UNITED STATES DEPARTMENT OF INTERIOR
GEOLOGICAL SURVEY

SUDS: Seismic Unified Data System

Peter L. Ward

U. S. Geological Survey
345 Middlefield Road
Menlo Park, CA. 94025

March 29, 1989

Open-File Report 89-188

Contents

SUDS: Seismic Unified Data System

Peter L. Ward

U.S. Geological Survey, MS 977
345 Middlefield Road
Menlo Park, Ca. 94025
415/329-4736

1. Note

This open-file report is a collection of documents written about **SUDS** at many different times. They are collected here as a formality to provide public release of this information.

2. Overview

The Seismic Unified Data System (**SUDS**) is a system for storing, accessing, and processing all types of seismic data efficiently. The goals of **SUDS** are:

1). To promote widespread exchange of data.

2). To promote widespread exchange of programs for analyzing and displaying data.

3). To make it easier for most seismologists to get their work done.

4). To reduce the effort required to write new programs.

5). To provide a transparent way of dealing with the proliferation of computer architectures now often being networked together.

SUDS is a bus. In the figurative sense it provides a vehicle for many people traveling together. Like a computer bus, **SUDS** provides a standardized way to plug together a wide variety of devices or modules so that they can communicate with each other efficiently. **SUDS** is also a workshop, a friendly place containing many of the specific tools needed to complete a project quickly and easily. While **SUDS** is based on standardization, it also provides numerous channels for growth and adaptation to fit the needs of an independent-minded research community. **SUDS** is developed around seismic data, but its methods and major tools apply equally well to any type of data in any field.

Overcoming the coefficient of static friction to learn about a new concept, language, or piece of hardware or software can appear difficult. It seems easier to stick with the familiar. There are many intricate details to **SUDS**, but most people will be able to derive the benefits of **SUDS** without having to learn these details. It is simply necessary to understand the overall concepts and then proceed to use the tools. You can fill in the details later when needed.

The building blocks of **SUDS** are natural, logical groups of data. These data groups are arranged end to end in a stream of information that can be stored and accessed sequentially, for example in a file or on a tape, and can be loaded sequentially into a database system where they can be accessed more randomly. Streams can be directed between different processes running on the same computer or over networks between processes running on different computers.

3. Introduction

The need for easy exchange of seismic data and data-analysis programs is increasing rapidly as digital recording systems are added to existing seismograph stations and networks and as we plan and build major global, national, topical, and portable networks. The advent of relatively inexpensive but powerful desktop workstations makes it feasible for most seismologists to do sophisticated waveform processing that we barely dreamed of a decade ago. In the last decade, major amounts of resources, especially in terms of trained scientific personnel, have been diverted to computer programming. This trend is likely to increase significantly as the data sources increase by orders of magnitude and our processing goals increase in complexity. In theory it is a relatively trivial programming job to convert from one data format to another and to create analysis programs that input or output (I/O) several optional data formats. In practice, however, input and output becomes onerous, and some popular programs contain little scientific code but major amounts of I/O code. Furthermore many scientists now wish to use

database management systems to provide rapid indexed access to large collections of data. Thus a whole new layer of software is becoming necessary.

A major step forward in seismic data organization was made at Lamont-Doherty Geological Observatory with the creation of the **ah** or **ad hoc** data format. Their approach was to group related information into C language structures and then to group structures into a header that is added to waveform data. This grouping leads to a data and file format that interfaces cleanly with dozens of programs or filters that they have developed.

The purpose of this document is to suggest an extension to Lamont's approach that is applicable to a wide variety of seismic data and seismic data processing. The primary concern is to define a **common but succinct** set of data that would typically be archived for seismic stations and networks and to group these data in a manner that will be efficient in terms of storage space, processing time, and programming time. The secondary concern is to make sure that this core format is readily extensible to meet the developing needs of research pursued by independent thinkers spread around the globe. If a common ground can be defined, we will all be able to exchange easily data and, even more important, data analysis programs. **Such exchange would provide the most rapid route to accomplishing our individual and collective scientific goals.**

4. Scope

My interest in a data system relates primarily to local seismograph networks and the flavor of this document reflects this interest. Most other users of earthquake related seismic data have similar needs, however. We all use waveforms, descriptors of waveforms, features picked from waveforms, and events calculated from these features. There are many areas where these interests overlap with people doing seismic refraction and reflection processing, strong ground motion analysis, etc. Thus I wish to encourage readers who have quite different interests not to be put off by the emphasis on local earthquakes and to provide advice on how their interests can be represented.

I have tried to take into account other major interests such as the database designed for the Center for Seismic Studies (Berger, J., North, R.G., Goff, R.C., and Tiberio, M.A., 1984, **A seismological data base management system:** Bull. Seism. Soc., v. 74, no. 5, p. 1849-1862)and the **Proposed Seismic Data Distribution and Interchange Format** developed at Albuquerque for world-wide data. I have also reviewed the typical PDE tape formats, ISC formats, and formats being used or developed for CALNET, the Southern Alaska Net, and SAC (Seismic Analysis Code). Some details from these representations will need to be added later to the structures discussed here.

The goal of this document is not to present THE ultimate data organization for seismology but to suggest and illustrate an approach to see if others would think it valuable to march to the same drummer. We could then work together to develop Version 1.

5. Levels of Agreement

There are four primary levels for agreement:

1.) Agree that a common, widely used, and extensible data format is desirable.

2.) Agree on the groups of data that are **most** important.

3.) Agree on the specific variables that most people need for present or anticipated routine and research processing.

4.) Agree on the actual bit representation of each variable, the manner in which variables will be grouped, and the manner in which groups will be combined to form files, archive tapes or disks, and databases.

In this document these levels are intermixed in order to organize the topics logically. Readers may wish to focus their attention at specific levels depending on their degree of interest.

6. Groupings of Seismic Data

Computing typically consists of grabbing groups of data, processing these data, and creating new groups of data or images of data. Thinking involves much the same process. Just as our thinking is typically clearer when we can quickly and easily grab a small group of coherent ideas often represented by a precise scientific term, computer programs tend to be more efficient and shorter when we can easily grab related pieces of data.

Seismic data have certain natural groupings. For example:

1). Information about the station and component on which data was collected: Name, location, gain settings, component orientation, instrument type, etc.

2). Information about a station component, such as calibration information, that typically involves interpretation and calculation and may not be available at the time a waveform is recorded.

3). The digital waveform data for an event often expressed in digital samples or volts.

4). Descriptive information about a waveform such as maximum and minimum amplitude, average noise, signal to noise ratio, number of clipped samples, etc. Such information is not only useful for scaling and plotting waveforms but also for searching for waveforms with certain characteristics.

5). Information derived from a waveform such as phase arrival time, phase first motion, phase period, amplitude, coda length, etc.

6). Event information resulting from analysis of a group of phases, etc.

As research processing continues more derivative forms of data arise:

7). The waveform data expressed as ground velocity or displacement or as spectra or in other numerically filtered forms such as rotated or summed components.

8). "Phase" type data derived by advanced processing techniques.

These groups of data are created at different times. Thus before an event is detected by an online computer, only group 1 exists. After the event is detected groups 1 and 3 exist. After a small amount of automatic processing groups 1, 3, and 4 exist. The other groups might be generated automatically but generally will only be believed after human interaction.

A general data format should recognize these natural groupings and should allow for the fact that not all groups exist or are utilized at the same time.

7. Overview

A suggested data model is summarized in Figure 1 that reflects the general flow of research which typically involves installation of instruments at many stations, observation of waveforms, interpretation of many features from each waveform, and processing these features to derive results. The topics in this figure are organized from those we create and thus know well at the top to those that involve the most interpretation and calculation at the bottom. Each box represents a topic or subject that has numerous attributes and each box will ultimately be represented by a data structure. The arrows with double stems designate the one-to-many relationships that are typical in these data. For one earthquake there may be many calculated origins. For one origin there are many arrival times. For one station there are many waveforms. For one waveform there may be many phase arrivals. Most database systems are built around these one-to-many relationships. A simple example is a table that presents data on a given topic or subject. The columns represent features of the subject and the rows represent instances of these features. In relational database terminology, the table is a relation, the columns are attributes, and the rows are tuples. The fundamental reason for establishing a new topic is to be able to represent succinctly these one-to-many relationships. The selection of topics is relatively straight-forward at the top and bottom of Figure 1 but becomes less clear in the middle. For example, most researchers group information on stations and events in different tables or files. On the other hand should information on moment be grouped with magnitude or even contained within the origin? Several principles appear to apply:

Logic: In many cases the data fall into natural groupings as discussed above.

History: Some features do not exist for much of the available data perhaps due to tradition or the development of new features such as moment. Thus while logically moment might be grouped with magnitude, it is typically not available for small earthquakes nor for most older earthquakes.

Extensibility: Uncertainties for earthquake locations logically should be included with the origin information, but there are a variety of ways to calculate and represent the precision. Thus it seems reasonable to include a simple measure of precision with the origin especially for uniform searching of the data but to assume that different researchers may need different ways to represent the precise errors such as by ellipsoids, by the covariance matrix, etc. Similarly there are many ways to represent the calibration of an instrument. While such extensibility will be needed to travel into the future, we should still strive where possible to find single, compact representations for common parameters from which a variety of traditional representations can be readily derived.

Storage Efficiency: Most computer efficient database and file formats involve allocation of a fixed record size for each instance. Inclusion of many attributes that exist rarely will typically waste significant storage space. If these same attributes are relegated to a new topic, then they need only be stored when they exist.

Half of the topics in Figure 1 can be readily defined to meet common needs either because the nature of these topics is closely related to experimental design or, as in the case of events, these topics have been traditionally tabulated for example by the ISC or NEIS. These topics represent the vast majority of the data and these are the topics that are most typically searched. Thus development of a unified data system for most of the data is not particularly difficult. **Probably the greatest difficulty and disagreement regarding common formats will center on the rows marked Interpretation and Calculation.** This is the area where there is the greatest diversity and where there will be the greatest need for expandability in the future. My approach here is to define groups commonly used making sure these groups can be indexed from the one side of the one-to-many relationships. New groups can then be defined as needed.

The box marked Association in Figure 1 represents the many ways we may chose to relate observations to calculations. Association is implementation dependent. In a database there might be a separate topic or relation that does this association. In the file system or on tape, this association is normally done by grouping the information in contiguous logical areas. Thus a disk or tape file might contain all data applicable to a single event.

8. Data Representation

The groups of data will exist in different forms with different goals:

A). Off-line archival storage that ideally should include all relevant information about a given event and its data in contiguous storage space.

B). On-line storage in a format that is indexed for rapid retrieval probably contained within a database management system. Typically for lack of sufficient storage devices, only waveforms of events being analyzed would be stored on line, but descriptive information of all events should be available.

C). On-line storage in files using the file system to specify groupings.

D). In-core storage as represented in analysis programs.

E). In some cases the original events as output by the online detector might be archived.

Choice of data storage format involves tradeoffs. Using **ascii** files for storage enhances readability, allows use of existing editors and filters, and makes portability among machines easy, but reduces efficiency in reading, writing, storing and retrieving data. Efficiency has become an issue on the larger networks and will be an increasing problem as we expect to collect increasing amounts of data and to process these data with increasingly sophisticated tools. Since waveform data are most naturally and normally stored in binary format, I believe it is not much more of a problem if all the data are stored in binary format. I like Lamont's approach to using a group of data structures, each of which groups related data. Furthermore the database that we are presently evaluating as most promising for seismic

purposes, **db_vista**, uses data structures as the primary mode of data storage and retrieval. One of db_vista's major attributes, is speed. We can assure maximum machine efficiency with data structures. These structures also provide for excellent programming efficiency in C, Pascal, Modula2, PL/1 and VMS Fortran. It is rumored that the new Fortran standard expected this year will include structures.

Thus from my point of view, chosing data formats means first agreeing on what data needs to be stored and then designing a set of data structures that can be used as modules in storage, retrieval, and processing of seismic data. The rest of this document concerns my suggestions for these structures. It is assumed that while these structures are being adopted, utilities will be written to readily convert these structures into and out of ascii format. A good example of such a filter is the Lamont command **ah2asc**. Similarly it is anticipated that a common library of low-level routines will be made available for identifying and utilizing these structures. The structures are listed with a description of the data to be stored in the comment field on the right side of the page and the actual allocation of memory space on the left side of the page. Many readers may wish to focus only on the right side of the pages to be sure the appropriate information is stored.

9. Some principles:

a). The structures should be self identifying so that in a general case such as a tape archive, the reading program could determine what structures exist, in which order they exist, and whether the program knows how to deal with them.

b). The structures should define common groupings of interest, but new structures should be definable in the future. The structures discussed below might form a core set that is very extensible. The programs using these structures must be able to identify each structure and to decide whether or not they can be interpreted by that particular program.

c). There is typically a conflict between storage efficiency and generality. These structures are an attempt to ride the middle ground. I encourage you to consider other variables that should be added but to temper your first urge with realism.

d). To enhance portability among machines, some effort should be made to align appropriate byte boundaries.

10. A Short Primer on C Structures

The appendix is written in C syntax although the ideas presented here do not depend on adoption of C. A C structure is a grouping of variables of mixed types in a contiguous section of memory so that the whole structure can be passed between subroutines as a unit or by a pointer. In its simplest form a C structure is similar to a Fortran common block. In fact on many compilers Fortran common blocks are referenced in C as structures and C structures can be made available to Fortran programmers either as common blocks or, when more complex, as a small library of subroutines. Passing structures between routines and on and off of disk or tape is highly efficient.

In C new variable types can be defined using **typedef**. **typedef x y** means henceforth define all variables of type y to be type x. Thus **typedef int phase; phase x;** means x is of type phase which is of type int. Typedefs are used to make the code more modular, readable, and portable and to isolate custom definitions. Use of typedefs will allow us in the future to take advantage of new languages such as C++ that allow complete specification of new variable types so that the compiler can check for improper usage. The following basic typedefs are used throughout this document:

typedef char	**CHAR**	A single ascii character
typedef char	**STRING**	A character string, null byte terminated

STRING c[10] means a character string with a maximum length of 10. The active length may to 0 to 9 characters terminated by a null byte. Thus note that if a station name is, for example, defined as STRING name[6], it can contain only 5 characters.

typedef char	**BITS8**	An 8 bit field, used for on-off flags
typedef short	**SH_INT**	A 16 bit signed integer
typedef long	**LG_INT**	A 32 bit signed integer
typedef float	**FLOAT**	A 32 bit floating point number, IEEE

typedef double **DOUBLE** A 64 bit double precision number, IEEE

11. Special Types of Variables

TIME: A single number representation of time provides the most efficient form for storage, search, and computation. Such a number is typically defined in terms of seconds from some standard time. One possible standard is that used by the UNIX operating system, which is now widely used in seismology. In UNIX time is kept in terms of the number of seconds since 00:00 GMT on January 1, 1970. The newest releases of UNIX also provide for a structure with two long integers, the number of seconds and the number of milliseconds. A double precision number takes the same storage as 2 longs and can provide time resolution finer than microseconds from the birth of Christ to a long time in the future. This precision is more accurate than needed in seismology. Thus I adopt the standard:

typedef DOUBLE ms_time;

The value of **ms_time** variables can then be easily defined from the system clock and **ms_time** variables can be easily converted to ascii strings and other representations using UNIX utility subroutines.

A common use of time is to identify when an instrument was changed or calibrated, when a location was calculated, etc. In this case accuracy to 1 second is more than adequate. Given the number of cases where such a timestamp is desirable, the use of only 4 bytes of storage becomes valuable. Thus I adopt a second type of time:

typedef LG_INT st_time

LONGITUDE-LATITUDE: A single number representation for degrees is also desirable for storage, searching, and computation. By defining

typedef LG_INT lonlat;

and counting in units of millionths of degrees, we obtain a variable that uses only 4 bytes, is easily derived from true degrees, and is accurate to about 0.1 meters. This same type could be used for all angles, but in most cases a FLOAT seems adequate.

IDENTIFIER: Each phase, waveform pick, etc. needs to be associated with a given station, instrument type, and component. It is useful to adopt a structure of this information:

```
typedef struct {
        STRING  network[4];
        STRING  name[5];
        CHAR    component;
        CHAR    type;
        CHAR    recorder;
} ident;
```

Component would typically be **v, n, e, r, t** but could be any other character. A list of common types of components and instruments would need to be developed and a subroutine written that returns a character string describing each code represented by a character or short integer.

12. Structures and Their Use

The structures are listed in the Appendix in a manner that is compilable. Before getting lost in the details of each structure, it is important to understand how these structures might be used. The reason for defining structures is to group related variables in a manner that they can be readily accessed as a unit. The bit representation of this unit would be identical whether the unit exists on disk or in memory, it should be identical on magnetic tape for the most commonly used computers, and probably would be identical when the unit exists in a database depending on the database design. User programs would read or write these units and would access individual variables by addressing them as components of these units. In Fortran a subroutine would be called to read the next unit, load the values into a common block, and return the number of the common block loaded. Of course the values could also be passed as subroutine arguments when desired. A fundamental feature of this design is that structures are identifiable, so that they can be combined in any order deemed appropriate and any subset of structure types could be used. Let's look at specific examples.

In some seismic networks the data are collected on a computer with a detector algorithm and written to tape for transport to another computer for processing. Each file on this tape might be

formated as follows:

For each station in order data are multiplexed:
strucktag; stationtrace;
strucktag; muxdata;
strucktag; data;

In a different application the multiplexed data might be transmitted to a central recording center via satellite. Since the station information is known to the receive site, the data format might simply be:

strucktag; muxdata;
strucktag; data;

Many seismic networks do automatic processing online. The data would be demultiplexed and might be stored in a disk or tape file as:

For each station:
strucktag; trace; strucktag; data;

We might wish to integrate the same data into a database. In this case the trace and and loctrace structures would be loaded into the database and the preceding file either kept as is or the trace structures could be removed.

A useful format for processing might be labeled by origin and contain:

strucktag; event; strucktag origin;
For each station:
strucktag; trace; structag; data;
For each marker:
strucktag; marker;

An archive tape on the other hand might contain most of the structures as follows:

structag model;
For each layer:
strucktag; layers;
For each event:
strucktag; event;
For each station:
strucktag; trace; structag; data;
For each origin:
strucktag; origin; strucktag; error;
For each station marker:
strucktag; marker; strucktag eventread;

The point is that the groupings can be set to meet local needs and to reflect the order and degree of processing. Programs would typically ask for the next structure and proceed according to what that structure is. A program could skip unknown structures or complain about them as desired.

The simplest way to link relevant structures is to group them together within a file. Another way is to include header information for example about each event within each marker. This is essentially how a relational database is usually set up. A network database, on the other hand, stores these links invisibly for the user. Putting the links within the structures creates less ambiguity but increase the complexity of implementation details and the storage space required. I have designed these structures to minimize redundancy and to use physical grouping in files to define logical grouping. Thus all arrivals pertain to the previous mentioned solution. All features refer to the previously described waveform, and so forth.

13. Physical Tape Format

The unified data system does not require any particular physical tape format. The data are treated as a stream of information and can be blocked onto tape in any number of ways. I think the easiest is to use physical block sizes of 20 times 512 or 10,240 bytes or less and to truncate the last block in a record. This upper limit is the largest block some systems will read. Unix **tar** tapes provide a nice way to label and manage multiple files but other standards such as an ANSI Labeled tape format might work. A standard byte order will need to be adopted and filters provided to swab bytes and change

data types as necessary. It would be most efficient to adopt the bit formats of the most heavily used computers in Seismology. Another more general approach would be to use the eXternal Data Representation language developed for networking.

14. Examples

An example of a structure viewed using **st2asc** follows:

```
% ls -l st.sample
-rw-r--r--  1 ward          76 May 20 13:41 st.sample


% st2asc st.sample
5 64 0 "kat" "ksv" 'v' 1 0 0 60.130000 -154.230000 675.000000 'b' 0 't'
   'u' 115 'g' 'v' 's' 'd' 'n' 4 1000000.000000 1024.000000 55.669998 581100999


% st2asc -v st.sample
    STRUCTURE: stationcomp (length=64+0=64 bytes)
    station/component    :
        STRUCTURE: station ident
        network              :   "kat"
        station name         :   "ksv"
        component            :   'v'
        instrument type      :   1        {sp usgs}
    component azimuth    :       0
    component incidence  :       0
    latitude             :       60.130000
    longitude            :       -154.230000
    elevation, meters    :       675.000000
    enclosure            :       'b'
    annotated comment    :       0        {no comment}
    recorder type        :       't'
    rock class           :       'u'
    rock type            :       115      {this code undefined}
    site condition       :       'g'
    sensor type          :       'v'
    data type            :       's'
    data units           :       'd'
    polarity             :       'n'
    status               :       4        {filtered}
    maximum gain         :       1000000.000000
    clipping value       :       1024.000000
    convert factor       :       55.669998
    effective date       :       May 31, 1988 16:56 39 GMT
```

An example of a data file viewed with **stdescribe** follows:

```
% stdescribe st.871005040607

File st.871005040607:
 0   5 stationcomp   length  64 +     0 bytes  kat.ksv.v.1
 1   7 descriptrace  length  52 +  4312 bytes  kat.ksv.v.1
 2   5 stationcomp   length  64 +     0 bytes  kat.kbte.e.1
```

```
 3   7 descriptrace  length  52 +  4312 bytes  kat.kbte.e.1
 4   5 stationcomp   length  64 +     0 bytes  kat.kbtn.n.1
 5   7 descriptrace  length  52 +  4312 bytes  kat.kbtn.n.1
 6   5 stationcomp   length  64 +     0 bytes  kat.kbtv.v.1
 7   7 descriptrace  length  52 +  4312 bytes  kat.kbtv.v.1
 8   5 stationcomp   length  64 +     0 bytes  kat.krbe.e.1
 9   7 descriptrace  length  52 +  4312 bytes  kat.krbe.e.1
10   5 stationcomp   length  64 +     0 bytes  kat.krbn.n.1
11   7 descriptrace  length  52 +  4312 bytes  kat.krbn.n.1
12   5 stationcomp   length  64 +     0 bytes  kat.krbv.v.1
13   7 descriptrace  length  52 +  4312 bytes  kat.krbv.v.1
14   5 stationcomp   length  64 +     0 bytes  kat.ksm.v.14
15   7 descriptrace  length  52 +  4312 bytes  kat.ksm.v.14
16   5 stationcomp   length  64 +     0 bytes  kat.ktp.v.1
17   7 descriptrace  length  52 +  4312 bytes  kat.ktp.v.1
18   5 stationcomp   length  64 +     0 bytes  kat.kkcn.n.1
19   7 descriptrace  length  52 +  4312 bytes  kat.kkcn.n.1
20   5 stationcomp   length  64 +     0 bytes  kat.pub.v.14
21   7 descriptrace  length  52 +  4312 bytes  kat.pub.v.14
22   5 stationcomp   length  64 +     0 bytes  kat.blm.v.14
23   7 descriptrace  length  52 +  4312 bytes  kat.blm.v.14
24   5 stationcomp   length  64 +     0 bytes  kat.kkce.e.1
25   7 descriptrace  length  52 +  4312 bytes  kat.kkce.e.1
26   5 stationcomp   length  64 +     0 bytes  kat.kkcv.v.1
27   7 descriptrace  length  52 +  4312 bytes  kat.kkcv.v.1
28   5 stationcomp   length  64 +     0 bytes  kat.www.v.14
29   7 descriptrace  length  52 +  4312 bytes  kat.www.v.14
30   5 stationcomp   length  64 +     0 bytes  kat.kkr.v.1
31   7 descriptrace  length  52 +  4312 bytes  kat.kkr.v.1
```

An example of a data files viewed with **st2asc** follows:

% st2asc -v st.871005040607

```
STRUCTURE: stationcomp (length=64+0=64 bytes)
station/component    :
    STRUCTURE: station ident
    network              :   "kat"
    station name         :   "ksv"
    component            :   'v'
    instrument type      :   1        {sp usgs}
component azimuth    :       0
component incidence  :       0
latitude             :       58.591702
longitude            :       -155.320007
elevation, meters    :       488.000000
enclosure            :       '_'
annotated comment    :       0        {no comment}
recorder type        :       '_'
rock class           :       '_'
rock type            :       0        {none given}
site condition       :       '_'
sensor type          :       '_'
```

```
    data type            :       's'
    data units           :       '_'
    polarity             :       '_'
    status               :       0       {none}
    maximum gain         :       NODATA
    clipping value       :       0.000000
    convert factor       :       NODATA
    channel              :       -32768
    spare                :       11648

    STRUCTURE: descriptrace (length=52+4312=4364 bytes)
    station/component    :
        STRUCTURE: station ident
        network              :   "kat"
        station name         :   "ksv"
        component            :   'v'
        instrument type      :   1       {sp usgs}
    beginning time       :       Oct 5, 1987 04:06 07.077 GMT
    local time diff      :       NODATA
    data type            :       's'
    data descriptor      :       '_'
    digitized by         :       0       {not given}
    processed by         :       0       {not given}
    number of samples    :       2156
    samples per second   :       100.000000
    minimum data value   :       -14.000000
    maximum data value   :       73.000000
    average noise        :       212.949997
    num clipped samples  :       0
    Waveform Data:
       263    131    -14    146    175    219    175     14    205    234
       234    175    146    161    234    234    175     29    219    190
       190    161    190    146    219    249    161    146    190    292
       190    175    146    131    146    249     58     43    146    131
       102     73    -14    102    205    146     73    -14     58    117
       205    131     14    205    117    190    205     14    117    131
       161     58     73     73    161    175     58    102    -14    161
       161    102     29    146     73    263    131    102     43    117
       190    263    161    131    205    307    307    190    117    190
       234    234    190    146    234    336    263    117    234    380
       351    175    161    322    307    366    234    205    380    424
       263    249    175    263    351    234    263    263    307    351
```

etc.....

