’ = "CUSP Tustin A/D #1"
’2’ = "CUSP Tustin A/D #2"
’3’ = "Pasadena Tustin #1"
’4’ = "Pasadena Tustin #2"
’5’ = "Pasadena 11/34 online"
’6’ = "Pasadena 11/34 online"
’7’ = "Pasadena Nova/Eclipse"
’8’ = "Pasadena VAX digitized"
’?’ = "Unknown or undefined"
’B’ = "UC Berkeley readings"
’C’ = "Pasadena hand timed"
’D’ = "Haliburton digital, Parkfield"
’E’ = "CUSP-Eclipse digitized"
’F’ = "CUSP-VAX/750 digitized"
’G’ = "CDMG readings"
’H’ = "Hand timed"
’J’ = "Jerry Eaton hand timed"
’L’ = "LLL readings"
’M’ = "Readings from Mexico"
’O’ = "Motorola RTP"
’P’ = "Prototype RTP"
’R’ = "Main RTP"
’S’ = "USC readings"

'T' = "Tera Corp. or PG&E readings"
'U' = "U. Nevada Reno readings"
'Y' = "Woodward-Clyde readings"

105:causes
'a' = "automatic"
'm' = "manual"

106:clarities
'c' = "clear"
'd' = "dubious"
'e' = "extremely clear"
'g' = "best guess"

107:clock_programs
'_' = "none"

108:clock_types
'c' = "cesium"
'd' = "digital"
'e' = "escapement"
'p' = "pendulum"
'r' = "radio"
's' = "satellite"

179:codelists
1 = "ann_com"
2 = "algorith"
3 = "ampunits"
4 = "authority"
5 = "comps"
6 = "datasrc"
7 = "datatyp"
8 = "descript"
9 = "enclosur"
10 = "equip_model"
11 = "equip_reason"
12 = "eventtyp"
13 = "feat_phase"
14 = "firstm"
15 = "inst_type"
16 = "mach_type"
17 = "mechan"
18 = "medium"
19 = "moment_constr"
20 = "moment_dtype"
21 = "onsetz"
22 = "polar"
23 = "reco_type"
24 = "rock_class"
25 = "rock_type"
26 = "sensor"
27 = "site_condit"

28 = "stat_stat"
29 = "stat_proc"
30 = "struct_names"
31 = "syncs"
32 = "tecton"
33 = "timequal"
34 = "timings"
35 = "velfunc"
36 = "waterwave"
101 = "amplitude_types"
102 = "authorities"
103 = "band_types"
104 = "cal_data_srcs"
105 = "causes"
106 = "clarities"
107 = "clock_programs"
108 = "clock_types"
109 = "components"
110 = "compression_types"
111 = "computer_types"
112 = "convergences"
113 = "coordinate_types"
114 = "data_types"
115 = "data_group_types"
116 = "decimation_types"
117 = "degree_types"
118 = "depth_controls"
119 = "detector_types"
120 = "enclosure_types"
121 = "endian_types"
122 = "equip_actions"
123 = "equip_models"
124 = "equip_reasons"
125 = "event_types"
126 = "filter_types"
127 = "first_motions"
128 = "float_types"
129 = "functions"
130 = "gain_types"
131 = "gain_unit_types"
132 = "horiz_datums"
133 = "hypo_controls"
134 = "hypo_programs"
135 = "input_subroutines"
136 = "instrument_types"
137 = "labels"
138 = "magnitude_types"
139 = "map_items"
140 = "map_projections"
141 = "mechanism_types"
142 = "media"
143 = "model_types"
144 = "onset_types"

145 = "pick_methods"
146 = "pick_types"
147 = "polarities"
148 = "precision_codes"
149 = "prime_meridians"
150 = "process_types"
151 = "recording_medias"
152 = "recorder_types"
153 = "region_types"
154 = "regions"
155 = "resolutions"
156 = "response_types"
157 = "rev_evidence"
158 = "rock_types"
159 = "sensor_types"
160 = "site_conditions"
161 = "site_types"
162 = "solution_qualities"
163 = "solution_types"
164 = "spectra_types"
165 = "spheroids"
166 = "status"
167 = "structure_names"
168 = "survey_methods"
169 = "synchronization_types"
170 = "taper_types"
171 = "time_types"
172 = "time_codes"
173 = "time_corrects"
174 = "time_func_types"
175 = "timing_qualities"
176 = "trigger_types"
177 = "units_types"
178 = "zero_weights"
179 = "codelists"

109:components

'B' = "beam"
'T' = "time"
'X' = "X triaxial (specify azimuth and dip)"
'Y' = "Y triaxial (specify azimuth and dip)"
'Z' = "Z triaxial (specify azimuth and dip)"
'd' = "dilatometer"
'e' = "east-west"
'h' = "horizontal"
'n' = "north-south"
'o' = "other (specify azimuth and dip)"
'r' = "radial (specify azimuth and dip)"
's' = "scalar calibrations"
't' = "transverse (specify azimuth and dip)"
'v' = "vertical"
'x' = "experimental"

110:compression_types
'1' = "Steim 1"
'2' = "Steim 2"
'_' = "no compression"
'j' = "Jiang"

111:computer_types
'6' = "PCSUDS on 80x86"
'i' = "ibm"
'v' = "vax"
'x' = "xdr"

112:convergences
'f' = "failed to reach min rms"
'i' = "too many iterations"
'p' = "convergence problems"

113:coordinate_types
'g' = "geographic: lat-lon"
'l' = "local grid: y-x"
'r' = "radial: dist-azm"
'u' = "UTM"

114:data_types /* created by the manual compiler */
-33 = "PAD4"
-32 = "PAD2"
-31 = "PAD1"
-30 = "INT3"
-29 = "IDXPTR"
-28 = "YESNO"
-27 = "UCHAR"
-26 = "UINT2"
-25 = "UINT4"
-24 = "MEMPTR"
-23 = "CHRPTR"
-22 = "GENPTR"
-21 = "CODESTR"
-20 = "INT2TM"
-19 = "LIST"
-18 = "LATIT"
-17 = "LONGIT"
-16 = "MS_TIME"
-15 = "ST_TIME"
-14 = "FLOAT8"
-13 = "FLOAT4"
-12 = "AUTHOR"
-11 = "DOMAIN"
-10 = "REFERS2"
-9 = "LABEL"
-8 = "CODE4"
-7 = "CODE2"
-6 = "CODE1"
-5 = "FIXED"

-4 = "INT4"
-3 = "INT2"
-2 = "STRING"
-1 = "CHAR"
2 = "structtag",
3 = "pc_terminator",
4 = "equipment",
5 = "stationcomp",
6 = "muxdata",
7 = "descriptrace",
8 = "loctrace",
9 = "pc_calibration",
10 = "feature",
11 = "residual",
12 = "pc_event",
13 = "evdescr",
14 = "origin",
15 = "error",
16 = "focalmech",
17 = "moment",
18 = "velmodel",
19 = "layers",
20 = "pc_comment",
23 = "calib",
25 = "triggers",
26 = "trigsetting",
27 = "eventsetting",
28 = "detector",
29 = "atodinfo",
30 = "timecorrection",
31 = "instrument",
32 = "chanset",
33 = "chansetentry",
104 = "sig_path_cmp",
105 = "signal_path",
106 = "mux_waveform",
107 = "waveform",
108 = "data_group",
109 = "response",
110 = "pick",
111 = "pick_residual",
112 = "event",
113 = "signif_event",
114 = "solution",
115 = "solution_err",
116 = "focal_mech",
118 = "vel_model",
119 = "vel_layer_data",
123 = "resp_pz_data",
125 = "lsa_detection",
126 = "lsa_setting",
130 = "clock_rate",
131 = "recorder",

200 = "**variable_info**",
201 = "**structure_info**",
202 = "**member_info**",
203 = "**stream**",
204 = "**file_index**",
205 = "**code_list**",
206 = "**gui_default**",
211 = "**comment**",
212 = "**structure_tag**",
213 = "**terminator**",
214 = "**code_data**",
300 = "**site**",
301 = "**spectra**",
302 = "**sig_cmp_data**",
303 = "**resp_fap_data**",
304 = "**lsa_set_data**",
305 = "**resp_cfs_data**",
306 = "**sig_path_data**",
307 = "**magnitude**",
308 = "**processing**",
309 = "**polarity**",
310 = "**ssam_setup**",
311 = "**ssam_output**",
312 = "**map_element**",
313 = "**seismometer**",
314 = "**resp_fir_data**",
315 = "**filter**",
316 = "**sig_path_ass**",
317 = "**ssam_band_data**",
318 = "**beam_data**",
319 = "**resp_sen_data**",
320 = "**calibration**",
321 = "**source**",
322 = "**user_vars**",
323 = "**service**",
324 = "**recorder_ass**",
325 = "**seismo_ass**",
326 = "**coordinate_sys**",

115:data_group_types
'E' = "**explosion**"
'd' = "**common depth-point gather**"
'e' = "**earthquake**"
'm' = "**common mid-point gather**"
'r' = "**receiver gather**"
's' = "**shot gather**"
't' = "**time interval**"

116:decimation_types
'a' = "**average**"
'e' = "**envelope**"
's' = "**simple**"

117:degree_types

'd' = "degrees"
'm' = "degrees minutes"
's' = "degrees minutes seconds"

118:depth_controls

'D' = "use 2 or more pP"
'G' = "fixed by geophysicist"
'N' = "fixed at 33 km"
'S' = "fixed at surface"
'e' = "known man-made source"
'f' = "fixed"

119:detector_types

'C' = "continuous interval"
'c' = "cross trigger"
'e' = "external trigger"
'l' = "fixed level trigger"
'm' = "murdock"
'r' = "radio trigger"
's' = "shortterm/longterm average"
't' = "fixed time trigger"
'z' = "log-z"

120:enclosure_types

'B' = "building"
'D' = "concrete arch dam"
'b' = "buried"
'd' = "concrete gravity dam"
'e' = "earth fill dam"
'g' = "bridge"
'n' = "nuclear power plant"
's' = "on surface"
'v' = "underground vault"

121:endian_types

'b' = "big endian"
'f' = "no change in endian"
'l' = "little endian"
't' = "change endian"

122:equip_actions

'A' = "initial installation"
'C' = "reinstallation after repairs"
'D' = "attenuation changed"
'E' = "vco battery changed"
'F' = "frequency adjusted"
'G' = "vco changed"
'H' = "calibrator changed"
'I' = "deviation changed"
'J' = "seismometer changed"
'K' = "phone drop changed"
'M' = "radio changed"

’N’ = "antenna changed"
’O’ = "landline changed"
’P’ = "grounding changed"
’R’ = "routine maintenance"
’S’ = "seismometer releveled"
’U’ = "amplifier changed"
’W’ = "summing amplifier changed"
’Y’ = "frequency compensator changed"
’Z’ = "tape channel changed"
’_’ = "none given"
’c’ = "configuration changed"
’e’ = "calibrator battery changed"
’s’ = "installed solar panels and solar vco"
’z’ = "signal zeroed"

123:equip_models
0 = "none given"
1 = "L4C geophone"
2 = "HS10 geophone"
3 = "EV17 geophone"
4 = "L22 geophone"
5 = "S-13 Geotech"
6 = "experimental L4C"
7 = "L28 geophone"
10 = "DILatometer Sacks-Evertson"
20 = "FBA-11 Kinemetrics"
21 = "FBA-13 Kinemetrics"
22 = "FBA-23 Kinemetrics"
23 = "SA-3000 Sprengnether"
24 = "SMA-1"
25 = "SMA-2"
26 = "C&GS Standard"
27 = "AR-240"
28 = "RFT-250"
29 = "RFT-350"
30 = "MO-2"
31 = "RMT-280"
32 = "SMA-2/3"
33 = "DSA-1/DSA-3"
34 = "DCA-300"
35 = "DCA-310"
36 = "DCA-333"
37 = "A-700"
38 = "SSA-1"
39 = "SSA-2"
40 = "CRA-1"
41 = "MO-2"
42 = "SSR-1"
43 = "BIDRA"
90 = "Tustin AtoD 2112-256-1S-110"
91 = "VR-3700B Bell and Howell tape"
92 = "3700E Bell and Howell tape"
99 = "j101amp/vco usgs" **/* J1 */**

```
100 = "j202 amp/vco usgs"                    /* J2 */
101 = "j302 mercury amp/vco usgs"            /* J3, J3* */
102 = "j302 lithium amp/vco usgs"/* J3L */
103 = "j302ml amp/vco usgs"                  /* J3M */
104 = "j312ml amp/vco usgs"                  /* J31 */
105 = "j402 amp/vco usgs"                    /* J4, J4* */
106 = "j402h amp/vco usgs"                   /* J4H, J4S */
107 = "j402l amp/vco usgs"                   /* J4L */
108 = "j402m amp/vco usgs"                   /* J4M */
109 = "j402h3 amp/vco usgs"                  /*  */
110 = "j412ml amp/vco usgs"                  /* J41 */
111 = "j501 amp/vco usgs"                    /* J5H serial 1000-1024 */
112 = "j502 amp/vco usgs"                    /* J5H serial 1025-1089 */
113 = "j502a amp/vco usgs"                   /* J5H serial 1090-9999 */
114 = "j512a amp/vco usgs"                   /* J51 */
115 = "6202 amp/vco develco"                 /* D */
116 = "low gain special at JMP"              /* J00 */
117 = "j303 special ??"                      /* J3C */
118 = "j401 T"                                         /* J4T */
119 = "j501 A"                                         /* J5A */
120 = "j501 amp/vco usgs, power mods"        /* J5M */
121 = "V02 amp/vco usgs"                     /* V02 */
122 = "V21 amp/vco usgs"                     /* V21 */
150 = "j202 calibrator usgs"                 /* J2 */
151 = "j302 mercury calibrator usgs"         /* J3, J3* */
152 = "j302 lithium calibrator usgs"         /* J3L */
153 = "j302ml calibrator usgs"               /* J3M */
154 = "j312ml calibrator usgs"               /* J31 */
155 = "j402 calibrator usgs"                 /* J4, J4* */
156 = "j402h calibrator usgs"                /* J4H, J4S */
157 = "j402l calibrator usgs"                /* J4L */
160 = "j412ml calibrator usgs"               /* J41 */
161 = "j501 calibrator usgs"                 /* J5H serial 1000-1024 */
162 = "j502 calibrator usgs"                 /* J5H serial 1025-1089 */
163 = "j502a calibrator usgs"                /* J5H serial 1090-9999 */
164 = "j512a calibrator usgs"                /* J51 */
167 = "j303 special ??"                      /* J3C */
200 = "PANDA high gain recorder, msu"
201 = "PANDA low gain recodrer, msu"
202 = "PANDA2 recorder, msu"
280 = "gs1 sum_amp usgs"
281 = "gs2 solar_sum_amp usgs"
300 = "r41f transmitter monitron"
301 = "r45f receiver monitron"
302 = "dt200 transmitter ritron"
303 = "dr200 receiver ritron"
304 = "ht200 transmitter motorola"
305 = "ht200 receiver motorola"
400 = "ca5-150h antenna scala"
401 = "ca5-150v antenna scala"
402 = "ca7-460 antenna scala"
403 = "cl-150hc antenna scala"
404 = "cl-150hr antenna scala"
```

405 = "cl-150v antenna scala"
406 = "ca5-450 antenna scala"
407 = "ra5-450 antenna scala"
408 = "pr-450u antenna scala"
500 = "1481 solar_panel arco"
501 = "435h solar_panel solarex"
600 = "01 discriminator usgs"
601 = "1 discriminator usgs"
602 = "1/2 discriminator usgs"
603 = "10 discriminator usgs"
604 = "10-20 discriminator usgs"
605 = "10-30 discriminator usgs"
606 = "10/20 discriminator usgs"
607 = "10/30 discriminator usgs"
608 = "20/1 discriminator usgs"
609 = "20/2 discriminator usgs"
610 = "21/2 discriminator usgs"
611 = "JJ discriminator usgs"

124:equip_reasons

'A' = "initial installation"
'C' = "reinstallation after repairs"
'E' = "vco battery low"
'F' = "frequency adjustment"
'G' = "vco problems"
'H' = "calibrator problems"
'I' = "change deviation"
'J' = "seismometer problems"
'K' = "phone drop problems"
'L' = "signal level problems"
'M' = "radio problems"
'N' = "antenna problems"
'O' = "landline problems"
'P' = "grounding problems"
'Q' = "flooding"
'R' = "routine maintenance"
'S' = "seismometer leveling problems"
'T' = "seismically dead"
'U' = "amplifier problems"
'V' = "vandalism"
'W' = "summing amplifier problem"
'X' = "seismic noise"
'Y' = "frequency compensator problems"
'Z' = "tape channel change"
'c' = "configuration change"
'e' = "calibrator battery low"
'h' = "signal too high"
's' = "install solar panels and solar vco"
'w' = "signal too weak"
'x' = "electronic noise"
'z' = "signal not zeroed"

125:event_types

'A' = "**nuclear post-shot event**"
'B' = "**nuclear pre-shot event**"
'C' = "**continuous data**"
'D' = "**dead trace**"
'E' = "**airborn chemical explosion**"
'F' = "**felt event**"
'G' = "**long period**"
'H' = "**volcano high frequency**"
'L' = "**landslide**"
'M' = "**mass drop**"
'N' = "**nuclear event**"
'O' = "**other**"
'Q' = "**shotgun**"
'R' = "**rifle**"
'S' = "**step calibration**"
'T' = "**tremor associated**"
'U' = "**USGS Menlo calibrator**"
'V' = "**artificial explosive other than quarry**"
'W' = "**water chemical explosive**"
'X' = "**emergent, low frequency near volcano**"
'a' = "**aftershock**"
'b' = "**b_type**"
'c' = "**calibration**"
'd' = "**borehole chemical explosion**"
'e' = "**earthquake**"
'f' = "**foreshock**"
'g' = "**airgun**"
'h' = "**hammer**"
'i' = "**icequake**"
'k' = "**volcanic blast**"
'l' = "**within net**"
'm' = "**sonic boom**"
'n' = "**noise**"
'o' = "**other**"
'p' = "**pseudo-random calibration**"
'q' = "**quarry shot**"
'r' = "**regional**"
's' = "**synthetic data**"
't' = "**teleseism**"
'v' = "**vibroseis frequency sweep**"
'w' = "**sine-wave calibration**"
'x' = "**experimental**"

126:filter_types
'f' = "**filtered**"

127:first_motions
'+' = "**probable up**"
'-' = "**probable down**"
'D' = "**down**"
'N' = "**nodal**"
'U' = "**up**"

128:float_types
'f' = "Change from VAX"
'i' = "IEEE float"
'n' = "No change"
't' = "Change to VAX"
'v' = "VAX float"

129:functions
'C' = "cosine squared"
'S' = "sine squared"
'c' = "constant"
'e' = "exponential"
'l' = "linear"
'o' = "other"
'p' = "parabolic"

130:gain_types
'U' = "ultra high"
'V' = "very high"
'h' = "high"
'l' = "low"
'm' = "medium"
'u' = "ultra low"
'v' = "very low"

131:gain_unit_types
'd' = "decibels"
'p' = "pure scalar multiplier"

132:horiz_datums
'r' = "reference site elevation"
's' = "sea level"

133:hypo_controls
'f' = "fixed"
's' = "known man-made source"

134:hypo_programs
'7' = "hypo71, Lee"
'e' = "hypoellipse, Lahr"
'i' = "hypoinverse, Klein"
'r' = "relp"
'u' = "Uhrhammer"

135:input_subroutines
'_' = "none"
'l' = "check_limits"

/* instrument types: use a number less than 100 except in equipment structures */
136:instrument_types
0 = "not specified"
1 = "sp usgs"