An example of reading structure in a C program:

```
stream = st_open(input_file, "r");
while ( st_get ( &st_ptr, &st_type, &st_len, stream)!=EOF) {
  switch ( st_type ) {
    case STATIONCOMP:
          load station list;
          break;
    case FEATURE:
          load feature list;
```

```
            break;
        case DESCRIPTRACE:
            plot trace;
            break;
        default: st_free ( st_ptr );
    }
}
st_close(stream);
```

```
/usr/suds
.
total 10
-rw-r--r--  1 ward          927 Mar 29 10:25 README
drwxr-sr-x  3 ward          512 Jan 24 17:05 SYS/
drwxr-sr-x  2 ward          512 Mar 23 11:23 bin/
drwxr-sr-x  2 ward          512 Mar 17 16:12 doc/
-rw-r--r--  1 ward           12 Mar 29 11:28 file.list
drwxr-sr-x  2 ward          512 Jan 26 16:01 files/
drwxr-sr-x  3 ward          512 Mar 23 11:40 include/
drwxr-sr-x  2 ward          512 Nov  1 10:54 lib/
drwxr-sr-x  4 ward          512 Dec 28 11:46 man/
drwxr-sr-x  4 ward          512 Nov  1 10:54 src/
./bin
total 888
-rwxr-xr-x  1 ward       131072 Nov 18 08:35 ah2st*
-rwxr-xr-x  1 ward       114688 Nov 18 08:36 asc2st*
-rwxr-xr-x  1 ward       122880 Nov 14 09:41 demux*
-rwxr-xr-x  1 ward       114688 Mar 23 11:23 st2ah*
-rwxr-xr-x  1 ward       114688 Nov 18 08:36 st2asc*
-rwxr-xr-x  1 ward        90112 Nov 18 08:36 stdescribe*
-rwxr-xr-x  1 ward       172032 Nov 18 08:36 stedit*
./doc
total 72
-rw-r--r--  1 ward        36158 Mar 29 10:22 suds.open.file
-rw-r--r--  1 ward        36158 Mar 29 10:22 suds.open.file%
./include
total 78
-rw-r--r--  1 ward           98 Nov 17 16:17 Makefile
-rw-r--r--  1 ward           98 Nov 17 16:17 Makefile%
drwxr-sr-x  2 ward          512 Mar 22 15:54 SCCS/
-rw-r--r--  1 ward         2906 Mar 22 15:51 ahheader.h
-rw-r--r--  1 ward         2895 Mar 22 15:51 ahheader.h%
-r--r--r--  1 ward          828 Nov 17 16:18 st_error.h
-rw-r--r--  1 ward        20704 Mar 22 15:55 suds.h
-rw-r--r--  1 ward        20703 Mar 22 15:55 suds.h%
-r--r--r--  1 ward         5954 Nov 17 16:16 suds_codes.h
-r--r--r--  1 ward        19586 Nov 17 16:16 suds_descr.h
./include/SCCS
total 56
-r--r--r--  1 ward            0 Nov 17 15:59 2036.595814378
-rw-r--r--  1 ward           31 Mar 22 15:50 p.ahheader.h
-rw-r--r--  1 ward           31 Mar 22 15:54 p.suds.h
-r--r--r--  1 ward         3177 Nov 17 16:10 s.ahheader.h
-r--r--r--  1 ward         1090 Nov 17 16:10 s.st_error.h
-r--r--r--  1 ward        20981 Nov 17 16:11 s.suds.h
-r--r--r--  1 ward         6216 Nov 17 16:11 s.suds_codes.h
-r--r--r--  1 ward        19850 Nov 17 16:11 s.suds_descr.h
./lib
total 0
./man
total 3
-rwxrwxr-x  1 ward          539 Nov 17 11:19 MAKELINKS*
drwxr-sr-x  2 ward          512 Mar 29 11:00 man1/
drwxr-sr-x  2 ward          512 Nov 17 11:14 man3/
./man/man1
total 31
-rw-r--r--  1 ward         1213 Nov  1 10:54 ah2st.1
-rw-r--r--  1 ward         1651 Nov  1 10:54 asc2st.1
-rw-r--r--  1 ward          882 Nov  1 10:54 cdd.1
-rw-r--r--  1 ward         1784 Nov  1 10:54 plotatod.1
-rw-r--r--  1 ward        13956 Nov  1 10:54 rsx.1
-rw-r--r--  1 ward         2845 Nov  9 09:22 scanseis.1
-rw-r--r--  1 ward          794 Mar 23 12:29 st2ah.1
-rw-r--r--  1 ward         1984 Nov  1 10:54 st2asc.1
-rw-r--r--  1 ward          712 Nov  1 10:54 stdescribe.1
-rw-r--r--  1 ward         2403 Nov  1 10:54 stedit.1
./man/man3
total 31
-rw-r--r--  1 ward          911 Nov  1 10:54 asc2field.3
-rw-r--r--  1 ward         3456 Nov  1 10:54 atod.3
-rw-r--r--  1 ward         1024 Nov  1 10:54 descr_trace.3
-rw-r--r--  1 ward          907 Nov  1 10:54 find_code.3
-rw-r--r--  1 ward          926 Nov  1 10:54 st_close.3
-rw-r--r--  1 ward         2698 Nov  1 10:54 st_error.3
-rw-r--r--  1 ward         1511 Nov  1 10:54 st_get.3
-rw-r--r--  1 ward          875 Nov  1 10:54 st_init.3
-rw-r--r--  1 ward         7940 Nov  1 10:54 st_intro.3
-rw-r--r--  1 ward         2033 Nov  1 10:54 st_open.3
-rw-r--r--  1 ward          787 Nov  1 10:54 st_put.3
-rw-r--r--  1 ward         1715 Nov  1 10:54 st_seek.3
-rw-r--r--  1 ward         2378 Nov  1 10:54 st_time.3
-rw-r--r--  1 ward          896 Nov  1 10:54 st_unget.3
./src
total 2
drwxr-sr-x  6 ward          1024 Mar 29 10:30 cmd/
drwxr-sr-x  3 ward           512 Nov  1 10:54 lib/
./src/cmd
total 1000
-rw-r--r--  1 ward           815 Nov 18 08:26 ,Makefile
-r--r--r--  1 ward           891 Mar 22 09:45 Makefile
drwxr-sr-x  2 ward           512 Mar 22 09:44 SCCS/
-rwxr-xr-x  1 ward        131072 Nov 18 08:27 ah2st*
-r--r--r--  1 ward          8628 Nov 17 15:55 ah2st.c
-rwxr-xr-x  1 ward        114688 Nov 18 08:27 asc2st*
-r--r--r--  1 ward          5699 Nov 17 16:18 asc2st.c
drwxr-sr-x  2 ward           512 Mar 17 14:56 atod/
drwxr-sr-x  2 ward           512 Mar 29 10:44 demux/
drwxr-sr-x  3 ward          1024 Mar 23 11:30 eqdetect/
-rw-r--r--  1 ward        154368 Mar 23 08:34 oldah
-rwxr-xr-x  1 ward        114688 Mar 23 11:08 st2ah*
-rw-r--r--  1 ward          5591 Mar 23 12:31 st2ah.c
-rw-r--r--  1 ward         20776 Mar 23 11:08 st2ah.o
-rwxr-xr-x  1 ward        114688 Nov 18 08:27 st2asc*
-r--r--r--  1 ward         10375 Nov 17 16:18 st2asc.c
-rwxr-xr-x  1 ward         90112 Nov 18 08:27 stdescribe*
-r--r--r--  1 ward          1704 Nov 17 16:18 stdescribe.c
-rwxr-xr-x  1 ward        172032 Nov 18 08:28 stedit*
-r--r--r--  1 ward         19324 Nov 17 16:19 stedit.c
./src/cmd/atod
total 765
-rw-r--r--  1 ward          3154 Nov  1 10:55 CDDatod.c
-rw-r--r--  1 ward           372 Nov  1 10:55 Makefile
-rw-r--r--  1 ward          6756 Nov  1 10:56 atod.installation
-rwxr-xr-x  1 ward         98304 Nov  1 10:56 cdd*
-rw-r--r--  1 ward          5412 Nov  1 10:56 cdd.c
-rw-r--r--  1 ward          2110 Nov  1 10:56 parkfield.stats
-rwxr-xr-x  1 ward        638976 Nov  1 10:56 plotatod*
-rw-r--r--  1 ward         12582 Nov  1 10:56 plotatod.c
-rwxr-xr-x  1 ward            71 Nov  1 10:55 showatod*
-rw-r--r--  1 ward          1280 Nov  1 10:55 st.sc.parkfield
./src/cmd/eqdetect
total 788
-rw-r--r--  1 ward           895 Dec 28 12:18 Makefile
-rw-r--r--  1 ward           343 Jan 18 14:51 README
-rw-r--r--  1 ward          1076 Nov  1 10:56 Testnet.c
-rwxr-xr-x  1 ward         32768 Nov  1 10:56 a.out*
-rw-r--r--  1 ward          5622 Nov  1 10:56 args.c
-rw-r--r--  1 ward          5922 Nov  1 10:56 cache.c
-rw-r--r--  1 ward          4304 Nov  1 10:56 diag.c
```

```
-rw-r--r--  1 ward        2762 Nov  1 10:56 dynflag.c
-rw-r--r--  1 ward      286720 Nov  1 10:56 eq.test
-rwxr-xr-x  1 ward          99 Dec 28 12:10 eqd.park*
-rwxr-xr-x  1 ward      180224 Nov  1 10:56 eqdetector*
-rw-r--r--  1 ward        3481 Nov  1 10:56 eqdetector.ls
-rw-r--r--  1 ward        9821 Nov  1 10:56 eqdetector.c
-rw-r--r--  1 ward        8217 Nov  1 10:56 eqdetector.h
-rw-r--r--  1 ward        1738 Dec 28 12:14 eqdetector.options
-rw-r--r--  1 ward        6947 Feb  1 13:35 getsweep.c
-rw-r--r--  1 ward         449 Oct 25 14:44 gettim_.c
-rw-r--r--  1 ward        2054 Nov  1 10:56 header.h
-rwxr-xr-x  1 ward       32768 Nov  1 10:56 inspect*
-rw-r--r--  1 ward          94 Nov  1 10:56 katmai.subnets
-rw-r--r--  1 ward         284 Nov  1 10:56 key.h
-rw-r--r--  1 ward        2127 Nov  1 10:56 lpa.h
-rw-r--r--  1 ward        1924 Nov  1 10:56 lpsopen.c
-rw-r--r--  1 ward          74 Nov  1 10:56 online.run
-rw-r--r--  1 ward         123 Dec 28 12:26 online.run.vpi
-rw-r--r--  1 ward         174 Nov  1 10:56 online.sta
-rw-r--r--  1 ward         366 Nov  1 10:56 online.sta.china
-rw-r--r--  1 ward        7177 Nov  1 10:56 out.c
-rw-r--r--  1 ward         103 Nov  1 10:56 parkfield.subnets
drwxr-sr-x  2 ward         512 Mar 17 15:59 rtp/
-rw-r--r--  1 ward         229 Nov  1 10:56 rtp.s
-rw-r--r--  1 ward      141528 Nov  1 10:56 st.880805221942
-rw-r--r--  1 ward         890 Nov  1 10:56 testtime.c
-rw-r--r--  1 ward       16004 Nov  1 10:56 triggers
./src/cmd/eqdetect/rtp
total 61
-rw-r--r--  1 ward        1289 Nov  1 10:56 Makefile
-rw-r--r--  1 ward          47 Mar 17 15:59 README
-rw-r--r--  1 ward        2716 Nov  1 10:56 eqdaemon.c
-rw-r--r--  1 ward        1713 Nov  1 10:56 eqdetector.c
-rw-r--r--  1 ward        1872 Nov  1 10:56 eqspool.c
-rw-r--r--  1 ward        3941 Nov  1 10:56 ppcheck.c
-rw-r--r--  1 ward       13112 Nov  1 10:56 ppick.c
-rwxr-xr-x  1 ward       32768 Nov  1 10:56 sigtest*
-rw-r--r--  1 ward         631 Nov  1 10:56 sigtest.c
-rw-rw-r--  1 ward         861 Nov  1 10:56 touch.c
./src/cmd/demux
total 144
-rw-r--r--  1 ward         233 Mar 22 08:08 Makefile
-rwxr-xr-x  1 ward      114688 Mar 22 08:11 demux*
-rw-r--r--  1 ward        6083 Mar 23 11:35 demux.c
-rw-r--r--  1 ward       16886 Mar 22 08:11 demux.o
./src/cmd/SCCS
total 50
-r--r--r--  1 ward        1096 Mar 22 09:44 s.Makefile
-r--r--r--  1 ward        8878 Nov 17 15:54 s.ah2st.c
-r--r--r--  1 ward        5948 Nov 17 15:54 s.asc2st.c
-r--r--r--  1 ward       10624 Nov 17 15:54 s.st2asc.c
-r--r--r--  1 ward        1949 Nov 17 15:54 s.stdescribe.c
-r--r--r--  1 ward       19573 Nov 17 15:54 s.stedit.c
./src/lib
total 1
drwxrwsrwx  3 ward        1024 Nov 18 08:24 suds/
./src/lib/suds
total 167
-r--r--r--  1 ward         826 Nov 18 08:24 Makefile
drwxr-sr-x  2 ward         512 Nov 18 08:23 SCCS/
-rw-r--r--  1 ward        3225 Nov 17 12:29 asc2field.c
-rw-r--r--  1 ward        2180 Nov 17 12:29 descr_trace.c
-rw-r--r--  1 ward         867 Nov 17 12:29 find_code.c
-rw-r--r--  1 ward      121700 Nov 18 08:19 libsuds.a
-rw-r--r--  1 ward        1835 Nov 17 12:30 st_error.c
-rw-r--r--  1 ward        2091 Nov 17 12:30 st_init.c
-rw-r--r--  1 ward        6327 Nov 17 12:31 st_time.c
-rw-r--r--  1 ward       16532 Nov 17 12:31 suds_io.c
./src/lib/suds/SCCS
total 2
-r--r--r--  1 ward        1030 Nov 18 08:23 s.Makefile
./files
total 231
-rw-r--r--  1 ward      154368 Nov  1 10:54 ah.871005040607
-rw-r--r--  1 ward       71232 Nov  1 10:54 st.871005040607
-rw-r--r--  1 ward          76 Nov  1 10:54 st.sample
./SYS
total 10
drwxrwxrwx  3 ward         512 Jan 24 17:04 Templates/
-rw-r--r--  1 ward         249 Jan 24 17:05 hosts
-rw-r--r--  1 ward        2580 Jan 24 17:05 rc.boot
-rw-r--r--  1 ward        4753 Jan 24 17:05 rc.local
./SYS/Templates
total 3
-rw-r--r--  1 ward        1229 Jan 24 17:04 README
drwxrwsrwx  2 ward         512 Jan 24 17:04 root/
./SYS/Templates/root
total 0
```

NAME

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

SYNOPSIS

ah2st [**-c clipvalue**] [**-n netname**] [**-o** *file*] [-s factor] [*files*]

DESCRIPTION

ah2st converts a stream of **ah or ad hoc** structures in the Lamont-Doherty format into **suds** structures. For each waveform the **suds** structures **stationcomp, origin,** and **descripttrace** followed by the waveform are created. If calibration information is included in the **ah** structure, a **suds** structure **calibration** is created. If the **ah** fields **ecomment** and **rcomment** include information, **suds** comment structures are also created.

OPTIONS

-c Make the next argument the plus and minus value where clipping of the waveform begins.

-n Make the next argument the network name.

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

-s Convert the waveform to short integers multiplying each point by *factor.*

SEE ALSO

st_intro(3s), asc2st(1s)

BUGS

The **ah** variable **rmin** is not translated into the **suds** structures.

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(3s), st2asc(1s)

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

cdd – contol CDD analog-to-digital converter

SYNOPSIS

cdd

DESCRIPTION

cdd is an interactive program for controlling the Cutler Digital Design analog-to-digital converter. It provides a menu:

o open(type msec/sample and number of channels)
r rezero atod
g get data
h get headers
s send control information to atod
t set time (Type day hour min sec)
T set time to computer's clock(GMT)
c change time(type milliseconds)
q quit

A typical sequence of commands to run the atod at 100 samples per second for 16 channels might be:

o 10 16
t 145 22 12 0
r
h
g

SEE ALSO

plotatod(1s), atod(3s)

AUTHOR

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

NAME

plotatod – plot output of analog-to-digital converter in a window

SYNOPSIS

plotatod *options stations device*

DESCRIPTION

plotatod collects data from the analog-to-digital converter and plots it in a window on the SUN workstation. The program reads 1000 samples of data and calculates the average noise and DC voltage offset for each channel and scales the plot automatically so that the average noise fits within the part of the window in the vertical direction assigned to the station. The channel number and station name are plotted at the left together with the scale factor over the millivolts of DC offset. The scale factor is the number of millivolts from the bar above the name to the bar below the name.

stations is the name of a file containing STATIONCOMP structures (See st_intro.3s). The fields used that must be set are **sc _name.st_name, con_mvolts, and channel.**

device is the pathname of the atod, normally **/dev/rst1**

By pushing the right mouse button a menu is displayed to allow changing the vertical and horizontal scales by factors of 2, to begin autoscaling again, and to exit.

OPTIONS

–d *n* plot every nth point. Default is 2.

–f *number* of first station to be plotted counting from one.

–g *value* Fix scale factor (gain) for all stations at this value in millivolts.

–l *number* of last station to be plotted counting from one.

–m *value* of milliseconds per sample atod is to be set for.

–n *number* of stations atod is to be set for.

–p print average noise and DC offset of the stations.

SEE ALSO

cdd(1s), atod(3s)

BUGS

AUTHOR

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

NAME

scanseis – scan digital seismic waveform files on a SUN3 workstation

SYNOPSIS

scanseis [**-t** *tapename*] [*filenames*]

DESCRIPTION

scanseis looks in the current directory, or on a **tar(1)** tape mounted on /dev/rst0 if the **-t** option is specified, for files beginning with **eq.** or **ah.** or ending with **D** that are assumed to contain seismic waveform data in the USGS online format, Lamont adhoc format, or University of Washington format respectively. The seismic data in each file are displayed and the user may choose descriptors for the file using the right mouse button and may then decide to save, delete, or compact the file. If compaction is chosen, the user may specify the beginning and end of the record to be saved and which stations are to be deleted. **compactseis** is then executed in a spawned, asynchronous process. A log of all transactions is appended to the file **scanseis.log** in the current directory.

Many aspects of the plotting and the contents of the descriptive menu can be specified in a file **.scanseis** in the current directory or if not there then in the user's home directory. The default for each item is used unless a different value is specified in a **.scanseis** file. The format for each line of **.scanseis** is label, number, and comment if desired. A label **menu** may be followed by a list of up to 50 menu items separated by commas. These may be clustered into as many as 5 groups separated by a slash character, /. Multiple lines may be used separated by . Only one item in each group may be selected from the menu by the user. This item is displayed in a different manner in the menu when selected. All selected items are appended to the line in **scanseis.log** describing the processing of the event. A complete sample of **.scanseis** including all default options is given below.

```
maxfiles      4     /* Maximum number of files to process at once */
decimation    4     /* Decimation factor */
labelx        1000  /* Pixels in X to first station name */
eachlabel     1000  /* Pixels in X between station names */
labely        6     /* Pixels in Y from center of strip to base of name */
overlap       100   /* percent overlap of traces in Y direction */
height        47    /* Pixels per trace window in Y direction */
second        50    /* Pixels per second in X direction */
tickmark      25    /* Length of time tics in pixels on Y */
xdelete       920   /* Pixels in X to first delete message for compacting */
menu EVENT TYPE,  Local event,  Regional event,  Teleseism,\
  Seismic noise,  Telemetry noise,  Calibration/SIGNAL LEVEL,\
  Weak signal,  Clear signal,  Clipped signal
```

EXAMPLES

FILES

SEE ALSO

DIAGNOSTICS

BUGS

AUTHOR

Peter Ward, US Geological Survey, Menlo Park, Ca.

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, STATIONCOMP and DESCRIPTRACE. The ORIGIN structure, if it exists, must preceed the STATIONCOMP and DESCRIPTRACE structures and the STATIONCOMP structure must preceed the DESCRIPTRACE structure. Data types allowed for waveforms are short, long, and float.

SEE ALSO

st_intro(3s), ah2st(1s)

BUGS

This is a quick hack for moving local network data into sunpick and needs to be made more general.

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] [-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

–c The following file contains a control file similar to **/usr/include/suds/suds_descr.h.** The output format and order can be changed from this control file as well as the content of label strings used in the verbose option. NOT IMPLEMENTED YET.

–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.

–n Do not list waveform data, simply list headers.

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

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

–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_codes.h** are followed by the appropriate explanatory string within brackets.

SEE ALSO

st_intro(3s), asc2st(1s)

DIAGNOSTICS

BUGS

EXAMPLE

Separate the fields by commas:

st2asc -s "," myfile

AUTHOR

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

NAME

stdescribe – describe the suds structures in a file

SYNOPSIS

stdescribe [**-o** *file*] [*file...*]

DESCRIPTION

stdescribe lists the file name followed by a list of structures within the file. The number of the structure from the beginning of the file is given followed by the structure number, name, the length of the structure, and the length of any ensuing data. If the first field in the structure is STATIDENT, this substructure is listed.

OPTIONS

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

SEE ALSO

st_intro(3s), st2asc(1s)

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. The next structure is displayed by pushing the key **F7**. The editor may be exited without change by pressing the key **F2**. You change fields by pressing **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. Note that you may only type within certain parts of the screen.

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 **stform** in **suds_descr.h** to get field labels, lengths, types, etc. The codes are listed in **suds_codes.h**

EDITORIAL

This editor is a quick hack of some other code I had in order to demonstrate 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 is not device independent and thus may not work on some terminals. UNIX System V **curses** might improve this. *stedit* may not start properly in a SUN-CMD window. Use Shelltool.

This editor 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.

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

st_intro, suds – standard buffered input/output package for suds structures

SYNOPSIS

#include <suds/suds.h>

DESCRIPTION

suds or the Seismic Unified Data System consists of an extensible set of structures that associate related variables into logical groups. The initial structures relate to information about earthquakes, seismic waveforms, crustal structures, etc., but the methods and programs used apply equally well to any grouping of related variables pertaining to any subject. Data typically have natural logical groupings. The basic concept of **suds** is to simplify programming, data exchange, and program exchange by taking advantage of these logical groupings. The programmer can then simply say: "I want to deal with information pertaining to hypocenters, stations, or waveforms", for example, and not have to worry about the messy details of input, output, and data storage. Many general purpose utilities can be written to display, edit, store, filter, and massage these structures. The structures are designed to be extensible in order to allow them to grow and change with the whims of a wide variety of independent researchers. They also allow for a variety of styles of implementation so that individual institutions can decide on the best mode of use to meet their needs.

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 strucures are:

st_close: close an st_stream.

st_flush: flush an st_stream.

st_free: free memory associated with a structure pointer allocated by st_get.

st_get: get the next structure in an st_stream.

st_open: open an st_stream.

st_peek: return the identification of the next structure in an st_stream.

st_put: put a structure into an st_stream.

st_rewind: rewind an st_stream.

st_seek: find the nth structure in an st_stream.

st_tell: return the number of the current structure's position in the st_stream.

st_unget: back up only 1 structure in an st_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. **st_error** provides an easy to use, general error handling capability.

Programs using the st_routines should be compiled with the **-lsuds** flag.

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.

Many utility commands or filters are available to operate on **suds** streams. 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.

st2ah: Convert suds streams to Lamont ah streams.

ping2st: Convert ping (Carl Johnson andUniversity 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.

st2xdr: Convert suds streams to eXternal Data Representation streams (A machine independent binary standard used in the Network File System).

xdr2st: Convert eXternal Data Representation streams to suds streams.

st2db: Load suds streams into the db_vista database.

db2st: Extract structures from the db_vista 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.

stsubset Extract a subset of structures from a suds stream.

Subroutines for use with **suds** structures include

st_create create a new suds structure and initialize it.

st_init initialize a suds structure.

make_mstime convert years through seconds to a **suds** time variable.

decode_mstime decode a **suds** time variable into years through seconds.

list_mstime convert a **suds** time variable to an ascii string.

get_mstime get the present time-of-day as a **suds** time variable.

find_code find number representing an ascii string.

list_code find ascii string explaining a **suds** code.

descr_trace calulate many fields in a DESCRIPTRACE structure from a waveform.

asc2field convert an ascii string to a suds structure field.

field2asc convert a suds structure field to an ascii string.

suds is designed to take advantage of the flexibility of the UNIX shell command language and the speed of C language I/O. Commands and subroutines that extend these advantages to FORTRAN programmers and other operating systems include:

The seismic structures and related constants are defined in **suds.h** The core structures include:

TAG: A label to identify structures and provide a basis to check for errors.

TERMINATOR: A marker to identify the logical end of a group of structures.

EQUIPMENT: Equipment making up a station/component.

STATIONCOMP: Station component information available before a waveform is recorded.

MUXDATA: Header for multiplexed data.

DESCRIPTRACE: Descriptive information about a seismic trace.

LOCTRACE: Location of a seismic trace within the filesystem.

CALIBRATION: Calibration information for a station component

FEATURE: Observed phase arrival time, amplitude, and period.

RESIDUAL: Calculated residuals for arrival times, magnitudes, etc.

EVENTREADINGS: Relation of features to a specific event

EVENT: General information about an event.

EV_DESCRIPT: Descriptive information about an event.

ORIGIN: Information about a specific solution for a given event.

ERROR: Errors related to a specific ORIGIN.

FOCALMECH: Focal mechanism solution.

MOMENT: Information on the seismic moment.

VELMODEL: Crustal velocity model.

LAYERS: Velocity characteristics of individual layers.

COMMENT: Free form comment that can be related to any field of any structure.

SHOTGATHER: Grouping of waveforms related to a given source.

PROFILE: Grouping of SHOTGATHERS into lines.

Extension of **suds** structures can be done by adding fields to the end of existing structures or adding new structures. Some care and close coordination with other institutions need to be exercised so that the wide portability can be maintained and so that old and new programs using these structures can co-exist harmoniously. Programs that use newly extended structures must check the length of structures on input and handle the older versions appropriately or give error messages. We strongly urge that you propose changes to these structures only after you are completely convinced that the existing structures can not meet your needs. In this way we can all work together to keep future confusion and complexity to a minimum.

SEE ALSO

ah2st(1s), asc2field(3s), asc2st(1s), descr_trace(3s), find_code(3s), st2asc(1s), st_describe(1s), st_edit(1s), st_close(3s), st_error(3s), st_flush(3s), st_get(3s), st_init(3s), st_open(3s), st_put(3s), st_rewind(3s), st_seek(3s), st_tell(3s), st_time(3s), st_unget(3s)

FILES

/usr/lib/libsuds.a

/usr/include/suds/suds.h

/usr/include/suds/suds_descr.h

/usr/include/suds/suds_codes.h

/usr/include/suds/st_error.h

AUTHOR

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

NAME

asc2field, field2asc – convert ascii string to a suds field, convert a suds field to an ascii string

SYNOPSIS

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

asc2field(string,ptr,type)
char *ptr,*string;
int type;

char *field2asc(ptr,type,verbose)
char *ptr;
int type,verbose;
```

DESCRIPTION

asc2field converts the ascii *string* to a variable of a specific *type* pointed to by *ptr* which is typically an address of a field in a **suds** structure. *field2asc* converts the field pointed to by *ptr* to an ascii string. *type* is defined by the "Integer defines for standard variable types" in **<suds/suds.h>.**

SEE ALSO

DIAGNOSTICS

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

EXAMPLES

AUTHOR

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

NAME

atod_open, atod_settime, atod_incrtime, atod_geterr, atod_reset – contol analog-to-digital converter

SYNOPSIS

int atod_open(device,ms_sample,numchan)
char *device;
int ms_sample, numchan;

int atod_settime(atod_fd,day,hour,min,seconds)
int atod_fd,day,hour,min,seconds;

int atod_incrtime(atod_fd,milliseconds)
int atod_fd,milliseconds;

int atod_geterr(atod_fd)
int atod_fd;

int atod_reset(atod_fd)
int atod_fd;

DESCRIPTION

These routines provide the software interface to the Cutler Digital Design (CDD) Data Sampler. *atod_open* opens and initializes the atod with the pathname given by *device*, typically **/dev/rst1**, and returns an integer file descriptor, *atod_fd*, used in the other subroutines. *atod_open* checks the byte order on the host machine and adjusts the CDD atod accordingly. The year is initialized in the atod with the last digit of the current year in the host machine's clock. The *ms_sample*, milliseconds per sample, may be any value such that the total throughput of the atod is 25,600 samples per second or less. Thus for 256 channels *ms_sample* may be **10** or larger, for 128 channels *ms_sample* may be **5** or larger, and for 16 channels (the minimum number of channels) *ms_sample* may be **1** or larger. *numchan* is the number of individual channels of data to be collected and may be any number from 1 to the maximum number of input channels installed in the Data Sampler. Since the Data Sampler digitizes in units of 16 channels, this number is rounded up to the next multiple of 16 by the software.

atod_settime sets the internal clock in the Data Sampler. If day is a negative number, the internal clock is disabled. *atod_settime* returns **1** if the time specified is impossible, such as a day of 34 or hour of 25, returns **4** if the atod buffer is overflowed, although the time will still be reset, and returns **0** if there are no errors.

atod_incrtime changes the internal clock by *milliseconds*, which may be positive or negative. *atod_incrtime* returns **4** or **0** as discussed above.

atod_reset clears the buffer in the Data Sampler and resets the digitizing process but leaves sample-rate, time, and other constants unchanged.

atod_geterr returns the error code from the Data Sampler. When an I/O error occurs the Data Sampler sends a message to the host machine that an error occurred. The host machine then asks the sampler for the error number and stores the value of the error in the driver software. *atod_geterr* returns this error value and resets the value to **0** so that a second call will return **0**. Other return values are **4** meaning the atod's buffer overflowed and **5** meaning an illegal command was sent to the atod.

DIAGNOSTICS

These routines use **st_error(3s)** to report errors and call **die**, which you must provide, after fatal errors.

SEE ALSO

cdd(1s), plotatod(1s), st_error(3s)

BUGS

The maximum throughput rate of 25,600 samples per second may not be attainable on a continuous basis on a SUN 3/50, depending on the use of the SCSI bus by other peripherals such as the disk and tape drive. Digitization of 32 stations at 100 samples per second uses less than 10% of the system resources.

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 in a suds waveform

SYNOPSIS

#include <suds/suds.h>

int descr_trace(sc,dt,waveform)
struct stationcomp *sc;
struct descriptrace *dt;
char *waveform;

DESCRIPTION

descr_trace calculates and sets the values in **struct descriptrace** for **mindata** (minimun value of waveform), **maxdata** (maximum value of the waveform), **avenoise** (average value of the first 200 samples), and **numclip** (number of clipped samples). If the waveform type is not short, long, or float, *descr_trace* has a return value of 1 and no change was made. Otherwise the return value is 0. If the pointer **waveform** is 0, the waveform is assumed to be contiguous to the end of **struct descriptrace.** 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

NAME

find_code, list_code – find or list codes used in suds structures

SYNOPSIS

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

int find_code(string,list)
char *string;
struct codes *list;

char *list_code(code,list)
int code;
struct codes *list;
```

DESCRIPTION

Many fields in suds structures contain a **char, short or int** that represent a specific line of a list. *find_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, a 0 is returned. *list_code* returns a pointer to the string corresponding to the specified *code*. If the code is not found a string **this code undefined** is returned. *list* is a codelist specified in **suds_codes.h.**

AUTHOR

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

NAME

st_close, st_flush – close or flush a stream of suds structures

SYNOPSIS

#include <stdio.h>

st_close(stream)
FILE *stream;

st_flush(stream)
FILE *stream;

DESCRIPTION

st_close causes any buffered data for the named *stream* to be written out, and the named *stream* to be closed. Buffers allocated by the standard input/output system are freed.

st_close is performed automatically for all open files upon calling *exit*(3).

st_flush causes any buffered data for the named output *stream* to be written out. The named *stream* remains open.

SEE ALSO

fclose(3s), st_open(3S)

DIAGNOSTICS

Errors are reported by *st_error* and then a user supplied subroutine called *die(errno)* is called (See *st_error(3s)*).

BUGS

AUTHOR

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

NAME

st_error – general purpose error reporting and handling

SYNOPSIS

#include <suds/st_error.h>

st_error(function,errnum,format [,arg]...)

int (*function)();
int errnum;
char *format;

extern char *progname;

extern char *st_errout;

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 nextline is the message given in the *printf* type *format* and by *args*. The third line is the system error associated with *errnum* if *errnum!=0*. The following line is the system error associated with *errno* (INTRO(2)) if *errnum!=errno*. *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, 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:

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

If the user has set special tty modes, these should be restored in *die*.

errnum may be equal to *errno(INTRO(2))* or one of the following manifest constants defined in *st_error.h:*

0 ENO_ERR Ignor errnum

1001 EBADDAY Illegal day

1002 EBADMONTH Illegal month

1003 EBADYEAR Illegal year

1004 EBADVALUE Illegal value

1005 EFEWDATA Not enough data

1006 ENOCONVRG Convergence failure

1007 EUNDERFLOW Underflow

1008 ENOSTATION No such station

1009 ESIZE Structure sizes mismatch

1010 ESINGULAR Singular matrix

1011 ETOODEEP Focal depth too great

1012 EBADPHASE No such phase

1013 ENOFEAS Constraints are inconsistent

1014 EINFIN Solution is unbounded

1015 EBADSYNC Structure tag sync code wrong

1016 ST_ERR Serious suds I/O error

SEE ALSO

intro(2), st_intro(3s)

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.