2 = "sp wwssn"
3 = "lp wwssn"
4 = "sp dwwssn"
5 = "lp dwwssn"
6 = "hglp lamont"
7 = "lp hglp lamont"
8 = "sp sro"
9 = "lp sro"
10 = "sp asro"
11 = "lp asro"
12 = "sp rstn"
13 = "lp rstn"
14 = "sp uofa U of alaska"
15 = "STS-1/UVBB"
16 = "STS-1/VBB"
17 = "STS-2"
18 = "FBA-23"
19 = "Wilcoxin"
50 = "USGS cassette"
51 = "GEOS"
52 = "EDA"
53 = "Sprengnether refraction"
54 = "Teledyne refraction"
55 = "Kinemetrics refraction"
300 = "amplifier"
301 = "amp/vco"
302 = "filter"
303 = "summing amp"
304 = "transmitter"
305 = "receiver"
306 = "antenna"
307 = "battery"
308 = "solar cell"
309 = "discriminator"
310 = "discr. rack"
311 = "paper recorder"
312 = "film recorder"
313 = "smoked glass recorder"
314 = "atod converter"
315 = "computer"
316 = "clock"
317 = "time receiver"
318 = "magnetic tape"
319 = "magnetic disk"
320 = "optical disk"

137:labels /* must be in strict numerical order */
0 = "unknown"
1 = "clock_rate_id"
2 = "comment_id"
3 = "data_group_id"
4 = "detection_id"
5 = "event_id"

6 = "focal_mech_id"
7 = "map_element_id"
8 = "mux_waveform_id"
9 = "pick_id"
10 = "recorder_id"
11 = "response_id"
12 = "seismometer_id"
13 = "sig_path_cmp_id"
14 = "signal_path_id"
15 = "solution_id"
16 = "ssam_setup_id"
17 = "site_id"
18 = "vel_model_id"
19 = "waveform_id"
20 = "service_id"
21 = "polarity_id"
22 = "calibration_id"
23 = "coordinate_id"
24 = "lsa_setting_id"
25 = "magnitude_id"
26 = "processing_id"
27 = "source_id"
28 = "user_vars_id"
29 = "spectra_id"
30 = "code_list_id"
31 = "default_id"
32 = "menu_id"
33 = ""
34 = ""
35 = ""
36 = ""
37 = ""
38 = ""

138:magnitude_types
'A' = "average coda and amplitude"
'S' = "Msz"
'a' = "amplitude"
'b' = "Mb"
'c' = "coda"
'l' = "Ml"
'm' = "moment"
's' = "Ms"
'w' = "Mw"

139:map_items
1 = "city"
2 = "state capitol"
3 = "country capitol"
4 = "village"
10 = "river"
11 = "stream"
12 = "drainage ditch"

100 = "topographic contour line"
101 = "geologic fprmation boundary"
102 = "magnetic contour line"
103 = "bouguer gravity contour line"
104 = "free-air gravity contour line"
105 = "submarine contour line"
200 = "strike and dip symbol"
201 = "mine symbol"

140:map_projections
’0’ = "user defined transformation"
’1’ = "linear x vs linear y"
’2’ = "linear x vs log y"
’3’ = "linear x vs ln y"
’4’ = "log x vs linear y"
’5’ = "log x vs log y"
’6’ = "log x vs ln y"
’7’ = "ln x vs linear y"
’8’ = "ln x vs log y"
’9’ = "ln x vs ln y"
’A’ = "albers equal area conic"
’D’ = "equal distance azimuthal"
’E’ = "polar cc from east, x deg, y dist"
’G’ = "gnomonic azimuthal"
’K’ = "kavraisky equal interval conic"
’L’ = "lambert conformal conic"
’M’ = "mercator cylindrical"
’N’ = "polar from north, x deg, y dist"
’O’ = "orthographic azimuthal"
’P’ = "polyconic"
’R’ = "rotate coordinates about a pole"
’S’ = "sinusoidal"
’T’ = "transverse mercator cylindrical"
’U’ = "universal transverse mercator"
’W’ = "wulff equal angle net,x azm,y dip"
’Z’ = "equal area azimuthal"
’e’ = "schmidt equal area net,x azm,y dip"
’m’ = "miller cylindrical"
’p’ = "perspective azimuthal"
’s’ = "stereographic azimuthal"
’t’ = "ptolemy equal interval conic"

141:mechanism_types
’P’ = "Pasyanos surface-wave inversion"
’d’ = "Dreger waveform inversion"
’h’ = "Harvard CMT"
’p’ = "P-wave first motion"
’u’ = "Urhammer near-field inversion"

142:media
’7’ = "7-track, 1/2 inch tape"
’9’ = "9-track, 1/2 inch tape"
’C’ = "disk cartridge"

’c’ = "1/4 inch cartridge tape"
’d’ = "DAT video tape"
’e’ = "Exabyte video tape"
’o’ = "optical read/write"
’r’ = "CD ROM"
’w’ = "WORM disk"

143:model_types
’a’ = "rectangular area, diagonal corners at A and B"
’p’ = "profile from A to B"

144:onset_types
’E’ = "emergent"
’I’ = "impulsive"
’e’ = "noisy emergent"
’i’ = "noisy impulsive"
’n’ = "noisy"

145:pick_methods
’A’ = "interactive automatic"
’a’ = "automatic offline"
’i’ = "interactive"
’r’ = "realtime processor"

146:pick_types
0 = "not given"
1 = "window" /* amplitude is the duration of the window */
2 = "f finis" /* for coda mag, time when signal is about twice noise */
3 = "x maximum amplitude"
4 = "increase gain step"
5 = "decrease gain step"
50 = "p first arrival" /* could be p, pn, pg, etc. */
51 = "p"
52 = "p*"
53 = "pp"
54 = "ppp"
55 = "pppp"
56 = "pps"
57 = "pg"
58 = "pn"
59 = "pdiffracted"
60 = "pcp"
61 = "pcppkp"
62 = "pcs"
63 = "pp"
64 = "ppp"
65 = "pkp"
66 = "pkppkp"
67 = "pkppks"
68 = "pkpsks"
69 = "pks"
70 = "ppks"
71 = "pkkp"

72 = "**pkks**"
73 = "**pcppkp**"
74 = "**pcspkp**"
100 = "**s first s wave**"
101 = "**s**"
102 = "**s***"
103 = "**ss**"
104 = "**sss**"
105 = "**ssss**"
106 = "**sg**"
107 = "**sn**"
108 = "**scs**"
109 = "**spcs**"
110 = "**ss**"
111 = "**sss**"
112 = "**ssss**"
113 = "**sscs**"
114 = "**scspkp**"
115 = "**scp**"
116 = "**sks**"
117 = "**skks**"
118 = "**skkks**"
119 = "**skssks**"
120 = "**skp**"
121 = "**skkp**"
122 = "**skkkp**"
201 = "**lg**"
202 = "**lr**"
203 = "**lr2**"
204 = "**lr3**"
205 = "**lr4**"
206 = "**lq**"
207 = "**lq2**"
208 = "**lq3**"
209 = "**lq4**"
301 = "**t**"

147:polarities

'n' = "**normal**"
'r' = "**reversed**"

148:precision_codes

'a' = "**0.000001**"
'b' = "**0.00001**"
'c' = "**0.0001**"
'd' = "**0.001**"
'e' = "**0.01**"
'f' = "**0.1**"
'g' = "**1.0**"
'h' = "**10.0**"
'i' = "**100.0**"
'j' = "**1000.0**"
'k' = "**10000.0**"

’l’ = "100000.0"
’m’ = "1000000.0"

149:prime_meridians
’g’ = "Greenwich"

150:process_types
’e’ = "error message"
’h’ = "hypoprogram"
’s’ = "sql"
’w’ = "waveform"

151:recording_medias
’5’ = "5-day tape playback"
’M’ = "32 mm microfilm"
’f’ = "1 in FM tape playback"
’h’ = "heated-pen recorder"
’i’ = "ink-jet recorder"
’m’ = "16 mm microfilm"
’s’ = "smoked-paper recorder"

152:recorder_types
’a’ = "analog tape"
’c’ = "cusp"
’j’ = "jade"
’p’ = "pdas"
’s’ = "sun-cdd"
’w’ = "willie"

153:region_types
’e’ = "Eaton station regions"
’f’ = "Flynn-Engdahl region in world"
’k’ = "Klein regions in California"

154:regions
’_’ = "none"

/* unsigned means 0 to some positive number, but stored in a signed integer "
since SUDS does not recognise unsigned data type since FORTRAN does not allow"
them"
***/**
155:resolutions
’8’ = "10 bit unsigned"
’T’ = "12 bit unsigned"
’l’ = "32 bit signed"
’m’ = "10 bit unsigned"
’s’ = "16 bit signed"
’t’ = "12 bit signed"

156:response_types
’a’ = "analog response in hertz"
’l’ = "Laplace transform analog response in radians per second"
’r’ = "ratio of gain to a Wood-Anderson seismograph"

’z’ = "Z-transform digital response"

157:rev_evidence

’c’ = "calibration signal"
’e’ = "explosion"
’i’ = "inconsistent mechanisms"
’t’ = "teleseism"
’w’ = "wiring error found"

158:rock_types

’S’ = "soil"
’a’ = "alluvium"
’b’ = "sand"
’g’ = "granite"
’i’ = "igneous rock"
’m’ = "metamorphic rock"
’s’ = "sedimentary rock"

159:sensor_types

’B’ = "bolometer"
’C’ = "local clock"
’H’ = "humidity"
’P’ = "pressure sensor"
’R’ = "rainfall"
’S’ = "linear strain meter"
’T’ = "temperature sensor"
’V’ = "volumetric strain meter"
’W’ = "wind"
’a’ = "accelerometer"
’c’ = "creep meter"
’d’ = "displacement sensor"
’g’ = "gravimeter"
’i’ = "tilt meter/inclinometer"
’m’ = "magnetic field"
’r’ = "radon sensor"
’s’ = "satellite time code"
’t’ = "tidal meter"
’v’ = "velocity seismometer"
’w’ = "torsion"
’x’ = "experimental"

160:site_conditions

’p’ = "permafrost"

161:site_types

’S’ = "source"
’e’ = "equipment, no sensor"
’r’ = "recorder"
’s’ = "sensor"

162:solution_qualities

’*’ = "8.5km< mean horiz error <=16km"
’0’ = "excellent"

'1' = "good"
'2' = "fair"
'3' = "poor"
'4' = "unacceptable"
'?' = "mean horiz error >16km"
'A' = "SEH SEZ <=1.34"
'B' = "SEH SEZ <=2.67"
'C' = "SEH SEZ <=5.35"
'D' = "SEH SEZ > 5.35"

163:solution_types
'C' = "catalog or final"
'I' = "ISC"
'N' = "NEIC"
'a' = "automatic"
'i' = "insufficent data"
'm' = "waiting for more data"
'n' = "not of key interest"
'p' = "preliminary"

164:spectra_types
'a' = "fourier amplitude"
'r' = "response"

165:spheroids
'_' = "none"

166:status
'a' = "active"
'p' = "planned"
'r' = "replaced by new name"

167:structure_names /* created by the manual compiler" */
104 = "sig_path_cmp",
105 = "signal_path",
106 = "mux_waveform",
107 = "waveform",
108 = "data_group",
109 = "response",
110 = "pick",
111 = "pick_residual",
112 = "event",
113 = "signif_event",
114 = "solution",
115 = "solution_err",
116 = "focal_mech",
118 = "vel_model",
125 = "lsa_detection",
126 = "lsa_setting",
130 = "clock_rate",
131 = "recorder",
300 = "site",
301 = "spectra",

307 = "magnitude",
308 = "processing",
309 = "polarity",
310 = "ssam_setup",
311 = "ssam_output",
312 = "map_element",
313 = "seismometer",
315 = "filter",
316 = "sig_path_ass",
320 = "calibration",
321 = "source",
322 = "user_vars",
323 = "service",
324 = "recorder_ass",
325 = "seismo_ass",
326 = "coordinate_sys",

168:survey_methods
’g’ = "GPS"
’m’ = "map"
’s’ = "survey"

169:synchronization_types
’W’ = "WWV"
’c’ = "comparision by computer"
’g’ = "guess"
’i’ = "Irig"
’p’ = "phase-lock loop"
’r’ = "rewiring"
’v’ = "visual"
’w’ = "WWVB"

170:taper_types
’l’ = "linear"

171:time_types
0 = "a floating-point number"
1 = "yrmodyhrmnsc.000"
2 = "yrmodyhrmnsc"
3 = "yr mo dy hr mn sc.000"
4 = "yr mo dy hr mn sc"
5 = "mo/dy/yr hr:mn sc.000 GMT"
6 = "mo/dy/yr hr:mn sc GMT"
7 = "month_name dy, year hr:mn sc.000 GMT"
8 = "month_name dy, year hr:mn sc GMT"
9 = "year/month_name/day/yrmody.hrmnsc"
10 = "yrmody.hrmnsc"

172:time_codes
’E’ = "IRIG-E"
’H’ = "IRIG-H"
’S’ = "BCD satellite"
’b’ = "WWVB"

'h' = "WWVH"
'l' = "local clock"
'r' = "WWV"
's' = "satellite"
'u' = "unknown, relative only"

173:time_corrects
'U' = "unknown, relative only"
'c' = "see comment structure"
'l' = "corrected to local clock"
'q' = "questionable"
's' = "corrected to satellite clock"
'u' = "uncorrected"

174:time_func_types
'b' = "half-boxcar"
's' = "half-sine"
't' = "triangle"
'z' = "trapezoid"

175:timing_qualities
'0' = "excellent"
'1' = "good"
'2' = "fair"
'3' = "poor"
'4' = "unacceptable"
'n' = "ignor timing"

176:trigger_types
'l' = "longterm/shortterm average"
't' = "teleseismic trigger"

177:units_types
'D' = "degrees"
'I' = "inches per second"
'M' = "miles"
'N' = "nanometers/sec/sec"
'S' = "samples per second"
'V' = "volts"
'a' = "analog"
'd' = "digital counts"
'e' = "millimeters"
'f' = "feet"
'g' = "nanometers"
'i' = "inches"
'k' = "kilometers"
'm' = "meters"
'n' = "nanometers/sec"
's' = "seconds"
'v' = "millivolts"

178:zero_weights
'B' = "boxcar weighting"

’D’ = "distance weighting"
’G’ = "dist too far but reduces gap"
’J’ = "Jeffrey’s weighting"
’M’ = "truncation weighting"
’R’ = "computed weight<0.0005"
’X’ = "critical station weighting"

DESCRIPTION

The code_list **data_types** is generated during compilation of the manual from **variable_info** and **structure_info**.

SEE ALSO

asc2field(3S) and (3S)field2asc(3S)

NAME

pc_codelists – initialization of codelists used in PC_SUDS

C_SYNOPSIS

1:ann_com

0 = "not given",

2:algorith

'e' = "eqdetect",
'm' = "mdetect",
'x' = "xdetect",

3:ampunits

'd' = "digital counts",
'm' = "millimeters on develocorder",
'n' = "nanometers (/sec or /sec/sec)",
'v' = "millivolts",

4:authority

0 = "not given",
101 = "calnet usgs menlo park, ca",
102 = "alaska net usgs menlo park, ca",
103 = "katmai net usgs menlo park, ca",
104 = "scalnet usgs pasadena, ca.",
120 = "shumagin net lamont palisades,ny",

5:comps

'E' = "east-west",
'N' = "north-south",
'V' = "vertical",
'e' = "east-west",
'n' = "north-south",
'v' = "vertical",

6:datasrc

'A' = "interactive automatic",
'a' = "automatic offline",
'i' = "interactive",
'r' = "realtime processor",

7:datatyp

'c' = "complex",
'd' = "double (64 bit IEEE real)",
'f' = "float (32 bit IEEE real)",
'i' = "16 bit signed stored as short int, -32767 to 32767",
'l' = "long (32 bit signed integer)",
'q' = "12 bit signed stored as short int, -2048 to 2048",
'r' = "12 bit data, 4 lsb time stored as short int",
's' = "12 bit unsigned stored as short int, 0 to 4096",
't' = "tensor",
'u' = "16 bit unsigned stored as short int, 0 to 65536",
'v' = "vector",

8:descript

'c' = "calibration",
'g' = "good",
't' = "telemetry noise",

9:enclosur

'B' = "building",
'D' = "concrete arch dam",
'b' = "buried",
'd' = "concrete gravity dam",
'e' = "earth fill dam",
'g' = "bridge",
'n' = "nuclear power plant",
's' = "on surface",
'v' = "underground vault",

10:equip_model

0 = "none given",
1 = "l4 geophone",
2 = "hs10 geophone",
3 = "ev17 geophone",
100 = "j302 amp/vco usgs",
101 = "j302ml amp/vco usgs",
102 = "j402 amp/vco usgs",
103 = "j402l amp/vco usgs",
104 = "j402h amp/vco usgs",
105 = "j402h3 amp/vco usgs",
106 = "j501 amp/vco usgs",
107 = "j502 amp/vco usgs",
108 = "j502a amp/vco usgs",
200 = "gs1 sum_amp usgs",
201 = "gs2 solar_sum_amp usgs",
300 = "r41f transmitter monitron",
301 = "r45f receiver monitron",
302 = "dt200 transmitter ritron",
303 = "dr200 receiver ritron",
304 = "ht200 transmitter motorola",
305 = "ht200 receiver motorola",
400 = "ca5-150h antenna scala",
401 = "ca5-150v antenna scala",
402 = "ca7-460 antenna scala",
403 = "cl-150hc antenna scala",
404 = "cl-150hr antenna scala",
405 = "cl-150v antenna scala",
406 = "ca5-450 antenna scala",
407 = "ra5-450 antenna scala",
408 = "pr-450u antenna scala",
500 = "1481 solar_panel arco",
501 = "435h solar_panel solarex",

11:equip_reason

0 = "none given",
1 = "initial installation",
2 = "routine site visit",

3 = "battery change needed",
4 = "vandalism",
5 = "flooding",
6 = "landslide",
7 = "seismic noise",
8 = "electronic noise",
9 = "seismically dead",
10 = "signal not zeroed",
11 = "signal too high",
12 = "signal too weak",
13 = "unit replacement",
14 = "reinstall after repair",
15 = "configuration change",

12:eventtyp
'E' = "explosion",
'N' = "nuclear",
'b' = "b-type",
'c' = "calibration",
'e' = "earthquake",
'f' = "free run",
'i' = "icequake",
'n' = "noise",
'r' = "regional",
't' = "teleseism",

13:feat_phase
0 = "not given",
1 = "window", /* amplitude is the duration of the window */
2 = "f finis", /* for coda mag, time when signal is about twice noise */
3 = "x maximum amplitude",
50 = "p first arrival", /* could be p, pn, pg, etc. */
51 = "p",
52 = "p*",
53 = "pp",
54 = "ppp",
55 = "pppp",
56 = "pps",
57 = "pg",
58 = "pn",
59 = "pdiffracted",
60 = "pcp",
61 = "pcppkp",
62 = "pcs",
63 = "pp",
64 = "ppp",
65 = "pkp",
66 = "pkppkp",
67 = "pkppks",
68 = "pkpsks",
69 = "pks",
70 = "ppks",
71 = "pkkp",

72 = "pkks",
73 = "pcppkp",
74 = "pcspkp",
100 = "s first s wave",
101 = "s",
102 = "s*",
103 = "ss",
104 = "sss",
105 = "ssss",
106 = "sg",
107 = "sn",
108 = "scs",
109 = "spcs",
110 = "ss",
111 = "sss",
112 = "ssss",
113 = "sscs",
114 = "scspkp",
115 = "scp",
116 = "sks",
117 = "skks",
118 = "skkks",
119 = "skssks",
120 = "skp",
121 = "skkp",
122 = "skkkp",
201 = "lg",
202 = "lr",
203 = "lr2",
204 = "lr3",
205 = "lr4",
206 = "lq",
207 = "lq2",
208 = "lq3",
209 = "lq4",
301 = "t",

14:firstm
'+' = "probable up",
'-' = "probable down",
'D' = "down",
'N' = "nodal",
'U' = "up",

/* instrument types: use a number less than 100 except in equipment structures */
15:inst_type
0 = "not specified",
1 = "sp usgs",
2 = "sp wwssn",
3 = "lp wwssn",
4 = "sp dwwssn",
5 = "lp dwwssn",

6 = "hglp lamont",
7 = "lp hglp lamont",
8 = "sp sro",
9 = "lp sro",
10 = "sp asro",
11 = "lp asro",
12 = "sp rstn",
13 = "lp rstn",
14 = "sp uofa U of alaska",
201 = "acceleration sensor",
202 = "velocity sensor",
203 = "displacement sensor",
204 = "strain sensor",
205 = "temperature sensor",
206 = "pressure sensor",
207 = "tilt sensor",
208 = "gravity sensor",
209 = "magnetic sensor",
210 = "radon sensor",
300 = "amplifier",
301 = "amp/vco",
302 = "filter",
303 = "summing amp",
304 = "transmitter",
305 = "receiver",
306 = "antenna",
307 = "battery",
308 = "solar cell",
309 = "discriminator",
310 = "discr. rack",
311 = "paper recorder",
312 = "film recorder",
313 = "smoked glass recorder",
314 = "atod converter",
315 = "computer",
316 = "clock",
317 = "time receiver",
318 = "magnetic tape",
319 = "magnetic disk",
320 = "optical disk",

16:mach_type
'6' = "80x86",

17:mechan
'e' = "explosive",
'n' = "normal",
's' = "strike-slip",
't' = "thrust",

18:medium
'r' = "rock",
's' = "soil",

19:moment_constr
'c' = "double couple",
'd' = "deviatoric",

20:moment_dtype
'1' = "polarities",
'2' = "amplitudes",
'3' = "polarities + amplitudes",
'4' = "waveforms",
'5' = "waveforms + polarities",
'6' = "waveforms + amplitudes",
'7' = "waveforms + polarities + amplitudes",

21:onsetz
'e' = "emersive",
'i' = "impulsive",

22:polar
'n' = "normal",
'r' = "reversed",

23:reco_type
'n' = "not known",

24:rock_class
'i' = "igneous",
'm' = "metamorphic",
's' = "sedimentary",

25:rock_type
0 = "none given",
1 = "soil",

26:sensor
'a' = "accelerometer",
'd' = "displacement sensor",
't' = "time",
'v' = "velocity seismometer",

27:site_condit
'p' = "permafrost",

28:stat_stat
'd' = "dead",
'g' = "good",

/* logical sum of these fields
***/**
29:stat_proc
0 = "none",
1 = "dc offset removed",
2 = "corrected for instr. resp.",
3 = "dc offset removed and corrected for instr. resp.",

4 = "filtered",
5 = "filtered and dc offset removed",
6 = "filtered and corrected for instr. resp.",
7 = "filtered, dc offset removed and corrected for instr. resp.",

30:struct_names

0 = "no struct",
1 = "station ident",
2 = "structure tag",
3 = "terminator",
4 = "equipment",
5 = "stationcomp",
6 = "muxdata",
7 = "descriptrace",
8 = "loctrace",
9 = "calibration",
10 = "feature",
11 = "residual",
12 = "event",
13 = "ev_descript",
14 = "origin",
15 = "error",
16 = "focalmech",
17 = "moment",
18 = "velmodel",
19 = "layers",
20 = "comment",
21 = "profile",
22 = "shotgather",
23 = "calib/points",
24 = "complex number",
25 = "triggers",
26 = "trigsetting",
27 = "eventsetting",
28 = "detector",
29 = "atodinfo",
30 = "timecorrection",
31 = "instrument",

31:syncs

’0’ = "total failure",
’1’ = "1 second synch",
’2’ = "10 second synch",
’3’ = "minute synch",
’4’ = "successful decode",
’5’ = "successful decode",

32:tecton

’N’ = "normal faulting",
’S’ = "strike-slip faulting",
’T’ = "thrust faulting",
’s’ = "subsidence",
’u’ = "uplift",

33:timequal

’0’ = "excellent",
’1’ = "good",
’2’ = "fair",
’3’ = "poor",
’4’ = "unacceptable",
’n’ = "ignor timing",

34:timings

’e’ = "external",
’i’ = "internal",

35:velfunc

’0’ = "constant",
’1’ = "linear",
’2’ = "exponential",

36:waterwave

’s’ = "seiche",
’t’ = "tsunami",

SUDS

Chapter 7: Mappings from other formats to SUDS

The final details of these mappings need to be worked out by users of these data sets and the implementer of the filters that convert to and from SUDS. Some minor additions to SUDS may still be necessary.

This page left blank intentionally.

NAME

ah_to_suds – mapping of AH (Lamont-Dougherty Ad Hoc) to SUDS

DESCRIPTION

MEMBERS

```
char    station.code[6];          = station.station_name       /* station code */
char    station.chan[6];          = signal_path.sensor_azimuth /* lpz,spn, etc. last char z, n, or e */
                                  = signal_path.sensor_dip
                                  = signal_path.component
char    station.stype[8];         = signal_path.sensor
                                  = signal_path.signal_name

float   station.slat;             = station.lat                /* station latitude */
float   station.slon;             = station.long               /*    "    longitude */
float   station.elev;             = station.elev               /*    "    elevation */
                                  = signal_path.total_resp_id
float   station.DS;               = response.maximum_gain      /* maximum gain at peak of calibration curve */
float   station.A0;               = response.normalization     /* normalization factor */
struct  station.calib cal[30];    = response_pz                /* calibration curve */

                                  event
float   event.lat;                = solution.origin_lat        /* event latitude */
float   event.lon;                = solution.origin_lon        /*   "   longitude */
float   event.dep;                = solution.origin_depth      /*   "   depth */
struct  event.time ot;            = solution.origin_time       /*   "   origin time */
char    event.ecomment[80];       = solution.comment           /* comment line*/

                                  data_group
short   record.type;              = waveform.data_type         /* data type */
long    record.ndata;             = waveform.data_length       /* number of samples */
float   record.delta;             = waveform.nom.dig.rate      /* sampling interval */
float   record.maxamp;            = waveform.max_val_data      /* maximum amplitude of record */
struct  record.time abstime;      = waveform.begin_time;       /* start time of record section */
float   record.rmin;              ??                           /* minimum value of abscissa */
char    record.rcomment[80];      = waveform.comment;          /* comment line */
char    record.log[202];          = processing;                /* log of data manipulations */
float   extra[21];                = user_var or comment        /* freebies */
```

SEE ALSO

st2ah(1), ah2st(1)

NAME

css_to_suds – mapping of CSS (Center for Seismic Studies) Database to SUDS

DESCRIPTION

The CSS database is described in **Center for Seismic Studies Version 3 Database: Schema Reference Manual,** by J. Anderson, W.E. Farrell, K. Garcia, J. Given, and H. Swanger, *Technical Report C90-01*, 61 pages, September, 1990.

lddate or load date is not included in SUDS structures because it is not as meaningful and is hard to enforce when the structures exist outside of the database. A database implementation can carry this information when needed. DB_VISTA for example allows structures or records to have a **timestamp**.

Question marks mean I was unclean on the CSS usage or there is another problem to resolve.

MEMBERS

Relation: affiliation
Description: Network station affiliations

char	affiliation.net[8]	= station.station_dc and authorities code_list
char	affiliation.sta[6]	= do this within SUDS
date	affiliation.lddate	= not included (see above)

Relation: arrival
Description: Summary information on a seismic arrival

char	arrival.sta[6]	= pick.signal_name[24]
double	arrival.time	= pick.time
long	arrival.arid	= pick.pick_id
long	arrival.jdate	= pick.time
long	arrival.stassid	= pick.signal_path_id
long	arrival.chanid	= pick.signal_name[24]
char	arrival.chan[8]	= pick.signal_name[24]
char	arrival.iphase[8]	= pick.observ_phase
char	arrival.stype	= pick.?
float	arrival.deltim	= pick_residual.residual
float	arrival.azimuth	= pick.obs_azimuth
float	arrival.delaz	= pick_residual.azm_2_stat
float	arrival.slow	= pick.obs_slowness
float	arrival.delslo	= pick_residual.
float	arrival.ema	= pick_residual.angle_emerg
float	arrival.rect	= pick.rectilinearity
float	arrival.amp	= pick.amplitude
float	arrival.per	= pick.frequency
float	arrival.logat	= pick.?
char	arrival.clip	= waveform.num_pos_clip
char	arrival.fm[2]	= pick.first_motion, pick.onset_type
float	arrival.snr	= pick.signal_2_noise
char	arrival.qual	= pick.obs_time_qual
char	arrival.auth[15]	= pick.authority
long	arrival.commid	= pick.comment_id
date	arrival.lddate	= not included (see above)

Relation: assoc
Description: Data associating arrivals with origins

long	assoc.arid	= pick_residual.pick_id
long	assoc.orid	= pick_residual.solution_id
char	assoc.sta[6]	= pick.signal_name[24]
char	assoc.phase[8]	= pick.observ_phase

float	assoc.belief	=
float	assoc.delta	= pick_residual.dist_2_stat
float	assoc.seaz	= pick_residual.?
float	assoc.esaz	= pick_residual.azm_2_stat
float	assoc.timeres	= pick_residual.residual
char	assoc.timedef	= pick_residual.?
float	assoc.azres	= pick_residual.?
char	assoc.azdef	= pick_residual.?
float	assoc.slores	= pick_residual.?
char	assoc.slodef	= pick_residual.?
float	assoc.emares	= pick_residual.?
float	assoc.wgt	= pick_residual.weight_used
char	assoc.vmodel[15]	= pick_residual.vel_model_id
long	assoc.commid	= pick_residual.comment_id
date	assoc.lddate	= not included (see above)

Relation: event
Description: Event identification

long	event.evid	= event.event_id
char	event.evname[15]	= signif_event.eq_name
long	event.prefor	= event.cat_sol_id
char	event.auth[15]	= solution.authority
long	event.commid	= event,comment_id
date	event.lddate	= not included (see above)

Relation: gregion
Description: Geographic region

long	gregion.grn	= use code_list
char	gregion.grname[40]	= use code_list
date	gregion.lddate	= not included (see above)

Relation: instrument
Description: Generic (default) calibration information about a station

long	instrument.inid	= sensor_id, signal_path_id, recorder_id
char	instrument.insname[50]	= code_list equip_models
char	instrument.instype[6]	= signal_path.signal_name[24]
char	instrument.band	= signal_path.band_type
char	instrument.digital	=
float	instrument.samprate	= waveform.nom_dig_rate
float	instrument.ncalib	= calibration.amplitude
float	instrument.ncalper	= calibration.frequency
char	instrument.dir[64]	= calibration_id
char	instrument.dfile[32]	= calibration_id
char	instrument.rsptype[6]	= calibration_id
date	instrument.lddate	= not included (see above)

lastid.Relation: lastid
lastid.Description: Counter values (Last value used for keys)

char	lastid.keyname[15]	= ?
long	lastid.keyvalue	= ?
date	lastid.lddate	= not included (see above)

Relation: netmag

Description: Network magnitude

long	netmag.magid	= magnitude.magnitude_id
char	netmag.net[8]	= magnitude.magnitude_dc
long	netmag.orid	= solution.solution_id
long	netmag.evid	= solution.event_id
char	netmag.magtype[6]	= magnitude.magnitude_type
long	netmag.nsta	= magnitude.num_used
float	netmag.magnitude	= magnitude.magnitude
float	netmag.uncertainty	= magnitude.mag_error
char	netmag.auth[15]	= magnitude.authority
long	netmag.commid	= magnitude.comment_id
date	netmag.lddate	= not included (see above)

Relation: network

Description: Network description and identification

char	network.net[8]	= codelist authorities 5 letters only
char	network.netname[80]	= codelist authorities, any length
char	network.nettype[4]	= NOT AVAILABLE is this necessary?
char	network.auth[15]	= codelist authorities, any length
long	network.commid	= station.comment_id
date	network.lddate	= not included (see above)

Relation: origerr

Description: Summary of errors in origin estimations

long	origerr.orid	= solution_err.solution_id
float	origerr.sxx	= solution_err.covar_xx
float	origerr.syy	= solution_err.covar_yy
float	origerr.szz	= solution_err.covar_zz
float	origerr.stt	= solution_err.covar_tt
float	origerr.sxy	= solution_err.covar_xy
float	origerr.sxz	= solution_err.covar_xz
float	origerr.syz	= solution_err.covar_yz
float	origerr.stx	= solution_err.covar_tx
float	origerr.sty	= solution_err.covar_ty
float	origerr.stz	= solution_err.covar_tz
float	origerr.sdobs	= solution_err.std_error
float	origerr.smajax	= solution_err.semi_major
float	origerr.sminax	= solution_err.semi_minor
float	origerr.strike	= solution_err.major_strike
float	origerr.sdepth	= solution_err.depth_error
float	origerr.stime	= solution_err.time_error
float	origerr.conf	= solution_err.confidence
long	origerr.commid	= solution_err.comment_id
date	origerr.lddate	= not included (see above)

Relation: origin

Description: Data on event location and confidence bounds

float	origin.lat	= solution.origin_lat
float	origin.lon	= solution.origin_long
float	origin.depth	= solution.origin_depth
double	origin.time	= solution.origin_time
long	origin.orid	= solution.solution_id
long	origin.evid	= solution.event_id

long	origin.jdate	= solution.origin_time	
long	origin.nass	= solution.num_p_rep_good etc	
long	origin.ndef	= solution.num_p_used etc	
long	origin.ndp	= solution.?	
long	origin.grn	= solution.region	
long	origin.srn	= solution. both needed?	
char	origin.etype[7]	= solution.?	
float	origin.depdp	= solution.?	
char	origin.dtype	= solution.?	
float	origin.mb	= magnitude.magnitude_id	Any number of different
long	origin.mbid	= magnitude.magnitude_id	types of magnitude can
float	origin.ms	= magnitude.magnitude_id	be stored for this
long	origin.msid	= magnitude.magnitude_id	solution.
float	origin.ml	= magnitude.magnitude_id	
long	origin.mlid	= magnitude.magnitude_id	
char	origin.algorithm[15]	= solution.hypo_program	
char	origin.auth[15]	= solution.authority	
long	origin.commid	= solution.comment_id	
date	origin.lddate	= not included (see above)	

Relation: remark
Description: Comments

long	remark.commid	= all suds structures can have comment
long	remark.lineno	= structures following them uniquely
char	remark.remark[80]	= identified by comment_id and comment_dc
date	remark.lddate	= not included (see above)

Relation: sensor
Description: Specific calibration information for physical channels

char	sensor.sta[6]	= signal_path.station_name[8]
char	sensor.chan[8]	= signal_path.signal_name[20]
double	sensor.time	= signal_path.from_time
double	sensor.endtime	= signal_path.thru_time
long	sensor.inid	= signal_path.sensor_id ?
long	sensor.chanid	= signal_path.signal_path_id
long	sensor.jdate	= signal_path.from_time ?
float	sensor.calratio	= response.maximum_gain
float	sensor.calper	= response.frequency_max
float	sensor.tshift	= signal_path.?
char	sensor.instant	= signal_path.?
date	sensor.lddate	= not included (see above)

Relation: site
Description: Station location information

char	site.sta[6]	= station.station_name[8]
long	site.ondate	= station.from_time
long	site.offdate	= station.thru_time
float	site.lat	= station.station_lat
float	site.lon	= station.station_long
float	site.elev	= station.station_elev
char	site.staname[50]	= station.site_descrip[40]
char	site.statype[4]	= station.?
char	site.refsta[6]	= station.ref_stat_id

float	site.dnorth	= station.dist_north
float	site.deast	= station.dist_east
date	site.lddate	= not included (see above)

Relation: sitechan
Description: Station-channel information

char	sitechan.sta[6]	= signal_path.station_name[8]
char	sitechan.chan[8]	= signal_path.signal_name[20]
long	sitechan.ondate	= signal_path.from_time
long	sitechan.chanid	= signal_path.signal_name[20]
long	sitechan.offdate	= signal_path.thru_time
char	sitechan.ctype[4]	= signal_path.sensor_type, band_type, gain_type, etc
float	sitechan.edepth	= signal_path.sensor_depth
float	sitechan.hang	= signal_path.sensor_azimuth
float	sitechan.vang	= signal_path.sensor_dip
char	sitechan.descrip[50]	= station.site_descrip[40] or signal_path.comment_id
date	sitechan.lddate	= not included (see above)

Relation: sregion
Description: Seismic region

long	sregion.srn	= solution.region, solution.region_type
char	sregion.srname[40]	= use code_list for names
date	sregion.lddate	= not included (see above)

Relation: stamag
Description: Station magnitude

long	stamag.magid	= pick_residual.pick_id
char	stamag.sta[6]	= signal_path.station_name
long	stamag.arid	= pick.pick_id
long	stamag.orid	= pick_residual.solution_id
long	stamag.evid	= solution.event_id
char	stamag.phase[8]	= pick_residual.pick_id
char	stamag.magtype[6]	= pick_residual.mag_type
float	stamag.magnitude	= pick_residual.magnitude
float	stamag.uncertainty	= pick_residual.?
char	stamag.auth[15]	= pick_residual.authority
long	stamag.commid	= pick_residual.comment_id
date	stamag.lddate	= not included (see above)

Relation: stassoc
Description: Arrivals from a single station grouped into an event

long	stassoc.stassid	=
char	stassoc.sta[6]	=
char	stassoc.etype[7]	= probably need a new structure to do this
char	stassoc.location[32]	=
float	stassoc.dist	=
float	stassoc.azimuth	=
float	stassoc.lat	=
float	stassoc.lon	=
float	stassoc.depth	=
double	stassoc.time	=
float	stassoc.imb	=
float	stassoc.ims	=

float	stassoc.iml	=
char	stassoc.auth[15]	=
long	stassoc.commid	=
date	stassoc.lddate	= not included (see above)

Relation: wfdisc
Description: Waveform file header and descriptive information

char	wfdisc.sta[6]	= waveform.station_name[8]
char	wfdisc.chan[8]	= waveform.signal_name[20]
double	wfdisc.time	= waveform.begin_time
long	wfdisc.wfid	= waveform.waveform_id
long	wfdisc.chanid	= waveform.signal_path_id
long	wfdisc.jdate	= waveform.begin_time
double	wfdisc.endtime	= waveform.end_time
long	wfdisc.nsamp	= waveform.data_length
float	wfdisc.samprate	= waveform.nom_dig_rate
float	wfdisc.calib	= response.maximum_gain
float	wfdisc.calper	= response.frequency_max
char	wfdisc.instype[6]	= signal_path.sensor_type etc.
char	wfdisc.segtype	= waveform.?
char	wfdisc.datatype[2]	= waveform.data_type
char	wfdisc.clip	= waveform.num_pos_clip etc.
char	wfdisc.dir[64]	= data_group.online_path
char	wfdisc.dfile[32]	= data_group.data_group_id
long	wfdisc.foff	= waveform.file_offset
long	wfdisc.commid	= waveform.comment_id
date	wfdisc.lddate	= not included (see above)

Relation: wftag
Description: Waveform mapping file

char	wftag.tagname[8]	= not supported. Could make an abbreviation
long	wftag.tagid	= structure if important or could use
long	wftag.wfid	= code_list
date	wftag.lddate	= not included (see above)

Relation: wftape
Description: Waveform tape file header and descriptive information

char	wftape.sta[6]	= waveform.station_name[8]
char	wftape.chan[8]	= waveform.signal_name[20]
double	wftape.time	= waveform.begin_time
long	wftape.wfid	= waveform.waveform_id
long	wftape.chanid	= waveform.signal_path_id
long	wftape.jdate	= waveform.begin_time
double	wftape.endtime	= waveform.end_time
long	wftape.nsamp	= waveform.data_length
float	wftape.samprate	= waveform.nom_dig_rate
float	wftape.calib	= response.maximum_gain
float	wftape.calper	= response.frequency_max
char	wftape.instype[6]	= signal_path.sensor_type etc.
char	wftape.segtype	= waveform.?
char	wftape.datatype[2]	= waveform.data_type
char	wftape.clip	= waveform.num_pos_clip etc.
char	wftape.dir[64]	= data_group.online_path

char	wftape.dfile[32]	= data_group.data_group_id
char	wftape.volname[6]	= data_group.media_label
long	wftape.tapefile	= data_group.media_path
long	wftape.tapeblock	= data_group.media_block
long	wftape.commid	= waveform.comment_id
date	wftape.lddate	= not included (see above)

SEE ALSO

BUGS

NAME

css_to_suds – mapping of CUSP Database to SUDS

DESCRIPTION

The following FORTRAN structures are designed by Allan Walters, of the USGS Menlo Park for use in his programs that read original VAX-formatted CUSP data on SUN computers for the purpose of translating these data into other formats.