NAME

st_get – get the next suds structure from stream

SYNOPSIS

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

int st_get(st_ptr,type,length,stream)
char **st_ptr;
int *type,*length;
FILE *stream; st_free(ptr,length)
char *ptr;
int length;
```

DESCRIPTION

st_get returns a pointer *st_ptr* to the next suds structure in the stream, the *type* of the structure, and the *length* in bytes of the structure. The return value of *st_get* is the total length of the structure plus the length of any data following the structure such as for structure type DESCRIPTRACE and MUXDATA.

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

SEE ALSO

st_open(3S), st_put(3S), st_error(3S), st_unget(3S)

DIAGNOSTICS

The most common error is likely to be a **Segmentation Violation** caused by not passing the addresses of **type** and **length.** 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 *type* and *length* are all set equal to zero.

BUGS

EXAMPLE

```
#include <stdio.h>

char *ptr;

int type,length,in_num;

in_num=st_get(&ptr,&type,&length,stdin);
```

AUTHOR

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

NAME

st_init – initialize a suds structure

SYNOPSIS

#include <suds/suds.h>

st_init(type,pointer)
int type;
char *pointer;

st_create(type,pointer,datalength)
int type,datalength;
char **pointer;

DESCRIPTION

st_init initializes all fields of a **suds** structure to the default values defined in the file **suds_descr.h** This initialization is important since fields with no data should be initialized to one of the constants **NODATA, NOTIME, NOCHAR, NOSTRG, or NOLIST** defined in the file **suds.h.** *st_create* uses *malloc* to create space for a new structure and then initializes the structure. *datalength* is the length of data in bytes that follow the structure when appropriate. **suds.h**

AUTHOR

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

NAME

st_open – open a stream containing suds structures

SYNOPSIS

#include <stdio.h>

FILE *st_open(filename, type)
char *filename, *type;

DESCRIPTION

st_open opens the file named by *filename* and associates a stream with it. If the open succeeds, *st_open* returns a pointer to be used to identify the stream in subsequent operations.

filename points to a character string that contains the name of the file to be opened.

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

"r"	open for reading
"w"	truncate or create for writing
"a"	append: open for writing at end of file, or create for writing
"r+"	open for update (reading and writing)
"w+"	truncate or create for update
"a+"	append; open or create for update at end-of-file

When a file is opened for update, both input and output may be done on the resulting stream. However, output may not be directly followed by input without an intervening *fseek* or *rewind*, and input may not be directly followed by output without an intervening *st_seek*, *st_rewind*, or an input operation which encounters end-of-file.

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.

SEE ALSO

fopen(3s), st_close(3S), st_seek(3S),

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

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

NAME

st_putc – put a suds structure on a stream

SYNOPSIS

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

int st_put(st_ptr,type,length,stream)
char *st_ptr;
int type;
int len;
FILE *stream;
```

DESCRIPTION

st_put writes the structure pointed to by *st_ptr* onto the named output *stream*

SEE ALSO

st_open(3S), st_close(3S), st_get(3S)

DIAGNOSTICS

Errors are reported by *st_error* and a user supplied subroutine called *die(errno)* is called (See *st_error(3s)*).

EXAMPLE

```
#include <stdio.h>

char *ptr;

int type,length,out_num;

out_num=st_put(ptr,type,length,stdout);
```

AUTHOR

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

NAME

st_seek, st_tell, st_rewind, st_peek – reposition a stream of suds structures

SYNOPSIS

```
#include <stdio.h>

st_seek(stream, offset,direction)
FILE *stream;
int offset,direction;

int st_tell(stream)
FILE *stream;

st_rewind(stream)
FILE *stream;
```

DESCRIPTION

st_seek sets the position of the next input or output operation on the *stream*. The new position is just before the nth structure specified by *offset* counting the first structure in the stream as zero. The offset is relative to the beginning of the file if direction==0, relative to the present position in the file if direction==1, and relative to the end of the file if direction==2.

st_rewind(*stream*) is equivalent to *st_seek*(*stream*,0,*0*).

st_seek and *st_rewind* undo any effects of *st_unget*(3S).

After *st_seek* or *st_rewind*, the next operation on a file opened for update may be either input or output.

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

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

SEE ALSO

fseek(3S), st_open(3S), st_unget(3S)

DIAGNOSTICS

st_seek returns –1 for improper seeks, otherwise zero. An improper seek can be, for example, an *st_seek* done on a file that has not been opened via *st_open*.

DIAGNOSTICS

Errors are reported by *st_error* and a user supplied subroutine called *die(errno)* is called (See *st_error(3s)*).

BUGS

AUTHOR

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

NAME

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

SYNOPSIS

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

ms_time make_mstime(year,month,day,hour,min,second)
int year,month,day,hour,min;
double second;

int decode_mstime(time,year,month,day,hour,min,second)
ms_time time;
int *year,*month,*day,*hour,*min;
double *second;

char *list_mstime(time,form)
ms_time time;
int form;

ms_time get_mstime();
```

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) as defined by *typedef* in **suds.h.** These routines provide simple conversion to and from other forms. **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 of seconds with resolution to microseconds or better.

make_mstime returns an ms_time variable. *decode_mstime* returns through pointers the components of the time variable. *list_mstime* returns a pointer to a null terminated character string in one of several formats specified by *form. get_mstime* returns the computer's clock as an *ms_time* variable.

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	yrmodyhrmnsc.000
1	yrmodyhrmnsc
2	yr mo dy hr mn sc.000
3	yr mo dy hr mn sc
4	mo/dy/yr hr:mn sc.000 GMT
5	mo/dy/yr hr:mn sc GMT
6	Aug 10, 1988 hr:mn sc.000 GMT
7	Aug 10, 1988 hr:mn sc GMT

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

To use *st_time* simply cast the values appropriately:

```
st_time stamp;
```

```
stamp=(long)get_mstime();
printf("%s\n",list_mstime((double)stamp,1);
```

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

SYNOPSIS

#include <stdio.h>

st_unget(st_ptr, stream)
char *st_ptr;
FILE *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*(3S) erases all memory of pushed back characters.

SEE ALSO

st_get(3S), st_seek(3S)

DIAGNOSTICS

Errors are reported by *st_error*. *st_unget* returns EOF if it can't push a character back and *ST_ERR* (defined in st_error.h) if *stream* is an illegal file descriptor.

BUGS

AUTHOR

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

Connection of the Analog-to-Digital Converter to the Computer and to the Software

There are four simple steps to connect the A/D converter to the computer:

1. Reconfigure the operating system so that it knows that the device is attached.
2. Create a symbolic name that can be referenced by users of the computer.
3. Physically attach the A/D converter to the SCSI bus.
4. Use the system software and special routines provided with this tape to control the sampler and read data from it.

You must be sure your computer is running SUN Operating System version 3.5. The interface does not work for Version 3.4 because major, bug ridden changes to the SCSI interface software were made for that release. The interface may work for Versions 3.2 and 4.0, but has not been tested for these versions.

The computer reads data from the A/D converter sequentially in a manner similar to reading data from a magnetic tape recorder. Thus the command-set used for the SCSI bus interface is that specified in ANSI Standard X3.131-1986 for Sequential-Access Devices. For this reason the A/D converter uses the operating system driver for SCSI based magnetic tape drives. In UNIX this driver is located in /usr/sys/OBJ/st.o. This routine also calls the low level SCSI driver package /usr/sys/OBJ/si.o. To install these routines in the kernal of your version of UNIX and point to the new device to be used, you need to add one line to the configuration file for your system located in /usr/sys/conf and rebuild the operating system. On Sun computers this process is explained in detail in **Chapter 8 Configuring the System Kernal** in the manual section **Installing UNIX on the Sun Workstation**. The steps are as follows.

Login as **root**. Edit the file in /usr/sys/conf that has been customized for your computer. If the system has not been customized and you are using a SUN 3/50 computer with one disk and one archive tape, you can use the file SEISMIC included on this software tape. If you do not use this file then add a line in your config file at least after the line

controller si0 at obio ? csr 0x140000 priority 2

and preferable immediately after the line

tape st0 at si0 drive 32 flags 1

if it exists. The new line should read

tape st1 at si0 drive 40 flags 1

If a line using drive 40 on this SCSI controller already exists it must be deleted if no device is actually attached or you must change the bus address of the A/D converter. There may be eight controllers attached to each SCSI bus, numbered from 0 to 7. The highest priority is assigned to controller 0 which is usually the system disk. A single tape unit is usually put at level 4 and the computer's SCSI controller is at level 7. Pick a level appropriate for your hardware configuration. Set the DIP switch inside the A/D converter to that number and change the 40 in the above line to 8 times the number set in the DIP switch. Now write out the new configuration file and type:

/etc/config SEISMIC

but substitute the name of your configuration file for the word SEISMIC. This name will appear on the banner line during login telling which version of the operating system you are running. /etc/config creates a new directory in /usr/sys with this config file name. Change to this directory. For example type:

cd /usr/sys/SEISMIC

and then type

make

Make compiles and links a new kernal called **vmunix**. You must move this to the / directory after you save the old version. Type:

mv /vmunix /vmunix.old
mv vmunix /vmunix

You may only be able to keep two system kernals in / at one time because of the size of the disk partion used. Be sure to keep the old version there until you know the new version works. If the new version does not work you can then simply respond to the boot command with

bsd(0,0,0)vmunix.old

WARNING: If you did not save the old version in / and the new version does not work, you may need to reload all software and software distributions!

Now you must create two entries in the /dev directory that can be used by the software to access the A/D converter. Type:

cd /dev
mknod nrst1 c 18 5
mknod rst1 c 18 1
chmod 0666 nrst1 rst1

Of course you may use **st2** or **st3** if two other tape drives already exist on the SCSI bus. In these cases add 1 or 2 to the last number, the minor device number. The minor device number should be the number following st in the name for **rst** and that number plus 4 for **nrst1**. Only four tape devices may be used with the st.o kernal software. The major device number 18 should be the same used for **rst0.**

ls -l rst0

Type:

to check that the number is the same. You may use a different name for these files if you wish such as **rsa0** and **nrsa0** meaning SCSI analog-to-digital converter. We recommend keeping the st designation, however, because that is the driver software being used. The programs provided with the A/D converter assume the name **/dev/rst1** and **/dev/nrst1. The difference between these two names is that the** rst1 device resets the A/D converter when it is opened or closed and the rst1 device does not do an automatic reset. If you use different names, you will need to change the names used in the software provided.

Now shutdown UNIX by typing:

/etc/halt

When the system halts turn off the power to the computer and all peripheral devices. Connect the SCSI bus cables between your computer and the A/D converter. The SCSI bus is a daisy chain with terminators at each end. The total length of the bus should not be longer than 6 meters (19.7 feet). Terminators are included typically within the computer and the system disk. The terminators have been left out of the A/D converter so that it can be located between the computer and the system disk or between and other two devices on the bus. Connect the SCSI bus cable from the computer to the SCSI IN connector of the A/D converter. Connect the new SCSI bus cable provided to the A/D converter SCSI OUT connector and to the connector formerly attached to the computer. Turn on the power to the peripherals and then to the computer. The computer should reboot automatically. During the boot process a line should appear showing the tape port for the address you specified, such as

st1 at si0 slave 40

The A/D converter should now be ready for use.

```
/*   structure for data file header for Lamont ah or ad hoc system
     witten, 11 June 85
     headers and data for events are stored in same file typically
     named ah.YRMODYHRMN
*/
static char Ahheader_h[] = "%W% %G%";   /* Sccs Id */
#define AHHEADSIZE 1024
#define TYPEMIN   1
#define TYPEMAX   6
#define LOGSIZE 202
#define LOGENT   10
#define NEXTRAS  21
#define NOCALPTS 30

#define ahFLOAT    1
#define ahCOMPLEX  2
#define ahVECTOR   3
#define ahTENSOR   4
#define ahDOUBLE   6

typedef   struct   {
   float   x;
   float   y;
   } vector;

typedef   struct   {
   float   r;
   float   i;
   } complex;

typedef struct  {
   double r;
   double i;
   } d_complex;

typedef   struct   {
   float   xx;
   float   yy;
   float   xy;
   } tensor;

struct   time   {
   short     yr;          /* year    */
   short     mo;          /* month   */
   short     day;         /* day     */
   short     hr;          /* hour    */
   short     mn;          /* minute  */
   float     sec;         /* second  */
   };


struct   calib   {
   complex   pole;        /* pole    */
   complex   zero;        /* zero    */
   };

struct   station_info   {
   char      code[6];     /* station code                                  */
   char      chan[6];     /* lpz,spn, etc. last char either z, n, or e     */
   char      stype[8];    /* wwssn,hglp,rstn,sro,asro,dwwssn etc.          */
   float     slat;        /* station latitude                              */
   float     slon;        /*    "    longitude                             */
   float     elev;        /*    "    elevation                             */
   float     DS;          /* maximum gain at peak of calibration curve     */
   float     A0;          /* normalization, factor to make calibration curve at
                             peak equal 1 for a specific frequency         */
   struct    calib  cal[NOCALPTS]; /* calibration curve                    */
   };

struct   event_info   {
   float    lat;          /* event latitude    */
   float    lon;          /*   "   longitude   */
   float    dep;          /*   "   depth       */
   struct   time   ot;    /*   "   origin time */
   char     ecomment[80]; /* comment line      */
   };

struct   record_info   {
   short    type;          /* data type float=1, complex=2, etc as defined above */
   long     ndata;         /* number of samples                                  */
   float    delta;         /* sampling interval                                  */
   float    maxamp;        /* maximum amplitude of record                        */
   struct   time  abstime; /* start time of record section                       */
   float    rmin;          /* minimum value of abscissa                          */
   char     rcomment[80];  /* comment line                                       */
   char     log[LOGSIZE];  /* log of data manipulations                          */
   };

typedef struct {
   struct   station_info   station;  /* station info */
   struct   event_info   event;      /* event info   */
   struct   record_info   record;    /* record info  */
   float    extra[NEXTRAS];          /* freebies     */
   } ahhed;
```

```
/* SUDS_DESCRIPT.H: data structures used by utility programs to access and
   describe any of the suds structures
 */
#ifndef SUDS_DESCRIP_H
#define SUDS_DESCRIP_H

static char Suds_descrip_h[] = "@(#)suds_descr.h        1.2 11/17/88";  /* Sccs Id */

#include <stdio.h>
#include "/usr/suds/include/suds.h"

/* special formats */
#define STIFM   ""
#define STTFM   "%ld"
#define MSTFM   "%lf"
#define LTTFM   "%lf"
#define CALFM   ""
#define COMFM   ""

#define DECIMAL "0-9. NODATA"
#define INTEGER "0-9 NODATA"
#define NAMES   "a-z A-Z"
#define ASCII   " --"
#define TYPICAL "0-9 a-zA-Z.,"
#define TIMES   "0-9.NOTIME"
#define S(x) (sizeof(x)/sizeof(struct codes))

extern CODES inst_type[],mach_type[],equip_model[],equip_reason[],ann_com[],rock_type[],stat_proc[],
   authority[],feat_phase[];

/* SUDSFORM: Information on each suds structure used for input from ascii,
   output to ascii, loading or unloading the database, etc.
 */
SUDS_FORM stform[] = {
   NO_STRUCT,   "junk",                 STR,   1,     0,      NULL,   NOSTRG, "%s",   0,      0,0,    3,25,DECIMAL,
   STAT_IDENT,  "network",              STR,   4,     0,      NULL,   "unk",  "%s",   0,      0,0,    0,10,NAMES,
   STAT_IDENT,  "station name",         STR,   5,     4,      NULL,   NOSTRG, "%s",   0,      0,0,    0,28,NAMES,
   STAT_IDENT,  "component",            CHR,   1,     9,      NULL,   NOCHAR, "%c",   0,      0,0,    0,44,"vneVNE",
   STAT_IDENT,  "instrument type",      SHT,   1,     10, inst_type,  NOLIST, "%d",   0,      0,0,    2,25,INTEGER,
   STRUCTTAG,   "sync char",            CHR,   1,     0,      NULL,   "S",    "%c",   0,      0,0,    3,25,DECIMAL,
   STRUCTTAG,   "machine type",         CHR,   1,     1, mach_type,   "MACHINE","%c", 0,      0,0,    5,25,DECIMAL,
   STRUCTTAG,   "struct number",        SHT,   1,     2,      NULL,   NODATA, "%d",   0,      0,0,    7,25,INTEGER,
   STRUCTTAG,   "struct length",        LNG,   1,     4,      NULL,   NODATA, "%ld",  0,      0,0,    9,25,INTEGER,
   TERMINATOR,  "beginning struct",     SHT,   1,     0,      NULL,   NODATA, "%d",   0,      0,0,    3,25,INTEGER,
/*10*/TERMINATOR,"unused",              SHT,   1,     2,      NULL,   NODATA, "%d",   0,      0,0,    5,25,INTEGER,
   EQUIPMENT,   "this station/comp",    STI,   1,     0,      NULL,   NOSTRG, STIFM,STAT_IDENT,0,0,   3,10,DECIMAL,
   EQUIPMENT,   "last station/comp",    STI,   1,     8,      NULL,   NOSTRG, STIFM,STAT_IDENT,0,0,   7,10,DECIMAL,
   EQUIPMENT,   "next station/comp",    STI,   1,     16,     NULL,   NOSTRG, STIFM,STAT_IDENT,0,0,   11,10,DECIMAL,
   EQUIPMENT,   "serial number",        STR,   8,     24,     NULL,   NOSTRG, "%s",   0,      0,0,    15,25,DECIMAL,
   EQUIPMENT,   "model",                SHT,   1,     32,equip_model, NOLIST, "%d",   0,      0,0,    17,25,INTEGER,
   EQUIPMENT,   "knob 1 setting",       SHT,   1,     34,     NULL,   NODATA, "%d",   0,      0,0,    19,25,INTEGER,
   EQUIPMENT,   "knob 2 setting",       SHT,   1,     36,     NULL,   NODATA, "%d",   0,      0,0,    21,25,INTEGER,
   EQUIPMENT,   "repair reason",        SHT,   1,     38,equip_reason,NOLIST, "%d",   0,      0,0,    23,25,INTEGER,
   EQUIPMENT,   "frequency",            FLT,   1,     40,     NULL,   NODATA, "%f",   0,      0,0,    25,25,DECIMAL,
/*20*/EQUIPMENT,"date effective",       STT,   1,     44,     NULL,   NOTIME, STTFM,  0,      0,0,    27,25,TIMES,
   STATIONCOMP, "station/component",    STI,   1,     0,      NULL,   NOSTRG, STIFM,STAT_IDENT,0,0,   3,10,DECIMAL,
   STATIONCOMP, "component azimuth",    SHT,   1,     12,     NULL,   NODATA, "%d",   0,      0,0,    7,25,INTEGER,
   STATIONCOMP, "component incidence",  SHT,   1,     14,     NULL,   NODATA, "%d",   0,      0,0,    7,60,INTEGER,
   STATIONCOMP, "latitude",             LLT,   1,     16,     NULL,   NODATA, LTTFM,  0,      0,0,    9,25,DECIMAL,
   STATIONCOMP, "longitude",            LLT,   1,     24,     NULL,   NODATA, LTTFM,  0,      0,0,    9,55,DECIMAL,
   STATIONCOMP, "elevation, meters",    FLT,   1,     32,     NULL,   NODATA, "%f",   0,      0,0,    11,25,DECIMAL,
   STATIONCOMP, "enclosure",            CHR,   1,     36,     NULL,   NOCHAR, "%c",   0,      0,0,    13,25,DECIMAL,
   STATIONCOMP, "annotated comment",    MIN,   1,     37,  ann_com,   NOLIST, "%d",   0,      0,0,    15,25,DECIMAL,
   STATIONCOMP, "recorder type",        CHR,   1,     38,     NULL,   NOCHAR, "%c",   0,      0,0,    17,25,DECIMAL,
/*30*/STATIONCOMP,"rock class",         CHR,   1,     39,     NULL,   NOCHAR, "%c",   0,      0,0,    19,25,DECIMAL,
```

```
   STATIONCOMP, "rock type",              SHT,    1,      40,rock_type,   NOLIST, "%d",   0,      0,0,    21,25,INTEGER,
   STATIONCOMP, "site condition",         CHR,    1,      42,     NULL,   NOCHAR, "%c",   0,      0,0,    23,25,DECIMAL,
   STATIONCOMP, "sensor type",            CHR,    1,      43,     NULL,   NOCHAR, "%c",   0,      0,0,    25,25,DECIMAL,
   STATIONCOMP, "data type",              CHR,    1,      44,     NULL,   NOCHAR, "%c",   0,      0,0,    27,25,DECIMAL,
   STATIONCOMP, "data units",             CHR,    1,      45,     NULL,   NOCHAR, "%c",   0,      0,0,    29,25,DECIMAL,
   STATIONCOMP, "polarity",               CHR,    1,      46,     NULL,   NOCHAR, "%c",   0,      0,0,    31,25,DECIMAL,
   STATIONCOMP, "status",                 MIN,    1,      47,stat_proc,   NOLIST, "%d",   0,      0,0,    33,25,DECIMAL,
   STATIONCOMP, "maximum gain",           FLT,    1,      48,     NULL,   NODATA, "%f",   0,      0,0,    35,25,DECIMAL,
   STATIONCOMP, "clipping value",         FLT,    1,      52,     NULL,   NODATA, "%f",   0,      0,0,    37,25,DECIMAL,
/*40*/STATIONCOMP,"conversion to mvolts",FLT,    1,      56,     NULL,   NODATA, "%f",   0,      0,0,    39,25,DECIMAL,
   STATIONCOMP, "channel",                SHT,    1,      60,     NULL,   NODATA, "%d",   0,      0,0,    41,25,INTEGER,
   STATIONCOMP, "spare",                  SHT,    1,      62,     NULL,   NODATA, "%d",   0,      0,0,    43,25,INTEGER,
   STATIONCOMP, "effective date",         STT,    1,      64,     NULL,   NOTIME, STTFM,  0,      0,0,    45,25,TIMES,
   MUXDATA,     "network",                STR,    4,      0,      NULL,   "unk",  "%s",   0,      0,0,    3,25,DECIMAL,
   MUXDATA,     "beginning time",         MST,    1,      4,      NULL,   NOTIME, MSTFM,  0,      0,0,    5,25,TIMES,
   MUXDATA,     "minutes west of GMT",    SHT,    1,      12,     NULL,   NODATA, "%d",   0,      0,0,    7,25,INTEGER,
   MUXDATA,     "number of channels",     SHT,    1,      14,     NULL,   NODATA, "%d",   0,      0,0,    9,25,INTEGER,
   MUXDATA,     "samples per second",     FLT,    1,      16,     NULL,   NODATA, "%f",   0,      0,0,    11,25,INTEGER,
   MUXDATA,     "type of data",           CHR,    1,      20,     NULL,   NOCHAR, "%c",   0,      0,0,    13,25,DECIMAL,
/*50*/MUXDATA,  "data descriptor",        CHR,    1,      21,     NULL,   NOCHAR, "%c",   0,      0,0,    15,25,DECIMAL,
   MUXDATA,     "spareG",                 SHT,    1,      22,     NULL,   NODATA, "%d",   0,      0,0,    17,25,INTEGER,
   MUXDATA,     "number of samples",      LNG,    1,      24,     NULL,   NODATA, "%ld",  0,      0,0,    19,25,INTEGER,
   DESCRIPTRACE,"station/component",      STI,    1,      0,      NULL,   NOSTRG, STIFM,STAT_IDENT,0,0,   3,10,DECIMAL,
   DESCRIPTRACE,"beginning time",         MST,    1,      12,     NULL,   NOTIME, MSTFM,  0,      0,0,    7,25,TIMES,
   DESCRIPTRACE,"local time diff",        SHT,    1,      20,     NULL,   NODATA, "%d",   0,      0,0,    9,25,INTEGER,
   DESCRIPTRACE,"data type",              CHR,    1,      22,     NULL,   NOCHAR, "%c",   0,      0,0,    11,25,DECIMAL,
   DESCRIPTRACE,"data descriptor",        CHR,    1,      23,     NULL,   NOCHAR, "%c",   0,      0,0,    13,25,DECIMAL,
   DESCRIPTRACE,"digitized by",           SHT,    1,      24,authority,   NOLIST, "%d",   0,      0,0,    15,25,INTEGER,
   DESCRIPTRACE,"processed by",           SHT,    1,      26,authority,   NOLIST, "%d",   0,      0,0,    17,25,INTEGER,
/*60*/DESCRIPTRACE,"number of samples",   LNG,    1,      28,     NULL,   NODATA, "%ld",  0,      0,0,    19,25,INTEGER,
   DESCRIPTRACE,"samples per second",     FLT,    1,      32,     NULL,   NODATA, "%f",   0,      0,0,    21,25,DECIMAL,
   DESCRIPTRACE,"minimum data value",     FLT,    1,      36,     NULL,   NODATA, "%f",   0,      0,0,    23,25,DECIMAL,
   DESCRIPTRACE,"maximum data value",     FLT,    1,      40,     NULL,   NODATA, "%f",   0,      0,0,    25,25,DECIMAL,
   DESCRIPTRACE,"average noise",          FLT,    1,      44,     NULL,   NODATA, "%f",   0,      0,0,    27,25,DECIMAL,
   DESCRIPTRACE,"num clipped samples",LNG,        1,      48,     NULL,   NODATA, "%ld",  0,      0,0,    29,25,DECIMAL,
   LOCTRACE,    "station/component",      STI,    1,      0,      NULL,   NOSTRG, STIFM,STAT_IDENT,0,0,   3,10,DECIMAL,
   LOCTRACE,    "pathname",               LNG,    1,      8,      NULL,   NODATA, "%ld",  0,      0,0,    5,25,INTEGER,
   LOCTRACE,    "tape name",              LNG,    1,      12,     NULL,   NODATA, "%ld",  0,      0,0,    7,25,INTEGER,
   LOCTRACE,    "offset",                 LNG,    1,      16,     NULL,   NODATA, "%ld",  0,      0,0,    9,25,INTEGER,
/*70*/CALIBRATION,      "station/component",    STI,    1,      0,      NULL,   NOSTRG, STIFM,STAT_IDENT,0,0,   3,25,DECIMAL,
   CALIBRATION, "normalization factor", FLT,      1,      8,      NULL,   NODATA, "%f",   0,      0,0,    5,25,DECIMAL,
   CALIBRATION,"calibration",   CAL,NOCALPTS,      12,     NULL,   NODATA, CALFM,  CALIB,  0,0,    7,25,DECIMAL,
   CALIBRATION, "beginning time",         STT,    1,      512,    NULL,   NOTIME, STTFM,  0,      0,0,    21,25,TIMES,
   CALIBRATION, "ending time",            STT,    1,      516,    NULL,   NOTIME, STTFM,  0,      0,0,    23,25,TIMES,
   FEATURE,     "station/component",      STI,    1,      0,      NULL,   NOSTRG, STIFM,STAT_IDENT,0,0,   3,10,DECIMAL,
   FEATURE,     "observed phase",         SHT,    1,      8,feat_phase,   NOLIST, "%d",   0,      0,0,    5,25,INTEGER,
   FEATURE,     "onset descriptor",       CHR,    1,      10,     NULL,   NOCHAR, "%c",   0,      0,0,    7,25,DECIMAL,
   FEATURE,     "first motion",           CHR,    1,      11,     NULL,   NOCHAR, "%c",   0,      0,0,    9,25,DECIMAL,
   FEATURE,     "signal/noise",           SHT,    1,      12,     NULL,   NODATA, "%d",   0,      0,0,    11,25,INTEGER,
/*80*/FEATURE,  "data source",            CHR,    1,      14,     NULL,   NOCHAR, "%c",   0,      0,0,    13,25,DECIMAL,
   FEATURE,     "timing quality",         CHR,    1,      15,     NULL,   NOCHAR, "%c",   0,      0,0,    15,25,DECIMAL,
   FEATURE,     "amplitude quality",      CHR,    1,      16,     NULL,   NOCHAR, "%c",   0,      0,0,    17,25,DECIMAL,
   FEATURE,     "amplitude units",        CHR,    1,      17,     NULL,   NOCHAR, "%c",   0,      0,0,    19,25,DECIMAL,
   FEATURE,     "gain range factor",      SHT,    1,      18,     NULL,   NODATA, "%d",   0,      0,0,    21,25,INTEGER,
   FEATURE,     "phase time",             MST,    1,      20,     NULL,   NOTIME, MSTFM,  0,      0,0,    23,25,TIMES,
   FEATURE,     "phase amplitude",        FLT,    1,      28,     NULL,   NODATA, "%f",   0,      0,0,    25,25,DECIMAL,
   FEATURE,     "phase period",           FLT,    1,      32,     NULL,   NODATA, "%f",   0,      0,0,    27,25,DECIMAL,
   RESIDUAL,    "event number",           LNG,    1,      0,      NULL,   NODATA, "%ld",  0,      0,0,    3,25,DECIMAL,
   RESIDUAL,    "station/component",      STI,    1,      4,      NULL,   NOSTRG, STIFM,STAT_IDENT,0,0,   5,10,DECIMAL,
/*90*/RESIDUAL, "phase code",             SHT,    1,      12,feat_phase,  NOLIST, "%d",   0,      0,0,    7,25,INTEGER,
   RESIDUAL,    "timing quality",         CHR,    1,      14,     NULL,   NOCHAR, "%c",   0,      0,0,    9,25,DECIMAL,
   RESIDUAL,    "amplitude quality",      CHR,    1,      15,     NULL,   NOCHAR, "%c",   0,      0,0,    11,25,DECIMAL,
   RESIDUAL,    "traveltime residual",    FLT,    1,      16,     NULL,   NODATA, "%f",   0,      0,0,    13,25,DECIMAL,
   RESIDUAL,    "solution weight",        FLT,    1,      20,     NULL,   NODATA, "%f",   0,      0,0,    15,25,DECIMAL,
   RESIDUAL,    "delay time",             FLT,    1,      24,     NULL,   NODATA, "%f",   0,      0,0,    17,25,DECIMAL,
```

```
   RESIDUAL,     "azimuth to station",    FLT,    1,      28,      NULL,    NODATA, "%f",    0,      0,0,    19,25,DECIMAL,
   RESIDUAL,     "distance to station",   FLT,    1,      32,      NULL,    NODATA, "%f",    0,      0,0,    21,25,DECIMAL,
   RESIDUAL,     "angle of emergence",    FLT,    1,      36,      NULL,    NODATA, "%f",    0,      0,0,    23,25,DECIMAL,
   EVENT,        "organization",          SHT,    1,      0,authority,      NOLIST, "%d",    0,      0,0,    3,25,INTEGER,
/*100*/EVENT,    "event number",          LNG,    1,      2,       NULL,    NODATA, "%ld",   0,      0,0,    5,25,INTEGER,
   EVENT,        "number felt reports",   SHT,    1,      6,       NULL,    NODATA, "%d",    0,      0,0,    7,25,INTEGER,
   EVENT,        "MM intensity",          CHR,    1,      8,       NULL,    NOCHAR, "%c",    0,      0,0,    9,25,DECIMAL,
   EVENT,        "event type",            CHR,    1,      9,       NULL,    NOCHAR, "%c",    0,      0,0,    11,25,DECIMAL,
   EVENT,        "tectonism",             CHR,    1,      10,      NULL,    NOCHAR, "%c",    0,      0,0,    13,25,DECIMAL,
   EVENT,        "waterwaves",            CHR,    1,      11,      NULL,    NOCHAR, "%c",    0,      0,0,    15,25,DECIMAL,
   EVENT,        "mechanism type",        CHR,    1,      12,      NULL,    NOCHAR, "%c",    0,      0,0,    17,25,DECIMAL,
   EVENT,        "explosive medium",      CHR,    1,      13,      NULL,    NOCHAR, "%c",    0,      0,0,    19,25,DECIMAL,
   EVENT,        "magnitude or lbs TNT",  FLT,    1,      14,      NULL,    NODATA, "%f",    0,      0,0,    21,25,DECIMAL,
   EV_DESCRIPT,  "earthquake name",       STR,    20,     0,       NULL,    NOSTRG, "%s",    0,      0,0,    3,25,DECIMAL,
/*110*/EV_DESCRIPT,"county",              STR,    16,     20,      NULL,    NOSTRG, "%s",    0,      0,0,    5,25,DECIMAL,
   EV_DESCRIPT,  "state",                 STR,    16,     36,      NULL,    NOSTRG, "%s",    0,      0,0,    7,25,DECIMAL,
   EV_DESCRIPT,"local time diff",         SHT,    1,      52,      NULL,    NODATA, "%d",    0,      0,0,    9,25,INTEGER,
   EV_DESCRIPT,  "unused",                SHT,    1,      54,      NULL,    NODATA, "%d",    0,      0,0,    11,25,INTEGER,
   ORIGIN,       "event number",          LNG,    1,      0,       NULL,    NODATA, "%ld",   0,      0,0,    3,25,INTEGER,
   ORIGIN,       "organization",          SHT,    1,      4,authority,      NOLIST, "%d",    0,      0,0,    4,25,INTEGER,
   ORIGIN,       "version",               CHR,    1,      6,       NULL,    NOCHAR, "%c",    0,      0,0,    5,25,DECIMAL,
   ORIGIN,       "processing status",     CHR,    1,      7,       NULL,    NOCHAR, "%c",    0,      0,0,    6,25,DECIMAL,
   ORIGIN,       "preferred location",    CHR,    1,      8,       NULL,    NOCHAR, "%c",    0,      0,0,    7,25,DECIMAL,
   ORIGIN,       "processing program",    CHR,    1,      9,       NULL,    NOCHAR, "%c",    0,      0,0,    8,25,DECIMAL,
/*120*/ORIGIN,   "depth control",         CHR,    1,      10,      NULL,    NOCHAR, "%c",    0,      0,0,    9,25,DECIMAL,
   ORIGIN,       "convergence",           CHR,    1,      11,      NULL,    NOCHAR, "%c",    0,      0,0,    10,25,DECIMAL,
   ORIGIN,       "region",                LNG,    1,      12,      NULL,    NODATA, "%ld",   0,      0,0,    11,25,INTEGER,
   ORIGIN,       "origin time",           MST,    1,      16,      NULL,    NOTIME, MSTFM,   0,      0,0,    12,25,TIMES,
   ORIGIN,       "latitude",              LLT,    1,      24,      NULL,    NODATA, LTTFM,   0,      0,0,    13,25,DECIMAL,
   ORIGIN,       "longitude",             LLT,    1,      32,      NULL,    NODATA, LTTFM,   0,      0,0,    14,25,DECIMAL,
   ORIGIN,       "depth, km",             FLT,    1,      40,      NULL,    NODATA, "%f",    0,      0,0,    15,25,DECIMAL,
   ORIGIN,       "horizontalerror",       FLT,    1,      44,      NULL,    NODATA, "%f",    0,      0,0,    16,25,DECIMAL,
   ORIGIN,       "error in depth",        FLT,    1,      48,      NULL,    NODATA, "%f",    0,      0,0,    17,25,DECIMAL,
   ORIGIN,       "rms of residuals",      FLT,    1,      52,      NULL,    NODATA, "%f",    0,      0,0,    18,25,DECIMAL,
/*130*/ORIGIN,   "crustal model used",    STR,    6,      56,      NULL,    NOSTRG, "%s",    0,      0,0,    19,25,DECIMAL,
   ORIGIN,       "azimuthal gap",         SHT,    1,      62,      NULL,    NODATA, "%d",    0,      0,0,    20,25,INTEGER,
   ORIGIN,       "closest station, km",   FLT,    1,      64,      NULL,    NODATA, "%f",    0,      0,0,    21,25,DECIMAL,
   ORIGIN,       "number of stations",    SHT,    1,      68,      NULL,    NODATA, "%d",    0,      0,0,    22,25,INTEGER,
   ORIGIN,       "number p phases rept.", SHT,    1,      70,      NULL,    NODATA, "%d",    0,      0,0,    23,25,INTEGER,
   ORIGIN,       "number p phases used",  SHT,    1,      72,      NULL,    NODATA, "%d",    0,      0,0,    23,55,INTEGER,
   ORIGIN,       "number s phases rept",  SHT,    1,      74,      NULL,    NODATA, "%d",    0,      0,0,    24,25,INTEGER,
   ORIGIN,       "number s phases used",  SHT,    1,      76,      NULL,    NODATA, "%d",    0,      0,0,    24,55,INTEGER,
   ORIGIN,       "magnitude type",        SHT,    1,      78,      NULL,    NODATA, "%d",    0,      0,0,    25,25,INTEGER,
   ORIGIN,       "number mag readings",   SHT,    1,      80,      NULL,    NODATA, "%d",    0,      0,0,    26,25,INTEGER,
/*140*/ORIGIN,   "number mag s used",     SHT,    1,      82,      NULL,    NODATA, "%d",    0,      0,0,    26,55,INTEGER,
   ORIGIN,       "magnitude",             FLT,    1,      84,      NULL,    NODATA, "%f",    0,      0,0,    27,25,DECIMAL,
   ORIGIN,       "magnitude weight",      FLT,    1,      88,      NULL,    NODATA, "%f",    0,      0,0,    28,25,DECIMAL,
   ORIGIN,       "magnitude rms",         FLT,    1,      92,      NULL,    NODATA, "%f",    0,      0,0,    29,25,DECIMAL,
   ORIGIN,       "date solution done",    STT,    1,      96,      NULL,    NOTIME, STTFM,   0,      0,0,    30,25,TIMES,
   ERROR,        "covariance matrix",     FLT,    10,     0,       NULL,    NODATA, "%f",    0,      0,0,    3,25,DECIMAL,
   FOCALMECH,    "strike plane a",        FLT,    1,      0,       NULL,    NODATA, "%f",    0,      0,0,    3,25,DECIMAL,
   FOCALMECH,    "dip plane a",           FLT,    1,      4,       NULL,    NODATA, "%f",    0,      0,0,    5,25,DECIMAL,
   FOCALMECH,    "rake plane a",          FLT,    1,      8,       NULL,    NODATA, "%f",    0,      0,0,    7,25,DECIMAL,
   FOCALMECH,    "strike plane b",        FLT,    1,      12,      NULL,    NODATA, "%f",    0,      0,0,    9,25,DECIMAL,
/*150*/FOCALMECH,"dip plane b",           FLT,    1,      16,      NULL,    NODATA, "%f",    0,      0,0,    11,25,DECIMAL,
   FOCALMECH,    "rake plane b",          FLT,    1,      20,      NULL,    NODATA, "%f",    0,      0,0,    13,25,DECIMAL,
   FOCALMECH,    "preferred plane",       CHR,    1,      24,      NULL,    NOCHAR, "%c",    0,      0,0,    15,25,DECIMAL,
   FOCALMECH,    "unused",                CHR,    3,      25,      NULL,    NOCHAR, "%c",    0,      0,0,    17,25,DECIMAL,
   MOMENT,       "datatypes",             BTS,    1,      0,       NULL,    "0",    "%x",    0,      0,0,    3,25,DECIMAL,
   MOMENT,       "constraints",           CHR,    1,      1,       NULL,    NOCHAR, "%c",    0,      0,0,    5,25,DECIMAL,
   MOMENT,       "unused",                CHR,    2,      2,       NULL,    NOCHAR, "%c",    0,      0,0,    7,25,DECIMAL,
   MOMENT,       "scalar moment",         FLT,    1,      4,       NULL,    NODATA, "%f",    0,      0,0,    9,25,DECIMAL,
   MOMENT,       "moment tensor",         FLT,    6,      8,       NULL,    NODATA, "%f",    0,      0,0,    11,25,DECIMAL,
   VELMODEL,     "network name",          STR,    4,      0,       NULL,    NOSTRG, "%s",    0,      0,0,    3,25,DECIMAL,
/*160*/VELMODEL,"model name",             STR,    6,      4,       NULL,    NOSTRG, "%s",    0,      0,0,    5,25,DECIMAL,
```

```
   VELMODEL,    "unused",                CHR,    1,      10,     NULL,   NOCHAR, "%c",   0,      0,0,    7,25,DECIMAL,
   VELMODEL,    "type",                  CHR,    1,      11,     NULL,   NOCHAR, "%c",   0,      0,0,    9,25,DECIMAL,
   VELMODEL,    "latitude pt A",         LLT,    1,      12,     NULL,   NODATA, LTTFM,  0,      0,0,    11,25,DECIMAL,
   VELMODEL,    "longitude pt A",        LLT,    1,      20,     NULL,   NODATA, LTTFM,  0,      0,0,    13,25,DECIMAL,
   VELMODEL,    "latitude pt B",         LLT,    1,      28,     NULL,   NODATA, LTTFM,  0,      0,0,    15,25,DECIMAL,
   VELMODEL,    "longitude pt B",        LLT,    1,      36,     NULL,   NODATA, LTTFM,  0,      0,0,    16,25,DECIMAL,
   LAYERS,      "thickness, km",         FLT,    1,      0,      NULL,   NODATA, "%f",   0,      0,0,    3,25,DECIMAL,
   LAYERS,      "p velocity at top",     FLT,    1,      4,      NULL,   NODATA, "%f",   0,      0,0,    5,25,DECIMAL,
   LAYERS,      "p velocity at base",    FLT,    1,      8,      NULL,   NODATA, "%f",   0,      0,0,    7,25,DECIMAL,
/*170*/LAYERS,  "s velocity at top",     FLT,    1,      12,     NULL,   NODATA, "%f",   0,      0,0,    9,25,DECIMAL,
   LAYERS,      "s velocity at base",    FLT,    1,      16,     NULL,   NODATA, "%f",   0,      0,0,    11,25,DECIMAL,
   LAYERS,      "velocity function",     SHT,    1,      20,     NULL,   NODATA, "%d",   0,      0,0,    13,25,INTEGER,
   LAYERS,      "spare",                 SHT,    1,      24,     NULL,   NODATA, "%d",   0,      0,0,    15,25,INTEGER,
   COMMENT,     "structure identifier",  SHT,    1,      0,      NULL,   NODATA, "%d",   0,      0,0,    3,25,INTEGER,
   COMMENT,     "field number",          SHT,    1,      2,      NULL,   NODATA, "%d",   0,      0,0,    5,25,INTEGER,
   COMMENT,     "comment length",        SHT,    1,      4,      NULL,   NODATA, "%d",   0,      0,0,    7,25,INTEGER,
   COMMENT,     "unused",                SHT,    1,      6,      NULL,   NODATA, "%d",   0,      0,0,    9,25,INTEGER,
   PROFILE,     "not right",             SHT,    1,      6,      NULL,   NODATA, "%d",   0,      0,0,    3,25,INTEGER,
   SHOTGATHER,  "not right",             SHT,    1,      6,      NULL,   NODATA, "%d",   0,      0,0,    3,25,INTEGER,
/*180*/CALIB,   "pole",                  CPX,    1,      0,      NULL,   NODATA, COMFM,COMPLEX,  0,0,    0,25,DECIMAL,
   CALIB,       "zero",                  CPX,    1,      4,      NULL,   NODATA, COMFM,COMPLEX,  0,0,    1,25,DECIMAL,
   COMPLEX,     "real",                  FLT,    1,      0,      NULL,   NODATA, "%f",   0,      0,0,    0,25,DECIMAL,
   COMPLEX,     "imaginary",             FLT,    1,      4,      NULL,   NODATA, "%f",   0,      0,0,    0,55,DECIMAL,
   TRIGGERS,    "station/component",     STI,    1,      0,      NULL,   NOSTRG, STIFM,STAT_IDENT,0,0,   3,10,DECIMAL,
   TRIGGERS,    "short-term average",    SHT,    1,      12,     NULL,   NODATA, "%d",   0,      0,0,    7,25,INTEGER,
   TRIGGERS,    "long-term average",     SHT,    1,      14,     NULL,   NODATA, "%d",   0,      0,0,    9,25,INTEGER,
   TRIGGERS,    "sh-term abs average",   SHT,    1,      16,     NULL,   NODATA, "%d",   0,      0,0,    11,25,INTEGER,
   TRIGGERS,    "lg-term abs average",   SHT,    1,      18,     NULL,   NODATA, "%d",   0,      0,0,    13,25,INTEGER,
   TRIGGERS,    "trigger level",         SHT,    1,      20,     NULL,   NODATA, "%d",   0,      0,0,    15,25,INTEGER,
/*190*/TRIGGERS,"number of triggers",     SHT,    1,      22,     NULL,   NODATA, "%d",   0,      0,0,    17,25,INTEGER,
   TRIGGERS,    "time of first trigger",MST,    1,      24,     NULL,   NOTIME, MSTFM,  0,      0,0,    19,25,TIMES,
   TRIGSETTING, "network name",          STR,    4,      0,      NULL,   "unk",  "%s",   0,      0,0,    3,25,DECIMAL,
   TRIGSETTING, "time of first trigger",MST,    1,      4,      NULL,   NOTIME, MSTFM,  0,      0,0,    5,25,TIMES,
   TRIGSETTING, "const1",               SHT,    1,      12,     NULL,   NODATA, "%d",   0,      0,0,    7,25,INTEGER,
   TRIGSETTING, "const2",               SHT,    1,      14,     NULL,   NODATA, "%d",   0,      0,0,    9,25,INTEGER,
   TRIGSETTING, "threshold",            SHT,    1,      16,     NULL,   NODATA, "%d",   0,      0,0,    11,25,INTEGER,
   TRIGSETTING, "const3",               SHT,    1,      18,     NULL,   NODATA, "%d",   0,      0,0,    13,25,INTEGER,
   TRIGSETTING, "const4",               SHT,    1,      20,     NULL,   NODATA, "%d",   0,      0,0,    15,25,INTEGER,
   TRIGSETTING, "weighted ave increment",SHT,   1,      22,     NULL,   NODATA, "%d",   0,      0,0,    17,25,INTEGER,
   TRIGSETTING, "sweep in seconds",     FLT,    1,      24,     NULL,   NODATA, "%f",   0,      0,0,    19,25,DECIMAL,
   TRIGSETTING, "aperture",             FLT,    1,      28,     NULL,   NODATA, "%f",   0,      0,0,    21,25,DECIMAL,
   TRIGSETTING, "spare",                LNG,    1,      32,     NULL,   NODATA, "%ld",  0,      0,0,    22,25,INTEGER,
};

int stform_len=(sizeof(stform)/sizeof(struct sudsform));

/* Indecies in stform for beginning of each structure type
 */
LG_INT beg_struct[]={
   0,1,5,9,11,21,44,53,66,70,75,88,99,109,114,145,146,154,159,167,174,178,179,180,182,184,192,
};

/* width of ascii field for input and output of each variable type
 */
LG_INT width_field[]={
   0,1,5,25,5,10,12,16,16,12,16,16,16,12,12,
};

LG_INT size_struct[]={
   0,
   sizeof(STATIDENT),
   sizeof(struct structtag),
   sizeof(struct terminator),
   sizeof(struct equipment),
   sizeof(struct stationcomp),
```

```
    sizeof(struct muxdata),
    sizeof(struct descriptrace),
    sizeof(struct loctrace),
    sizeof(struct calibration),
    sizeof(struct feature),
    sizeof(struct residual),
    sizeof(struct event),
    sizeof(struct ev_descript),
    sizeof(struct origin),
    sizeof(struct error),
    sizeof(struct focalmech),
    sizeof(struct moment),
    sizeof(struct velmodel),
    sizeof(struct layers),
    sizeof(struct comment),
    sizeof(struct profile),
    sizeof(struct shotgather),
    sizeof(struct calib),
    sizeof(COMPLEXX),
    sizeof(struct triggers),
    sizeof(struct trigsetting),
};

#endif SUDS_DESCRIP_H
```

```
/*  ST_ERROR.H
 * Bruce R. Julian, USGS Menlo Park Calif., 15 Dec. 1983
 * Modified PLWard, USGS Menlo Park Calif., April 8, 1988
 */
#ifndef SUDSERR_H
#define SUDSERR_H

static char St_error_h[] = "@(#)st_error.h      1.2 11/17/88";  /* Sccs Id */

#define ERRBASE          1000
#define EBADDAY          (ERRBASE + 1)
#define EBADMONTH        (ERRBASE + 2)
#define EBADYEAR         (ERRBASE + 3)
#define EBADVALUE        (ERRBASE + 4)
#define EFEWDATA         (ERRBASE + 5)
#define ENOCONVRG        (ERRBASE + 6)
#define EUNDERFLOW       (ERRBASE + 7)
#define ENOSTATION       (ERRBASE + 8)
#define ESIZE            (ERRBASE + 9)
#define ESINGULAR        (ERRBASE + 10)
#define ETOODEEP         (ERRBASE + 11)
#define EBADPHASE        (ERRBASE + 12)
#define ENOFEAS          (ERRBASE + 13)
#define EINFIN           (ERRBASE + 14)
#define EBADSYNC         (ERRBASE + 15)
#define ST_ERR           (ERRBASE + 16)
#define ECANTBRACKET     (ERRBASE + 17)

#endif SUDSERR_H
```

15

```
/* SUDS: Version 1.2 AUG 10,1988
   Some care has been taken to keep bytes aligned on 4 byte boundaries for
      portabilty.
   Individual fields within all structures have unique names for compatability
      with the database system.
   Expandability is accomplished by adding new structures and by adding new
      fields to the end of these structures. Such additions need to be
      coordinated among all users and should not be taken lightly.
   PLNard, U.S. Geological Survey, Menlo Park, Ca. 94025
 */
#ifndef SUDS_H
#define SUDS_H

static char Suds_h[] = "%W% %G%";       /* Sccs Id */

/* Define code for type of machine suds is being compiled on.
   See suds_codes.h  CODES mach_type[]
   THIS DEFINE MUST BE CHANGED FOR MACHINES NOT A SUN 3
 */
#define MACHINE '3'

/* Define symbols for missing data. See suds_descr.h
 */
#define NODATA "-32767."
#define NOTIME "-2147472000"    /* Dec 14, 1901 00:00 00 GMT, near largest long */
#define NOCHAR "_"
#define NOSTRG ""
#define NOLIST "0"

/* Standard variable types redefined for portability and clarity
 */
typedef char    CHAR;   /* A single ascii character                      */
typedef char    MINI;   /* A 1 byte integer (0 to 255)                   */
typedef char    STRING; /* A character string, null byte terminated      */
typedef char    BITS8;  /* An 8 bit field                                */
typedef short   SH_INT; /* A 16 bit signed integer                       */
typedef long    LG_INT; /* A 32 bit signed integer                       */
typedef float   FLOAT;  /* A 32 bit floating point number, IEEE          */
typedef double  DOUBLE; /* A 64 bit double precision number, IEEE        */
typedef struct {
   FLOAT   fx;
   FLOAT   fy;
   } VECTOR;

typedef struct {
   FLOAT   cr;
   FLOAT   ci;
   } COMPLEXX;

typedef struct {
   DOUBLE  dr;
   DOUBLE  di;
   } D_COMPLEX;

typedef struct {
   FLOAT   xx;
   FLOAT   YY;
   FLOAT   xy;
   } TENSOR;

/* Special variable types
 */
typedef struct {        /* station component identifier                  */
   STRING  network[4];  /* network name                                  */
   STRING  st_name[5];  /* name of station where equipment is located    */
   CHAR    component;   /* component v,n,e                               */
   SH_INT  inst_type;   /* instrument type                               */
   } STATIDENT;

typedef LG_INT ST_TIME; /*  stamp GMT time in seconds before or after Jan 1,
                               1970, resolution is one second          */

typedef DOUBLE MS_TIME; /* GMT time in seconds before or after Jan 1,
                               1970, resolution finer than microseconds*/

typedef DOUBLE LONLAT;  /* latitude or longitude in degrees, N and E positive*/

typedef struct codes {
   LG_INT num;
   STRING *meaning;
   } CODES;

/* SUDSFORM: Information on each suds structure used for input from ascii,
   output to ascii, loading or unloading the database, etc.
 */
typedef struct sudsform {
   LG_INT  fstype;      /* Structure types or identifiers                */
   STRING  *fname;      /* Name used to identify structure item          */
   LG_INT  ftype;       /* Type of variable in field                     */
   LG_INT  flength;     /* Length of variable in the field               */
   LG_INT  offset;      /* Offset of variable pointer from beginning of structure
                           mneasured in bytes, where first byte of structure=0.
                           This field is potentail machine dependent and ideally
                           should be determined by a program like the ddlp
                           (Data Description Language Processor) of db_vista.
                           Asd long as structures are carefully defined to
                           align 4 bytes boundaries, we can get away without
                           writing such a complex program for most machines     */
   CODES   *codelist;   /* If value indexes a codes list, this is list   */
   STRING  *initval;    /* Value to initialize structure                 */
   STRING  *fformat;    /* Printf type format to read + write field      */
   LG_INT  nextfstype;  /* If structure, this is fstype                  */
   LG_INT  frecord;     /* database record for this field                */
   LG_INT  ffield;      /* database field for this field                 */
   LG_INT  form_row;    /* row of field in forms editor                  */
   LG_INT  form_col;    /* beginning column of field in forms editor     */
   STRING  *allowchar;  /* allowed characters on input                   */
   } SUDS_FORM;

/* Integer defines for standard variable types
 */
#define  CHR    1       /* char 1 byte                  */
#define  MIN    2       /* char number 1 byte           */
#define  STR    3       /* string                       */
#define  BTS    4       /* char 1 byte                          */
#define  SHT    5       /* short 2 bytes                        */
#define  LNG    6       /* long 4 bytes                         */
#define  FLT    7       /* float 4 bytes                        */
#define  DBL    8       /* double 8 bytes                       */
#define  STI    9       /* struct stat_ident 8 bytes            */
#define  STT    10      /* st_time 4 bytes                      */
#define  MST    11      /* ms_time 8 bytes                      */
#define  LLT    12      /* lon_lat 8 bytes                      */
#define  CAL    13      /* struct calib 500 bytes               */
#define  CPX    14      /* struct complex 16 bytes              */

/* Structure types or identifiers
 */
#define NO_STRUCT       0
#define STAT_IDENT      1
```

```
#define STRUCTTAG       2
#define TERMINATOR      3
#define EQUIPMENT       4
#define STATIONCOMP     5
#define MUXDATA         6
#define DESCRIPTRACE    7
#define LOCTRACE        8
#define CALIBRATION     9
#define FEATURE         10
#define RESIDUAL        11
#define EVENT           12
#define EV_DESCRIPT     13
#define ORIGIN          14
#define ERROR           15
#define FOCALMECH       16
#define MOMENT          17
#define VELMODEL        18
#define LAYERS          19
#define COMMENT         20
#define PROFILE         21
#define SHOTGATHER      22
#define CALIB           23
#define COMPLEX         24
#define TRIGGERS        25
#define TRIGSETTING     26

#define TOTAL_STRUCTS   26

 /* STRUCTTAG to identify structures when archived together
 */
struct structtag {
   CHAR sync;           /* The letter S. If not present, error exists. Use
                           to unscramble damaged files or tapes    */
   CHAR machine;        /* code for machine writing binary file for use in
                           identifying byte order and encoding.    */
   SH_INT id_struct;    /* structure identifier: numbers defined above  */
   LG_INT len_struct;   /* structure length in bytes for fast reading and
                           to identify new versions of the structure */
   LG_INT len_data;     /* length of data following structure in bytes  */
   };

#define ST_MAGIC   'S'  /* magic character for sync in structtag        */

/* TERMINATOR: Structure to end a sequence of related structures when
      loaded in a serial file or on a serial device
 */
struct terminator {
   SH_INT  structid;    /* id for structure at beginning of this sequence*/
   SH_INT  spareA;
   };

/* EQUIPMENT: Equipment making up a station/component. Primarily used for
      maintenance but may be referenced by researcher. One or more structures
      exist for each piece of equipment making up a station/component.
 */
struct equipment {
   STATIDENT this;      /* identifier of this piece of equipment        */
   STATIDENT previous;  /* next piece of equipment toward sensor        */
   STATIDENT next;      /* next piece of equipment toward recorder      */
   STRING  serial[8];   /* serial number                                */
   SH_INT  model;       /* model such as L4, HS10, etc.                 */
   SH_INT  knob1;       /* knob setting or series resistor value of Lpad*/
   SH_INT  knob2;       /* knob setting or shunt  resistor value of Lpad*/
   SH_INT  reason;      /* reason change was made                       */
   FLOAT   frequency;   /* sensor corner frequency, vco freq, transmitter
                           frequency, etc.                              */
   ST_TIME effective;   /* date/time these values became effective      */
   };

/* STATIONCOMP: generic station component information available before an
      event is detected and added to the waveforms when a detected event
      is stored. Primary station information used by researcher.
 */
struct stationcomp {
   STATIDENT sc_name;   /* station component identification             */
   SH_INT  azim;        /* component azimuth clockwise from north,
                           0 for vertical                               */
   SH_INT  incid;       /* component angle of incidence from vertical
                           0 is vertical, 90 is horizontal              */
   LONLAT  st_lat;      /* latitude, north is plus                      */
   LONLAT  st_long;     /* longitude, east is plus                      */
   FLOAT   elev;        /* elevation in meters                          */
   CHAR    enclosure;   /* d=dam, n=nuclear power plant, v=underground
                           vault, b=buried, s=on surface, etc.          */
   CHAR    annotation;  /* annotated comment code                       */
   CHAR    recorder;    /* type device data recorded on                 */
   CHAR    rockclass;   /* i=igneous, m=metamorphic, s=sedimentary      */
   SH_INT  rocktype;    /* code for type of rock                        */
   CHAR    sitecondition;/* p=permafrost, etc.                          */
   CHAR    sensor_type; /* sensor type: d=displacement, v=velocity,
                           a=acceleration                               */
   CHAR    data_type;   /* s=short(16 bit), r=12 bit data, 4 lsb time,
                           l=long(32 bit), f=float,
                           d=double, c=complex, v=vector, t=tensor      */
   CHAR    data_units;  /* data units: d=digital counts, v=millivolts,
                           n=nanometers (/sec or /sec/sec)              */
   CHAR    polarity;    /* n=normal, r=reversed                         */
   CHAR    st_status;   /* d=dead, g=good                               */
   FLOAT   max_gain;    /* maximum gain of the amplifier                */
   FLOAT   clip_value;  /* +-value of data where clipping begins        */
   FLOAT   con_mvolts;  /* conversion factor to millivolts: mv per counts
                           0 means not defined or not appropriate       */
   SH_INT  channel;     /* a2d channel number                           */
   SH_INT  spareH;      /* spare                                        */
   ST_TIME effective;   /* date/time these values became effective      */
   };

/* MUXDATA: header for multiplexed data
 */
struct muxdata {
   STRING  netname[4];  /* network name                                 */
   MS_TIME begintime;   /* time of first data sample                    */
   SH_INT  loctime;     /* minutes to add to GMT to get local time      */
   SH_INT  numchans;    /* number of channels: if !=1 then multiplexed  */
   FLOAT   dig_rate;    /* samples per second                           */
   CHAR    typedata;    /* s=short(16 bit), r=12 bit data, 4 lsb time,
                           l=long(32 bit), f=float,
                           d=double, c=complex, v=vector, t=tensor      */
   CHAR    descript;    /* g=good, t=telemetry noise, c=calibration, etc*/
   SH_INT  spareG;      /* spare                                        */
   LG_INT  numsamps;    /* number of sample sweeps. Typically not known
                           when header is written, but can be added later*/
   };

/* DESCRIPTRACE: descriptive information about a seismic trace.
      Normally followed by waveform. Thus length is usually
      sizeof(struct descriptrace) plus length of trace in bytes.
 */
struct descriptrace {
   STATIDENT dt_name;   /* station component identification             */
```

```
   MS_TIME begintime;   /* time of first data sample                    */
   SH_INT  localtime;   /* minutes to add to GMT to get local time      */
   CHAR    datatype;    /* s=short(16 bit), r=12 bit data, 4 lsb time,
                           l=long(32 bit), f=float,
                           d=double, c=complex, v=vector, t=tensor      */
   CHAR    descriptor;  /* g=good, t=telemetry noise, c=calibration, etc*/
   SH_INT  digi_by;     /* agency code who digitized record; 0=original */
   SH_INT  processed;   /* processing done on this waveform             */
   LG_INT  length;      /* number of samples in trace                   */
   FLOAT   rate;        /* samples per second                           */
   FLOAT   mindata;     /* minimum value of data (type s,l,f only)      */
   FLOAT   maxdata;     /* maximum value of data (type s,l,f only)      */
   FLOAT   avenoise;    /* average value of first 200 samples (type s,l,f only)*/
   LG_INT  numclip;     /* number of clipped datapoints                 */
   };

/* LOCTRACE: location of trace
 */
struct loctrace {
   STATIDENT lt_name;   /* station component identification             */
   STRING  *fileloc;    /* pointer to pathname in file system           */
   STRING  *tapeloc;    /* pointer to name of tape or offline storage   */
   LG_INT  beginloc;    /* bytes from begining of file to trace         */
   };