MEMBERS

```
STRUCTURE /TAG/
  UNION
    MAP
      INTEGER*4          TID      structure id ; type
    END MAP
    MAP
      BYTE               C4(4)    3-character id
    END MAP
  END UNION
END STRUCTURE

STRUCTURE /CSTR4/
  INTEGER*4              NC       characters
  BYTE C4(4)             4-byte string
END STRUCTURE

STRUCTURE /CSTR8/
  INTEGER*4              NC       characters
  BYTE C8(8)             8-byte string
END STRUCTURE

STRUCTURE /DATETIME/                                        = MS_TIME in suds
  INTEGER*4              DATE     Year,month,day
  INTEGER*4              HRMN     hour,minute
  REAL*4                 SEC      seconds
END STRUCTURE

HID: Event id summary data structure
STRUCTURE /HID/
  RECORD /TAG/           TAG      Structure id              = not applicable
  INTEGER*4              NB       total bytes in header     = not applicable
  INTEGER*4              ID       mem id                    = ??
  RECORD /CSTR4/         WHO      id of analyst             = solution.authority
  RECORD /DATETIME/      T        time header creation      = ??
  INTEGER*4              FILL(8)  undefined fields
END STRUCTURE

HST: Event set specific data structure
STRUCTURE /HST/
  RECORD /TAG/           TAG      Structure id              = not applicable
  INTEGER*4              SET      set number                = waveform.data_group_id
  RECORD /DATETIME/      T        set start date-time       = waveform.begin_time
  RECORD /CSTR4/         NET      network name              = signal_path_dc
  RECORD /CSTR4/         DEV      device name               = signal_path.signal_name
  REAL*4                 DT       secs/sample               = waveform.nom_dig_rate
  INTEGER*4              INC      bytes/sample              = waveform.data_type
```

```
  INTEGER*4          B        digitizer bits                   = ??
  INTEGER*4          MC       max counts                       = ??
  REAL*4             VM       max volts                        = ??
  INTEGER*4          SYN      time code synch ident            = ??
  INTEGER*4          FILL(1)  undefined fields
END STRUCTURE

HPN: Event pin specific data structure
STRUCTURE /HPN/
  RECORD /TAG/       TAG      Structure id                     = not applicable
  INTEGER*4          SET      set number                       = waveform.data_group_id
  INTEGER*4          PIN      pin number                       = signal_path.record_num_in
  RECORD /CSTR8/     NAM      site name 6 bytes                = signal_path.station_name
  RECORD /CSTR4/     TYP      component 3 bytes                = signal_path.signal_name
  INTEGER*4          RTC      dig time cnts to 1st sample      = waveform.begin_time
  INTEGER*4          KEY      grm offset=HID.NB+HPN.KEY bytes  = waveform.file_offset
  INTEGER*4          N        length of trace (bytes)          = waveform.data_length
  INTEGER*4          MSK      triggering mask                  = ??
  INTEGER*4          TRC      digitizer time cnts of trigger   = ??
  INTEGER*4          FILL(3)  undefined fields
END STRUCTURE
End of GRM file header section.

HHY : Event hypocenter summary structure
STRUCTURE /HHY/
  RECORD /TAG/       TAG      Structure id                     = not applicable
  INTEGER*4          MSK      event type; fix                  = solution.origin_status
  RECORD /DATETIME/  T        origin time                      = solution.origin_time
  REAL*4             LAT      latitude                         = solution.origin_lat
  REAL*4             LON      longitude                        = solution.origin_long
  REAL*4             Z        depth (- down)                   = solution.origin_depth
  REAL*4             RMS      rms of solution                  = solution.rms_of_resids
  INTEGER*4          NP       num phases in solut              = solution.num_p_used etc.
  REAL*4             GAP      azimuthal gap                    = solution.gap_of_stations
  REAL*4             DMN      dist to closest stn              = solution.dist_near_stat
  REAL*4             ELT      error lat                        = solution.horiz_error also solution_err
  REAL*4             ELN      error lon                        = solution.horiz_error also solution_err
  REAL*4             EZ       error depth                      = solution.depth_error also solution_err
  REAL*4             ET       error time                       = solution_err.time_err
END STRUCTURE

HMG: Event magnitude summary structure
STRUCTURE /MAG/
  REAL*4             M        magnitude                        = magnitude.mag_value
  INTEGER*4          NP       number of phases                 = magnitude.num_used
  REAL*4             RMS      rms of calculation               = magnitude.rms_of_mag
END STRUCTURE

STRUCTURE /HMG/
  RECORD /TAG/       TAG      Structure id                     = not applicable
  RECORD /MAG/       MD       coda duration mag                = magnitude.mag_value
                                                               magnitude.magnitude_type
  RECORD /MAG/       MC       coda amplitude mag               = magnitude.mag_value
```

			magnitude.magnitude_type
RECORD /MAG/	ML	wood-anderson mag	= magnitude.mag_value magnitude.magnitude_type
RECORD /MAG/	MH	helicorder magnitude	= magnitude.mag_value magnitude.magnitude_type
RECORD /MAG/	M	other defined mag	= magnitude.mag_value magnitude.magnitude_type

END STRUCTURE

HPX: Pin phase arrival time data structure
STRUCTURE /HPX/

RECORD /TAG/	TAG	Structure id	= not applicable
INTEGER*4	SET	set number	= waveform.data_group_id
INTEGER*4	PIN	pin number	= signal_path.record_num_in
INTEGER*4	RTC	samp count of arrival	= pick.time
RECORD/DATETIME/	T	time of arrival	= pick.time
RECORD/CSTR8/	PHZ	phase descriptor	= pick.observ_phase
INTEGER*4	AZ	azm to station(0 = N)	= pick_residual.azm_2_stat
INTEGER*4	IA	take-off angl(0 = UP)	= pick_residual.angle_emerg
REAL*4	X	distance to stn (KM)	= pick_residual.dist_2_stat
REAL*4	TTR	traveltime resid (S)	= pick_residual.residual
REAL*4	TC	delay time correct(S)	= pick_residual.delay_used
REAL*4	ERR	timing error est(S)	= pick_residual. ??

END STRUCTURE

HCD: Pin coda duration data structure SC=sample count
STRUCTURE /HCD/

RECORD /TAG/	TAG	Structure id	= not applicable
INTEGER*4	SET	set number	= waveform.data_group_id
INTEGER*4	PIN	pin number	= signal_path.record_num_in
INTEGER*4	RTC	SC start of coda	= pick.time
INTEGER*4	NEQ	number of coda windows	= ??
REAL*4	AMP	amp of S in digital counts	= pick.amplitude
REAL*4	AFX	nominal minimun ampl	= ??
REAL*4	QFX	fixed coda decay const	= ??
REAL*4	AFR	free amplitude	= pick.amplitude
REAL*4	QFR	free coda decay const	= ??
REAL*4	RMS	residual of fit	= ??
REAL*4	TAU	length of coda	= pick.time
REAL*4	RBB	amp of final sample	= ??
RECORD/CSTR8/	PHZ	phase descriptor	= pick.observ_phase etc.

END STRUCTURE

HAP: Pin amplitude/period data structure SC=sample count
STRUCTURE /HAP/

RECORD /TAG/	TAG	Structure id	= not applicable
INTEGER*4	SET	set number	= waveform.data_group_id
INTEGER*4	PIN	pin number	= signal_path.record_num_in
INTEGER*4	RTC0	SC start of window	= ??
INTEGER*4	A1	min/max amp at RTC2 (DC/microns)	= ??
INTEGER*4	A2	min/max amp at RTC4 (DC/microns)	= ??
INTEGER*4	DC	window bias offset digital counts	= ??
INTEGER*4	WIN	offset from start window	= ??

INTEGER*4	RTC1	1st zero crossing SC	= ??
INTEGER*4	RTC2	1st max/min SC	= ??
INTEGER*4	RTC3	2nd zero crossing SC	= ??
INTEGER*4	RTC4	2nd max/min SC	= ??
INTEGER*4	RTC5	3rd zero crossing SC	= ??
RECORD /CSTR8/	PHZ	phase descriptor	= pick.observ_phase etc.
END STRUCTURE	period=(RTC5-RTC1)*HST.DT		= pick.frequency

SEE ALSO

BUGS

NAME

sac_to_suds – mapping of SAC (Seismic Analysis Code) to SUDS

DESCRIPTION

The Seismic Analysis Code is a widely used program described in **Users manual,** by Joseph E. Tull, *Lawrence Livermore National Laboratory, MS L-208, Livermore, CA 94550,* September 15, 1987 and other documents. The variables listed are for header version 6.

MEMBERS

int	NPTS	= waveform.data_length
float	B	= waveform.begin_time
float	E	= waveform.end_time
enum	IFTYPE	= waveform.data_type
logical	LEVEN	= assumed true, should we allow uneven spacing?
float	DELTA	= waveform.nom_dig_rate
float	ODELTA	= waveform.prec_dig_rate
enum	IDEP	= waveform.data_units
float	SCALE	= needed?
float	DEPMIN	= waveform.min_val_data
float	DEPMAX	= waveform.max_val_data
float	DEPMEN	= needed?
int	NZYEAR	= waveform.begin_time
int	NZJDAY	= waveform.begin_time
int	NZHOUR	= waveform.begin_time
int	NZMIN	= waveform.begin_time
int	NZSEC	= waveform.begin_time
int	NZMSEC	= waveform.begin_time
int	NZDTTM	= waveform.begin_time
auxil	KADATE	= waveform.begin_time
auxil	KATIME	= waveform.begin_time
float	O	= solution.origin_time
auxil	KO[8]	= needed?
float	A	= pick.time
auxil	KA[8]	= pick.observ_phase
float	F	= pick.time pick.observ_phase = coda length
auxil	KF[8]	= from pick_types code_list
float	T[10]	= pick.time
auxil	KT[10][8]	= pick.observ_phase
int	IZTYPE	= not needed since times are absolute
char	KINST[8]	= waveform.signal_name signal_path.sensor_type etc
int	IINST	= recorder.recorder_id
float	RESP[10]	= response. various values
char	KNETWK[8]	= in domain values
char	KSTNM[8]	= waveform.station_name[8]
int	ISTREG	= station.region
float	STLA	= station.station_lat
float	STLO	= station.station_long
float	STEL	= station.station_elev
float	STDP	= signal_path.sensor_depth
float	CMPAZ	= signal_path.sensor_azimuth
float	CMPINC	= signal_path.sensor_dip
char	KCMPNM[8]	= signal_path.signal_name[20]
auxil	KSTCMP	= signal_path.signal_name[20]
logical	LPSPOL	= needed, calculate from signal_path.polarity ?
char	KEVNM[16]	= signif_event.eq_name[20]

int	IEVREG	= solution.region
float	EVLA	= solution.origin_lat
float	EVLO	= solution.origin_long
float	EVEL	= event_source.origin_depth
float	EVDP	= solution.origin_depth
int	IEVTYP	= event.event_type
char	KHOLE[8]	= event_source.event_name[12]
float	DIST	= pick_residual.dist_2_stat
float	AZ	= pick_residual.azm_2_stat
float	BAZ	= pick_residual.
float	GCARC	= needed?
logical	LCALDA	= ?
int	IQUAL	= pick.obs_time_qual
int	ISYNTH	= waveform.signal_type
char	KDATRD[8]	= should this be included?
float	USER[10]	= user_vars
char	KUSER[3][8]	= comment_id
logical	LOVROK	= not applicable?
int	NVHDR	= not applicable

SEE ALSO

BUGS

NAME

seed_to_suds – mapping of SEED (Standard for the Exchange of Earthquake Data) to SUDS

DESCRIPTION

SEED allows any arbitrary data format for time series, requiring the reading programs to be fairly complex. SUDS, for simplicity of data use, requires that any type of data be written in XDR binary format if it is to be used on many machines and with most utility programs. SUDS allows writing data in native binary format if there is no time for conversion on the recording computer, but SUDS requires conversion to XDR binary format before the data are used with most utility programs on a different type of computer.

SUDS allows multiplexed data but strongly encourages that it be used only for transfering data from the digitizing system to the first other system that uses the data, whereupon the data are demultiplexed. Nearly all utility programs use demultiplexed data.

SEED allows several blockettes with similar function, but slightly varying implementation. SUDS permits this but strongly discourages it in order to minimize specialized programming to handle several flavors of the same basic concept. Thus SUDS structures tend to be more versatile and lengthy than SEED blockettes.

Station/Component Identifiers

SEED

- Network Identifier Code, given in abbreviation table
- Station_name, 5 letters, globally unique
- Location_Id, 2 letter array subcode
- Channel_Id, 3 letters
  - Band_code, passband of instrument
  - Source_code, sensor family
  - Orientation_code, axis of sensor

SUDS

- Network or domain, globally unique number and 6 letter abbreviation
- Station_name, 7 letter, unique to network
- Component, 1 letter, orientation of sensor axis
- Sensor, 1 letter, type of sensor
- Band_type, 1 letter, passband and general digitization rate
- Gain_type, 1 letter, which output of multigain sensor or amplifier
- Path_type, 1 letter, unique to network to designate different signal_paths

These fields are catenated into a 19 letter signal_name as net_stat_CSBP

SEED Types

- A Fixed length character string, length to 5
- D Integer, length 1-4 can be a short, length 5-7 must be a long
- F Floating point number
- V Variable length character string, length to 70

SEED Lengths are in numbers of bytes, typically ASCII characters

MEMBERS

SEED Blockettes and Fields	Type	Length	SUDS Structures and Members
5 Field Volume Id Blockette			
1 Blockette type - 005	D	3	= not applicable
2 Length of blockette	D	4	= not applicable
3 Version of format	D	4	= contained in structure_tag
4 Logical record length	D	2	= not applicable

5 Beginning of volume	V	1-22	= waveform.begin_time

8 Telemetry Volume Id Blockette

1 Blockette type - 008	D	3	= not applicable
2 Length of blockette	D	4	= not applicable
3 Version of format	D	4	= contained in structure_tag
4 Logical record length	D	2	= not applicable
5 Station Id	A	5	= station.station_name
6 Location Id	A	2	= add to station_name
7 Channel Id	A	3	= signal_path.component, sensor, band_type
8 Beginning of volume	V	1-22	= waveform.begin_time
9 End of volume	V	1-22	= waveform.end_time
10 Station info effective	V	1-22	= station.from_time
11 Channel info effective	V	1-22	= signal_path.from_time

10 Volume Id Blockette

1 Blockette type - 010	D	3	= not applicable
2 Length of blockette	D	4	= not applicable
3 Version of format	D	4	= contained in structure_tag
4 Logical record length	D	2	= not applicable
5 Beginning time	V	1-22	= signal_path or station from_time
6 End time	V	1-22	= signal_path or station thru_time

11 Volume Station Header Index Blockette

1 Blockette type - 011	D	3	= not applicable
2 Length of blockette	D	4	= not applicable
3 Number of stations	D	3	= not applicable
Repeat 4-5 for each station			
4 Station Id code	A	5	= not applicable
5 Sequence of station header	D	6	= not applicable

12 Volume Time Span Index Blockette

1 Blockette type - 012	D	3	= not applicable
2 Length of blockette	D	4	= not applicable
3 Number of spans in table	D	4	= not applicable
Repeat 4-6 for each span			
4 Beginning of span	V	1-22	= not applicable
5 End of span	V	1-22	= not applicable
6 Sequence of time span header	D	6	= not applicable

30 Data Format Dictionary Blockette

1 Blockette type - 030	D	3	= not applicable
2 Length of blockette	D	4	= not applicable
3 Short descriptive name	V	1-50	= not applicable
4 Data format lookup code	D	4	= not applicable
5 Data family type	D	3	= not applicable
6 Number of decoder keys	D	2	= not applicable
Repeat 7 for each decoder key			
7 Decoder keys	V	any	= not applicable

In SUDS waveforms and all structures can be written in the native format of any machine as specified in the structure_tag. The input/output routines currently on recognise XDR or the native format of the machine used for analysis. Where write-time efficiency is a serious problem, a filter might be written to convert format at a later time.

31 Comment Description Blockette

1 Blockette type - 031	D	3	= not applicable
2 Length of blockette	D	4	= not applicable
3 Comment code key	D	4	= not applicable
4 Comment class code	A	1	= not applicable
5 Description of comment	V	1-70	= add comments where appropriate to
6 Units of comment level	D	3	= waveform, station, signal_path, etc.

32 Cited Source Dictionary Blockette

1 Blockette type - 032	D	3	= not applicable
2 Length of blockette	D	4	= not applicable
3 Source lookup code	D	2	= put in comment structure or code_lists
4 Name of publication/author	V	1-70	= put in comment structure for station
5 Date published/catalog	V	1-70	= put in comment structure for station
6 Publisher name	V	1-50	= put in comment structure for station

33 Generic Abbreviation Blockette

1 Blockette type - 033	D	3	= not applicable
2 Length of blockette	D	4	= not applicable
3 Abbreviation lookup code	D	3	= should be entered in code_lists
4 Abbreviation description	V	1-50	= should be entered in code_lists

34 Units Abbreviations Blockette

1 Blockette type - 034	D	3	= not applicable
2 Length of blockette	D	4	= not applicable
3 Unit lookup code	D	3	= should be entered in code_lists
4 Unit name	V	1-20	= should be entered in code_lists
5 Unit description	V	0-50	= should be entered in code_lists

35 Beam Configuration Blockette

1 Blockette type - 035	D	3	= not applicable
2 Length of blockette	D	4	= not applicable
3 Beam lookup code	D	3	= signal_path.path_type or station_name
4 Number of components	D	4	= signal_path.data_length
Repeat 5-9 for number of components			
5 Station Id	A	5	= station.station_name
6 Location Id	A	2	= add to station_name
7 Channel Id	A	3	= signal_path.component, sensor, band_type
8 Sub-channel Id	D	4	= beam_comp.beam_comp_id
9 Component weight	D	5	= beam_comp.delay

43 Response (Poles & Zeros) Dictionary Blockette

1 Blockette type - 043	D	3	= not applicable
2 Length of blockette	D	4	= not applicable
3 Response lookup key	D	4	= response.response_id
4 Response name	V	1-25	= response.name_response
5 Response type	A	1	= response.response
6 Stage signal input units	D	3	= response.input_units
7 Stage signal output units	D	3	= response.output_units
8 AO normalization factor 1.0 if none	F	12	= response.normalization

9 Normalization frequency	F	12	= response.frequency_max
10 Number of complex zeros	D	3	= response.data_length
Repeat 11-14 for each response			
11 Real zero	F	12	= response_pz.zero_r
12 Imaginary zero	F	12	= response_pz.zero_i
13 Real zero error	F	12	= response_pz.zero_err_r
14 Imaginary zero error	F	12	= response_pz.zero_err_i
15 Number of complex poles	D	3	= response.data_length
Repeat 16-19 for each response			
16 Real pole	F	12	= response_pz.pole_r
17 Imaginary pole	F	12	= response_pz.pole_i
18 Real pole error	F	12	= response_pz.pole_err_r
19 Imaginary pole error	F	12	= response_pz.pole_err_i

44 Response (Coefficients) Dictionary Blockette

1 Blockette type - 044	D	3	= not applicable
2 Length of blockette	D	4	= not applicable
3 Response lookup key	D	4	= response.response_id
4 Response name	V	1-25	= response.name_response
5 Response type	A	1	= response.response
6 Signal input units	D	3	= response.input_units
7 Signal output units	D	3	= response.output_units
8 Number of numerators	D	3	= response.data_length
Repeat 9-10 for each numerator			
9 Numerator coefficient	F	12	= response_fir.numer_coef
10 Numerator error	F	12	= response_fir.numer_coef_err
11 Number of denominators	D	3	= response.data_length
Repeat 12-13 for each denominator			
12 Denominator coefficient	F	12	= response_fir.denom_coef
13 Denominator error	F	12	= response_fir.denom_coef_err