/* CALIBRATION: calibration information for a station component
 */
#define NOCALPTS 30

struct calibr   {
   COMPLEXX   pole;     /* pole                                         */
   COMPLEXX   zero;     /* zero                                         */
   };

struct calibration {
   STATIDENT ca_name;   /* station component identification             */
   FLOAT maxgain;       /* maximum gain of calibration curve            */
   FLOAT normaliz;      /* factor to multiply standard calib by to make
                           peak at given frequency=1                    */
   struct calib cal[NOCALPTS]; /* calibration info                      */
   ST_TIME   begint;    /* time this calibration becomes effective      */
   ST_TIME   endt;      /* time this calibration is no longer effective */
   };


/* FEATURE: Observed phase arrival time, amplitude, and period.
 */
struct feature {
   STATIDENT fe_name;   /* station component identification             */
   SH_INT  obs_phase;   /* observed phase code                          */
   CHAR    onset;       /* wave onset descriptor, i or e                */
   CHAR    direction;   /* first motion: U,D,+,-                        */
   SH_INT  sig_noise;   /* ratio ampl. of first peak or trough to noise */
   CHAR    data_source; /* i=interactive,a=automatic,r=rtp, or user code*/
   CHAR    tim_qual;    /* timing quality given by analyst: 0-4, etc.
                           n=ignore timing                              */
   CHAR    amp_qual;    /* amplitude quality given by analyst: 0-4, etc.
                           n=ignor amplitude information                */
   CHAR    ampunits;    /* units amplitude measured in: d=digital counts
                           m=mm on develocorder, etc.                   */
   SH_INT  gain_range;  /* 1 or gain multiplier if gain range in effect */
   MS_TIME time;        /* phase time, x value where pick was made      */
   FLOAT   amplitude;   /* peak-to-peak amplitude of phase              */
   FLOAT   period;      /* period of waveform measured                  */
   };

/* RESIDUAL: Calculated residuals for arrival times, magnitudes, etc.
 */
struct residual {
   LG_INT  event_num;   /* unique event number                          */
   STATIDENT re_name;   /* station component identification             */
   SH_INT  set_phase;   /* phase code set for this solution             */
   CHAR    set_tim_qual;/* timing quality assigned for this solution: 0-4*/
   CHAR    set_amp_qual;/* amplitude quality assigned for this solution: 0-4*/
   FLOAT   residual;    /* traveltime residual or phase magnitude       */
   FLOAT   weight_used; /* weight used in this solution                 */
   FLOAT   delay;       /* delay time or station correction used        */
   FLOAT   azimuth;     /* azimuth event to station, 0 north            */
   FLOAT   distance;    /* distance in km event to station              */
   FLOAT   emergence;   /* angle of emergence from source, 0=down,180=up*/
   };

/* EVENT: general information about an event
 */
struct event{
   SH_INT  authority;   /* organization processing the data             */
   LG_INT  number;      /* unique event number assigned by organization */
   SH_INT  felt;        /* number of felt reports                       */
   CHAR    mintensity;  /* maximum Modified Mercali Intensity           */
   CHAR    ev_type;     /* e=earthquake, E=explosion, n=nuclear,
                           i=icequake, b=b_type, n=net, r=regional,
                           t=teleseism, c=calibration, n=noise          */
   CHAR    tectonism;   /* observed u=uplift, s=subsidence, S=strikeslip
                           faulting, N=normal faulting, T=thrust        */
   CHAR    waterwave;   /* seiche, tsunami, etc.                        */
   CHAR    mechanism;   /* t=thrust, s=strike-slip, n=normal, e=explosive*/
   CHAR    medium;      /* medium containing explosion or event         */
   FLOAT   size;        /* magnitude or pounds TNT for explosions       */
   };

/* EV_DESCRIPT: descriptive information about an event typically used for
   major, destructive earthquakes. This structure is typically associated
   with EVENT structure.
 */
struct ev_descript {
   STRING  eqname[20];  /* Popular name used to refer to this earthquake*/
   STRING  country[16]; /* country of earthquake                        */
   STRING  state[16];   /* state, province or other political subdivision*/
   SH_INT  localtime;   /* hours to add to GMT to get local time        */
   SH_INT  spareB;
   };

/* ORIGIN: information about a specific solution for a given event
 */
struct origin {
   LG_INT  number;      /* unique event number assigned by organization */
   SH_INT  authority;   /* organization processing the data             */
   CHAR    version;     /* version of solution within organization      */
   CHAR    or_status;   /* processing status: f=final, a=automatic, etc */
   CHAR    preferred;   /* p=preferred location                         */
   CHAR    program;     /* name of processing progrsm  h=hypo71, l=hypolayer,
                           i=isc, c=centroid, etc.                      */
   CHAR    depcontrl;   /* depth control: f=fixed, etc.                 */
   CHAR    convergence; /* hypocentral convergence character            */
   LG_INT  region;      /* geographic region code assigned locally      */
   MS_TIME orgtime;     /* origin time                                  */
   LONLAT  or_lat;      /* latitude, north is plus                      */
   LONLAT  or_long;     /* longitude, east is plus                      */
   FLOAT   depth;       /* depth in kilometers, + down                  */
```

```
   FLOAT   err_horiz;   /* horizontal error in km                          */
   FLOAT   err_depth;   /* vertical error in km                            */
   FLOAT   res_rms;     /* rms of residuals                                */
   STRING  crustmodel[6];/* code for model used in this location           */
   SH_INT  gap;         /* azimuthal gap in degrees                        */
   FLOAT   nearstat;    /* distance in km to nearest station               */
   SH_INT  num_stats;   /* number of stations reporting phases             */
   SH_INT  rep_p;       /* number of p phases reported                     */
   SH_INT  used_p;      /* number of p times used in the solution          */
   SH_INT  rep_s;       /* number of s phases reported                     */
   SH_INT  used_s;      /* number of s times used in the solution          */
   SH_INT  mag_type;    /* magnitude type: coda,tau,xmag ml,mb,ms,mw       */
   SH_INT  rep_m;       /* number of magnitude readings reported           */
   SH_INT  used_m;      /* number of magnitude readings used               */
   FLOAT   magnitude;   /* magnitude value                                 */
   FLOAT   weight;      /* average magnitude weight                        */
   FLOAT   mag_rms;     /* rms of magnitude                                */
   ST_TIME effective;   /* time this solution was calculated               */
   };

/* ERROR
 */
struct error {
   FLOAT   covarr[10];  /* covariance matrix                               */
   };


/* FOCALMECH
 */
struct focalmech {
   FLOAT   astrike;     /* strike of plane a                               */
   FLOAT   adip;        /* dip of plane a                                  */
   FLOAT   arake;       /* rake of plane a                                 */
   FLOAT   bstrike;     /* strike of plane b                               */
   FLOAT   bdip;        /* dip of plane b                                  */
   FLOAT   brake;       /* rake of plane b                                 */
   CHAR    prefplane;   /* preferred plane a or b or blank                 */
   CHAR    spareC[3];
   };

/* MOMENT TENSOR (Might be used instead of focal mechanism)
 */
struct moment{
   BITS8   datatypes;   /* sum of: 1=polarities, 2=amplitudes,
                           4=waveforms, etc.                               */
   CHAR    constraints; /* solution constrained: d=deviatoric,
                           c=double couple                                 */
   CHAR    spareD[2];
   FLOAT   sc_moment;   /* scalar moment                                   */
   FLOAT   norm_ten[6]; /* normalized moment tensor                        */
   };

/* VELMODEL: Velocity model
 */
struct velmodel {
   STRING  netname[4];  /* network name                                    */
   STRING  modelname[6];/* model name                                      */
   CHAR    spareE;
   CHAR    modeltype;   /* p=profile A to B, a=area within corners A B     */
   LONLAT  latA;        /* latitude of point A, north is plus              */
   LONLAT  longA;       /* longitude of point A, east is plus              */
   LONLAT  latB;        /* latitude of point B, north is plus              */
   LONLAT  longB;       /* longitude of point B, east is plus              */
   };
```

```
/* LAYERS: Velocity layers
 */
struct layers {
   FLOAT   thickness;   /* thickness in kilometers                         */
   FLOAT   pveltop;     /* p velocity at top of layer                      */
   FLOAT   pvelbase;    /* p velocity at base of layer                     */
   FLOAT   sveltop;     /* s velocity at top of layer                      */
   FLOAT   svelbase;    /* s velocity at base of layer                     */
   SH_INT  function;    /* velocity function in layer: 0=constant,
                           1=linear, 2=exponential, etc.                   */
   SH_INT  spareF;
   };

/* COMMENT: Comment tag to be followed by the bytes of comment
 */
struct comment {
   SH_INT  refer;       /* structure identifier comment refers to          */
   SH_INT  item;        /* item in structure comment refers to             */
   SH_INT  length;      /* number of bytes in comment                      */
   SH_INT  unused;
   };


/* PROFILE: Grouping of shotgathers by profile
 */
struct profile {
   int junk1;
   /* What is your suggestion? */
   };


/* SHOTGATHER: Grouping of waveforms by source event
 */
struct shotgather {
   int junk2;
   /* What is your suggestion? */
   };

/* TRIGSETTING: Settings for earthquake trigger system
 */
struct trigsetting {
   STRING  netwname[4]; /* network name                                    */
   MS_TIME beginttime;  /* time these values in effect                     */
   SH_INT  const1;      /* trigger constant 1                              */
   SH_INT  const2;      /* trigger constant 2                              */
   SH_INT  threshold;   /* trigger threshold                               */
   SH_INT  const3;      /* trigger constant 3                              */
   SH_INT  const4;      /* trigger constant 4                              */
   SH_INT  wav_inc;     /* weighted average increment                      */
   FLOAT   sweep;       /* trigger sweep time in seconds                   */
   FLOAT   aperture;    /* seconds for coincident station triggers         */
   LG_INT  spareI;      /* spare                                           */
   };

/* TRIGGERS: Earthquake detector trigger statistics
 */
struct triggers {
   STATIDENT tr_name;   /* station component identification                */
   SH_INT  sta; /* short term average                                  */
   SH_INT  lta; /* long term average                                   */
   SH_INT  abs_sta;     /* short term absolute average                     */
   SH_INT  abs_lta;     /* long term absolute average                      */
   SH_INT  trig_value;  /* value of trigger level (sta)                    */
   SH_INT  num_triggers;/* number of times triggered during this event     */
   MS_TIME trig_time;   /* time of first trigger                           */
```

```
};

/* Misc. defines to protect sloppy programmers
 */
#include <stdio.h>
extern FILE *st_open(),*efopen();
extern LG_INT errno,die();
extern DOUBLE atof(),make_mstime();
extern SUDS_FORM stform[];
extern LG_INT size_struct[],beq_struct[],width_field[];
extern CODES inst_type[],mach_type[],equip_model[],equip_reason[],ann_com[],
   rock_type[],stat_proc[],authority[],feat_phase[],struct_names[];

#endif SUDS_H
```

LG

```
/* codes: structure relating numeric codes to strings
   EACH CODELIST MUST BE TERMINATED BY A "0,    0," to designate the end of list
   When adding new code arrays add new names to extern at end of suds.h
   and put names in suds_descr.h. You must then recompile suds_io.c
   PLWard, U.S. Geological Survey, Menlo Park, Ca 94025
 */
#ifndef SUDS_CODES_H
#define SUDS_CODES_H

static char Suds_codes_h[] = "@(#)suds_codes.h  1.2 11/17/88";  /* Sccs Id */

#include "/usr/suds/include/suds.h"

CODES 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",
   0,   0,
};

/* instrument types: use a number less than 100 except in equipment structures
 */
CODES 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",
   0,   0,
  };

CODES ann_com[]={
   0,   "no comment",
   };

CODES rock_type[]={
   0,   "none given",
   1,   "soil",
   };

CODES mach_type[]={
   ' ', "not given",
   '3', "sun 3",
   '4', "sun 4",
   '6', "80386",
   'i', "ibm",
   'v', "vax",
   'x', "xdr",
   0,   0,
   };

CODES equip_model[]={
   0,   "none given",
   1,   "14     geophone",
   2,   "hs10   geophone",
   3,   "ev17   geophone",
   100, "j302   amp/vco usgs",
   101, "j302m1 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",
   0,   0,
   };

CODES 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",
   0,   0,
   };

CODES reco_type[]={
   0,   "not specified",
   };

CODES rock_class[]={
   0,   "not specified",
   };

/* logical sum of these fields */
CODES 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.",
   0,   0,
   };

CODES feat_phase[]={
   0,   "not given",
   1,   "window",       /* amplitude is the duration of the window       */
   2,   "f finis",      /* for cods mag, time when signal is about twice noise*/
   3,   "z 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",
   0,   0,
   };

CODES 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",
    0,    0,
    };

#endif SUDS_CODES_H
```

```
SCCSID = "@(#)Makefile  1.2 11/18/88"
CFLAGS = -g -f68881
INCDIR = /usr/suds/include

INC = $(INCDIR)/suds_descr.h $(INCDIR)/suds_codes.h $(INCDIR)/suds.h \
        $(INCDIR)/st_err.h

LIBOBJ = asc2field.o descr_trace.o find_code.o st_error.o st_init.o st_time.o \
        suds_io.o

SRC = suds_io.c st_time.c st_init.c st_error.c find_code.c descr_trace.c \
        asc2field.o

libsuds.a: $(LIBOBJ)
        ar rc libsuds.a $(LIBOBJ)
        ranlib libsuds.a

install:
        cp libsuds.a /usr/lib
        ranlib /usr/lib/libsuds.a

list:
        enscriptW $(SRC) Makefile

descr_trace.o find_code.o st_init.o              : $(INCDIR)/suds.h
st_time.o suds_io.o                              : $(INCDIR)/suds.h
suds_descr.h suds_codes.h stedit.o asc2field.o   : $(INCDIR)/suds.h
st_time.o suds_io.o st_error.o                   : $(INCDIR)/st_error.h
suds_io.o                                        : $(INCDIR)/suds_descr.h
suds_io.o                                        : $(INCDIR)/suds_codes.h
```

```
/* convert a structure field to an ascii string and visa-versa
   PLWard, U.S. Geological Survey, Menlo Park, Ca 94025  5/18/88
 */
static char SccsId[] = "%W%\t%G%";
#define MAXSTR 36

#include "/usr/suds/include/suds.h"

extern double atof();
extern int atoi();

char *field2asc(ptr,type,verbose)
   char *ptr;
   int  type,verbose;
{
   CHAR    *ch;
   STRING  *st;
   SH_INT  *sh;
   LG_INT  *lg;
   FLOAT  *ft;
   DOUBLE *db;
   char temp[MAXSTR];

   switch(type) {
      case CHR: ch=ptr; sprintf(temp,"%c",*ch); break;
      case BTS:
      case MIN: ch=ptr; sprintf(temp,"%d",(int)*ch); break;
      case STR: st=ptr; sprintf(temp,"%s",st); break;
      case SHT: sh=(short *)(ptr);
                if(*sh==atoi(NODATA))sprintf(temp,"NODATA");
                else sprintf(temp,"%d",*sh); break;
      case LNG: lg=(long *)(ptr);
                if(*lg==atoi(NODATA))sprintf(temp,"NODATA");
                else sprintf(temp,"%ld",*lg); break;
      case FLT: ft=(float *)(ptr);
                if(*ft==atoi(NODATA))sprintf(temp,"NODATA");
                else sprintf(temp,"%f",*ft); break;
      case LLT:
      case DBL: db=(double *)(ptr);
                if(*db==atoi(NODATA))sprintf(temp,"NODATA");
                else sprintf(temp,"%lf",*db); break;
      case STT: lg=(long *)(ptr);
                if(verbose) {
                   *db= *lg;
                   if(*lg==atoi(NOTIME))sprintf(temp,"NOTIME");
                   else sprintf(temp,"%s",list_mstime(*db,7));
                }
                else {
                   if(*lg==atof(NOTIME))sprintf(temp,"NOTIME");
                   else sprintf(temp,"%ld",*lg);
                }
                break;
      case MST: db=(double *)(ptr);
                if(verbose) {
                   if(*db==atof(NOTIME))sprintf(temp,"NOTIME");
                   else sprintf(temp,"%s",list_mstime(*db,6));
                }
                else {
                   if(*db==atof(NOTIME))sprintf(temp,"NOTIME");
                   else sprintf(temp,"%lf",*db);
                }
                break;
      case CAL:
      case CPX:
      case STI:
      default:  strcpy(temp,"ERROR");
                break;

   }
   return(temp);
}

asc2field(string,ptr,type)
   char *ptr,*string;
   int  type;
{
   char *field;

   field=string;
   switch(type) {
      case CHR: if(strcmp(string,"NOCHAR")==0) field=NOCHAR;
                sscanf(field,"%c",(CHAR *)(ptr));
                break;
      case STR: if(strcmp(string,"NOSTRG")==0) field=NOSTRG;
                strcpy(ptr,field);
                break;
      case BTS:
      case MIN: *((MINI *)(ptr))=atoi(field);            break;
      case SHT: *((SH_INT *)(ptr))=atoi(field);          break;
      case LNG: *((LG_INT *)(ptr))=atoi(field);          break;
      case FLT: *((FLOAT *)(ptr))=atof(field);           break;
      case DBL: *((DOUBLE *)(ptr))=atof(field);          break;
      case STT: if(strcmp(field,"NOTIME")==0) field=NOTIME;
                *((ST_TIME *)(ptr))=atof(field);                  break;
      case MST: if(strcmp(field,"NOTIME")==0) field=NOTIME;
                *((MS_TIME *)(ptr))=atof(field);         break;
      case LLT: *((LONLAT *)(ptr))=atof(field);          break;
      default:  st_error(die,0,"unrecognized type %d",type);
   }
}
```

```
/* descr_trace: put mindata, maxdata, avenoise, numclip in descriptrace
   structure for short, long, and float type traces. Structure must include
   other values. trace follows descriptrace structure unless argument trace!=0.

   PLNard U.S.Geological Survey, Menlo Park, Ca 94025 5/4/88
 */
static char SccsId[] = "%W%\t%G%";
#include "/usr/suds/include/suds.h"

#define NOISELEN 200    /* number of samples to get average noise over  */

int descr_trace(sc,dt,trace)
   struct stationcomp *sc;
   struct descriptrace *dt;
   char *trace;
{
   register i;
   short *sh;
   long  *lg,clipnum;
   float *ft,min,max;
   double ave;

   ave=0.0;
   dt->numclip=0;
   switch(sc->data_type) {
      case 's': sh=(short *)(trace==0? (char *)(dt+1):trace);
                for(i=0,dt->mindata=sh[0],dt->maxdata=sh[0];i<dt->length;i++) {
                   if(sh[i]<min) dt->mindata=sh[i];
                   if(sh[i]>max) dt->maxdata=sh[i];
                   if(sc->clip_value!=0 &&
                      (sh[i]>sc->clip_value || sh[i]<sc->clip_value))dt->numclip++;
                   if(i<NOISELEN) ave+=sh[i];
                }
                break;
      case 'l': lg=(long *)(trace==0? (char *)(dt+1):trace);
                for(i=0,dt->mindata=lg[0],dt->maxdata=lg[0];i<dt->length;i++) {
                   if(lg[i]<min) dt->mindata=lg[i];
                   if(lg[i]>max) dt->maxdata=lg[i];
                   if(sc->clip_value!=0 &&
                      (lg[i]>sc->clip_value || lg[i]<sc->clip_value))dt->numclip++;
                   if(i<NOISELEN) ave+=lg[i];
                }
                break;
      case 'f': ft=(float *)(trace==0? (char *)(dt+1):trace);
                for(i=0,dt->mindata=ft[0],dt->maxdata=ft[0];i<dt->length;i++) {
                   if(ft[i]<min) dt->mindata=ft[i];
                   if(ft[i]>max) dt->maxdata=ft[i];
                   if(sc->clip_value!=0 &&
                      (ft[i]>sc->clip_value || ft[i]<sc->clip_value))dt->numclip++;
                   if(i<NOISELEN) ave+=ft[i];
                }
                break;
      default: return(1);
   }
   dt->avenoise=ave/(dt->length<NOISELEN?dt->length:NOISELEN);
   return(0);
}
```

```
/* Find numeric or ascii codes in codelists stored in suds_codes.h
   PLWard, U.S.Geological Survey, Menlo Park, Ca 94025 5/4/88
 */
static char SccsId[] = "%W%\t%G%";
#include "/usr/suds/include/suds.h"
#include <ctype.h>
#include <stdio.h>

#define MAXSTR 100

int find_code(str,list)
   char *str;
   struct codes *list;
{
   register i,j;
   char temp[MAXSTR];

   j=strlen(str);
   for(i=0;i<MAXSTR && i<j;i++)temp[i]=isupper(str[i])?str[i]+'a'-'A':str[i];
   temp[MAXSTR-1]='\0';
   j=strlen(temp);
   for(i=0;list[i].meaning!=0;i++)
      if(strncmp(temp,list[i].meaning,j)==0) {
         return(list[i].num);
      }
   return(0);
}

char *list_code(code,list)
   int code;
   struct codes *list;
{
   register i;

   for(i=0;list[i].num!=code && list[i].meaning!=0;i++);
   if(list[i].num==code)return(list[i].meaning);
   return("this code undefined");
}
```

```
/*
 * Initialize list of error messages
 * Bruce R. Julian, USGS Menlo Park Calif., 15 Dec 1983
 * Revision 1.1  86/05/16  16:45:46  julian
 * Initial revision
 * Mod PLWard 4/7/88
 */
static char SccsId[] = "%W%\t%G%";
#include "/usr/suds/include/st_error.h"

char    *errlist[] = {
        "Error 0",
        "Illegal day",
        "Illegal month",
        "Illegal year",
        "Illegal value",
        "Not enough data",
        "Convergence failure",
        "Underflow",
        "No such station",
        "Structure sizes mismatch",
        "Singular matrix",
        "Focal depth too great",
        "No such phase",
        "Constraints are inconsistent",
        "Solution is unbounded",
        "Structure tag sync code wrong",
        "Serious suds I/O error",
        "Can not bracket root",
};
int     st_nerr = { sizeof errlist/sizeof errlist[0] };
int     st_errbase = ERRBASE;

#include <stdio.h>
#include <varargs.h>

char *progname=NULL;
char *st_errout="stderr";
FILE *eout;
int  initout=1;

st_error(va_alist)
   va_dcl
{
   int (*fcn)();
   char *fmt;
   int errnum;
   extern int errno,sys_nerr,nerr;
   extern char *sys_errlist[];
   va_list ap;

   if(initout) {
      if(strcmp(st_errout,"stderr")!=0) {
         if((eout=fopen(st_errout,"w"))==NULL) {
           fprintf(stderr,"st_error: unable to open error out file %s\n",st_errout);
           eout=stderr;
         }
      }
      else eout=stderr;
      initout=0;
   }
   va_start(ap);
   fcn=(int (*)())va_arg(ap,int *);
   errnum=va_arg(ap,int);
   fmt=va_arg(ap,char *);
   fprintf(eout,"\nERROR");
   if(progname) fprintf(eout," in %s",progname);
   fprintf(eout,":\n");
   _doprnt(fmt,ap,eout);
   fprintf(eout,"\n    ");
   if(errno>0 && errno<sys_nerr) fprintf(eout,"   %s\n",sys_errlist[errno]);
   if(errnum!=errno && errnum >=ERRBASE && errnum <= st_nerr+ERRBASE)
      fprintf(eout,"   %s\n",errlist[errnum-ERRBASE]);
   errno=0;
   if(fcn)(*fcn)(errnum);
   va_end(ap);
}
```

```
/* initialize a structure based on table in suds_descr.h
 */
static char SccsId[] = "%W%\t%G%";
#include <stdio.h>
#include "/usr/suds/include/suds.h"

extern struct sudsform stform[];
extern int beg_struct[];
extern double atof();
extern int atoi();

initvalue(i,ptr)
   int i;
   char *ptr;
{
   short *sh;
   long *lg;
   float *ft;
   double *db;
   ST_TIME *st;
   MS_TIME *ms;

   switch(stform[i].ftype) {
      case CHR: *ptr=stform[i].initval[0];
                break;
      case BTS:
      case MIN: *ptr=atoi(stform[i].initval);
                break;
      case STR: strcpy(ptr,stform[i].initval);
                break;
      case SHT: *((short *)ptr)=atoi(stform[i].initval);
                break;
      case LNG: *((long *)ptr)=atoi(stform[i].initval);
                break;
      case FLT: *((float *)ptr)=atof(stform[i].initval);
                break;
      case LLT:
      case DBL: *((double *)ptr)=atof(stform[i].initval);
                break;
      case STT: *((ST_TIME *)ptr)=atoi(stform[i].initval);
                break;
      case MST: *((MS_TIME *)ptr)=atof(stform[i].initval);
                break;
      case CAL:
      case CPX:
      case STI:
      default:
                break;
   }
}

st_init(type,ptr)
   int type;
   char *ptr;
{
   register i;

   char *ptrloc;

   ptrloc=ptr;
   for(i=beg_struct[type];stform[i].fstype==type;i++) {
      if(stform[i].nextfstype!=0) {
         st_init(stform[i].nextfstype,ptrloc+stform[i].offset);
      }
      else initvalue(i,ptrloc+stform[i].offset);
   }
}

st_create(type,ptr,datalength)
   int type;
   char **ptr;
{
   extern char *malloc();

   *ptr=malloc(size_struct[type]+datalength);
   if(*ptr==NULL) {
      st_error(0,0,"Unable to malloc pointer in st_create of length %d",
         size_struct[type]+datalength);
      return;
   }
   st_init(type,*ptr);
}

#ifdef DEBUG

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

main()
{
   struct stationcomp sc;
   FILE *st,*st_open();

   st_init(STATIONCOMP,&sc);
   st=st_open("stdout","w");
   st_put(&sc,STATIONCOMP,sizeof(struct stationcomp),st);
}

#endif DEBUG
```

```
/* st_time: time and date utility routines for suds
   Bruce Julian and PLWard, U. S. Geological Survey, Menlo Park, Ca. 94025
 */
static char SccsId[] = "%W%\t%G%";
#include "/usr/suds/include/suds.h"
#include "/usr/suds/include/st_error.h"
#include <sys/time.h>
#include <math.h>
#include <stdio.h>

#define BASEJDN         2440588                 /* 1 January 1970       */

/* Calendar codes */
#define GREGORIAN       1                       /* Modern calendar      */
#define JULIAN          0                       /* Old-style calendar   */

/* Time intervals */
#define SECOND          1
#define MINUTE          (60*SECOND)
#define HOUR            (60*MINUTE)
#define DAY             (24*HOUR)

typedef int bool;

/* floor(x/y), where x, y>0 are integers, using integer arithmetic */
#define qfloor(x, y)  (x>0 ? (x)/y : -((y-1-(x))/y))

static int      ecom[2][15] = {
   { 0, 31, 59, 90, 120, 151, 181, 212, 243, 273, 304, 334, 365, 396, 424 },
   { 0, 31, 60, 91, 121, 152, 182, 213, 244, 274, 305, 335, 366, 397, 425 },
};

/* Recognize leap years */
bool
isleap(yr, cal)
   int  yr;             /* Year                                 */
   int  cal;            /* Calendar type                        */
{
   bool l;

   if (yr < 0)
      yr++;
   l = (yr%4 == 0);
   if (cal == GREGORIAN)
      l = l && (yr%100 != 0 || yr%400 == 0);
   return l;
}

/* Compute day of year */
int
yrday(mo, day, lp)
   int  mo, day;        /* Month, day                           */
   bool lp;             /* Leap year?                           */
{
   return ecom[lp][mo-1] + day;
}

/* Calculate month, day from day of year */
void
mnday(d, lp, pm, pd)
   int  d;              /* Day of year                          */
   bool lp;             /* Leap year?                           */
   int  *pm, *pd;       /* Month, day                           */
{
   int  i;
   int  *et = ecom[lp];

   for (i=1; d>et[i]; i++)
            ;
   *pm = i;
   *pd = d - et[i-1];
}

/* Compute Julian Day number from calendar date */
long
jdn(yr, mo, day, cal)
   int  yr, mo, day;    /* Year, month, day                     */
   int  cal;            /* Calendar type                        */
{
   long ret;

   if (yr < 0)
      yr++;

   /* Move Jan. & Feb. to end of previous year */
   if (mo <= 2) {
      --yr;
      mo += 12;
   }

   ret=qfloor((long)(4*365+1)*(yr+4712), 4) + ecom[0][mo-1] + day +
      (cal==GREGORIAN ? -qfloor(yr, 100) + qfloor(yr, 400) + 2 : 0);
   return(ret);
}

/* Compute calendar date from Julian Day Number */
void
date(n, py, pm, pd, cal)
   long n;              /* Julian day number                    */
   int  *py, *pm, *pd;  /* Year, month, day                     */
   int  cal;            /* Calendar type                        */
{
   long d, t;
   int  y;

/* Find position within cycles that are nd days long */
#define CYCLE(n, nd) { t=qfloor(d-1,nd); y+=t*n; d-=t*nd; }

/* The same, with bound on cycle number */
#define LCYCLE(n, nd, l) { t=qfloor(d-1,nd); if (t>l) t=l; y+=t*n; d-=t*nd; }

   y = -4799;
   if (cal == GREGORIAN) {
        d = n + 31739;          /* JD -31739 = 31 Dec 4801 B.C. */
      CYCLE(400, 146097)        /* Four-century cycle           */
      LCYCLE(100, 36524, 3)     /* 100-year cycle               */
   }
   else d = n + 31777;          /* JD -31777 = 31 Dec 4801 B.C. */

   CYCLE(4, 1461)               /* Four-year cycle              */
   LCYCLE(1, 365, 3)            /* Yearly cycle                 */

   if (y <= 0) --y;
   *py = y;
   mnday((int)d, isleap(y, cal), pm, pd);
}

extern int errno;

MS_TIME get_mstime()
```

```
{
   struct timeval tp;
   struct timezone tzp;

   if(gettimeofday(&tp,&tzp) == -1) {
      st_error(0,errno,"Unable to read system clock");
      return(0.0);
   }
   return((MS_TIME)tp.tv_sec+((MS_TIME)tp.tv_usec)/1000000.0);
}

MS_TIME make_mstime(year,month,day,hour,min,second)
   int year, month, day, hour, min;
   double second;
{
   int err;
   int  *et = ecm[isleap(year,GREGORIAN)];

   err=0;
   if(year==0) {
      st_error(0,EBADYEAR,"Year is %d",year); err=1;
   }
   if(month==0) mnday(day,isleap(year,GREGORIAN),&month,&day);
   if(month<1 || month>12) {
      st_error(0,EBADMONTH,"Month is %d",month); err=1;
   }
   if(day<1 || day>et[month]-et[month-1]) {
      st_error(0,EBADDAY,"Day is %d",day); err=1;
   }
   if(err) return(0.0);
   return((double) DAY*(jdn(year,month,day,GREGORIAN)-BASEJDN) +
      HOUR*hour + MINUTE*min + SECOND*second);
}

int decode_mstime(time,year,month,day,hour,min,second)
   MS_TIME time;
   int *year,*month,*day,*hour,*min;
   double *second;
{
   long d;

   d = floor(time/DAY);
   date(d+BASEJDN,year,month,day,GREGORIAN);
   time -= d*DAY;
   *hour = time/HOUR;
   time -= (*hour)*HOUR;
   *min = time/MINUTE;
   *second = time - (*min)*MINUTE;
   return(1);
}

#define MAXFORM 7

char *mon[]={"","Jan","Feb","Mar","Apr","May","Jun","Jul","Aug","Sep","Oct",
             "Nov","Dec"};
char outstr[34];

char *list_mstime(time,form)
   MS_TIME time;
   int form;
{
   int year,month,day,hour,min;
   double second;

   if(form<0 || form > MAXFORM) form=0;
   if(decode_mstime(time,&year,&month,&day,&hour,&min,&second)==0) return(0);
   switch(form) {
      case 0: sprintf(outstr,"%02d%02d%02d%02d%02d%06.3f",year-1900,month,day,
                 hour,min,second);
              break;
      case 1: sprintf(outstr,"%02d%02d%02d%02d%02d%02d",year-1900,month,day,
                 hour,min,(int)second);
              break;
      case 2: sprintf(outstr,"%02d %02d %02d% 02d %02d %06.3f",year-1900,month,day,
                 hour,min,second);
              break;
      case 3: sprintf(outstr,"%02d %02d %02d% 02d %02d %02d",year-1900,month,day,
                 hour,min,(int)second);
              break;
      case 4: sprintf(outstr,"%02d/%02d/%02d %02d:%02d %06.3f",month,day,year-1900,
                 hour,min,second);
              break;
      case 5: sprintf(outstr,"%02d/%02d/%02d %02d:%02d %02d",month,day,year-1900,
                 hour,min,(int)second);
              break;
      case 6: sprintf(outstr,"%s %d, %d %02d:%02d %06.3f GMT",mon[month],day,year,
                 hour,min,second);
              break;
      case 7: sprintf(outstr,"%s %d, %d %02d:%02d %02d GMT",mon[month],day,year,
                 hour,min,(int)second);
              break;
   }
   return(outstr);
}

#ifdef DEBUG
main()
{
   double tim,timl,sec;
   int i,year,month,day,hour,min;

   tim=get_mstime();
   printf("system time = %f\n",tim);
   for(i=0;i<9;i++)printf("%s\n",list_mstime(tim,i));
   decode_mstime(tim,&year,&month,&day,&hour,&min,&sec);
   timl=make_mstime(year,month,day,hour,min,sec);
   printf("remade time is %f which differs by %f\n",timl,timl-tim);
   day=yrday(month,day,isleap(year,GREGORIAN));
   printf("day of year is %d\n",day);
   timl=make_mstime(year,0,day,hour,min,sec);
   printf("remade time is %f which differs by %f\n",timl,timl-tim);
}
#endif DEBUG
```

```
/* BASIC SUDS INPUT AND OUTPUT ROUTINES
   These routines are meant to be used like fopen, fread, fwrite, fflush, fclose,
   fseek, ftell, and rewind except that basic error handling is done for you.