45 Response List Dictionary Blockette

1 Blockette type - 045	D	3	= not applicable
2 Length of blockette	D	4	= not applicable
3 Response lookup key	D	4	= response.response_id
4 Response name	V	1-25	= response.name_response
5 Signal input units	D	3	= response.input_units
6 Signal output units	D	3	= response.output_units
7 Number of responses listed	D	4	= response.data_length
Repeat 8-12 for each response			
8 Frequency (Hz)	F	12	= response_fap.frequency
9 Amplitude	F	12	= response_fap.amplitude
10 Amplitude error	F	12	= response_fap.amplitude_err
11 Phase angle (degrees)	F	12	= response_fap.phase
12 Phase error (degrees)	F	12	= response_fap.phase_error

46 Generic Response Dictionary Blockette

1 Blockette type - 046	D	3	= not applicable
2 Length of blockette	D	4	= not applicable
3 Response lookup key	D	4	= response.response_id
4 Response name	V	1-25	= response.name_response
5 Signal input units	D	3	= response.input_units
6 Signal output units	D	3	= response.output_units

7 Number of corners listed	D	4	= response.data_length
Repeat 8-9 for each response			
8 Corner frequency (Hz)	F	12	= response_cfs.corner_freq
9 Corner slope (db/decade)	F	12	= response_cfs.db_per_decade

47 Decimation Dictionary Blockette

1 Blockette type - 047	D	3	= not applicable
2 Length of blockette	D	4	= not applicable
3 Response lookup key	D	4	= response.response_id
4 Response name	V	1-25	= response.name_response
5 Input sample rate	F	10	= response.inp_samp_rate
6 Decimation factor	D	5	= response.decim_factor
7 Decimation offset	D	5	= response.decim_offset
8 Estimated delay (seconds)	F	12	= response.estim_delay
9 Correction applied (secs)	F	12	= response.used_delay

48 Channel Sensitivity/Gain Dictionary Blockette

1 Blockette type - 048	D	3	= not applicable
2 Length of blockette	D	4	= not applicable
3 Response lookup key	D	4	= response.response_id
4 Response name	V	1-25	= response.name_response
5 Sensitivity/gain	F	12	= response.normalization
6 Frequency (Hz)	F	12	= response.frequency_max
7 Number of history values	D	2	= response.data_length
Repeat 8-10 for each response			
8 Sensitivity for calibration	F	12	= response_sen.sensitivity
9 Frequency of calib sensitiv	F	12	= response_sen.frequency
10 Time of above calibration	V	1-22	= response_sen.cal_time

50 Station Id Blockette

1 Blockette type - 050	D	3	= not applicable
2 Length of blockette	D	4	= not applicable
3 Station call letters	A	5	= station.station_name[8]
4 Latitude	D	10	= station.station_lat
5 Longitude	D	11	= station.station_lon
6 Elevation	D	7	= station.station_elev
7 Number of Channels	D	4	= not applicable
8 Number of Station Comments	D	3	= not applicable
9 Site name	V	1-60	= station.site_descrip[40]
10 Network name	D	3	= domain
11 Longword (32-bit) order	D	4	= not applicable
12 Word (32-bit) order	D	2	= not applicable
13 Start effective date	V	1-22	= station.from_time
14 End effective date	V	0-22	= station.thru_time
15 Update flags	A	1	= should trigger update of database

51 Station Comment Blockette

1 Blockette type - 051	D	3	= not applicable
2 Length of blockette	D	4	= not applicable
3 Beginning effective time	V	1-22	= include in comment if necessary
4 End effective time	V	1-22	= include in comment if necessary
5 Comment code key	D	4	= station.comment_id
6 Comment Level	D	6	= include in comment if necessary

52 Channel Id Blockette

In SUDS, the signal_path structures explain different components at one station, which is a physical location.

1 Blockette type - 052	D	3	= not applicable
2 Length of blockette	D	4	= not applicable
3 Location Id	A	2	= add to station_name
4 Channel Id	A	3	= signal_path.component, sensor, band_type
5 SubChannel Id	D	4	=
6 Instrument Id	D	3	= signal_path.sensor_id
7 Optional Comment	V	0-30	= signal_path.comment_id
8 Units of Signal Response	D	3	= waveform.data_units
9 Units of Calibration Input	D	3	= waveform.data_units
10 Latitude (degs)	D	10	= station.station_lat
11 Longitude (degs)	D	11	= station.station_lon
12 Elevation (m)	D	7	= station.station_elev

In SUDS, a station is a physical location and if the latitude, longitude, or elev have changed, the station name must be changed. This can be done by adding a character to the SEED name.

13 Local depth (m)	D	5	= signal_path.sensor_depth
14 Azimuth	D	5	= signal_path.sensor_azimuth
15 Dip	D	5	= signal_path.sensor_dip
16 Data Format Id code	D	4	= waveform.data_type
17 Data record length	D	2	= not applicable
18 Sample Rate	F	10	= waveform.nom_dig_rate
19 Max Clock Drift	F	10	=
20 Number of Comments	D	4	= signal_path.comment_id
21 Channel Flags	V	0-26	= waveform.signal_type
22 Start Date	V	1-22	= signal_path.from_time
23 End Date	V	0-22	= signal_path.thru_time
24 Update Flags	A	1	= not applicable

53 Response (Poles Zeros) Blockette

1 Blockette type - 053	D	3	= not applicable
2 Length of blockette	D	4	= not applicable
3 Transfer function type	A	1	= response.response
4 Stage sequence number	D	2	= response.data_length
5 Stage signal input units	D	3	= response.input_units
6 Stage signal output units	D	3	= response.output_units
7 AO normalization factor	F	12	= response.normalization
8 Normalization frequency (Hz)	F	12	= response.frequency_max
9 Number of complex zeros	D	3	= response.data_length
Repeat 10-13 for each complex zero			
10 Real zero	F	12	= response_pz.zero_r
11 Imaginary zero	F	12	= response_pz.zero_i
12 Real zero error	F	12	= response_pz.zero_err_r
13 Imaginary zero error	F	12	= response_pz.pole_err_i
14 Number of complex poles	D	3	= response.data_length
Repeat 15-18 for each complex pole			
15 Real pole	F	12	= response_pz.pole_r
16 Imaginary pole	F	12	= response_pz.pole_i
17 Real pole error	F	12	= response_pz.pole_err_r
18 Imaginary pole error	F	12	= response_pz.pole_err_i

54 Response (Coefficients) Blockette

1 Blockette type - 054	D	3	= not applicable
2 Length of blockette	D	4	= not applicable
3 Response type	A	1	= response.response
4 Stage sequence number	D	2	= response.data_length
5 Signal input units	D	3	= response.input_units
6 Signal output units	D	3	= response.output_units
7 Number of numerators	D	4	= response.data_length
Repeat 8-9 for each numerator			
8 Numerator coefficient	F	12	= response_fir.numer_coef
9 Numerator error	F	12	= response_fir.numer_coef_err
10 Number of denominators	D	4	= response.data_length
Repeat 11-12 for each denominator			
11 Denominator coefficient	F	12	= response_fir.denom_coef
12 Denominator error	F	12	= response_fir.denom_coef_err

55 Response List Blockette

1 Blockette type - 055	D	3	= not applicable
2 Length of blockette	D	4	= not applicable
3 Stage sequence number	D	2	= response.data_length
4 Signal input units	D	3	= response.input_units
5 Signal output units	D	3	= response.output_units
6 Number of responses listed	D	4	= response.data_length
Repeat 7-11 for each response			
7 Frequency (Hz)	F	12	= response_fap.frequency
8 Amplitude	F	12	= response_fap.amplitude
9 Amplitude error (Absolute)	F	12	= response_fap.amplitude_err
10 Phase angle (degrees)	F	12	= response_fap.phase
11 Phase error (degrees)	F	12	= response_fap.phase_error

56 Generic Response Blockette

1 Blockette type - 056	D	3	= not applicable
2 Length of blockette	D	4	= not applicable
3 Stage sequence number	D	2	= response.data_length
4 Signal input units	D	3	= response.input_units
5 Signal output units	D	3	= response.output_units
6 Number of corners listed	D	4	= response.data_length
Repeat 7-8 for each corner			
7 Corner frequency (Hz)	F	12	= response_cfs.corner_freq
8 Corner slope (db/decade)	F	12	= response_cfs.db_per_decade

57 Decimation Blockette

1 Blockette type - 057	D	3	= not applicable
2 Length of blockette	D	4	= not applicable
3 Stage sequence number	D	2	= response.data_length
4 Input sample rate	F	10	= response.inp_samp_rate
5 Decimation factor	D	5	= response.decim_factor
6 Decimation offset	D	5	= response.decim_offset
7 Estimated delay (seconds)	F	11	= response.estim_delay
8 Correction applied (seconds)	F	11	= response.used_delay

58 Channel Sensitivity/Gain Blockette

1 Blockette type - 058	D	3	= not applicable

2 Length of blockette	D	4	= not applicable
3 Stage sequence number	D	2	= response.data_length
4 Sensitivity/gain	F	12	= response.normalization
5 Frequency (Hz)	F	12	= response.frequency_max
6 Number of history values	D	2	= response.data_length
Repeat 7-9 for each history value			
7 Sensitivity for cal	F	12	= response_sen.sensitivity
8 Frequency of cal sensitivity	F	12	= response_sen.frequency
9 Time of above cal	V	1-22	= response_sen.cal_time

59 Channel Comment Blockette

1 Blockette type - 059	D	3	= not applicable
2 Length of blockette	D	4	= not applicable
3 Beginning effective time	V	1-22	= waveform.comment_id
4 End effective time	V	0-22	= waveform.comment_id
5 Comment code key	D	4	= waveform.comment_id
6 Comment level	D	6	= waveform.comment_id

60 Response Reference Blockette

1 Blockette type - 059	D	3	= not applicable
2 Length of blockette	D	4	= not applicable
3 Number of stages	D	2	= response.data_length
Repeat 4 for each stage			
4 Stage sequence number	D	2	= response.data_length
5 Number of responses	D	2	= response.data_length
Repeat 6 for each stage			
6 Response lookup key	D	2	= response.response_id

70 Time Span Id Blockette

1 Blockette type - 070	D	3	= not applicable
2 Length of blockette	D	4	= not applicable
3 Time span flag	A	1	= waveform.signal_type
4 Beginning time of data span	V	1-22	= waveform.begin_time
5 End Time of data span	V	1-22	= waveform.end_time

71 Hypocenter Info Blockette

1 Blockette type - 071	D	3	= not applicable
2 Length of blockette	D	4	= not applicable
3 Origin time of event	V	1-22	= solution.origin_time
4 Hypocenter source Id	D	2	= solution.authority
5 Latitude of event	D	10	= solution.origin_lat
6 Longitude of event	D	11	= solution.origin_long
7 Depth (Km)	D	7	= solution.origin_depth
8 Number of magnitudes	D	2	= solution.data_length
Repeat 9-11 for each magnitude			
9 Magnitude	D	5	= magnitude.magnitude
10 Magnitude type	V	1-10	= magnitude.magnitude_type
11 Magnitude source	D	2	= magnitude.authority

72 Event Phases Blockette

1 Blockette type - 072	D	3	= not applicable
2 Length of blockette	D	4	= not applicable
3 Station Id	A	5	= station.station_name

4 Location Id	A	2	= add to station_name
5 Channel Id	A	3	= signal_path.component, sensor, band_type
6 Arrival time of phase	V	1-22	= pick.time
7 Amplitude of signal	F	10	= pick.amplitude
8 Period of signal (Seconds)	F	10	= pick.period
9 Signal to noise ratio	F	10	= pick.signal_2_noise
10 Name of phase	V	1-20	= pick.observ_phase

73 Time Span Data Start Index Blockette

1 Blockette type - 073	D	3	= not applicable
2 Length of blockette	D	4	= not applicable
3 Number of data pieces	D	4	= data_group is used to associate pieces
Repeat 4-9 for each data piece			
4 Station Id of data piece	A	5	= station.station_name
5 Location Id	A	2	= add to station_name
6 Channel Id	A	3	= signal_path.component, sensor, band_type
7 Time of record	V	1-22	= waveform.begin_time
8 Sequence of first record	D	6	= groups of waveforms are created
9 Sub-sequence	D	2	= using data_group

74 Time Series Index Blockette

1 Blockette type - 074	D	3	= not applicable
2 Length of blockette	D	4	= not applicable
3 Station Id	A	5	= station.station_name
4 Location Id	A	2	= add to station_name
5 Channel Id	A	3	= signal_path.component, sensor, band_type
6 Series start time	V	1-22	= waveform.begin_time
7 Sequence of first data	D	6	= in SUDS each waveform contains all the
8 Sub-sequence number	D	2	= information needed to use it
9 Series end time	V	1-22	= waveforms are grouped using
10 Sequence of last data	D	6	= data_group. Pieces of waveform
11 Sub-sequence number	D	2	= can be used by creating a new
12 Number accelerator repeats	D	3	= waveform as a part of an old one.
Repeat 13-15 for each accelerator repeat			
13 Record start time	V	1-22	= waveform.begin_time
14 Sequence of record	D	6	= not applicable
15 Sub-sequence number	D	2	= not applicable

Fixed Section of Data Header

1 Sequence number	A	6	= not applicable
2 Data header indicator	A	1	= not applicable
3 Reserved byte	A	1	= not applicable
4 Station Id	A	5	= station.station_name
5 Location Id	A	2	= add to station_name
6 Channel Id	A	3	= signal_path.component, sensor, band_type
7 Reserved byte	A	2	= not applicable
8 Record start time	B	10	= waveform.begin_time
9 Number of samples	B	2	= waveform.data_length
10 Sample rate factor	B	2	= waveform.nom_dig_rate
11 Sample rate multiplier	B	2	= waveform.nom_dig_rate
12 Activity flags	B	1	
Bit 0: Calibration Signals Present			= waveform.signal_type
Bit 1: Time Error Caused by Clock Correction			= waveform.time_status

(amount recorded in the time correction field. If not set, correction field has not been added to time yet)			
Bit 2: Beginning of Event			= not applicable
Bit 3: End of Event			= not applicable
Bit 4: A positive leap second happened during this record. (A 61 second minute)			= not allowed for
Bit 5: A negative leap second happened during this record. (A 59 second minute)			= not allowed for
Bit 6: Event in Progress			= not applicable
Other bits reserved and must be zero.			
13 I/O flags	B	1	
Bit 0: Station Tape Parity Error Possible			= not applicable
Bit 1: Long Record Read (Possibly no problem)			= not applicable
Bit 2: Short Record Read (Record padded)			= not applicable
Other bits reserved and must be zero.			
14 Data quality flags	B	1	
Bit 0: Amplifier Saturation Detected (Station dependent)			
Bit 1: Digitizer Clipping Detected			= waveform.num_pos_clip
Bit 2: Spikes Detected			= waveform.num_spikes
Bit 3: Glitches Detected			= waveform.num_glitches
Bit 4: Missing/Padded Data Present			= waveform.comment_id
Bit 5: Telemetry Syncronization Error			= not applicable
Bit 6: A digital filter may be charging			= waveform.comment_id
Bit 7: Time Tag is Questionable			= waveform.time_status
15 Number of blockettes follow	B	1	= not applicable
16 Time correction	B	4	= waveform.time_status
17 Beginning of data	B	2	= not applicable
18 First blockette	B	2	= not applicable

200 Generic Event detection blockette (28 bytes)

1 Blockette type - 200	B	2	= not applicable
2 Next blockette's number	B	2	= not applicable
3 Signal amplitude	B	4	= trigger.amplitude
4 Signal period	B	4	= trigger.period
5 Background estimate	B	4	= trigger.signal_2_noise
6 Event detection flags	B	1	= see below
Bit 0: Set: Dilatation wave; Unset: Compression			= trigger.first_motion
Bit 1: Set: units above are after deconvolution (see Channel Id); If unset: digital counts			= ?
Bit 2 -- Set: bit 0 is undetermined			= not applicable
Other bits reserved and must be zero.			
7 Reserved byte	B	1	= not applicable
8 Signal onset time	B	10	= trigger.trigger_time

201 Murdock Event detection blockette (36 bytes)

1 Blockette type - 201	B	2	= not applicable
2 Next blockette's number	B	2	= not applicable
3 Signal amplitude	B	4	= trigger.amplitude
4 Signal period	B	4	= trigger.period
5 Background estimate	B	4	= trigger.signal_2_noise
6 Event detection flags	B	1	= see below
Bit 0: Set: Dilatation wave; Unset: Compression			= trigger.first_motion

Other bits reserved and must be zero.			
7 Reserved byte	B	1	= not applicable
8 Signal onset time	B	10	= trigger.trigger_time
9 Signal-to-noise ratio values	B	6	= trigger.local_1 to local_6
10 Lookback value	B	1	= trigger.level
11 Pick algorithm	B	1	= trig_setting.algorithm

202 Log-Z Event Detection Blockette

Reserved for future use

300 Step Calibration Blockette

1 Blockette type - 300	B	2	= not applicable
2 Next blockette's number	B	2	= not applicable
3 Beginning calibration time	B	10	= calibration.begin_time
4 Number of step calibrations	B	1	= calibrations.number
5 Calibration flags	B	1	= see below
Bit 0: Set: first pulse is positive			= calibration.first_motion
Bit 1: Set: calibration's alternate sign			= not applicable
Bit 2: Set: calibration automatic; Unset: manual			= calibration.cause
Bit 3: Set: continued from previous records			= calibration.continuation
Other bits reserved and must be zero.			
6 Step duration	B	4	= calibration.end_time-begin_time
7 Interval duration	B	4	= calibration.end_time-begin_time
8 Calibration signal amplitude	B	4	= calibration.amplitude
9 Channel with calibration input	A	3	= waveform.signal_path_id
10 Reserved byte	B	1	= not applicable

310 Sine Cal blockette (32 bytes)

1 Blockette type - 310	B	2	= not applicable
2 Next blockette's number	B	2	= not applicable
3 Beginning calibration time	B	10	= calibration.begin_time
4 Reserved byte	B	1	= not applicable
5 Calibration flags	B	1	= see below
Bit 2: Set: calibration automatic; Unset: manual			= calibration.cause
Bit 3: Set: continued from previous records			= calibration.continuation
Bit 4: Set: peak-to-peak amplitude			= calibration.amplitude_type
Bit 5: Set: zero-to-peak amplitude			= calibration.amplitude_type
Bit 6: Set: RMS amplitude			= calibration.amplitude_type
Other bits reserved and must be zero.			
6 Calibration duration	B	4	= calibration.end_time-begin_time
7 Period of signal	B	4	= calibration.frequency
8 Amplitude of signal	B	4	= calibration.amplitude
9 Channel with calibration input	A	3	= waveform.signal_path_id
10 Reserved byte	B	1	= not applicable

320 Pseudo random cal blockette (28 bytes)

1 Blockette type - 320	B	2	= not applicable
2 Next blockette's number	B	2	= not applicable
3 Beginning calibration time	B	10	= calibration.begin_time
4 Reserved byte	B	1	= not applicable
5 Calibration flags	B	1	= see below
Bit 2: Set: calibration automatic; Unset: manual			= calibration.cause
Bit 3: Set: continued from previous records			= calibration.continuation

Bit 4: Set: random amplitudes			= calibration.amplitude_type
Other bits reserved and must be zero.			
6 Calibration duration	B	4	= calibration.end_time-begin_time
7 Peak-to-peak amplitude step	B	4	= calibration.amplitude
8 Channel with calibration input	A	3	= waveform.signal_path_id
9 Reserved byte	B	1	= not applicable

390 Generic Calibration Blockette

1 Blockette type - 390	B	2	= not applicable
2 Next blockette’s number	B	2	= not applicable
3 Beginning calibration time	B	10	= calibration.begin_time
4 Reserved byte	B	1	= not applicable
5 Calibration flags	B	1	= see below
Bit 2: Set: calibration automatic; Unset: manual			= calibration.cause
Bit 3: Set: continued from previous records			= calibration.continuation
Other bits reserved and must be zero.			
6 Calibration duration	B	4	= calibration.end_time-begin_time
7 Calibration signal amplitude	B	4	= calibration.amplitude
8 Channel with calibration input	A	3	= waveform.signal_path_id
9 Reserved byte	B	1	= not applicable

395 Cal Abort blockette (16 bytes)

1 Blockette type - 395	B	2	= not applicable
2 Next blockette’s number	B	2	= not applicable
3 Ending calibration time	B	10	= calibration.end_time
4 Reserved bytes	B	2	= not applicable

400 Beam Blockette (16 bytes)

1 Blockette type - 400	B	2	= not applicable
2 Next blockette’s number	B	2	= not applicable
3 Beam azimuth(degrees)	B	4	= signal_path.sensor_azimuth
4 Beam slowness(sec/deg)	B	4	= signal_path.sensor_dip
5 Beam configuration	B	2	= beam_comp and signal_path
6 Reserved bytes	B	2	= not applicable

405 Beam Delay Blockette

1 Blockette type - 300	B	2	= not applicable
2 Next blockette’s number	B	2	= not applicable
3 Array of delay values	B	2	= beam_comp and signal_path

BUGS

SEE ALSO

NAME

segy_to_suds – mapping of SEGY (Society of Exploration Geophysicists format Y) to SUDS

DESCRIPTION

SEG-Y is defined in **Recommended standards for digital tape formats,** by K.M. Barry, D.A. Carver, and C.W. Kneale, *Geophysics, v. 40,* no. 2, p.344-352, 1975. SEGY-LDS Version 2.0 is defined by C. Spencer, I. Asudeh, and T. Cote, Geological Survey of Canada, Manual Version 1.00, 17p., January 31, 1989.