   WARNING: Do not mix these routines with the standard I/O routines for the
   same files at the same time. While these routines use the standard I/O
   library, you may mess up these routines by interspersing the standard
   I/O calls. First close with st_close and then open the same file with
   fopen or close with fclose and then open with st_open.

   Error handling uses st_error. Errors that should cause termination call die,
   a user supplied program that might simply call exit but might also
   reset terminal characteristics, clear buffers, etc.
               die(n) int n; {exit(n);}
   To get a warning only, but continue, make die not call exit. To allow for
   this option, calls to st_error in this file are followed by return(ST_ERR).

   PLWard, U S Geological Survey, Menlo Park, Ca. 4/8/88
 */
static char SccsId[] = "%W%\t%G%";
#include <stdio.h>
#include <string.h>
#include <sys/types.h>
#include <sys/stat.h>

#include "/usr/suds/include/suds.h"
#include "/usr/suds/include/st_error.h"
#include "/usr/suds/include/suds_codes.h"
#include "/usr/suds/include/suds_descr.h"

#define ST_EOF   (-999)          /* internal code for EOF kept in next.id_struct */
#define ST_NOTAG (-843)             /* internal code for initialization of stdin */

char *malloc(),*realloc();
extern int errno,die();

/* streamtags structure: one instance is created for each open file, even those
   not opened by this software package
*/
struct streamtags {
   struct structtag this;                          /* structtag for data read */
   struct structtag next;                       /* structtag for data to be read */
   char *st_ptr;                                 /* last data put back on stream */
   int num_struct;                          /* number of structures read or written*/
   int st_mode;                        /* 0=closed, 1=open to read, 2=open to write */
   char *filename;                                                  /* filename */
} *avail;
int s_fdsize=0;

/* st_first: read first strucktag in a stream. Should be not called by user.
 */
int st_first(stream,fd)
   FILE *stream;
   int fd;
{
   int ret;

   ret=fread((char *)&avail[fd].next,1,sizeof(struct strucktag),stream);
   if(ret!=sizeof(struct strucktag)) {
      if(ret==0 && errno==0) return(0);
      st_error(die,errno,"st_get: read strucktag returns %d",ret);
      return(ST_ERR);
   }
   if(aveil[fd].next.sync!=ST_MAGIC)
      st_error(die,EBADSYNC,"st_get: initial strucktag.sync=%d",
         avail[fd].next.sync);
   if(avail[fd].next.machine!=MACHINE)
      st_error(0,0,"st_get: machine type %c is different from current type %c",
         avail[fd].next.machine,MACHINE);
   return(ST_ERR);
}

/* st_initit: open stdio. Should not be called by user.
 */
char *st_names[]={"stdin","stdout","stderr"};

int st_initit()
{
   int i,fd;

   if(s_fdsize>=3) return(0);
   avail=(struct streamtags *)malloc(3*sizeof(struct streamtags));
   if(avail==NULL) {
      st_error(die,errno,"st_get: malloc of streamtags for stdio");
      return(ST_ERR);
   }
   for(i=0;i<3;i++) {
      avail[i].this.id_struct=0;
      avail[i].next.id_struct=i>0?0:ST_NOTAG;
      avail[i].num_struct=0;
      avail[i].st_ptr=NULL;
      avail[i].filename=st_names[i];
      avail[i].st_mode=i>0?2:1;
   }
   s_fdsize=3;
}

/* efopen: fopen with error handling
 */
FILE *efopan(filename,type)
   char *filename, *type;
{
   int fd;
   FILE *input;

   errno=0;
   if(strcmp(filename,"stdin")==0) {
      if(type[0]=='r') return(stdin);
      else st_error(die,errno,"cannot fopen stdin with type=%s",type);
      return(NULL);
   }
   if(strcmp(filename,"stdout")==0) {
      if(type[0]=='w' || type[0]=='a') return(stdout);
      else st_error(die,errno,"cannot fopen stdout with type=%s",type);
      return(NULL);
   }
   if(strcmp(filename,"stderr")==0) {
      if(type[0]=='w' || type[0]=='a') return(stderr);
      else st_error(die,errno,"cannot fopen stderr with type=%s",type);
      return(NULL);
   }
   if((input=fopan(filename,type))==NULL) {
      st_error(die,errno,"cannot fopen file %s with type=%s",filename,type);
      return(NULL);
   }
   return(input);
}

/* st_open: open a suds file for reading or writing structures
 */
```

```
FILE *st_open(filename,type)
   char *filename, *type;
{
   int fd;
   FILE *input;

   if(s_fdsize<3)st_initit();
   if((input=efopen(filename,type))==NULL) return(NULL);
   fd=fileno(input);
   if(fd>=s_fdsize) {                    /* increase allocation of avail */
      free(avail);
      avail=(struct streamtags *)realloc(avail,(fd+1)*sizeof(struct streamtags));
      if(avail==NULL) {
         st_error(die,errno,"st_get: realloc of streamtags");
         return(NULL);
      }
      s_fdsize=fd+1;
   }
   avail[fd].this.id_struct=0;
   avail[fd].next.id_struct=0;
   avail[fd].num_struct=0;
   avail[fd].st_ptr=NULL;
   avail[fd].filename=filename;
   switch (type[0]) {
      case 'r': avail[fd].st_mode=1; st_first(input,fd); break;
      case 'w':
      case 'a': avail[fd].st_mode=2; break;
      default: st_error(die,0,"st_open: unrecognized file type %s",type);
               avail[fd].st_mode=0;
               return(NULL);
   }
   return(input);
}

/* efclose: close with error handling
 */
int efclose(stream)
   FILE *stream;
{
   int ret;

   if((ret=fclose(stream))!=0) st_error(0,errno,"fclose returns %d",ret);
   return(ret);
}

/* st_close: close a suds file
 */
int st_close(stream)
   FILE *stream;
{
   int ret,fd;

   fd=fileno(stream);
   if(fd<0 || fd>=s_fdsize) {
      st_error(0,0,"st_tell: illegal file descriptor",fd);
      return(ST_ERR);
   }
   avail[fd].this.id_struct=0;
   avail[fd].next.id_struct=0;
   avail[fd].num_struct=0;
   avail[fd].st_ptr=NULL;
   avail[fd].filename="";
   avail[fd].st_mode=0;
   return(efclose(stream));
}

/* st_flush: flush the structure stream
 */
int st_flush(stream)
   FILE *stream;
{
   int ret,fd;

   fd=fileno(stream);
   if(fd<0 || fd>=s_fdsize) {
      st_error(0,0,"st_flush: illegal file descriptor",fd);
      return(ST_ERR);
   }
   if(avail[fd].st_mode!=2) {
      st_error(0,0,"st_flush: file %s not open for writing",fd);
      return(EOF);
   }
   if((ret=fflush(stream))!=0)
      st_error(0,errno,"st_flush: fflush of file %s returns %d",
        avail[fd].filename,ret);
   return(ret);
}

/* st_unget: put one structure back on the stream
 */
int st_unget(st_ptr,stream)
   char *st_ptr;        /* pointer to structure                              */
   FILE *stream;        /* input stream from fopen()                         */
{
   int fd;

   fd=fileno(stream);
   if(fd<0 || fd>=s_fdsize) {
      st_error(0,0,"st_unget: illegal file descriptor",fd);
      return(ST_ERR);
   }
   if(avail[fd].st_mode!=1) {
      st_error(0,0,"st_unget: file %s not open for reading",avail[fd].filename);
      return(EOF);
   }
   if(avail[fd].num_struct==0) {
      st_error(0,errno,"st_unget: cannot unget at beginning of file %s",
        avail[fd].filename);
      return(EOF);
   }
   if(avail[fd].st_ptr!=NULL) {
      st_error(0,errno,"st_unget: can only unget one structure for file %s",
        avail[fd].filename);
      return(EOF);
   }
   avail[fd].st_ptr=st_ptr;
   avail[fd].num_struct--1;
   return(0);
}

/* st_tell: find the number of the structure that is about to be read counting
            the first structure as 0.
 */
long st_tell(stream)
   FILE *stream;
{
   int fd;

   if(s_fdsize<3)st_initit();
   fd=fileno(stream);
```

```
   if(fd<0 || fd>=s_fdsize) {
      st_error(0,0,"st_tell: illegal file descriptor",fd);
      return(ST_ERR);
   }
   if(avail[fd].st_mode==0) {
      st_error(0,0,"st_tell: file %s is not open",avail[fd].filename);
      return(ST_ERR);
   }
   return(avail[fd].num_struct);

}

/* st_rewind: return to the beginning of the file. Can not be used on stdin, stdout,
              stderr, or pipes.
 */
int st_rewind(stream)
   FILE *stream;
{
   int fd;

   fd=fileno(stream);
   if(fd<0 || fd>=s_fdsize) {
      st_error(0,0,"st_tell: illegal file descriptor",fd);
      return(ST_ERR);
   }
   if(fd<3) {
      st_error(0,0,"st_rewind: you can not rewind %s",avail[fd].filename);
      return(ST_ERR);
   }
   if(avail[fd].st_mode==0) {
      st_error(0,0,"st_rewind: file %s is not open",avail[fd].filename);
      return(ST_ERR);
   }
   avail[fd].this.id_struct=0;
   avail[fd].next.id_struct=0;
   avail[fd].num_struct=0;
   avail[fd].st_ptr=NULL;
   rewind(stream,0L,0);
   st_first(stream,fd);
   return(0);
}

/* st_seek: position to read the nth structure in a file counting the first as 0.
            Can not be used on pipes.
 */
int st_seek(stream,offset,ptrname)
   FILE *stream;
   long offset;
   int ptrname;
{
   int fd,numwant,typ,len,inp,i;
   char *ptr;

   fd=fileno(stream);
   if(fd<0 || fd>=s_fdsize) {
      st_error(0,0,"st_tell: illegal file descriptor",fd);
      return(ST_ERR);
   }
   if(s_fdsize<3)st_initit();
   if(avail[fd].next.id_struct==ST_NOTAG) st_first(stdin,0);
   switch(ptrname) {
      case 0: numwant=offset; break;
      case 1: numwant=offset+st_tell(stream); break;
      case 2: if(fd<3) {
                 st_error(0,0,"st_seek: you can not seek from end of %s",
                    avail[fd].filename);
                 return(ST_ERR);
              }
              st_rewind(stream);
              for(numwant=0;(inp=st_get(&ptr,&typ,&len,stream))!=EOF;numwant++)
                 st_free(ptr,inp);
              st_rewind(stream);
              if(offset<0)offset= -offset;
              numwant=numwant-offset;
              break;
      default: st_error(die,0,"st_seek: ptrname must be 0, 1, or 2, not %d",
                 ptrname);
              return(ST_ERR);
   }
   if(numwant<avail[fd].num_struct && fd<3) {
      st_error(0,0,"st_seek: you can not seek backwards on %s",avail[fd].filename);
      return(ST_ERR);
   }
   if(numwant==avail[fd].num_struct) return(0);
   if(numwant< avail[fd].num_struct) st_rewind(stream);
   while(numwant>st_tell(stream)) {
      inp=st_get(&ptr,&typ,&len,stream);
      st_free(ptr,inp);
   }
   return(0);
}

/* st_peek: what is the id number of the next structure to be read
 */
int st_peek(stream)
   FILE *stream;
{
   int fd;

   if(s_fdsize<3)st_initit();
   fd=fileno(stream);
   if(fd<0 || fd>=s_fdsize) {
      st_error(0,0,"st_tell: illegal file descriptor",fd);
      return(ST_ERR);
   }
   if(avail[fd].st_mode!=1) {
      st_error(0,0,"st_tell: file %s is not open for reading",avail[fd].filename);
      return(ST_ERR);
   }
   if(avail[fd].next.id_struct==ST_NOTAG) st_first(stdin,0);
   if(avail[fd].st_ptr!=NULL) return((int)avail[fd].this.id_struct);
   return((int)avail[fd].next.id_struct);
}

/* st_get: get the next structtag and structure from opened stream. Returns
   through arguments the pointers to the structure type and the structure.
   Return value is
                if > 0  length of the structure
                if ==0  end of file
                if < 0  error

   This routine uses malloc(3) to allocate space for each new structure.
   The user must use st_free(3) to deallocate the space.

   This routine reads the structtag for a file to identify the structure.
   It then reads the structure and the next structtag to be sure the structure
   ends properly. This latter structtag is kept in the array structure avail,
   with one element for each open stream. When an EOF is encountered,
   the structtag entry is modified so that on the next call, the EOF is returned
   to the caller.
```

```
 */
int st_get(st_ptr,st_typ,st_len,stream)
   char **st_ptr;       /* pointer to structure returned. Space malloc'd      */
   int  *st_typ;        /* structure id returned                              */
   int  *st_len;        /* structure length                                   */
   FILE *stream;        /* input stream from fopen()                          */
{
   int ret,i,fd,numin,numget;
   char *temp,*tempp;
   struct stat stt;
   struct muxdata *mx;

   *st_typ=0;
   *st_len=0;
   if(s_fdsize<3)st_initit();
   fd=fileno(stream);
   if(fd<0 || fd>=s_fdsize) {
      st_error(0,0,"st_get: illegal file descriptor",fd);
      return(ST_ERR);
   }
   if(avail[fd].st_mode!=1) {
      st_error(0,0,"st_get: file %s is not open for reading. Mode=%d",
         avail[fd].filename,avail[fd].st_mode);
      return(ST_ERR);
   }
   if(avail[fd].st_ptr!=NULL) { /* read structtag put back by st_unget */
      *st_typ=avail[fd].this.id_struct;
      *st_len=avail[fd].this.len_struct;
      *st_ptr=avail[fd].st_ptr;
      avail[fd].st_ptr=NULL;
      return(avail[fd].this.len_struct+avail[fd].this.len_data);
   }
   if(avail[fd].next.id_struct==ST_NOTAG) st_first(stdin,0);
   if(avail[fd].next.id_struct==ST_EOF) return(EOF);

   avail[fd].this.sync=avail[fd].next.sync;
   avail[fd].this.machine=avail[fd].next.machine;
   avail[fd].this.id_struct=avail[fd].next.id_struct;
   avail[fd].this.len_struct=avail[fd].next.len_struct;
   avail[fd].this.len_data=avail[fd].next.len_data;

   /* If reading MUXDATA structure and len_data not set (=0), assume data goes to
      end of this file, and add that length to the structure
   */
   if(avail[fd].this.id_struct==MUXDATA && avail[fd].this.len_data==0) {
      if(fstat(fd,&stt)== -1) st_error(0,0,"Cannot get length of input file %d",fd);
      avail[fd].this.len_data=stt.st_size-ftell(stream)-avail[fd].this.len_struct;
   }
   numget=avail[fd].this.len_struct+avail[fd].this.len_data;
   temp=malloc(numget);                  /* malloc space for new struct  */
   if(temp==NULL) {
      st_error(die,errno,"st_get: malloc fails for struct %d with length %d",
         avail[fd].this.id_struct,numget);
      return(ST_ERR);
   }
   numin=fread(temp,1,numget,stream);                    /* read struct */
   if(numin!=numget) {
      st_error(die,errno,"st_get: reading struct id=%d returns %d expect %d",
         avail[fd].this.id_struct,numin,numget);
      return(ST_ERR);
   }
   if(avail[fd].this.id_struct==MUXDATA) {
      tempp=temp;
      mx=(struct muxdata *)tempp;
      switch(mx->typedata) {
         case 'r':
         case 's': i=2; break;
         case 'l':
         case 'f': i=4; break;
         case 'v':
         case 'c':
         case 'd': i=8; break;
         case 't': i=12; break;
         default : st_error(0,0,"Unknown data type %c in structure type %d",
                      mx->typedata,MUXDATA);
      }
      mx->numsamps=avail[fd].this.len_data/i;
   }
                                                        /* read next structtag */
   ret=fread((char *)&avail[fd].next,1,sizeof(struct structtag),stream);
   if(ret!=sizeof(struct structtag)) {
      if(ret==0 && errno==0) avail[fd].next.id_struct=ST_EOF;
      else {
         st_error(die,errno,"st_get: read structtag returns %d",ret);
         return(ST_ERR);
      }
   }
   if(avail[fd].next.sync!=ST_MAGIC) {
      st_error(die,EBADSYNC,"st_get: structtag.sync=%d",avail[fd].next.sync);
      return(ST_ERR);
   }
   if(avail[fd].next.machine!=MACHINE)
      st_error(0,0,"st_get: machine type %c is different from current type %c",
         avail[fd].next.machine,MACHINE);
   avail[fd].num_struct+=1;
   *st_typ=avail[fd].this.id_struct;
   *st_len=avail[fd].this.len_struct;
   *st_ptr=temp;
   return(numin);
}

/* st_put: write out structtag and structure
 */
int st_put(st_ptr,st_typ,st_len,stream)
   char *st_ptr;        /* pointer to structure                               */
   int  st_typ;         /* structure id                                       */
   int  st_len;         /* structure length                                   */
   FILE *stream;        /* output stream from fopen()                         */

{
   int ret1,ret2,fd;
   struct structtag temp;

   if(s_fdsize<3)st_initit();
   fd=fileno(stream);
   if(fd<0 || fd>=s_fdsize) {
      st_error(0,0,"st_put: illegal file descriptor",fd);
      return(ST_ERR);
   }
   if(avail[fd].st_mode!=2) {
      st_error(0,0,"st_put: file %s is not open for writing. Mode=%d",
         avail[fd].filename,avail[fd].st_mode);
      return(ST_ERR);
   }
   temp.sync=ST_MAGIC;
   temp.machine=MACHINE;
   temp.id_struct=st_typ;
   temp.len_struct=size_struct[st_typ];
   temp.len_data=st_len-size_struct[st_typ];
   ret2=sizeof(struct structtag);
```

```
   if((ret1=fwrite(&temp,1,ret2,stream))!=ret2)
      st_error(0,errno,"st_put: tag: wrote only %d out of %d",ret1,ret2);
   if((ret2=fwrite(st_ptr,1,st_len,stream))!=st_len)
      st_error(0,errno,"st_put: data: wrote only %d out of %d",ret2,temp.len_data);
   avail[fd].num_struct+=1;
   return(ret1+ret2);
}

/* st_free: free a block of memory malloc'd in st_get
   While this routine simply calls free, it is important to use it to
   allow for future code that might optimize the malloc and realloc calls
   for massive but predictable utilization.
 */
st_free(ptr,length)
   char *ptr;
   int length;
{
   return(free(ptr));
}
```

```
SCCSID = "@(#)Makefile  1.2 3/22/89"
CFLAGS = -g -f68881

INC =

SRC = stedit.c stdescribe.c st2asc.c asc2st.c ah2st.c

all: ah2st asc2st st2asc stdescribe stedit

ah2st: ah2st.o
	$(CC) $(CFLAGS) -o ah2st ah2st.o -lsuds

asc2st: asc2st.o
	$(CC) $(CFLAGS) -o asc2st asc2st.o -lsuds

st2asc: st2asc.o
	$(CC) $(CFLAGS) -o st2asc st2asc.o -lsuds

st2ah: st2ah.o
	$(CC) $(CFLAGS) -o st2ah st2ah.o -lsuds

stdescribe: stdescribe.o
	$(CC) $(CFLAGS) -o stdescribe stdescribe.o -lsuds

stedit: stedit.o
	$(CC) $(CFLAGS) -o stedit stedit.o -lsuds  -lcurses -ltermcap

edit:
	sccs edit $(SRC)

install:
	cp ah2st /usr/suds/bin
	cp asc2st /usr/suds/bin
	cp st2asc /usr/suds/bin
	cp stdescribe /usr/suds/bin
	cp stedit /usr/suds/bin

list:
	enscriptW $(SRC) $(INC) Makefile

ah2st.o asc2st.o st2asc.o stdescribe.o stedit.o : /usr/suds/include/suds.h
st2asc.o                                        : /usr/suds/include/st_error.h
```

```
static char SccsId[] = "@(#)ah2st.c     1.2 11/17/88";
#include "/usr/suds/include/suds.h"
#include <time.h>
#include <stdio.h>
#include <string.h>

#define MAXCOM  300
#define AHHEADSIZE 1024
#define TYPEMIN   1
#define TYPEMAX   6
#define LOGSIZE 202
#define NEXTRAS  21
#define NOCALPTS 30

struct   time   {
   short     yr;          /* year    */
   short     mo;          /* month   */
   short     day;         /* day     */
   short     hr;          /* hour    */
   short     mn;          /* minute  */
   float     sec;         /* second  */
   };

struct   station_info   {
   char      code[6];     /* station code                                    */
   char      chan[6];     /* lpz,spn, etc. last char either z, n, or e       */
   char      stype[8];    /* wwssn,hglp,rstn,sro,asro,dwwssn etc.            */
   float     slat;        /* station latitude                                */
   float     slon;        /*    "    longitude                               */
   float     elev;        /*    "    elevation                               */
   float     DS;          /* maximum gain at peak of calibration curve       */
   float     A0;          /* normalization, factor to make calibration curve at
                             peak equal 1 for a specific frequency           */
   struct    calib  cal[NOCALPTS]; /* calibration curve                      */
   };

struct   event_info   {
   float    lat;          /* event latitude    */
   float    lon;          /*   "   longitude   */
   float    dep;          /*   "   depth       */
   struct   time   ot;    /*   "   origin time */
   char     ecomment[80]; /* comment line      */
   };

struct   record_info   {
   short   type;          /* data type float=1, complex=2, etc as defined above */
   long    ndata;         /* number of samples            */
   float   delta;         /* sampling interval            */
   float   maxamp;        /* maximum amplitude of record  */
   struct  time  abstime; /* start time of record section */
   float   rmin;          /* minimum value of abscissa    */
   char    rcomment[80];  /* comment line                 */
   char    log[LOGSIZE];  /* log of data manipulations    */
   };

typedef struct {
   struct   station_info   station;  /* station info */
   struct   event_info   event;      /* event info   */
   struct   record_info   record;    /* record info  */
   float    extra[NEXTRAS];          /* freebies     */
   } ahhed;

extern int errno,die();
extern double make_mstime();
```

```
ah2st(infile,outfile,network,makeint,clipval,factor)
   char *infile,*outfile,*network;
   int makeint;
   float clipval,factor;
{
   register i;
   ahhed ah;
   struct stationcomp sc;
   struct descriptrace *dt;
   struct calibration ca;
   struct origin or;
   struct tm t;
   int    ret,size,siz,yr;
   char   temp[20],dattype;
   short  *out;
   float  *in;
   FILE   *inf,*outf,*efopen(),*st_open();
   char   com[MAXCOM];

   inf= efopen(infile,"r");
   outf=st_open(outfile,"w");
   while((ret=fread(&ah,1,AHHEADSIZE,inf))==AHHEADSIZE) {
      st_init(STATIONCOMP,&sc);
      sc.clip_value=clipval;
      if(network[0]!='\0') {
         strncpy(sc.sc_name.network,network,4);
         sc.sc_name.network[3]='\0';
      }
      strncpy(sc.sc_name.st_name,ah.station.code,5); sc.sc_name.st_name[4]='\0';
      temp[0]=ah.station.chan[0]; temp[1]=ah.station.chan[1]; /* pickup sp or lp */
      strcat(temp,ah.station.stype);
      sc.sc_name.inst_type=(short)find_code(temp,inst_type);
      ret=strlen(ah.station.chan);
      if(ret>=1) switch (ah.station.chan[ret-1]) {
         case 'v':
         case 'Z':
         case 'z':
         case 'V': sc.sc_name.component='v';
                   sc.azim=0;
                   sc.incid=0;
                   break;
         case 'e':
         case 'E': sc.sc_name.component='e';
                   sc.azim=90;
                   sc.incid=90;
                   break;
         case 'n':
         case 'N': sc.sc_name.component='n';
                   sc.azim=0;
                   sc.incid=90;
                   break;
      }
      sc.st_lat=ah.station.slat;
      sc.st_long=ah.station.slon;
      sc.elev=ah.station.elev;

      switch(ah.record.type) {
         case 1: dattype='f'; size=4; break;
         case 2: dattype='c'; size=8; break;
         case 3: dattype='v'; size=8; break;
         case 4: dattype='t'; size=12; break;
         case 6: dattype='d'; size=8; break;
         default: dattype='f'; size=4; break;
      }
      size=size*ah.record.ndata;
```

```
    dt=(struct descriptrace *)malloc(size+sizeof(struct descriptrace));
    if(dt==NULL)st_error(die,errno,"Unable to malloc struct descriptrace");
    st_init(DESCRIPTRACE,dt);
    dt->datatype=dattype;
    for(i=0;i<sizeof(STATIDENT);i++) dt->dt_name.network[i]=sc.sc_name.network[i];
    yr=ah.record.abstime.yr;
    if(yr<1900)yr+=1900;
    dt->begintime=make_mstime(yr,ah.record.abstime.mo,
       ah.record.abstime.day,ah.record.abstime.hr,
       ah.record.abstime.mn,(double)ah.record.abstime.sec);
    dt->length=ah.record.ndata;
    dt->rate= 1.0/ah.record.delta;
    fread(dt+1,1,size,inf);
    if(makeint && dt->datatype=='f') {
       in=(float *)(dt+1);
       out=(short *)in;
       for(i=0;i<dt->length;i++)out[i]=in[i]*factor;
       dt->datatype='s';
       size=2*dt->length;
    }
    sc.data_type=dt->datatype;
    descr_trace(&sc,dt,dt+1);
    st_put(&sc,STATIONCOMP,sizeof(struct stationcomp),outf);

/*  ah.record.rmin    not used        */

    st_put(dt,DESCRIPTRACE,size+sizeof(struct descriptrace),outf);
    if(strlen(ah.record.rcomment)>0) {
       siz=make_comment(com,DESCRIPTRACE,0,ah.record.rcomment);
       st_put(com,COMMENT,siz,outf);
    }
    if(strlen(ah.record.log)>0) {
       siz=make_comment(com,DESCRIPTRACE,0,ah.record.log);
       st_put(com,COMMENT,siz,outf);
    }
    if(ah.event.lat!=0.0 && ah.event.lon!=0.0 &&  ah.event.dep!=0.0) {
       st_init(ORIGIN,&or);
       or.or_lat=ah.event.lat;
       or.or_long=ah.event.lon;
       or.depth=ah.event.dep;
       yr=ah.event.ot.yr;
       if(yr<1900)yr+=1900;
       or.orgtime=make_mstime(yr,ah.event.ot.mo,ah.event.ot.day,
          ah.event.ot.hr,ah.event.ot.mn,(double)ah.event.ot.sec);
       st_put(&or,ORIGIN,sizeof(struct origin),outf);
       if(strlen(ah.event.ecomment)>0) {
          siz=make_comment(com,ORIGIN,0,ah.event.ecomment);
          st_put(com,COMMENT,siz,outf);
       }
    }
    if(ah.station.DS!=0.0 && ah.station.A0!=0.0) {
       st_init(CALIBRATION,&ca);
       for(i=0;i<sizeof(STATIDENT);i++) ca.ca_name.network[i]=sc.sc_name.network[i];
       ca.maxgain=ah.station.DS;
       ca.normaliz=ah.station.A0;
       for(i=0;i<NOCALPTS;i++) {
          ca.cal[i].pole.cr=ah.station.cal[i].pole.cr;
          ca.cal[i].pole.ci=ah.station.cal[i].pole.ci;
          ca.cal[i].zero.cr=ah.station.cal[i].zero.cr;
          ca.cal[i].zero.ci=ah.station.cal[i].zero.ci;
       }
       st_put(&ca,CALIBRATION,sizeof(struct calib),outf);
    }
    free(dt);
}

int make_comment(str,refer,item,comment)
   int  refer,item;
   char *str,*comment;
{
   struct comment *cm;

   cm=(struct comment *)str;
   cm->refer=refer;
   cm->item=item;
   cm->length=strlen(comment);
   cm->unused=atoi(NODATA);
   strncat(cm+1,comment,cm->length);
   return(sizeof(struct comment)+cm->length);
}

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

main(argc,argv)
   int argc;
   char **argv;
{
   register i;
   int numfil,makeint;
   char *network,*ofile;
   extern char *progname;
   float clipval,factor;

   progname=argv[0];
   numfil=0;
   ofile="stdout";
   network="";
   makeint=0;
   clipval=0.0;
   factor=1.0;
   for(i=1;i<argc;i++){
      if(argv[i][0]=='-') {
         switch(argv[i][1]) {
            case 'c': if(argv[i][2]!='\0') clipval= atof(&argv[i][2]);
                      else clipval=atof(argv[++i]);
                      break;
            case 'n': if(argv[i][2]!='\0') network= &argv[i][2];
                      else network=argv[++i];
                      break;
            case 'o': if(argv[i][2]!='\0') ofile= &argv[i][2];
                      else ofile=argv[++i];
                      break;
            case 's': makeint=1;
                      if(argv[i][2]!='\0') factor= atof(&argv[i][2]);
                      else factor=atof(argv[++i]);
                      break;
            default: st_error(0,0,"Unrecognized argument %s",argv[i]);
         }
      }
      else {
         ah2st(argv[i],ofile,network,makeint,clipval,factor);
         numfil++;
      }
   }
   if(numfil==0) ah2st("stdin",ofile,network,makeint,clipval,factor);
}
```

```
/* asc2st: convert an ascii file to suds format
   PLWard, U.S. Geological Survey, Menlo Park, Ca 94025
 */
static char SccsId[] = "@(#)asc2st.c    1.2 11/17/88";
#define MAXIN 256                       /* maximum length of input line */
#define DATATYPE ptr[22]        /* character representing datatype is offset
                                   22 bytes from beginning of MUXDATA and
                                   DESCRIPTRACE                          */
#include "/usr/suds/include/suds.h"
extern int stform_len;
extern SUDS_FORM stform[];

int ind=0;              /* index of character on input line            */
int babble;

char *next_field(stream,end)
   FILE *stream;
   int *end;
{
   register j;
   int isstr,ischr;
   char c;
   char buf[MAXIN];     /* buffer for actiive field in instr           */

   isstr=ischr=j=0;
   while(1) {
      switch (c=fgetc(stream)) {
         case '\n': break;
         case EOF: if(j!=0) goto out; *end=1; return(NULL);
         case '\t':
         case ' ': if(j==0) break;
         case ',': if(ischr) { fgetc(stream); goto out;}
                   if(isstr) { buf[j++]=c; break;}
                   goto out;
         case '\"':if(isstr) { fgetc(stream); goto out; } isstr=1; break;
         case '\'':if(ischr) { fgetc(stream); goto out; } ischr=1; break;
         default:  buf[j++]=c;
      }
   }
out:
   buf[j]='\0';
   return(buf);
}

int getfields(ptr,type,slen,inf)
   char *ptr;
   int type,slen;
   FILE *inf;
{
   register i;
   char     *field;
   int      end;

   end=0;
   for(i=beg_struct[type];stform[i].fstype==type && stform[i].offset <slen;i++) {
      if(stform[i].nextfstype!=0)
         getfields(ptr+stform[i].offset,stform[i].nextfstype,slen-stform[i].offset,
            inf);
      else {
         field=next_field(inf,&end);
         if(babble)
            fprintf(stderr,"Input(at %5d bytes): '%s'\n",stform[i].offset,field);
         if(end) st_error(die,0,"improper end to structure %d",type);
         if(strcmp(field,"NODATA")==0) field=NODATA;
         if(strcmp(field,"NOLIST")==0) field=NOLIST;
         asc2field(field,ptr+stform[i].offset,stform[i].ftype);
      }
   }
}

asc2st(infile,outf,cfile,sepstr,verbose)
   char *infile,*cfile,*sepstr;
   int verbose;
   FILE *outf;
{
   register i;
   FILE   *inf;         /* input stream                                  */
   int    type;         /* type of structure being read in               */
   int    slen;         /* length of structure being read in             */
   int    dlen;         /* length of data following this structure,bytes*/
   int    datlen;       /* length of data following this structure,words*/
   char   *ptr;         /* pointer to structure                          */
   char   *field;
   int    end;
   extern int beg_struct[];
   short  *sh;
   long   *lg;
   float  *ft;
   double *db;

   end=0;
   inf=efopen(infile,"r");

   while((field=next_field(inf,&end))!=NULL) {
      type=atoi(field);
      if(type<0 || type>TOTAL_STRUCTS)
         st_error(die,0,"illegal structure type %d",type);
      slen=atoi(next_field(inf,&end));
      dlen=atoi(next_field(inf,&end));
      if(babble)
         fprintf(stderr,"Struct %d length=%d data length=%d\n",type,slen,dlen);
      ptr=(char *)malloc(slen+dlen);
      if(ptr==NULL)st_error(die,0,"Unable to malloc for structure %d length=%d",
         type,slen+dlen);
      getfields(ptr,type,slen,inf);
      if(dlen) {
         if(type!=MUXDATA && type!=DESCRIPTRACE)
            st_error(die,0,"data should not follow structure type %d",type);
         switch(DATATYPE) {
            case 's': datlen=dlen/2; sh=(short *)(ptr+slen); break;
            case 'l': datlen=dlen/4; lg=(long *)(ptr+slen); break;
            case 'c':
            case 'v':
            case 't':
            case 'f': datlen=dlen/4; ft=(float *)(ptr+slen); break;
            case 'd': datlen=dlen/8; db=(double *)(ptr+slen); break;
            default:  st_error(die,0,"unknown data type %c after structure type %d",
                        DATATYPE,type);
         }
         for(i=0;i<datlen;i++) {
            switch(DATATYPE) {
               case 's': *(sh++)=(short)atoi(next_field(inf,&end)); break;
               case 'l': *(lg++)=(long)atoi(next_field(inf,&end)); break;
               case 'c':
               case 'v':
               case 't':
               case 'f': *(ft++)=(float)atof(next_field(inf,&end)); break;
               case 'd': *(db++)=(double)atof(next_field(inf,&end)); break;
            }
         }
```

```
        }
        st_put(ptr,type,slen+dlen,outf);
        free(ptr);
    }
}

extern char *progname;

die(in)
    int in;
{
    exit(in);
}

main(argc,argv)
    int argc;
    char **argv;
{
    register i;
    int verbose,numfil,outcl;
    char *sepstr,*ofile,*cfile;
    FILE *outf;

    progname=argv[0];
    numfil=0;
    verbose=0;
    babble=0;
    outcl=1;
    sepstr=" ";
    ofile="stdout";
    for(i=1;i<argc;i++){
        if(argv[i][0]=='-') {     /* c,s,v for future expansion */
            switch(argv[i][1]) {
/*              case 'c': if(argv[i][2]!='\0') cfile= &argv[i][2];
                          else cfile=argv[++i];
                          break;*/
                case 'o': if(argv[i][2]!='\0') ofile= &argv[i][2];
                          else ofile=argv[++i];
                          break;
/*              case 's': if(argv[i][2]!='\0') sepstr= &argv[i][2];
                          else sepstr=argv[++i];
                          break;*/
/*              case 'v': verbose=1;
                          break;*/
                case 'V': babble=1;
                default: st_error(0,0,"Unrecognized argument %s",argv[i]);
            }
        }
        else {
            if(outcl) {
                outf=st_open(ofile,"w");
                outcl=0;
            }
            asc2st(argv[i],outf,cfile,sepstr,verbose);
            numfil++;
        }
    }
    if(numfil==0) {
        outf=st_open(ofile,"w");
        asc2st("stdin",outf,cfile,sepstr,verbose);
    }
    st_close(outf);
}
```

```
/* st2ah: convert data in st (SUDS) format to ah (Lamont) format */
/* Event information, if present, must preceed traces
   Assumes input file name begins with st.   and then writes output file to ah.
   Writes data out as FLOAT, inputs as long, short, float, etc.
   PLWard 3/22/89
 */

#include <stdio.h>
FILE *infile,*outfile;

#include <rpc/rpc.h>
#include "/usr/suds/include/ahheader.h"
#include "/usr/suds/include/suds.h"
ahhed lam;

die(n) int n; {exit(n);}
extern char *progname;

main(argc,argv)
   int argc;
   char **argv;
{
   int i,st_type,st_len,tr_len,leng;
   char *ch,*st_ptr,outname[50];
   float ftrace;
   double sec;
   struct stationcomp *stcp;
   struct descriptrace *dstr;
   struct origin *org;
   short *sh;
   int *in;
   float *fl;
   int yer,mon,day,our,min;

   progname=argv[0];

   /* zero unused parts of ah structure */
   lam.event.lat=0.;
   lam.event.lon=0.;
   lam.event.dep=0.;
   lam.event.ot.yr=0;
   lam.event.ot.mo=0;
   lam.event.ot.day=0;
   lam.event.ot.hr=0;
   lam.event.ot.mn=0;
   lam.event.ot.sec=0.0;
   lam.event.ecomment[0]='\0';

   lam.station.A0=0.0;
   for(i=0;i<NOCALPTS;i++) {
      lam.station.cal[i].pole.r=0.0;
      lam.station.cal[i].pole.i=0.0;
      lam.station.cal[i].zero.r=0.0;
      lam.station.cal[i].zero.i=0.0;
   }

   for(i=1;i<argc;i++)
      if(strncmp(argv[i],"st.",3)!=0)
         st_error(0,0,"Illegal file %s: File name must begin with st.\n");
   else {
      strcpy(outname,argv[i]);
      outname[0]='a';
      outname[1]='h';
      infile=st_open(argv[i],"r");
      outfile=fopen(outname,"w");

   while((tr_len=st_get(&st_ptr,&st_type,&st_len,infile))!=EOF) {
      switch(st_type) {
         case STATIONCOMP: stcp=(struct stationcomp *)st_ptr;
                           strcpy(lam.station.code,stcp->sc_name.st_name);
                           lam.station.chan[0]=stcp->sc_name.component;
                           strcpy(lam.station.stype,
                              list_code(stcp->sc_name.inst_type,inst_type));
                           if(stcp->sc_name.inst_type==0)
                              lam.station.stype[0]='\0';
                           lam.station.slat=stcp->st_lat;
                           lam.station.slon=stcp->st_long;
                           lam.station.elev=stcp->elev;
                           break;
         case DESCRIPTRACE:dstr=(struct descriptrace *)st_ptr;
                           lam.record.type=ahFLOAT;
                           lam.record.ndata=dstr->length;
                           lam.record.delta=1./(dstr->rate);
                           lam.record.maxamp=dstr->maxdata-dstr->avenoise;
                           decode_mstime(dstr->begintime,&yer,&mon,&day,&our,
                              &min,&sec);
                           lam.record.abstime.yr=yer;
                           lam.record.abstime.mo=mon;
                           lam.record.abstime.day=day;
                           lam.record.abstime.hr=our;
                           lam.record.abstime.mn=min;
                           lam.record.abstime.sec=sec;
                           lam.record.rmin=0.0;
                           lam.record.rcomment[0]='\0';
                           lam.record.log[0]='\0';
                           lam.station.DS=0.0;
                           fwrite(&lam,1,AHHEADSIZE,outfile);
                           ch=st_ptr+st_len;
                           for(i=0,leng=0;i<lam.record.ndata;i++) {
                              ch=ch+leng;
                              switch(dstr->datatype) {
                                 case 'r': sh=(short  *)(ch); leng=2;
                                           ftrace= *sh/16.0; break;
                                 case 's': sh=(short  *)(ch); ftrace= *sh;
                                           leng=2; break;
                                 case 'l': in=( long  *)(ch); ftrace= *in;
                                           leng=4; break;
                                 case 'f': fl=(float  *)(ch); ftrace= *fl;
                                           leng=4; break;
                                 default : st_error(die,0,
                                    "Unknown datatype in struct descriptrace of %c",
                                    dstr->datatype);
                              }
                              ftrace=(ftrace-dstr->avenoise);
                              fwrite(&ftrace,4,1,outfile);
                           }
                           break;
         case ORIGIN:      org=(struct origin *)st_ptr;
                           lam.event.lat=org->or_lat;
                           lam.event.lon=org->or_long;
                           lam.event.dep=org->depth;
                           decode_mstime(dstr->begintime,&yer,&mon,&day,&our,
                              &min,&sec);
                           lam.event.ot.yr=yer;
                           lam.event.ot.mo=mon;
                           lam.event.ot.day=day;
                           lam.event.ot.hr=our;
                           lam.event.ot.mn=min;
                           lam.event.ot.sec=sec;
                           break;
         case CALIBRATION: st_error(0,0,"Calibration structure not translated.");
```

```
                        break;
                }
            }
            st_close(infile);
            fclose(outfile);
            st_free(st_ptr);
        }
    }
```

08

```
/* st2asc: convert suds structures to ascii
   PLWard, US Geological Survey, Menlo Park, Calif 4/29/88
 */
static char SccsId[] = "@(#)st2asc.c    1.2 11/17/88";
#define MAXOUTLINE      300

/* DEFAULTS */
int  linelength=72;
char indentinit[]="    ";
char headeri[]="";
char structhead[]="    STRUCTURE: ";

#include "/usr/suds/include/suds.h"
#include "/usr/suds/include/st_error.h"
#include <stdio.h>

die(in)
   int in;
{
   exit(in);
}

st2asc(infile,outfile,cfile,sepstr,verbose,nolist,indenti,header)
   char *infile,*outfile,*cfile,*sepstr,*indenti,*header;
   int verbose,nolist;
{
   int    type,len,inp;
   FILE   *inf,*outf,*conf,*st_open(),*efopen();
   char   *ptr;
   char outline[MAXOUTLINE];

   inf= st_open(infile,"r");
   outf=efopen(outfile,"w");
   while((inp=st_get(&ptr,&type,&len,inf))!=EOF) {
      if(type<0 || type>TOTAL_STRUCTS) {
         fprintf(outf,
            "Skipped unrecognized structure type %d of length %d\n",type,len);
         continue;
      }
      liststruct(type,ptr,len,inp,0,outf,outline,sepstr,verbose,nolist,indenti,header);
      st_free(ptr,inp);
   }
}

extern double atof();
extern int atoi();

liststruct(type,pointer,len,inp,level,outf,outline,sepstr,verbose,nolist,indenti,header)
   int type,len,inp,level,verbose,nolist;
   char *pointer,*sepstr,*outline,*indenti,*header;
   FILE *outf;
{
   register i,j;
   short  *sh,shh;
   long   *lg;
   float  *ft;
   double *db;
   char   *ch,*ctime();
   char   *ptr;
   int    sep,code,leng,dtype;
   char   indent[MAXOUTLINE];
   char   temp[MAXOUTLINE];
   extern char *list_code();
   struct descriptrace *dt;
   struct muxdsta *mx;

ptr=pointer;
if(strlen(outline)>=linelength) {
   fprintf(outf,"%s\n",outline);
   outline[0]='\0';
}
if(verbose) {
   for(i=0,indent[0]='\0';i<level;i++) sprintf(indent,"%s%s",indent,indenti);
   if(level==0) {
      fprintf(outf,"%s%s%s%s (length=%d+%d=%d bytes)\n",indent,header,
         structhead,struct_names[type].meaning,len,inp-len,inp);
   }
   else fprintf(outf,"%s%s%s\n",
      indent,structhead,struct_names[type].meaning);
   for(i=0,indent[0]='\0';i<=level;i++) sprintf(indent,"%s%s",indent,indenti);
}
else {
   if(level==0) {
      sprintf(outline,"%s%d%s%d%s%d%s",
         header,type,sepstr,len,sepstr,inp-len,sepstr);
   }
   if(strlen(outline)>=linelength) {
      fprintf(outf,"%s\n",outline);
      sprintf(outline,"%s",indenti);
   }
}
for(i=beg_struct[type];stform[i].fstype==type && stform[i].offset<len;i++) {
   sep=1;
   if(verbose) sprintf(outline,"%s%-20s:\t",indent,stform[i].fname);
   code= -1;
   switch(stform[i].ftype) {
      case CHR: ch=ptr+stform[i].offset;
                sprintf(temp,stform[i].fformat,*ch);
                sprintf(outline,"%s'%s'",outline,temp);
                if(stform[i].codelist!=0) code= *ch;
                break;
      case BTS:
      case MIN: ch=ptr+stform[i].offset;
                sprintf(temp,stform[i].fformat,(int)*ch);
                sprintf(outline,"%s%s",outline,temp);
                if(stform[i].codelist!=0) code= *ch;
                break;
      case STR: ch=ptr+stform[i].offset;
                sprintf(temp,stform[i].fformat,ch);
                sprintf(outline,"%s\"%s\"",outline,temp);
                break;
      case SHT: sh=(short *)(ptr+stform[i].offset);
                if(*sh==atoi(NODATA))sprintf(temp,"NODATA");
                else sprintf(temp,stform[i].fformat,*sh);
                sprintf(outline,"%s%s",outline,temp);
                if(stform[i].codelist!=0) code= *sh;
                break;
      case LNG: lg=(long *)(ptr+stform[i].offset);
                if(*lg==atoi(NODATA))sprintf(temp,"NODATA");
                else sprintf(temp,stform[i].fformat,*lg);
                sprintf(outline,"%s%s",outline,temp);
                if(stform[i].codelist!=0) code= *lg;
                break;
      case FLT: ft=(float *)(ptr+stform[i].offset);
                if(*ft==atoi(NODATA))sprintf(temp,"NODATA");
                else sprintf(temp,stform[i].fformat,*ft);
                sprintf(outline,"%s%s",outline,temp);
                break;
      case LLT:
      case DBL: db=(double *)(ptr+stform[i].offset);
```

```
                if(*db==atoi(NODATA))sprintf(temp,"NODATA");
                else sprintf(temp,stform[i].fformat,*db);
                sprintf(outline,"%s%s",outline,temp);
                break;
        case STT: lg=(long *)(ptr+stform[i].offset);
                if(verbose) {
                   *db= *lg;
                   if(*lg==atoi(NOTIME))sprintf(temp,"NOTIME");
                   else sprintf(temp,"%s",list_mstime(*db,7));
                   sprintf(outline,"%s%s",outline,temp);
                }
                else {
                   if(*lg==atof(NOTIME))sprintf(temp,"NOTIME");
                   else sprintf(temp,stform[i].fformat,*lg);
                   sprintf(outline,"%s%s",outline,temp);
                }
                break;
        case MST: db=(double *)(ptr+stform[i].offset);
                if(verbose) {
                   if(*db==atof(NOTIME))sprintf(temp,"NOTIME");
                   else sprintf(temp,"%s",list_mstime(*db,6));
                   sprintf(outline,"%s%s",outline,temp);
                }
                else {
                   if(*db==atof(NOTIME))sprintf(temp,"NOTIME");
                   else sprintf(temp,stform[i].fformat,*db);
                   sprintf(outline,"%s%s",outline,temp);
                }
                break;
        case CAL:
        case CPX:
        case STI:
        default:  if(verbose) {
                   fprintf(outf,"%s\n",outline);
                   outline[0]='\0';
                }
                if(stform[i].nextfstype!=0) {
                   liststruct(stform[i].nextfstype,ptr,len,inp,level+1,outf,outline,
                      sepstr,verbose,nolist,indenti,header);
                   sep=0;
                }
                break;
    };
    if(verbose && code>=0) {
       sprintf(outline,"%s\t{%s}",outline,
           list_code(code,stform[i].codelist));
    }
    if(verbose && outline[0]!='\0') {
       fprintf(outf,"%s\n",outline);
       outline[0]='\0';
    }
    else {
       if(sep)sprintf(outline,"%s%s",outline,sepstr);
       if(strlen(outline)>=linelength) {
          fprintf(outf,"%s\n",outline);
          sprintf(outline,"%s",indenti);
       }
    }
}
if((type==DESCRIPTRACE || type==MUXDATA) && nolist==0) {
   if(outline[0]!='\0') {
      fprintf(outf,"%s\n",outline);
      sprintf(outline,"%s",indenti);
   }
   else if(verbose) {
      fprintf(outf,"%sWaveform Data:\n",indenti);
      sprintf(outline,"%s",indenti);
   }
   if(type==DESCRIPTRACE) {
      dt=(struct descriptrace *)ptr;
      leng=dt->length;
      dtype=dt->datatype;
    }
   if(type==MUXDATA) {
      mx=(struct muxdata *)ptr;
      leng=mx->numsamps;
      dtype=mx->typedata;
   }
   ch=ptr+len;
   switch(dtype) {
      case 'r':
      case 's': sh=(short  *)(ch); break;
      case 'l': lg=( long  *)(ch); break;
      case 'f': ft=(float  *)(ch); break;
      case 'c':
      case 'v': ft=(float  *)(ch); leng=leng*2; break;
      case 't': ft=(float  *)(ch); leng=leng*3; break;
      case 'd': db=(double *)(ch); break;
      default : st_error(die,0,"Unknown datatype in struct descriptrace of %c",
                   dt->datatype);
   }
   for(i=0;i<leng;i++) {
      switch(dtype) {
         case 'r': shh=sh[i]>>4;
                   sprintf(outline,"%s%6d:%x%s",
                      outline,shh,sh[i] & 0xf,sepstr); break;
         case 's': sprintf(outline,"%s%6d%s",outline,sh[i],sepstr); break;
         case 'l': sprintf(outline,"%s%6ld%s",outline,lg[i],sepstr); break;
         case 'c':
         case 'v':
         case 't':
         case 'f': sprintf(outline,"%s%13.6e%s",outline,ft[i],sepstr); break;
         case 'd': sprintf(outline,"%s%13.6le%s",outline,db[i],sepstr); break;
      }
      if(strlen(outline)>=linelength) {
         fprintf(outf,"%s\n",outline);
         sprintf(outline,"%s",indenti);
      }
   }
}
else if(type==DESCRIPTRACE && verbose)fprintf(outf,"%sWaveform Data:\n",indenti);
if(level==0) {
   fprintf(outf,"%s\n",outline);
   outline[0]='\0';
}
}

main(argc,argv)
   int argc;
   char **argv;
{
   register i;
   int verbose,numfil,nolist;
   char *ofile,*cfile;
   extern char *progname;
   char *sepstr,*indenti,*header;              /* string separating fields     */

   indenti=indentinit;
   header=headeri;
   numfil=0;
```

```
    verbose=0;
    nolist=0;
    cfile="";
    ofile="stdout";
    sepstr=" ";
    progname=argv[0];
    for(i=1;i<argc;i++){
       if(argv[i][0]=='-') {
          switch(argv[i][1]) {
             case 'c': if(argv[i][2]!='\0') cfile= &argv[i][2];
                       else cfile=argv[++i];
                       break;
             case 'h': if(argv[i][2]!='\0') header= &argv[i][2];
                       else header=argv[++i];
                       break;
             case 'i': if(argv[i][2]!='\0') indenti= &argv[i][2];
                       else indenti=argv[++i];
                       break;
             case 'l': if(argv[i][2]!='\0') linelength= atoi(&argv[i][2]);
                       else linelength=atoi(argv[++i]);
                       break;
             case 'n': nolist=1;
                       break;
             case 'o': if(argv[i][2]!='\0') ofile= &argv[i][2];
                       else ofile=argv[++i];
                       break;
             case 's': if(argv[i][2]!='\0') sepstr= &argv[i][2];
                       else sepstr=argv[++i];
                       break;
             case 'v': verbose=1;
                       break;
             default: st_error(0,0,"Unrecognized argument %s",argv[i]);
          }
       }
       else {
          st2asc(argv[i],ofile,cfile,sepstr,verbose,nolist,indenti,header);
          numfil++;
       }
    }
    if(numfil==0) st2asc("stdin",ofile,cfile,sepstr,verbose,nolist,indenti,header);
}
```

```
/* stdescribe: list structures in a stream
 */
static char SccsId[] = "@(#)stdescribe.c        1.2 11/17/88";
#include "/usr/suds/include/suds.h"
#include <stdio.h>

die(n) int n; {exit(n);}
extern char *list_code();

stdescribe(infil,outfile)
   char *infil;
   FILE *outfile;
{
   int typ,len,inp,num;
   char *ptr;
   STATIDENT *si;
   FILE *infile;

   num=0;
   infile=st_open(infil,"r");
   fprintf(outfile,"File %s:\n",infil);
   while((inp=st_get(&ptr,&typ,&len,infile))!=EOF) {
      fprintf(outfile,"%3d  %3d  %-14s length %4d + %6d bytes",num++,typ,
         list_code(typ,struct_names),len,inp-len);
      if(stform[beg_struct[typ]].nextfstype==STAT_IDENT) {
         si=(STATIDENT *)ptr;
         fprintf(outfile,"  %s.%s.%c.%d",si->network,si->st_name,si->component,
            si->inst_type);
      }
      fprintf(outfile,"\n");
      st_free(ptr,inp);
   }
   st_close(infile);
}

main(argc,argv)
   int argc;
   char **argv;
{
   register i;
   int numfil,out;
   char *network,*ofile;
   extern char *progname;
   FILE *outfile;

   out=1;
   progname=argv[0];
   numfil=0;
   ofile="stdout";
   for(i=1;i<argc;i++){
      if(argv[i][0]=='-') {
         switch(argv[i][1]) {
            case 'o': if(argv[i][2]!='\0') ofile= &argv[i][2];
                      else ofile=argv[++i];
                      break;
            default: st_error(0,0,"Unrecognized argument %s",argv[i]);
         }
      }
      else {
         if(out) {
            outfile=st_open(ofile,"w");
            out=0;
         }
         stdescribe(argv[i],outfile);
         numfil++;
      }
   }
   if(numfil==0) {
      outfile=st_open(ofile,"w");
      stdescribe("stdin",outfile);
   }
   st_close(outfile);
}
```

```
/*stedit: display and edit suds files
   This program is a quick hack of another program I wrote to show how
   an stedit might function. It should be integrated with the Lamont waveform
   editor and thought out much more carefully.
   PLWard, U.S.Geological Survey, Menlo Park, Ca 94025  5/19/88
 */
static char SccsId[] = "@(#)stedit.c    1.2 11/17/88";
#include <stdio.h>
#include <curses.h>
#include <ctype.h>
#include "/usr/suds/include/suds.h"

#define MAXCOL 80
#define MAXOUT 100

/* form.h
*/
#define MAXLINK 20      /* maximum length of any link string            */
#define MAXFIELD 100    /* maximum length of any single input field     */
#define ERR_Y   1       /* row of error messages                        */
#define ERR_X   1       /* first column of error messages               */
#define INST_Y  2       /* row of instructions                          */
#define INST_X  1       /* first column of instructions                 */
#define CMD_Y   2       /* row of command line                          */
#define CMD_X   1       /* first column of command line                 */
#define SPACE   '_'     /* char to show a space for input               */
#define BIGGEST 30      /* dimension of largest string in database      */
#define NONE    '\0'    /* no char                                      */
#define BREAK   00
#define CNTRLC  03
#define CNTRLD  04
#define BELL    07
#define BSP     010
#define LFD     012
#define TAB     011
#define RET     015
#define ESC     033
#define SPA     ' '
#define DEL     0177
#define UPCUR   05
#define DNCUR   06
#define RTCUR   07
#define LFCUR   02
#define F2      020
#define F3      021
#define F4      022
#define F5      023
#define F6      024
#define F7      025
#define F8      026
#define F9      027
#define R1      030
#define R2      031
#define R3      032

FILE *cons;

extern int  beg_struct[];
extern int  width_field[];
extern char *list_code();
extern char *list_mstime();
extern int atoi();
extern double atof();

/* primary active indecies form addressing screen                       */
#define NCOL  stdscr->_curx     /* current cursor column                */
#define NROW  stdscr->_cury     /* current cursor row                   */

int refr =      1;              /* if true refresh screen next          */
int erractive = 0;              /* if true error written, remove        */

char AUPtoLO(c) char c; { return(isupper(c)?c-'A'+'a':c); }

beg_curses() {

   if(initscr()==ERR) {
      fprintf(stderr,"Insufficient core to allocate a window.\n");
      exit(ERR);
   }
   noecho();
   raw();
   crmode();
   nonl();
}

end_curses() {
   refresh();
   mvcur(0,COLS-1,LINES-1,0);
   echo();
   noraw();
   nocrmode();
   nl();
   endwin();
}

errform(fmt,ch)
   char *fmt,ch;
{
   int co,ro;

   co=NCOL;
   ro=NROW;
   move(ERR_Y,ERR_X);
   erractive=1;
   if(ch!=0)printw(fmt,ch);
   clrtoeol();
   move(ro,co);
   refresh();
}

errmess(fmt,ch)
   char *fmt,*ch;
{
   int co,ro;

   co=NCOL;
   ro=NROW;
   move(ERR_Y,ERR_X);
   erractive=1;
   if(ch!=0)printw(fmt,ch);
   clrtoeol();
   move(ro,co);
   refresh();
}

inform(fmt,a,b,c,d,e,f,g,h,i)
   char *fmt;
   int a,b,c,d,e,f,g,h,i;
{
   int co,ro;
```