MEMBERS

Reel Identification Header (Only one shot)

Part 1: 3200 bytes EBCDIC card image (40 cards)

Describes the data from a line of shotpoints. If desired to keep this information in SUDS, convert to ASCII and put in a comment associated with the source structure.

Card 1:	client	company	crew_number		
Card 2:	line	area	map_id		
Card 3:	reel_number	day_start_of_reel		year	observer
Card 4:	instrument_mfg	model	serial_number		
Card 5:	data_traces_per_rec	auxiliary_traces_per_rec		CDP_fold	
Card 6:	sample_interval	samples/trace bits/in		bytes_per_sample	
Card 7:	recording_format	format_this_reel		measurement_system	
Card 8:	sample_code floating_pt		fixed_pt	fixed_pt_gain	correlated
Card 9:	gain_type	fixed	binary	floating_pt	other
Card 10:	FILTERS:		alias_hz	notch_hz band_hz	slope_db_per_octave
Card 11:	SOURCE:	type	number_per_point		point_interval
Card 12:	PATTERN:	length	width		
Card 13:	SWEEP:	start_hz	end_hz	length_ms	channel_notype
Card 14:	TAPER:	start_length_ms	end_length_ms	type	
Card 15:	SPREAD:	offset	max_distance	group_interval	
Card 16:	GEOPHONES:	per_group	spacing	frequency	mfgmodel
Card 17:	PATTERN:	length	width		
Card 18:	TRACES SORTED BY:		record	cdp	other
Card 19:	AMPLITUDE RECOVERY:		none	spherical_div	agcother
Card 20:	map_projection	zone_id	coordinate_units		
Card 21:	PROCESSING:				
Card 22:	PROCESSING:				
Card 23:	unassigned				
...					
Card 39:	unassigned				
Card 40:	END EBCDIC				

InterBlock Gap

Part 2: 400 bytes binary

First 60 bytes assigned

1	long	job_id_number	= data_group.job_number also domains for keys
5	long	line_number	= data_group.line_number
9	long	reel_number	= data_group.reel_number
13	short	data_traces_per_record	= not applicable
15	short	auxiliary_traces_per_record	= not applicable
17	short	sample_interval_microsec	= waveform.prec_dig_rate ??
19	short	orig_sample_interval_microsec	= waveform.nom_dig_rate
21	short	samples_per_trace	= waveform.data_length

23	short	orig_samples_per_trace	= processing
25	short	format_code	= waveform.data_type
		1=FLOAT4	
		2=INT4	
		3=INT2	
		4=FLOAT4 with gain_code	
		etc.	
27	short	cdp_fold	= processing
29	short	trace_sorting_code	= data_group options?
		1=as recorded	
		2=CDP ensemble	
		3=single fold continuous profile	
		4=horizontally stacked	
31	short	vertical_sum_code	= processing
		1=no sum	
		2=two sum	
		N=N sum	
33	short	sweep_freq_at_start	= source.begin_freq
35	short	sweep_freq_at_end	= source.end_freq
37	short	sweep_length_ms	= source.sweep_length
39	short	sweep_type_code	= source.sweep_type
		1=linear	
		2=parabolic	
		3=exponential	
		4=other	
		5=borehole source	= source.event_type
		6=water explosive source	
		7=airgun source	
41	short	trace_num_of_sweep_channel	= waveform.signal_type
43	short	start_taper_length_ms	= source.begin_taper
45	short	end_taper_length_ms	= source.end_taper
47	short	taper_type	= source.taper_type
		1=linear	
		2=cos**2	
		3=other	
49	short	correlated_data_traces	= processing
		1=no	
		2=yes	
51	short	binary_gain_recovered	= processing
		1=yes	
		2=no	
53	short	amplitude_recovery_method	= processing
		1=none	
		2=spherical divergence	
		3=agc	
		4=other	
55	short	measurement_units	=
		1=meters	
		2=feet	
57	short	impulse_polarity	= signal_path.polarity
		1=pressure incr or geophone up is negative	
		2=pressure incr or geophone up is positive	
59	short	vibratory_polarity	= source.signal_lag

(seismic signal lags pilot signal by degrees)
1= 337.5 to 22.5
2= 22.5 to 67.5
3= 67.5 to 112.5
4= 112.5 to 157.5
5= 157.5 to 202.5
6= 202.5 to 247.5
7= 247.5 to 292.5
8= 292.5 to 337.5

Last 340 bytes unassigned by SEGY

61	short	number_traces_in_file	= not applicable
63	short	attribute	= waveform.data_type ?
		0=velocity/displacement data	
		1=instantaneous amplitude	
		2=instantaneous frequency	
		3=instantaneous phase	
		4=slowness (m/ms)	
		5=semblance (0-1000)	
65	float	mean_amplitude	= waveform.average_noise ??
69	short	domain	= waveform.data_type ?
		0=time/distance	
		1=FK or user friendly polar	
		2=TAU-P or slant stack	
71	short	sample_rate_exponent	= waveform.nom_dig_rate
73	long	vred ??	= ??
77	float	min_all_samples_in_file	= waveform.min_val_data
81	float	max_all_traces_in_file	= waveform.max_val_data
85	short	instrument_type	= signal_path.recorder_id
		1=EDA	
		2=USGS cassette	
		3=GEOS	
		4=Sprengnether	
		5=Teledyne	
		6=Kinemetrics	
87	short	file_creation_year	= not applicable
89	short	file_creation_month	= not applicable
91	short	file_creation_day	= not applicable
93	short	file_format	= not applicable
		1=reel header 3600 bytes	
		2=reel header and data padded to NNB bytes	
95	short	character_code	= not applicable
		1=EBCDIC	
		2=ASCII	
97	long	NNB=file_record_length	= not applicable
101	short	byte_order	= not applicable
		1=MSB first	
		256=LSB first	
296 bytes left free			
399	short	format_version	= not applicable
		99= Version .99	
		100=Version 1.0	
		200=Version 2.0	

SIO flavor

61	short	domain	= waveform.data_type
		0=time/distance	
		1=time	
		2=frequency-wavenumber, rectangular coord	
		3=frequency-wavenumber, polar coord	
		4=frequency, rectangular coord	
		5=frequency, polar coord	
		6=depth	
		7=TAU-P or slant stack	
		8=fk or "user friendly" polar	
63	short	num_wavenums_in_FK	=
65	short	tx_sample_interval_microsecs	=
67	short	tx_time_delay_ms	=
69	short	SIOSEIS_version	=
71	short	num_traces_tx_domain	=

InterBlock Gap

Trace Data Blocks

Trace Identification Header

1	long	line_trace_sequence_num	=
5	long	reel_trace_sequence_num	=
9	long	original_record_num	=
13	long	original_trace_sequence_num	=
17	long	source_point_number	=
21	long	cdp_ensemble_number	=
25	long	trace_within_ensemble	=
29	short	trace_id_code	= waveform.signal_type
		1=seismic data	
		2=dead	
		3=dummy	
		4=time break	
		5=uphole	
		6=sweep	
		7=timing	
		8=water break	
		>=9 optional	
31	short	vertical_sum	=
33	short	horizontal_stack (cdp fold)	=
35	short	data use	=
		1=production	
		2=test	
37	long	source_receiver_distance	=
41	long	surface_elevation_at_receiver	= station.station_elev
45	long	surface_elevation_at_source	= source.origin_elev
49	long	depth_below_surface	= signal_path.sensor_depth
53	long	datum_elev_receiver	=
57	long	datum_elev_source	= source.origin_depth ??
61	long	water_depth_source	= source.water_depth
65	long	water_depth_receiver	=
69	short	scalar_elevation_multiplier	=
71	short	scalar_position_multiplier	=

73	long	source_x_lat	= source.origin_lat
77	long	source_y_long	= source.origin_long
81	long	group_x_lat	= station.station_lat
85	long	group_y_long	= station.station_long
89	short	coordinate_units	=
		1=meters or feet	
		2=seconds of arc	
91	short	weathering_velocity	=
	(SIO: constant velocity from velan)		
93	short	subweathering_velocity	=
95	short	uphole_time_at_source	= source.
97	short	uphole_time_at_group	=
99	short	source_static	= source.static
101	short	group_static	= signal_path.time_delay
103	short	total_static	=
105	short	lag_time_a	=
107	short	lag_time_b	=
109	short	delay_recording_time	=
		(SIO: deep water delay)	
111	short	mute_time_start_ms	= processing
113	short	mute_time_end_ms	= processing
115	short	number_samples	= waveform.nom_dig_rate
117	short	sample_interval_microseconds	= waveform.nom_dig_rate
119	short	gain_type	=
		1=fixed	
		2=binary	
		3=floating	
		>=4 optional	
121	short	inst_gain_const	=
123	short	inst_early_gain	=
125	short	correlated	=
		1=no	
		2=yes	
127	short	sweep_freq_at_start	= source.begin_freq
129	short	sweep_freq_at_end	= source.end_freq
131	short	sweep_length_ms	= source.sweep_length
133	short	sweep_type_code	= source.sweep_type
		1=linear	
		2=parabolic	
		3=exponential	
		4=other	
		5=borehole source	= source.event_type
		6=water explosive source	
		7=airgun source	
135	short	start_taper_length_ms	= source.begin_taper
137	short	end_taper_length_ms	= source.end_taper
139	short	taper_type	= source.taper_type
		1=linear	
		2=cos**2	
		3=other	
141	short	alias_filter_frequency	= processing
143	short	alias_filter_slope	= processing
145	short	notch_filter_frequency	= processing

147	short	notch_filter_slope	= processing
149	short	low_cut_frequency	= processing
151	short	high_cut_frequency	= processing
153	short	low_cut_slope	= processing
155	short	high_cut_slope	= processing
157	short	year_recorded	= waveform.begin_time
159	short	day_of_year	= waveform.begin_time
161	short	hour	= waveform.begin_time
163	short	minute	= waveform.begin_time
165	short	second	= waveform.begin_time
167	short	time_code	=
		1=local	= waveform.local_time
		2=GMT	
		3=other	
169	short	trace_weighting_factor	= waveform.weight
171	short	geophone_group_num_on_roll_1st	=
173	short	orig_first_geophone_group_number=	
175	short	orig_last_geophone_group_number	=
177	short	gap_size	=
179	short	taper_overtravel	=
		1=down or behind	
		2=up or ahead	

181-240 unassigned by SEGY

SEGY-LDS version 2.0

181	long	microseconds_trace_start	=
185	short	time_correction_ms	=
187	short	charge_size_kg	= source.yield
189	short	shot_time_year	= source.origin_time
191	short	shot_time_day	= source.origin_time
193	short	shot_time_hour	= source.origin_time
195	short	shot_time_minute	= source.origin_time
197	short	shot_time_second	= source.origin_time
199	long	shot_time_microsecond	= source.origin_time
203	short	azimuth_shot_to_receiver	=
205	short	azimuth_geophone_from_north	= signal_path.sensor_azimuth
207	short	dip_geophone	= signal_path.sensor_dip
209	long	actual_trace_start	=
213	char	recording_inst_number[4]	=
217	char	deployment_name[4]	=
221	char	shot_point_name[4]	= source.source_name
225	char	receiver_site_name[4]	= station.station_name
229	char	shot_id[4]	= source.source_id
233	char	line_id[4]	=
237	char	geophone_orientation[4]	=

SEGY-SIO Scripps Institute of Oceanography flavor

181	long	deep_water_delay_secs	=
185	long	start_mute_secs	= processing
189	long	end_mute_secs	= processing
193	long	sample_interval_secs	= waveform.nom_dig_rate
197	long	water_bottom_time_secs	=

201	long	<0 end_of_gather, >0 number of traces stacked	=
205	long	smute_start_secs	= processing
209	long	smute_end_secs	= processing
213	long	SeaBeam_slant_range (closest beam depth)	=

Data Values of the Seismic or Auxiliary Channels

SEE ALSO

BUGS

Need to include other major flavors.

NAME

suds1_suds2 – mapping of SUDS 2.4 with PC-SUDS version 1.31

MEMBERS

FIELDS from PCSUDS NOT YET MAPPED to SUDS 2.4

```
LG_INT   form.nextfstype;/∗ SUDS2 is not recursive because of database∗/
LG_INT   form.frecord;          /∗ never used? ∗/
LG_INT   form.ffield;           /∗ never used? ∗/

STRING   statident.network[4]; /∗ components if statident are mapped ∗/
STRING   statident.st_name[5]; /∗ components if statident are mapped ∗/
CHAR     statident.component;  /∗ components if statident are mapped ∗/
SH_INT   statident.inst_type;  /∗ components if statident are mapped ∗/

SH_INT   atodinfo.base_address;
BITS16   atodinfo.device_flags;
SH_INT   atodinfo.extended_bufs;
SH_INT   atodinfo.external_mux;
CHAR     atodinfo.timing_source;
CHAR     atodinfo.trigger_source;

CHAR     detector.event_type;
CHAR     detector.net_node_id[10];
LG_INT   detector.event_number;

ST_TIME equipment.effective;

SUDS_STATIDENT instrument.in_name;
SH_INT   instrument.comps;
LG_INT   instrument.void_samp;
FLOAT    instrument.aa_corner;
FLOAT    instrument.aa_poles;
FLOAT    instrument.gain;
LOAT     instrument.local_z;
FLOAT    instrument.pre_event;
SH_INT   instrument.trig_num;
STRING   instrument.study[6];

SH_INT   muxdata.loctime;

STRING   origin.crustmodel[6];
FLOAT    origin.weight;                        of magnitudes

CHAR     stationcomp.annotation;               /∗ never used ? ∗/
CHAR     stationcomp.st_status;
LOAT     stationcomp.max_gain;
```

FIELDS MAPPED∗∗

```
typedef struct{
        INT4     number;                               LG_INT        codes.num;
        CHRPTR meaning;              STRING            codes.∗meaning;
} SUDS_CODE_LIST, ∗LIST;
```

```
typedef struct{
        FIXED   structure_type;
        FIXED   structure_len;
        INT4    struct_number;        LG_INT        form.fstype;
        INT2    member_number;
        INT2    member_type;          LG_INT        form.ftype;
        INT2    member_length;        LG_INT        form.flength;
        INT2    member_offset;        LG_INT        form.offset;
        INT2    pri_key_numb;
        INT2    for_key_numb;
        INT4    key_structure;
        INT4    key_member;
        CHAR    db_include;
        CHAR    db_must_be_in;
        CHAR    db_index_type;
        CHAR    db_delete_type;
        INT4    db_permission;
        INT2    editor_row;           LG_INT        form.form_row;
        INT2    editor_column;        LG_INT        form.form_col;
        FIXED   name_len;
        STRING  member_name[16]; STRING             form.*fname;
        FIXED   title_len;
        STRING  member_title[24];
        LIST    ptr_code_list;
        FIXED   list_len;
        STRING  code_list_name[24]; SUDS_CODES form.*codelist;
        FIXED   default_len;
        STRING  default_values[24]; STRING form.*initval;
        FIXED   format_len;
        STRING  print_format[20];   STRING form.*fformat;
        FIXED   allowed_len;
        STRING  allowed_chars[24];  STRING form.*allowchar;
        CODE1   checks_input;
        CHAR    spare_code;
        INT2    spare;
} SUDS_MEMBER_INFO, *MEMPTR;

typedef struct{
        FIXED   structure_type;
        FIXED   structure_len;
        CHAR    type;
        CHAR    mode;
        CODE1   endian_type;
        CODE1   float_type;
        INT4    byte_ptr;
        UINT4   pack;
        FIXED   len_io_name;
        STRING  io_name[24];
        CHRPTR  file_handle;
        CODE1   machine_type;
        CODE1   output_type;
        CODE1   endian_change;
        CODE1   float_change;
```

```
        CHAR     sync_char;
        CODE1    computer_type;
        INT2     suds_version;
        INT4     struct_number;
        INT4     struct_length;
        INT4     length_data;
} SUDS_STREAM;

typedef struct{
        FIXED    structure_type;
        FIXED    structure_len;
        INT4     struct_number;
        INT4     struct_length;
        MEMPTR   member_table;
        INT4     num_members;
        FIXED    len_struct_n;
        STRING   struct_name[16];
        FIXED    len_typedef;
        STRING   typedef_name[24];
        FIXED    len_define;
        STRING   define_name[20];
        INT4     data_only_to;
        INT4     db_permission;
        INT4     data_type_off;
        INT4     data_len_off;
        INT4     data_off_off;
        INT4     xdr_struct_len;
} SUDS_STRUCTURE_INFO;

typedef struct{
        FIXED    structure_type;
        FIXED    structure_len;
        FIXED    name_len;
        STRING   variable_name[8];
        FIXED    define_len;
        STRING   define_name[4];
        INT2     define_num;
        INT2     xdr_num_bytes;
        FIXED    c_type_len;
        STRING   c_type[20];
        FIXED    default_len;
        STRING   default_values[24];
        FIXED    min_len;
        STRING   min_value[24];
        FIXED    max_len;
        STRING   max_value[24];
        FIXED    format_len;
        STRING   print_format[16];
        FIXED    allowed_len;
        STRING   allowed_chars[24];
        INT2     spare;
        INT2     num_bytes;
} SUDS_VARIABLE_INFO;
```

```
typedef struct{
        FLOAT4  cr;                              same
        FLOAT4  ci;                              same
} COMPLEX;

typedef struct{
        FLOAT8  dr;                              same
        FLOAT8  di;                              same
} D_COMPLEX;

typedef struct{
        FLOAT4  fx;                              same
        FLOAT4  fy;                              same
} VECTOR;

typedef struct{
        FLOAT4  xx;                              same
        FLOAT4  yy;                              same
        FLOAT4  xy;                              same
} TENSOR;

typedef struct{
        FIXED   structure_type;
        FIXED   structure_len;
        LABEL   beam_comp_id;
        DOMAIN  beam_comp_dc;
        REFERS2 signal_path_id;
        DOMAIN  signal_path_dc;
        FLOAT4  delay;
        FLOAT4  weight;
} SUDS_BEAM_COMP;

typedef struct{
        FIXED   structure_type;
        FIXED   structure_len;
        LABEL   calibration_id;
        DOMAIN  calibration_dc;
        REFERS2 waveform_id;
        DOMAIN  waveform_dc;
        MS_TIME                 begin_time;
        MS_TIME                 end_time;
        FLOAT4  amplitude;
        FLOAT4  frequency;
        CODE1   event_type;
        CODE1   ampl_units;
        CODE1   amplitude_type;
        CODE1   cause;
        CODE1   first_motion;
        CHAR    continuation;
        INT2    number;
        INT4    spare;
        AUTHOR  authority;
        REFERS2 comment_id;
```

```
        DOMAIN  comment_dc;
} SUDS_CALIBRATION;

typedef struct{
        FIXED   structure_type;
        FIXED   structure_len;
        LABEL   clock_cor_id;        SUDS_STATIDENT timecorrection.tm_name;
        DOMAIN  clock_cor_dc;
        REFERS2 recorder_id;
        DOMAIN  recorder_dc;
        MS_TIME                      time_corr;       MS_TIME timecorrection.time_correct;
        ST_TIME from_time;           ST_TIME timecorrection.effective_time;
        ST_TIME thru_time;
        FLOAT4  rate_corr;           FLOAT            timecorrection.rate_correct;
        CHAR    sync_cd_type;        CHAR             timecorrection.sync_code;
        CHAR    program_type;        CHAR             timecorrection.program;
        INT2    spare;
        REFERS2 comment_id;
        DOMAIN  comment_dc;
} SUDS_CLOCK_CORRECT;

typedef struct{
        FIXED   structure_type;
        FIXED   structure_len;
        LABEL   comment_id;
        DOMAIN  comment_dc;
        CODE4   data_type;
        INT4    data_length;         SH_INT           comment.length;
        INT4    struct_number;       SH_INT           comment.refer;
        INT4    spare;                                SH_INT          comment.item;  use {}
} SUDS_COMMENT;

typedef struct{
        FIXED   structure_type;
        FIXED   structure_len;
        LABEL   data_group_id;
        DOMAIN  data_group_dc;
        CODE1   media_type;
        CHAR    spare_char;
        INT2    spare;
        FIXED   len_media_l;
        STRING  media_label[16];     STRING           loctrace.*tapeloc;
        FIXED   len_media_p;
        STRING  media_path[64];
        INT4    media_block;
        INT4    job_number;
        INT4    line_number;
        INT4    reel_number;
        FIXED   len_online_p;
        STRING  online_path[64];     STRING           loctrace.*fileloc;
        REFERS2 comment_id;
        DOMAIN  comment_dc;
} SUDS_DATA_GROUP;
```

```
typedef struct{
        FIXED   structure_type;
        FIXED   structure_len;
        LABEL   event_id;
        DOMAIN  event_dc;
        REFERS2 data_group_id;
        DOMAIN  data_group_dc;
        REFERS2 auto_sol_id;        CHAR        origin.version;
        DOMAIN  auto_sol_dc;
        REFERS2 cat_sol_id;         CHAR        origin.version;
                                                CHAR        origin.preferred;
        DOMAIN  cat_sol_dc;
        CODE1   event_type;
        CHAR    local_1_cd;
        CHAR    local_2_cd;
        CHAR    local_3_cd;
        CHAR    local_4_cd;
        CHAR    local_5_cd;
        CHAR    local_6_cd;
        CHAR    local_7_cd;
        REFERS2 comment_id;
        DOMAIN  comment_dc;
} SUDS_EVENT;

typedef struct{
        FIXED   structure_type;
        FIXED   structure_len;
        INT4    struct_number;
        INT4    tag_begin_at;
        FIXED   len_signal_n;
        STRING  signal_name[20];
} SUDS_FILE_INDEX;

typedef struct{
        FIXED   structure_type;
        FIXED   structure_len;
        REFERS2 waveform_id;
        DOMAIN  waveform_dc;
        REFERS2 response_id;
        DOMAIN  response_dc;
        REFERS2 prev_wave_id;
        DOMAIN  prev_wave_dc;
        AUTHOR  authority;
        INT2    position;
        CODE1   decim_type;
        CHAR    decim_points;
        INT2    decim_interv;
        INT2    decim_index;
        INT4    spare;
        REFERS2 comment_id;
        DOMAIN  comment_dc;
} SUDS_FILTER;
```

```
typedef struct{
        FIXED    structure_type;
        FIXED    structure_len;
        REFERS2  solution_id;
        DOMAIN   solution_dc;
        CHAR     prefer_plane;      CHAR          focalmech.prefplane;
        CHAR     spare_char;
        INT2     spare;
        FLOAT4   a_strike;          FLOAT         focalmech.astrike;
        FLOAT4   a_dip;                           FLOAT         focalmech.adip;
        FLOAT4   a_rake;            FLOAT         focalmech.arake;
        FLOAT4   b_strike;          FLOAT         focalmech.bstrike;
        FLOAT4   b_dip;                           FLOAT         focalmech.bdip;
        FLOAT4   b_rake;            FLOAT         focalmech.brake;
        INT4     spare_a;
        REFERS2  comment_id;
        DOMAIN   comment_dc;
} SUDS_FOCAL_MECH;

typedef struct{
        FIXED    structure_type;
        FIXED    structure_len;
        LABEL    contr_file_id;
        DOMAIN   contr_file_dc;
        FIXED    len_contr_n;
        STRING   control_name[20];
        REFERS2  comment_id;
        DOMAIN   comment_dc;
} SUDS_HYPO_CONTROL;

typedef struct{
        FIXED    structure_type;
        FIXED    structure_len;
        REFERS2  signal_path_id;    SUDS_STATIDENT triggers.tr_name;
        DOMAIN   signal_path_dc;
        REFERS2  lsa_setting_id;
        DOMAIN   lsa_setting_dc;
        MS_TIME                     lsa_onset_time; MS_TIME triggers.trig_time;
        FLOAT4   amplitude;         SH_INT        triggers.trig_value;
        FLOAT4   frequency;
        FLOAT4   signal_2_noise;
        FLOAT4   longterm_ave;      SH_INT        triggers.lta;
        FLOAT4   shortterm_ave;     SH_INT        triggers.sta;
        FLOAT4   other_ave;
        FLOAT4   level;
        INT2     local_1;           SH_INT        triggers.abs_sta;
        INT2     local_2;           SH_INT        triggers.abs_lta;
        INT2     local_3;
        INT2     local_4;
        INT2     local_5;
        INT2     local_6;
        CODE1    event_type;
        CODE1    first_motion;
```

```
        INT2    num_detections; SH_INT          triggers.num_triggers;
        AUTHOR  authority;
        REFERS2 comment_id;
        DOMAIN  comment_dc;
} SUDS_LSA_DETECTION;

typedef struct{
        FIXED   structure_type;
        FIXED   structure_len;
        LABEL   lsa_setting_id;
        DOMAIN  lsa_setting_dc;     STRING      trigsetting.netwname[4];
                                                STRING      eventsetting.netwname[4];
        CODE1   algorithm;          CHAR        trigsetting.algorithm;
                                                CHAR        eventsetting.algorithm;
        CHAR    spare_code;
        INT2    spare;
        FLOAT4  setting_1;          SH_INT      trigsetting.const1;
                                                SH_INT      eventsetting.const1;
        FLOAT4  setting_2;          SH_INT      trigsetting.const2;
                                                SH_INT      eventsetting.const2;
        FLOAT4  setting_3;          SH_INT      trigsetting.const3;
                                                SH_INT      eventsetting.const3;
        FLOAT4  setting_4;          SH_INT      trigsetting.const4;
        FLOAT4  setting_5;          FLOAT       eventsetting.minduration;
        FLOAT4  setting_6;          FLOAT       eventsetting.maxduration;
        FLOAT4  setting_7;
        FLOAT4  setting_8;
        FLOAT4  threshold;          SH_INT      trigsetting.threshold;
                                                SH_INT      eventsetting.threshold;
        FLOAT4  weighted_inc;       SH_INT      trigsetting.wav_inc;
        FLOAT4  sweep;                          FLOAT       trigsetting.sweep;
        FLOAT4  aperture;           FLOAT       trigsetting.aperture;
        FLOAT4  level;
        FLOAT4  spare_a;
        AUTHOR  authority;
        ST_TIME from_time;          MS_TIME trigsetting.beginttime;
                                                MS_TIME eventsetting.beginttime;
        ST_TIME thru_time;
        REFERS2 comment_id;
        DOMAIN  comment_dc;
} SUDS_LSA_SETTING;

typedef struct{
        FIXED   structure_type;
        FIXED   structure_len;
        LABEL   magnitude_id;
        DOMAIN  magnitude_dc;
        FLOAT4  mag_value;
        FLOAT4  mag_error;
        INT2    num_reports;        SH_INT      origin.rep_m;
        INT2    num_used;           SH_INT      origin.used_m;
        CODE1   magnitude_type;     SH_INT      origin.mag_type;
        CHAR    spare;
```

```
        INT2     spare_a;
        FLOAT4   rms_of_mag;          FLOAT       origin.mag_rms;
        AUTHOR   authority;
        REFERS2  comment_id;
        DOMAIN   comment_dc;
} SUDS_MAGNITUDE;

typedef struct{
        FIXED    structure_type;
        FIXED    structure_len;
        LABEL    map_element_id;
        DOMAIN   map_element_dc;
        LATIT    latitude;
        LONGIT   longitude;
        FLOAT4   elevation;
        CODE4    element;
        CODE4    map_source;
        INT4     map_scale;
        ST_TIME  time_mapped;
        ST_TIME  time_encoded;
        AUTHOR   authority;
        INT2     importance;
        CODE1    compression;
        CHAR     spare_char;
        CODE4    data_type;
        INT4     data_length;
        REFERS2  comment_id;
        DOMAIN   comment_dc;
} SUDS_MAP_ELEMENT;

typedef struct{
        FIXED    structure_type;
        FIXED    structure_len;
        REFERS2  solution_id;
        DOMAIN   solution_dc;
        CHAR     constraints;         CHAR        moment.constraints;
        CHAR     spare_code;          BITS8       moment.datatypes;
        INT2     spare;
        FLOAT4   scalar_moment;       FLOAT       moment.sc_moment;
        FLOAT4   norm_tens1_1;        FLOAT       moment.norm_ten[6];
        FLOAT4   norm_tens1_2;        FLOAT       moment.norm_ten[6];
        FLOAT4   norm_tens1_3;        FLOAT       moment.norm_ten[6];
        FLOAT4   norm_tens2_1;        FLOAT       moment.norm_ten[6];
        FLOAT4   norm_tens2_2;        FLOAT       moment.norm_ten[6];
        FLOAT4   norm_tens3_1;        FLOAT       moment.norm_ten[6];
        REFERS2  comment_id;
        DOMAIN   comment_dc;
} SUDS_MOMENT;

typedef struct{
        FIXED    structure_type;
        FIXED    structure_len;
        LABEL    mux_waveform_id;
```

```
        DOMAIN  mux_waveform_dc;  STRING         muxdata.netname[4];
        REFERS2 recorder_id;
        DOMAIN  recorder_dc;
        FIXED   len_contr_f;
        STRING  name_contr_f[12];
        REFERS2 clock_cor_id;
        DOMAIN  clock_cor_dc;
        MS_TIME                   nom_beg_time; MS_TIME muxdata.begintime;
        MS_TIME                   nom_end_time;
        FIXED   len_media_l;
        STRING  media_label[16];
        FIXED   len_media_p;
        STRING  media_path[64];
        CODE1   media;
        CODE1   detector;
        CODE1   trigger_type;
        CODE1   event_type;       CHAR           muxdata.descript;
        CODE1   compression;
        CODE1   data_units;
        CHAR    spare_charA;
        CODE1   clock_type;
        INT4    dc_offset;
        FLOAT4  nom_dig_rate;     FLOAT          muxdata.dig_rate;
        INT4    numb_stations;    SH_INT         muxdata.numchans;
        INT4    block_size;       LG_INT         muxdata.blocksize;
        CODE4   data_type;        CHAR           muxdata.typedata;
        INT4    data_length;      LG_INT         muxdata.numsamps;
        REFERS2 comment_id;
        DOMAIN  comment_dc;
} SUDS_MUX_DATA;

typedef struct{
        FIXED   structure_type;
        FIXED   structure_len;
        LABEL   pick_id;
        DOMAIN  pick_dc;
        REFERS2 event_id;
        DOMAIN  event_dc;
        REFERS2 signal_path_id;
        DOMAIN  signal_path_dc;
        REFERS2 waveform_id;
        DOMAIN  waveform_dc;
        FIXED   len_signal_n;
        STRING  signal_name[20];SUDS_STATIDENT feature.fe_name;
        MS_TIME                   time;                          MS_TIME feature.time;
        CODE2   observ_phase;     SH_INT         feature.obs_phase;
                                                 SH_INT          residual.set_phase;
        CODE1   onset_type;       CHAR           feature.onset;
        CODE1   first_motion;     CHAR           feature.direction;
        INT2    spare;
        CODE1   pick_method;      CHAR           feature.data_source;
        CODE1   obs_time_qual;    CHAR           feature.tim_qual;
        CODE1   obs_ampl_qual;    CHAR           feature.amp_qual;
```

```
        CODE1   ampl_units;        CHAR        feature.ampunits;
        INT2    gain_range;        SH_INT      feature.gain_range;
        FLOAT4  amplitude;         FLOAT       feature.amplitude;
        FLOAT4  frequency;         FLOAT       feature.period;
        FLOAT4  obs_azimuth;
        FLOAT4  obs_slowness;
        FLOAT4  rectilinearity;
        FLOAT4  spare_a;
        ST_TIME time_picked;       ST_TIME feature.time_of_pick;
        AUTHOR  authority;         SH_INT      feature.pick_authority; plus
                                               SH_INT      feature.pick_reader;
        FLOAT4  signal_2_noise;    SH_INT      feature.sig_noise;
        REFERS2 comment_id;
        DOMAIN  comment_dc;
} SUDS_PICK;

typedef struct{
        FIXED   structure_type;
        FIXED   structure_len;
        REFERS2 pick_id;           SUDS_STATIDENT residual.re_name;
        DOMAIN  pick_dc;
        REFERS2 solution_id;       LG_INT      residual.event_num;
        DOMAIN  solution_dc;
        REFERS2 vel_model_id;
        DOMAIN  vel_model_dc;
        CODE1   cal_time_qual;     CHAR        residual.set_tim_qual;
        CODE1   cal_ampl_qual;     CHAR        residual.set_amp_qual;
        CODE1   mag_type;
        CHAR    spare;
        FLOAT4  pick_magnitude;
        FLOAT4  residual;          FLOAT       residual.residual;
        FLOAT4  weight_used;       FLOAT       residual.weight_used;
        FLOAT4  delay_used;        FLOAT       residual.delay;
        FLOAT4  azm_2_stat;        FLOAT       residual.azimuth;
        FLOAT4  dist_2_stat;       FLOAT       residual.distance;
        FLOAT4  angle_emerg;       FLOAT       residual.emergence;
        REFERS2 comment_id;
        DOMAIN  comment_dc;
} SUDS_PICK_RESIDUAL;

typedef struct{
        FIXED   structure_type;
        FIXED   structure_len;
        LABEL   processing_id;
        DOMAIN  processing_dc;
        CODE1   process_type;
        CHAR    spare;
        INT2    spare_a;
        AUTHOR  authority;
        CODE4   data_type;
        INT4    data_length;
        REFERS2 comment_id;
        DOMAIN  comment_dc;
```

```
} SUDS_PROCESSING;

typedef struct{
        FIXED   structure_type;
        FIXED   structure_len;
        LABEL   recorder_id;          SH_INT        atodinfo.device_id;
        DOMAIN  recorder_dc;
        FIXED   len_name;
        STRING  recorder_name[12];
        FIXED   len_serial_n;
        STRING  serial_number[12];SH_INT instrument.in_serial;
        CODE4   model;
        FLOAT4  speed;
        CODE1   speed_units;
        CODE1   data_units;           CHAR          stationcomp.data_units;
        CODE1   polarity;
        CODE1   recorder_type;        CHAR          stationcomp.recorder;
        FLOAT4  conv_2_mvolts;        FLOAT         stationcomp.con_mvolts;
                                                    FLOAT         instrument.dig_con;
        FLOAT4  gain;                               SH_INT        stationcomp.atod_gain;
        FLOAT4  clip_value;           FLOAT         stationcomp.clip_value;
        FIXED   len_detect_p;
        STRING  name_detect_p[12];CHAR              detector.dalgorithm;
        INT2    ver_detect_p;         FLOAT         detector.versionnum;
        INT2    spare;
        CODE4   storage_type;         CHAR          instrument.datatype;
                                                    CHAR          stationcomp.data_type;
        REFERS2 comment_id;
        DOMAIN  comment_dc;
} SUDS_RECORDER;

typedef struct{
        FIXED   structure_type;
        FIXED   structure_len;
        LABEL   response_id;          SUDS_STATIDENT calibration.ca_name; and signal_path
        DOMAIN  response_dc;
        FIXED   len_name_resp;
        STRING  name_response[20];
        CODE1   response_type;
        CODE1   input_units;
        CODE1   output_units;
        CHAR    spare_char;
        FLOAT4  maximum_gain;         FLOAT         calibration.maxgain;
        FLOAT4  normalization;        FLOAT         calibration.normaliz;
        FLOAT4  frequency_max;
        FLOAT4  inp_samp_rate;
        INT2    decim_factor;
        INT2    decim_offset;
        FLOAT4  estim_delay;
        FLOAT4  used_delay;
        ST_TIME from_time;            ST_TIME       calibration.begint;
        ST_TIME thru_time;            ST_TIME       calibration.endt;
        AUTHOR  authority;
```

```
        INT4     spare;
        CODE4    data_type;
        INT4     data_length;          SUDS_CALIBR calibration.cal[NOCALPTS];
        REFERS2  comment_id;
        DOMAIN   comment_dc;
} SUDS_RESPONSE;

typedef struct{
        FIXED    structure_type;
        FIXED    structure_len;
        REFERS2  response_id;
        DOMAIN   response_dc;
        FLOAT4   corner_freq;
        FLOAT4   db_per_decade;
} SUDS_RESPONSE_CFS;

typedef struct{
        FIXED    structure_type;
        FIXED    structure_len;
        REFERS2  response_id;
        DOMAIN   response_dc;
        FLOAT4   frequency;
        FLOAT4   amplitude;
        FLOAT4   amplitude_err;
        FLOAT4   phase;
        FLOAT4   phase_error;
        INT4     spare;
} SUDS_RESPONSE_FAP;

typedef struct{
        FIXED    structure_type;
        FIXED    structure_len;
        REFERS2  response_id;
        DOMAIN   response_dc;
        INT4     position;
        INT4     spare;
        FLOAT4   numer_coef;
        FLOAT4   numer_coef_err;
        FLOAT4   denom_coef;
        FLOAT4   denom_coef_err;
} SUDS_RESPONSE_FIR;

typedef struct{
        FIXED    structure_type;
        FIXED    structure_len;
        REFERS2  response_id;
        DOMAIN   response_dc;
        FLOAT4   pole_r;                       COMPLEXX calibr.pole;
        FLOAT4   pole_i;
        FLOAT4   pole_err_r;
        FLOAT4   pole_err_i;
        FLOAT4   zero_r;                       COMPLEXX calibr.zero;
        FLOAT4   zero_i;
```

```
        FLOAT4  zero_err_r;
        FLOAT4  zero_err_i;
} SUDS_RESPONSE_PZ;

typedef struct{
        FIXED   structure_type;
        FIXED   structure_len;
        REFERS2 response_id;
        DOMAIN  response_dc;
        FLOAT4  sensitivity;
        FLOAT4  frequency;
        ST_TIME cal_time;
        INT4    spare;
} SUDS_RESPONSE_SEN;

typedef struct{
        FIXED   structure_type;
        FIXED   structure_len;
        LABEL   sensor_id;
        DOMAIN  sensor_dc;
        REFERS2 response_id;
        DOMAIN  response_dc;
        CODE4   sensor_model;
        FIXED   len_serial_n;
        STRING  serial_number[12];SH_INT instrument.sn_serial;;
        FLOAT4  free_frequency;     FLOAT       instrument.nat_freq;
        FLOAT4  motor_const;        FLOAT       instrument.mot_con;
        FLOAT4  eff_mo_const;
        FLOAT4  sensor_mass;
        CODE1   sensor_type;        CHAR        instrument.sens_type;
        CHAR    pad_type;
        INT2    spare;
        INT2    r_coil;
        INT2    r_crit_damp;
        FLOAT4  eff_damping;        FLOAT       instrument.damping;
        INT2    r_lpad;
        INT2    r_tpad;
        INT2    r_shunt;
        INT2    r_cal_coil;
        FLOAT4  cal_mo_const;
        AUTHOR  authority;
        ST_TIME from_time;          ST_TIME instrument.effective;
        ST_TIME thru_time;
        REFERS2 comment_id;
        DOMAIN  comment_dc;
} SUDS_SENSOR;

typedef struct{
        FIXED   structure_type;
        FIXED   structure_len;
        LABEL   service_id;
        DOMAIN  service_dc;
        REFERS2 station_id;
```

```
        DOMAIN  station_dc;
        FIXED   len_signal_n;
        STRING  signal_name[20];
        ST_TIME visit_time;
        CODE4   authority;
        FIXED   len_reasons;
        CODESTR                 reasons[20];    SH_INT          equipment.reason;
        FIXED   len_actions;
        CODESTR                 actions[20];
        REFERS2 comment_id;
        DOMAIN  comment_dc;
} SUDS_SERVICE;

typedef struct{
        FIXED   structure_type;
        FIXED   structure_len;
        REFERS2 signal_path_id;
        DOMAIN  signal_path_dc;
        REFERS2 component_id;
        DOMAIN  component_dc;
        CODE4   component_type;
        AUTHOR  authority;
        INT4    pos_in_path;
        INT4    channel_number;
        FLOAT4  frequency;
        FLOAT4  attenuation;
        ST_TIME from_time;
        ST_TIME thru_time;
        REFERS2 comment_id;
        DOMAIN  comment_dc;
} SUDS_SIG_PATH_ASS;

typedef struct{
        FIXED   structure_type;
        FIXED   structure_len;
        LABEL   sig_path_cmp_id;
        DOMAIN  sig_path_cmp_dc;
        REFERS2 response_id;
        DOMAIN  response_dc;
        AUTHOR  authority;
        INT2    spare;
        CODE1   polarity;
        CODE1   gain_units;
        FLOAT4  gain_multip;
        CODE4   model;                          SH_INT          equipment.model;
        FIXED   len_serial_n;
        STRING  serial_number[12];STRING equipment.serial[8];
        INT2    setting1;       SH_INT          equipment. knob1;
        INT2    setting2;       SH_INT          equipment.knob2;
        INT2    setting3;
        INT2    setting4;
        FLOAT4  frequency;      FLOAT           equipment.frequency;
        ST_TIME new_battery;
```

```
        CODE4   data_type;
        INT4    data_length;
        REFERS2 comment_id;
        DOMAIN  comment_dc;
} SUDS_SIG_PATH_CMP;

typedef struct{
        FIXED   structure_type;
        FIXED   structure_len;
        REFERS2 connected_id;      SUDS_STATIDENT equipment.this, previous, next;
        DOMAIN  connected_dc;
        INT2    pin_plus;
        INT2    pin_minus;
        INT2    pin_ground;
        CHAR    in_or_out;
        CHAR    spare_char;
} SUDS_SIG_PATH_IO;

typedef struct{
        FIXED   structure_type;
        FIXED   structure_len;
        LABEL   signal_path_id;
        DOMAIN  signal_path_dc;
        REFERS2 station_id;
        DOMAIN  station_dc;
        FIXED   len_signal_n;
        STRING  signal_name[20];SUDS_STATIDENT stationcomp.sc_name;
        FIXED   len_station_n;
        STRING  station_name[8];SUDS_STATIDENT stationcomp.sc_name;
        AUTHOR  network;           SUDS_STATIDENT stationcomp.sc_name;
        CODE1   component_type;
        CODE1   sensor_type;       CHAR           stationcomp.sensor_type
        CODE1   band_type;
        CODE1   gain_type;
        CODE1   polarity;          CHAR           stationcomp.polarity;
        CODE1   amp_response;
        CHAR    path_type;
        CHAR    spare;
        REFERS2 recorder_id;
        DOMAIN  recorder_dc;
        INT4    attenuator;
        FLOAT4  gain_multiplier;
        REFERS2 total_resp_id;
        DOMAIN  total_resp_dc;
        INT2    satellite_hops;
        INT2    sensor_depth;
        INT4    channel;           SH_INT         stationcomp.channel;
                                                  SH_INT        instrument.channel;
        FLOAT4  time_delay;        FLOAT          stationcomp.clock_correct;
        FLOAT4  seismic_delay;     FLOAT          stationcomp.station_delay;
        REFERS2 sensor_id;
        DOMAIN  sensor_dc;
        FLOAT4  sensor_azimuth;    SH_INT         stationcomp.azim;
```

```
        FLOAT4   sensor_dip;          SH_INT     stationcomp.incid;
        ST_TIME  from_time;           ST_TIME stationcomp.effective;
        ST_TIME  thru_time;
        CODE4    data_type;
        INT4     data_length;
        REFERS2  comment_id;
        DOMAIN   comment_dc;
} SUDS_SIGNAL_PATH;

typedef struct{
        FIXED    structure_type;
        FIXED    structure_len;
        REFERS2  event_id;            LG_INT     event.number;
        DOMAIN   event_dc;
        FIXED    len_eq_name;
        STRING   eq_name[20];         STRING     evdescr.eqname[20];
        FIXED    len_country;
        STRING   country[16];         STRING     evdescr.country[16];
        FIXED    len_state;
        STRING   state[16];           STRING     evdescr.state[16];
        INT2     local_time;          SH_INT     evdescr.localtime;
        INT2     num_felt_rep;        SH_INT     event.felt;
        AUTHOR   felt_authority;
        FLOAT4   event_magnitude;FLOAT           event.size;
        AUTHOR   mag_authority;       SH_INT     event.authority; Note many types of authority
        AUTHOR   mm_authority;
        INT2     mm_intensity;        CHAR       event.mintensity;
        CHAR     event_type;          CHAR       event.ev_type;
        CHAR     spare_code;
        CHAR     tectonism;           CHAR       event.tectonism;
        CHAR     waterwave;           CHAR       event.waterwave;
        CHAR     mechanism;           CHAR       event.mechanism;
        CHAR     medium;                         CHAR          event.medium;
        AUTHOR   tect_auth;
        AUTHOR   water_auth;
        AUTHOR   mech_auth;
        AUTHOR   medium_auth;
        INT4     spare;
        REFERS2  comment_id;
        DOMAIN   comment_dc;
} SUDS_SIGNIF_EVENT;

typedef struct{
        FIXED    structure_type;
        FIXED    structure_len;
        LABEL    solution_id;         LG_INT     origin.number;
        DOMAIN   solution_dc;
        REFERS2  event_id;
        DOMAIN   event_dc;
        ST_TIME  time_sol_done;       ST_TIME origin.effective;
        AUTHOR   authority;           SH_INT     origin.authority;
        MS_TIME                       origin_time;   MS_TIME origin.orgtime;
        LATIT    origin_lat;          LONLAT     origin.or_lat;
```

```
        LONGIT  origin_long;        LONLAT      origin.or_long;
        FLOAT4  origin_depth;       FLOAT       origin.depth;
        CODE1   origin_status;      CHAR        origin.or_status;
        CODE1   depth_control;      CHAR        origin.depcontrl;
        CODE1   time_control;
        CODE1   convergence;        CHAR        origin.convergence;
        CODE1   quality;
        CODE1   region_type;
        CHAR    spare_char;
        CODE1   hypo_program;       CHAR        origin.program;
        CODE4   region;                         LG_INT      origin.region;
        REFERS2 contr_file_id;
        DOMAIN  contr_file_dc;
        INT2    hypo_prog_vers;
        INT2    gap_of_stations;SH_INT          origin.gap;
        FLOAT4  rms_of_resids;      FLOAT       origin.res_rms;
        FLOAT4  horiz_error;        FLOAT       origin.err_horiz;
        FLOAT4  depth_error;        FLOAT       origin.err_depth;
        FLOAT4  depth_err_up;
        FLOAT4  depth_err_down;
        FLOAT4  dist_near_stat;     FLOAT       origin.nearstat;
        FLOAT4  near_s_p_time;
        FLOAT4  p2s_vel_ratio;
        INT2    num_stat_good;      SH_INT      origin.num_stats;
        INT2    num_p_rep_good;     SH_INT      origin.rep_p;
        INT2    num_p_used;         SH_INT      origin.used_p;
        INT2    num_s_rep_good;     SH_INT      origin.rep_s;
        INT2    num_s_used;         SH_INT      origin.used_s;
        INT2    num_resid_disc;
        INT2    spare_a;
        CHAR    spare1_char;
        CODE1   pref_mag_type;      SH_INT      origin.mag_type;
        FLOAT4  preferred_mag;      FLOAT       origin.magnitude;
        AUTHOR  pref_mag_auth;
        INT4    spare;
        CODE4   data_type;
        INT4    data_length;
        REFERS2 comment_id;
        DOMAIN  comment_dc;
} SUDS_SOLUTION;

typedef struct{
        FIXED   structure_type;
        FIXED   structure_len;
        REFERS2 solution_id;
        DOMAIN  solution_dc;
        FLOAT4  covar_xx;           FLOAT       error.covarr[10];
        FLOAT4  covar_yy;           FLOAT       error.covarr[10];
        FLOAT4  covar_zz;           FLOAT       error.covarr[10];
        FLOAT4  covar_tt;           FLOAT       error.covarr[10];
        FLOAT4  covar_xy;           FLOAT       error.covarr[10];
        FLOAT4  covar_xz;           FLOAT       error.covarr[10];
        FLOAT4  covar_yz;           FLOAT       error.covarr[10];
```

```
        FLOAT4  covar_tx;           FLOAT         error.covarr[10];
        FLOAT4  covar_ty;           FLOAT         error.covarr[10];
        FLOAT4  covar_tz;           FLOAT         error.covarr[10];
        FLOAT4  std_error;
        FLOAT4  semi_major;
        FLOAT4  semi_minor;
        FLOAT4  major_strike;
        FLOAT4  depth_error;
        FLOAT4  time_error;
        FLOAT4  confidence;
        FLOAT4  spare;
        REFERS2 comment_id;
        DOMAIN  comment_dc;
} SUDS_SOLUTION_ERR;

typedef struct{
        FIXED   structure_type;
        FIXED   structure_len;
        LABEL   source_id;
        DOMAIN  source_dc;
        REFERS2 data_group_id;
        DOMAIN  data_group_dc;
        FIXED   len_ev_name;
        STRING  source_name[12];
        MS_TIME                     origin_time;
        MS_TIME                     orig_org_time;
        LATIT   origin_lat;
        LONGIT  origin_long;
        FLOAT4  origin_elev;
        FLOAT4  origin_depth;
        FLOAT4  water_depth;
        FLOAT4  yield;
        CODE1   coordinates;
        CODE1   event_type;
        CODE1   sweep_type;
        CODE1   taper_type;
        INT2    begin_freq;
        INT2    end_freq;
        INT2    sweep_length;
        INT2    begin_taper;
        INT2    end_taper;
        INT2    signal_lag;
        FLOAT4  source_static;
        AUTHOR  authority;
        REFERS2 comment_id;
        DOMAIN  comment_dc;
} SUDS_SOURCE;

typedef struct{
        FIXED   structure_type;
        FIXED   structure_len;
        REFERS2 ssam_setup_id;
        DOMAIN  ssam_setup_dc;
```

```
        INT4      num_band_chan;
        INT4      spare;
        CODE4     data_type;
        INT4      data_length;
        REFERS2   comment_id;
        DOMAIN    comment_dc;
} SUDS_SSAM_DATA;

typedef struct{
        FIXED     structure_type;
        FIXED     structure_len;
        FLOAT4    upper_freq;
        FLOAT4    lower_freq;
} SUDS_SSAM_PASSBAND;

typedef struct{
        FIXED     structure_type;
        FIXED     structure_len;
        LABEL     ssam_setup_id;
        DOMAIN    ssam_setup_dc;
        FLOAT4    nom_dig_rate;
        INT2      num_band_chan;
        INT2      samp_per_fft;
        CODE4     data_type;
        INT4      data_length;
        REFERS2   comment_id;
        DOMAIN    comment_dc;
} SUDS_SSAM_SETUP;

typedef struct{
        FIXED     structure_type;
        FIXED     structure_len;
        LABEL     station_id;
        DOMAIN    station_dc;
        LATIT     station_lat;          LONLAT     stationcomp.st_lat;
        LONGIT    station_long;         LONLAT     stationcomp.st_long;
        FIXED     len_station_n;
        STRING    station_name[8];
        FIXED     len_old_name;
        STRING    old_name[8];
        CODE4     network;
        FLOAT4    station_elev;         FLOAT      stationcomp.elev;
        REFERS2   ref_stat_id;
        DOMAIN    ref_stat_dc;
        FLOAT4    dist_north;           FLOAT      instrument.local_y;
        FLOAT4    dist_east;            FLOAT      instrument.local_x;
        CODE1     site_precision;
        CODE1     enclosure;            CHAR       stationcomp.enclosure;
        CODE1     site_cond;            CHAR       stationcomp.sitecondition;
        CODE1     status;
        INT2      region;
        CODE2     rock_type;            SH_INT     stationcomp.rocktype; and stationcomp.rockclass;
        CODE1     region_type;
```

```
        CHAR     spare_c;
        INT2     spare;
        FIXED    len_site_d;
        STRING   site_descrip[40];
        ST_TIME  from_time;
        ST_TIME  thru_time;
        REFERS2  comment_id;
        DOMAIN   comment_dc;
} SUDS_STATION;

typedef struct{
        CHAR     sync_char;          CHAR        structtag.sync;
        CODE1    computer_type;      CHAR        structtag.machine;
        INT2     suds_version;
        INT4     struct_number;      SH_INT      structtag.id_struct;
        INT4     struct_length;      LG_INT      structtag.len_struct;
        INT4     length_data;        LG_INT      structtag.len_data;
} SUDS_STRUCTURE_TAG;

typedef struct{
        FIXED    structure_type;
        FIXED    structure_len;
        INT4     structure_num; SH_INT           terminator.structid;
        INT4     spare;
        REFERS2  comment_id;
        DOMAIN   comment_dc;
} SUDS_TERMINATOR;

typedef struct{
        FIXED    structure_type;
        FIXED    structure_len;
        LABEL    user_vars_id;
        DOMAIN   user_vars_dc;
        REFERS2  waveform_id;
        DOMAIN   waveform_dc;
        INT4     waveform_type;
        INT2     spare;
        CHAR     type_zero;
        CHAR     type_one;
        CHAR     type_two;
        CHAR     type_three;
        CHAR     type_four;
        CHAR     type_five;
        CHAR     type_six;
        CHAR     type_seven;
        CHAR     type_eight;
        CHAR     type_nine;
        FLOAT4   zero;
        FLOAT4   one;
        FLOAT4   two;
        FLOAT4   three;
        FLOAT4   four;
        FLOAT4   five;
```

```
        FLOAT4  six;
        FLOAT4  seven;
        FLOAT4  eight;
        FLOAT4  nine;
        REFERS2 comment_id;
        DOMAIN  comment_dc;
} SUDS_USER_VARS;

typedef struct{
        FIXED   structure_type;
        FIXED   structure_len;
        REFERS2 vel_model_id;
        DOMAIN  vel_model_dc;
        FLOAT4  depth_2_top;        FLOAT       layers.thickness;
        FLOAT4  p_vel_top;          FLOAT       layers.pveltop;
        FLOAT4  s_vel_top;          FLOAT       layers.sveltop;
        FLOAT4  depth_2_base;
        FLOAT4  p_vel_base;         FLOAT       layers.pvelbase;
        FLOAT4  s_vel_base;         FLOAT       layers.svelbase;
        CODE2   vel_function;       SH_INT      layers.function;
        CODE2   dens_function;
        FLOAT4  density;
        FLOAT4  attenuation;
        INT4    spare;
} SUDS_VEL_LAYER;

typedef struct{
        FIXED   structure_type;
        FIXED   structure_len;
        LABEL   vel_model_id;       STRING      velmodel.netname[4];
        DOMAIN  vel_model_dc;
        LATIT   A_latitude;         LONLAT      velmodel.latA;
        LONGIT  A_longitude;        LONLAT      velmodel.longA;
        LATIT   B_latitude;         LONLAT      velmodel.latB;
        LONGIT  B_longitude;        LONLAT      velmodel.longB;
        CODE1   model_type;         CHAR        velmodel.modeltype;
        CHAR    spare_char;
        INT2    spare;
        FIXED   len_model_n;
        STRING  model_name[16]; STRING          velmodel.modelname[6];
        ST_TIME from_time;          ST_TIME velmodel.time_effective;
        AUTHOR  authority;
        CODE4   data_type;
        INT4    data_length;
        REFERS2 comment_id;
        DOMAIN  comment_dc;
} SUDS_VEL_MODEL;

typedef struct{
        FIXED   structure_type;
        FIXED   structure_len;
        LABEL   waveform_id;
        DOMAIN  waveform_dc;
```

```
        REFERS2 signal_path_id;
        DOMAIN  signal_path_dc;
        REFERS2 mux_waveform_id;
        DOMAIN  mux_waveform_dc;
        REFERS2 data_group_id;
        DOMAIN  data_group_dc;
        REFERS2 calibration_id;
        DOMAIN  calibration_dc;
        FIXED   len_signal_n;
        STRING  signal_name[20];SUDS_STATIDENT descriptrace.dt_name;
                                               SUDS_STATIDENT lt_name;
        MS_TIME             begin_time;        MS_TIME descriptrace.time_correct; plus begintime
        MS_TIME             end_time;
        MS_TIME             nominal_time;      MS_TIME descriptrace.begintime;
        INT2    local_time;       SH_INT       descriptrace.localtime;
        CODE1   resolution;
        CODE1   data_units;
        AUTHOR  digitized_by;     SH_INT       descriptrace.digi_by;
        INT4    spare_a;
        FLOAT4  nom_dig_rate;     FLOAT        descriptrace.rate;
        FLOAT4  prec_dig_rate;    FLOAT        descriptrace.rate_correct;
        FLOAT4  min_val_data;     FLOAT        descriptrace.mindata;
        FLOAT4  max_val_data;     FLOAT        descriptrace.maxdata;
        FLOAT4  average_noise;    FLOAT        descriptrace.avenoise;
        FLOAT4  dc_removed;
        INT4    num_pos_clip;     LG_INT       descriptrace.numclip;
        INT4    num_neg_clip;
        INT2    num_spikes;
        INT2    num_glitches;
        FLOAT4  weight;                        CHAR        descriptrace.descriptor;
        CODE1   time_source;
        CODE1   gain_ranged;
        CODE1   signal_type;
        CODE1   filter_code;
        CODE1   compression;
        CODE1   time_status;
        CHAR    spare_b;
        CHAR    spare_c;
        INT4    file_offset;      LG_INT       loctrace.beginloc;
        CODE4   data_type;        CHAR         descriptrace.datatype;
        INT4    data_length;      LG_INT       descriptrace.length;
        REFERS2 processing_id;    SH_INT       descriptrace.processed;
        DOMAIN  processing_dc;
        REFERS2 comment_id;
        DOMAIN  comment_dc;
} SUDS_WAVEFORM;
```

SEE ALSO

NAME

ah_to_suds – mapping of UW (University of Washington format) to SUDS

DESCRIPTION

MEMBERS

```
struct muxhead {
        short       nchan;          = not applicable
        long        lrate;          = waveform.nom_dig_rate
        long        lmin;           = waveform.begin_time
        long        lsec;           = waveform.begin_time
        long        length;         = waveform.data_length
        short       tapenum;        = data_group.media_label
        short       eventnum;       = data_group.media_path
        short       flg[10];        = waveform.comment
        char        extra[10];      = user_vars?
        char        comment[80];    = waveform.comment
};

struct stahead {
        cha          name[5];       = signal_path.station_name
        short       lta;            = trigger.longterm_ave
        short       trig;           = trigger.signal_path_id
        short       bias;           = waveform.dc_removed ?
};
```

SEE ALSO