```
   co=NCOL;
   ro=NROW;
   move(ERR_Y,ERR_X);
   if(a!=0)printw(fmt,a,b,c,d,e,f,g,h,i);
   clrtoeol();
   move(ro,co);
   refresh();
}

int pshow=0;

/* get from keyboard */
char getcode(allowed)
   char *allowed;
{
   register char c,d,conv;
   char *allow;

getit:
   c=0177 & getch();                    /* read character          */
   if(erractive) {                      /* if error message, erase */
      errform("",0);
      erractive=0;
   }
   if(c>=' ' && c<='~') {               /* if char allowed return  */
      allow=allowed;
      for(conv=0; *allow != NONE;) {
         if(*allow == '!') conv= *(++allow) >= 'a' ? 'A'-'a':'a'-'A';
         if(*allow == '-' && *(++allow-2)<c && c< *allow)
            return(c+conv);
         if(*allow == '\\') allow++;
         if(*allow == c) return(c+conv);
         if(*(++allow) != '-') conv=0;
      }
      fputc(BELL,stderr);
      errmess("Only characters allowed are %s",allowed);
      goto getit;
   }
   switch(c) {                          /* treat control characters */
      case ESC:   c=0177 & getch();
                  if(c!='O' && c!='[') {
                     errform("\007Unexpected escape sequence %o",c);
                     goto getit;
                  }
                  switch(c=0177 & getch()) {
                     case '2':  c=0177 & getch(); d=0177 & getch();
                                if((c!='2' && c!='3')||(d<'0' && d>'9')) {
                                   errform("\007Unexpected escape sequence %o",c);
                                   goto getit;
                                }
                                if((0177 & getch())!='z') {
                                   errform("\007Unexpected escape sequence ");
                                   goto getit;
                                }
                                switch ((c-'0')*10+d-'0') {
                                   case  8: return(R1);
                                   case  9: return(R2);
                                   case 10: return(R3);
                                   case 25: return(F2);
                                   case 26: return(F3);
                                   case 27: return(F4);
                                   case 28: return(F5);
                                   case 29: return(F6);
                                   case 30: return(F7);
                                   case 31: return(F8);
                                   case 32: return(F9);
                                   default: errform(
                                               "\007Unexpected escape code %o",c);
                                            goto getit;
                                }
                     case 'A': return(UPCUR);   /* UPCUR */
                     case 'B': return(DNCUR);   /* DNCUR */
                     case 'C': return(RTCUR);   /* RTCUR */
                     case 'D': return(LFCUR);   /* LFCUR */
                     default:  errform("\007Unexpected escape code %o",c);
                               goto getit;
                  }
      case BSP:
      case LFD:
      case RET:
      case TAB:    return(c);
      case DEL:    return(BSP);
      case CNTRLD: pshow=pshow?0:1;
                   if(!pshow)inform("",0);
                   goto getit;
      case CNTRLC: die(100);
      case BREAK:
      default:     errform("\007Unexpected control code %o",c);
                   goto getit;
   }
}

outchar(c,field,minf,maxf,cnum,cmax)
   char c;
   int  *field,minf,maxf,*cnum,cmax;
{
   register i;
   int y;

   repeat: switch (c) {
      case UPCUR: *cnum= -2; goto out;
      case DNCUR: *cnum=cmax; goto out;
      case RTCUR: if(*cnum<cmax-1) {
                     move(NROW,++NCOL);
                     goto out;
                  }
                  else {
                     *cnum=cmax;
                     goto out;
                  }
      case LFCUR: if(*cnum>0) {
                     move(NROW,--NCOL);
                     *cnum-=2;
                     goto out;
                  }
                  else {
                     *cnum= -2;
                     goto out;
                  }
      case NONE:  goto out;
      default:    addch(c);
   }
out:
   (*cnum)++;
   if(refr)refresh();
}

getfield(index,ptr,type,baserow)
   int index;           /* index into stform[] table           */
```

```
    int type;           /* type of ptr                          */
    int baserow;        /* for recursive structures             */
    char *ptr;
{
    char temp[MAXOUT];
    register i,j ;
    int len;

    len=width_field[stform[index].ftype];
    if(stform[index].ftype==STR)len=stform[index].flength;
    j=stform[index].form_col;
    for(i=0;i<len;i++) {
       temp[i]=stdscr->_y[stform[index].form_row+baserow][j++];
       if(temp[i]==SPACE)temp[i]=SPA;
    }
    temp[len-1]='\0';
    if(strncmp(temp,"NOTIME",6)==0) {
       if(type==MST) *((MS_TIME *)(ptr))=atof(NOTIME);
       else          *((ST_TIME *)(ptr))=atoi(NOTIME);
       return;
    }
    asc2field(temp,ptr,type);
}

editstform(ptr,type,len,base,baserow)
    char *ptr;
    int type,len,base,baserow;
{
    register int c;
    int i,j;
    char d;
    char in[MAXFIELD];
    int minfield;       /* min row in stform[] in this structure*/
    int maxfield;       /* max row in stform[] in this structure*/
    int curfield;       /* current index in stform[] table      */
    int charnum;        /* current index in input word          */
    int charmax;        /* maximum index in input word          */
    int oldcurf;
    int oldtype;
    int code;
    long timl;
    double timd;

    charnum=0;
    charmax=0;
    oldcurf= -1;
    oldtype= -1;
begin:
    if(type!=oldtype) {
       curfield=beg_struct[type];
       minfield=curfield;
       for(maxfield=minfield;stform[maxfield].fstype==type;maxfield++);
       maxfield--;
       oldtype=type;
    }
    if(charnum>charmax) curfield++;
    else if(charnum<0)curfield--;
    if(curfield!=oldcurf) {
       if(curfield>maxfield) {
          if(base== -1)curfield=minfield;
          else return(c);
       }
       if(curfield<minfield) {
          if(base== -1)curfield=maxfield;
          else return(c);
       }
       charnum=0;
       charmax=width_field[stform[curfield].ftype];
       if(stform[curfield].ftype==STR)charmax=stform[curfield].flength;
       oldcurf=curfield;
       move(stform[curfield].form_row+baserow,stform[curfield].form_col);
       refresh();
    }
    if(pshow) {
       fprintf(cons,
          "row=%2d col=%2d curfield=%2d/%2d/%2d old=%2d index=%2d/%2d base=%d\n",
          NROW,NCOL,minfield,curfield,maxfield,oldcurf,charnum,charmax,base);
       fflush(cons);
       refresh();
    }
    if(charnum>=charmax) goto end;

    if(stform[curfield].nextfstype!=0) {
       c=editstform(ptr,stform[curfield].nextfstype,len,stform[curfield].offset,
          stform[curfield].form_row);
    }
    else c=getcode(stform[curfield].allowchar);      /* read new character */
    if(c < ' ' || c > '~') switch(c) {
       case F2:                                          /* f2: */
       case F7:                                          /* f7: */
       case EOF:    return(c);
       case R1:
       case R2:
       case R3:
       case F3:                                          /* f3: */
       case F4:                                          /* f4: */
       case F5:                                          /* f5: */
       case F6:                                          /* f6: */
       case F8:                                          /* f8: */
       case F9:     goto begin;                          /* f9: */
       case LFCUR:
       case RTCUR:
       case UPCUR:
       case DNCUR: outchar(c,&curfield,minfield,maxfield,&charnum,charmax);
                   goto begin;
       case NULL:
       case LFD:
       case RET:    goto end;
       case TAB:    while(charnum<charmax) {
                       outchar(stdscr->_y[NROW][charnum+stform[curfield].form_col],
                          &curfield,minfield,maxfield,&charnum,charmax);
                    }
                    c=RET;
                    goto end;
       case BSP:    refr=0;
                    outchar(LFCUR,&curfield,minfield,maxfield,&charnum,charmax);
                    outchar(SPACE,&curfield,minfield,maxfield,&charnum,charmax);
                    refr=1;
                    outchar(LFCUR,&curfield,minfield,maxfield,&charnum,charmax);
                    c=AUPtoLO(stdscr->_y[NROW][NCOL-1]);
                    goto begin;
    }
    else outchar(c,&curfield,minfield,maxfield,&charnum,charmax);
    goto begin;
end:
    refr=0;
    if(charnum>0) {
       while(charnum<charmax)
             outchar(SPACE,&curfield,minfield,maxfield,&charnum,charmax);
       if(stform[curfield].codelist!=0) {
```

```
            getfield(curfield,&code,LNG,baserow);
            printw("  {%s}",list_code(code,stform[curfield].codelist));
            clrtoeol();
         }
         if(stform[curfield].ftype==MST) {
            getfield(curfield,&timd,MST,baserow);
            if(timd!=atof(NOTIME))printw(" {%s}",list_mstime(timd,6));
            clrtoeol();
         }
         if(stform[curfield].ftype==STT) {
            getfield(curfield,&timl,STT,baserow);
            if(timl!=atoi(NOTIME))printw(" {%s}",list_mstime((double)timl,7));
            clrtoeol();
         }
         charnum++;
         refr=1;
         refresh();
         goto begin;
      }
      else charnum=charmax+1;
      refr=1;
      goto begin;
   }

initstform(ptr,type,len,base)
   char *ptr;
   int type,len,base;
{
   register i,j;
   int num,numf,k,col;

   for(i=beg_struct[type];stform[i].fstype==type && stform[i].offset<len;i++) {
      if(stform[i].nextfstype!=0) {
         initstform(ptr,stform[i].nextfstype,len,stform[i].form_row);
      }
      else {
         standout();
         move(stform[i].form_row+base,stform[i].form_col-strlen(stform[i].fname)-1);
         printw("%s",stform[i].fname);
         standend();
         col=stform[i].form_col;
         move(stform[i].form_row+base,stform[i].form_col);
         numf=1;
         num=width_field[stform[i].ftype];
         if(stform[i].ftype==STR)num=stform[i].flength;
         if(stform[i].ftype>=SHT && stform[i].ftype<=DBL) numf=stform[i].flength;
         for(k=0;k<numf;k++) {
            for(j=0;j<num;j++) {
               col++;
               printw("_");
            }
            if(col>(MAXCOL-num) && k<numf-1) base++;
         }
      }
   }
}


#include <sgtty.h>
#include <sys/file.h>

loadstform(ptr,type,len,base)
   char *ptr;
   int type,len,base;
{
   register i,j;
   int num,numf,sep,code,col,after;
   char temp[MAXOUT];
   char *ch;
   short *sh;
   long *lg;
   float *ft;
   double *db;

   for(i=beg_struct[type];stform[i].fstype==type && stform[i].offset<len;i++) {
      sep=1;
      code= -1;
      num=width_field[stform[i].ftype];
      if(stform[i].ftype==STR)num=stform[i].flength;
      switch(stform[i].ftype) {
         case CHR: ch=ptr+stform[i].offset;
                   sprintf(temp,stform[i].fformat,*ch);
                   if(stform[i].codelist!=0) code= *ch;
                   break;
         case BTS:
         case MIN: ch=ptr+stform[i].offset;
                   sprintf(temp,stform[i].fformat,(int)*ch);
                   if(stform[i].codelist!=0) code= *ch;
                   break;
         case STR: ch=ptr+stform[i].offset;
                   sprintf(temp,stform[i].fformat,ch);
                   break;
         case SHT: sh=(short *)(ptr+stform[i].offset);
                   if(*sh==atoi(NODATA))sprintf(temp,"NODATA");
                   else sprintf(temp,stform[i].fformat,*sh);
                   if(stform[i].codelist!=0) code= *sh;
                   numf=stform[i].flength; break;
         case LNG: lg=(long *)(ptr+stform[i].offset);
                   if(*lg==atoi(NODATA))sprintf(temp,"NODATA");
                   else sprintf(temp,stform[i].fformat,*lg);
                   if(stform[i].codelist!=0) code= *lg;
                   numf=stform[i].flength; break;
         case FLT: ft=(float *)(ptr+stform[i].offset);
                   if(*ft==atoi(NODATA))sprintf(temp,"NODATA");
                   else sprintf(temp,stform[i].fformat,*ft);
                   numf=stform[i].flength; break;
         case DBL: db=(double *)(ptr+stform[i].offset);
                   if(*db==atoi(NODATA))sprintf(temp,"NODATA");
                   else sprintf(temp,stform[i].fformat,*db);
                   numf=stform[i].flength; break;
         case LLT: db=(double *)(ptr+stform[i].offset);
                   if(*db==atoi(NODATA))sprintf(temp,"NODATA");
                   else sprintf(temp,stform[i].fformat,*db);
                   break;
         case STT: lg=(long *)(ptr+stform[i].offset);
                   if(*lg==atof(NOTIME))sprintf(temp,"NOTIME");
                   else sprintf(temp,stform[i].fformat,*lg);
                   break;
         case MST: db=(double *)(ptr+stform[i].offset);
                   if(*db==atof(NOTIME))sprintf(temp,"NOTIME");
                   else sprintf(temp,stform[i].fformat,*db);
                   break;
         case CAL:
         case CPX:
         case STI:
         default:  if(stform[i].nextfstype!=0) {
                      loadstform(ptr,stform[i].nextfstype,len,stform[i].form_row);
                      sep=0;
                   }
                   break;
```

```
      };
      if(sep) {
         temp[num]=NONE;
         for(after=j=0;j<num;j++) {
            if(temp[j]==NONE) after=1;
            if(after || temp[j]==SPA)temp[j]=SPACE;
         }
         move(stform[i].form_row+base,stform[i].form_col);
         printw("%s",temp);
         if(code>=0) {
            printw("  {%s}",list_code(code,stform[i].codelist));
            clrtoeol();
         }
      }
      if(stform[i].ftype==MST && strncmp(temp,"NOTIME",6)!=0)
         printw(" {%s}",list_mstime(*db,6));
      if(stform[i].ftype==STT && strncmp(temp,"NOTIME",6)!=0)
         printw(" {%s}",list_mstime((double)(*lg),7));
   }
   refresh();
}

savestform(ptr,type,len,base,baserow)
   char *ptr;
   int type,len,base,baserow;
{
   register i;

   for(i=beg_struct[type];stform[i].fstype==type && stform[i].offset<len;i++) {
      if(stform[i].nextfstype!=0) {
         savestform(ptr,stform[i].nextfstype,len,stform[i].offset+base,
            stform[i].form_row+baserow);
      }
      else {
         getfield(i,ptr+base+stform[i].offset,stform[i].ftype,baserow);
      }
   }
}

char *infilen=NULL;
char *outfile;
char template[]="stedit.XXXXXX";
FILE *st_inf,*st_outf;

stedit(infile)
   char *infile;
{
   int inp,type,len,stnum,ret;
   char *ptr;
   extern char *mktemp();
   char instr[5];

   infilen=infile;
   st_inf=st_open(infile,"r");
   if(strcmp("stdin",infile)==0) st_outf=st_open("stdout","w");
   else {
      outfile=mktemp(template);
      st_outf=st_open(outfile,"w");
   }
   beg_curses();
   clearok(stdscr,TRUE);
   clear();
   refresh();           /* needed to get sun-cmd window in proper sync  */
   for(stnum=0;(inp=st_get(&ptr,&type,&len,st_inf))!=EOF;stnum++) {
      clear();
      move(CMD_Y,CMD_X);
      printw("Edit structure %d of type %d (%s) of length %d bytes",
         stnum,type,list_code(type,struct_names),len);
      standout();
      move(0,15);
      printw("F7=next structure      F2=abort without change");
      standend();
      initstform(ptr,type,len,0);
      loadstform(ptr,type,len,0);
      move(stform[beg_struct[type]].form_row,stform[beg_struct[type]].form_col);
      refresh();
      ret=editstform(ptr,type,len,-1,0);
      if(ret==F2) break;
      savestform(ptr,type,len,0,0);
      st_put(ptr,type,len,st_outf);
   }
   refresh();
   end_curses();
   if(ret==F2) unlink(outfile);
   else if(strcmp("stdin",infile)!=0) {
      instr[0]=SPA;
      while(instr[0]!='y' && instr[0]!='n') {
         fprintf(stderr,"Do you want to save these changes? (Answer y or n) ");
         fscanf(stdin,"%s",instr);
      }
      if(instr[0]=='y') {
         st_close(st_outf);
         type=rename(outfile,infile);
         if(type== -1)st_error(die,0,"rename of %s to %s failed",outfile,infile);
      }
      else unlink(outfile);
   }
   st_close(st_inf);
}

die(in)
   int in;
{
   char instr[5];
   int type;

   end_curses();
   if(infilen!=NULL && strcmp("stdin",infilen)!=0) {
      while(instr[0]!='y' && instr[0]!='n') {
         fprintf(stderr,"Do you want to save these changes? (Answer y or n) ");
         fscanf(stdin,"%s",instr);
      }
      if(instr[0]=='y' || instr[0]=='Y') {
         st_close(st_outf);
         type=rename(outfile,infilen);
         if(type== -1)st_error(die,0,"rename of %s to %s failed",outfile,infilen);
      }
      else unlink(outfile);
   }
   st_close(st_inf);
   exit(in);
}

extern char *progname;

main(argc,argv)
   int argc;
   char **argv;
{
   register i;
```

```
   int numfil,outcl;

   cons=fopen("/dev/console","w");
   progname=argv[0];
   numfil=0;
   outcl=1;
   for(i=1;i<argc;i++){
      if(argv[i][0]=='-') {
         switch(argv[i][1]) {
            default: st_error(0,0,"Unrecognized argument %s",argv[i]);
         }
      }
      else {
         stedit(argv[i]);
         numfil++;
      }
   }
   if(numfil==0) {
      stedit("stdin");
   }
}
```

```
Change pathname to your pathname
Put these lines in /etc/rc.local
  just before the line
(echo -n 'starting rpc and net services:')                    >/dev/console

(echo -n 'starting online eq detector:')                      >/dev/console
if [ -f /usr/katmai/online.run ]; then
        /usr/katmai/online.run & (echo -n ' online.run')      >/dev/console
        (echo '.')                                            >/dev/console
fi
```

9

```
# Make eqdetector, real-time earthquake detector

OBJ=    eqdetector.o getsweep.o cache.o out.o args.o diag.o dynflag.o
O44=    eqdetector.o getsweep.o cache.o out.o args.o diag.o dynflag.o lpaopen.o rtp.o

# use DLPA line for a DEC PDP11/44 or 11/34
CFLAGS = -g -f68881
#CFLAGS = -O -DLPA

eqdetector: $(OBJ) Makefile
        cc $(CFLAGS) -o eqdetector $(OBJ) ../atod/CDDatod.o -lsuds -lm

on44:   $(O44) Makefile
        cc $(CFLAGS) -i -o eqdetector $(O44)

list:
        enscript2 Testnet.c rtp.s out.c eqdetector.c lpaopen.c getsweep.c \
           dynflag.c diag.c cache.c args.c eqdetector.h lpa.h key.h \
           header.h Makefile

install:
        @echo ""
        @echo "******* You must be superuser to do this. *******"
        @echo ""
        chown root eqdetector
        chmod 4755 eqdetector
        cp eqdetector /etc/eqdetector

out.o : header.h
args.o cache.o diag.o dynflag.o eqdetector.o getsweep.o out.o : eqdetector.h
lpaopen.o : key.h
```

```
/*  args.c  --  process command line arguments for online.c  */

#include "eqdetector.h"

#ifndef LPA
#include <stdio.h>
#endif

extern int die();

#define  AARG   argv[++i < argc ? i : st_error(die,0,"not enough args") ]
#define  IARG   atoi(AARG)
#define  FARG   atof(AARG)

proc_args(argc,argv) char **argv; {
   register i; int j,n;
   char *arg;
   float atof();

   if(argc < 3){prt_args(); exit(0);}
   for(i=1; i<argc; i++) {
      if(argv[i][0]=='-') {
         for(arg=argv[i]; *arg!=0; arg++) switch( *arg ) {
            case '-':                                   break;

            case 'a': after     = FARG;                 break;
            case 'b': before    = FARG;                 break;
            case 'd': decim     = IARG;                 break;
            case 'e': const1    = IARG;
                      const2    = IARG;
                      threshold = IARG;                 break;
            case 'g': diffsam   = 1;                    break;
            case 'j': conttape  = 1;                    break;
            case 'l': aperture  = FARG;                 break;
            case 'n': nchan     = IARG;                 break;
            case 'q': shortcut  = 1;                    break;
            case 'r': rate      = IARG;                 break;
            case 's': sweep     = FARG;                 break;
            case 'u': nbuf      = IARG;                 break;
            case 'w': const3    = IARG;
                      const4    = IARG;
                      wav_inc   = IARG;                 break;
            case 'x': xchg_tape = IARG;                 break;
            case 'y': dyn_flag  = 1;                    break;
            case 'z': bufsz     = IARG;                 break;

            case 'c': cach_file = AARG;                 break;
            case 'f': disk_file = AARG;                 break;
            case 'o': log_file  = AARG;                 break;
            case 't': tape_file = AARG;                 break;
#ifdef LPA
            case '0': tape_file = "/dev/nrmt1";         break;
            case '1': tape_file = "/dev/nrmt3";         break;
#endif LPA
#ifndef LPA
            case '0': tape_file = "/dev/nrst0";         break;
            case '1': tape_file = "/dev/nrst0";
                      fprintf(stderr,"Only one tape drive\n");
                      exit(1);                          break;
#endif
            case 'p': pseudo_a2d= AARG;                 break;

            case 'A': prt_cach = 1;
                      prt_a2d  = 1;
                      prt_out  = 1;
            case 'D': prt_diag = 1;
                      prt_etas = 1;
                      prt_strg = 1;
                      prt_trig = 1;
                      prt_vbse = 1;
                      prt_st_av = 1;
                      prt_st_abs_av = 1;                break;
            case 'C': prt_cach = 1;                     break;
            case 'E': prt_etas = prt_diag = 1;          break;
            case 'F': fake_trig= 1;                     break;
            case 'H': no_head  = 1;                     break;
            case 'L': prt_a2d  = 1;                     break;
            case 'N': not_rtp  = 1;                     break;
            case 'M': prt_mall = 1;                     break;
            case 'O': prt_out  = 1;                     break;
            case 'P': prt_parm = 1;                     break;
            case 'Q': fake_ok  = 1;                     break;
            case 'R': prt_st_abs_av = prt_diag = 1;     break;
            case 'S': prt_st_av = prt_diag = 1;         break;
            case 'T': prt_trig = 1;                     break;
            case 'U': prt_tim  = 1;                     break;
            case 'V': prt_vbse = prt_trig = 1;          break;
            case 'X': exit_early = 1;                   break;
            case 'Z': prt_strg = prt_diag = 1;          break;

            default : fprintf(stderr,"online: command '%c' not legal\n", *arg);
                      prt_args(); exit(1);
         }
      }
      else {
         if(i+2!=argc)
            st_error(die,0,"Incorrect number of file names. Station file is %s",argv[i]);
         stat_file=argv[i++];
         subnet_file=argv[i];
      }
   }
}

static char *doc_parm[] = {
   "PARAMETERS",
   "a after secs",
   "b before secs",
   "d decimation factor",
   "e eta consts (3)",
   "g diff sam bef compu",
   "l len aperture secs",
   "n num chan from a2d",
   "q quick short cuts",
   "r rate in hz",
   "s sweep len secs",
   "u num a2d bufs",
   "w weighted ave consts",
   "x xchg tape events",
   "y enable dyn flags",
   "z buf size in words",
   "- alone: defaults",
   0 };
static char *doc_file[] = {
   "FILES",
   "c cache",
   "f disk file out",
   "i input net",
   "p pseudo a2d file",
   "t tape out",
   "0 tape unit 0",
```

```
    "l tape unit 1",
    "j write cont tape file",
    0 };
static char *doc_diag[] = {
    "DIAGNOSTICS",
    "A all",
    "C cache",
    "D misc diagnstcs",
    "E etas",
    "F one fake trig",
    "H no header write",
    "L a2d",
    "N no RTP",
    "M print malloc",
    "o diag output file name",
    "O output",
    "P params",
    "Q quit: fake trig",
    "R st_abs_av, lt_abs_av",
    "S st_av, lt_av",
    "T trigger",
    "U time usage",
    "V verbose trigger",
    "X exit after startup",
    "Z station trigger",
    0 };

#define get(s)    ( *s ? *s++ : "" )

prt_args() {
    char **p=doc_parm, **f=doc_file, **d=doc_diag;
    fprintf(stderr,"Arguments for online processing program version 7/27/88:\n\n");

    while( *p || *f || *d)
        fprintf(stderr,"%-25s  %-25s  %-25s\n", get(p), get(f), get(d) );
}
```

```
/* cache.c -- cache routines for online seismic detection system
 *
 * There are three external routines to be called from online.c:
 *
 *    cach_init(file_name, nbufbef, nbufaft, bufsz)
 *       file_name: file name of disk cache or 0 for core cache
 *       nbufbef:   time before trigger to save data in buffer loads
 *       nbufaft:   time after  trigger to continue writing in bufs
 *       bufsz:     buffer size in words (same as lpa buffer size)
 *
 *    trig_on()
 *       this function is called each time the system triggers.
 *
 *    putcach(lpabuf)
 *       called from lparead with pointer to most recently filled buffer
 */
#include <sys/file.h>
char conttap[]="/dev/nrst0";
static short *conbuf;

#include "eqdetector.h"

static char *cach_file=NULL;/* null=in core cache, else file name      */
static int  size;           /* size of cache in buffers                */
       int  current;        /* index of next cache buffer to use       */
static int  valid;          /* number of valid buffers (usu == size)   */
       int  trig;           /* bool and number of future bufs to write */
       int  nbuftrig;       /* initial value for trig ie nbuf to write */
static int  bufszb;         /* buffer size in bytes (same size as lpa) */
static int  bufszw;         /* buffer size in words (same size as lpa) */
static int  cbufszb;        /* in core buffer size in bytes (cf cbuf)  */
static int  g_fd;           /* cache file descr for get if not in core */
static int  p_fd;           /* cache file descr for put if not in core */
static short  *cbuf;        /* pointer to in-core cache buffer space   */
static long cachszb;        /* cache size in bytes (cf in core malloc) */
extern int  prt_cach;       /* diagnostic switch for cache routines    */
extern int  prt_mall;       /* diagnostic switch for malloc printout   */

cach_init(file_name, nbufbef, nbufaft, bufsz) char *file_name; {
   cach_file= file_name;
   bufszw   = bufsz;                    /* xfer to global variable      */
   bufszb   = bufszw  * sizeof(short);  /* buf size on bytes from words */
   nbuftrig = nbufbef + nbufaft;        /* on trig output nbuftrig bufs */
   size     = nbufbef;                  /* cache size in bufs           */
   cachszb  = nbufbef * bufszb;         /* cache size in bytes          */

   if(cach_file) {
      cbufszb = bufszb;
      if( (p_fd =  open(cach_file,    1)) < 0 ){
         if((p_fd = creat(cach_file, 0664)) < 0 )
            st_error(die,0,"Unable to create cache file %s",cach_file);
      }
      if((g_fd =  open(cach_file,    0)) < 0 )
            st_error(die,0,"Unable to open cache file %s",cach_file);
   }
   else {
      cbufszb = cachszb;
      if(cbufszb != cachszb)perr("online/cache: not enough core"); }

   if(prt_mall) st_error(0,0,"cache:   malloc(%u)\n", cbufszb);
   cbuf = (short *)malloc(cbufszb);
   if(cbuf == 0 || cbufszb == 0){    /******* malloc error *******/
      st_error(0,0,"**********  In-core cache is too large.  **********\n");
      st_error(0,0,"Need to re-adjust parameters or use disk-file cache.\n\n");
      prt_mall = prt_cach = 1; }   /* call st_error() after more diags */
   if(prt_mall) fprintf(stderr,"cache:   malloc(%u) = %u\n\n\n", cbufszb, cbuf);

   if(prt_cach){
      fprintf(stderr,"cache nbufs nbytes bufszw bufszb nbuftrig cbufszb\n");
      fprintf(stderr,"%5s %5d %6D %6d %6d %8d %7d\n\n", cach_file?"disk":"core",
         size, cachszb, bufszw, bufszb, nbuftrig, cbufszb); }

   if(cbuf == 0 || cbufszb == 0)
      st_error(0,0,"error in init_cach: attempt to malloc(%u)\n", cbufszb);

   if(conttape)initconttape();
}

short *getcach(n) {
   if(cach_file){
      if(lseek(g_fd, (long)n * (long)bufszb, 0) < 0 )
         st_error(die,0,"Unable to lseek in cache file");
      if(read(g_fd, cbuf, bufszb) != bufszb )
         st_error(die,0,"Unable to read cache file");
      return(cbuf);
   }
   return(cbuf + n*bufszw);
}

putcach(lpabuf) short *lpabuf; {                  /* called from a2dread */
   if(cach_file){
      if(write(p_fd,lpabuf,bufszb) != bufszb)
         st_error(die,0,"Unable to write full buffer to cache file");
      if(current+1 >= size)lseek(p_fd, 0L, 0); }
   else copyw(lpabuf, cbuf+current*bufszw, bufszw);
   if(conttape)writecont(lpabuf);
   if(++current >= size)current=0;   /* now index of oldest buffer    */
   if(valid     <  size)valid++;     /* oldest valid when valid==size */

   if(trig && valid==size){
      wrtfile( getcach(current) );
      trig--;
      if(!trig)trig_off(); }
}

trig_on() {
   if(prt_cach)fprintf(stderr,"\ncache:trig_on: %s trigger\n",trig?"old":"new");

   if(!trig)newfile();
   trig=nbuftrig;
}

static trig_off() {
   if(prt_cach)st_error(0,0,"\ncache:trig_off: trigger just went off\n");
   endfile();
}

static copyw(from,to,nwords) short *from, *to; {
   short *f, *t;
   f=from; t=to;
   while(nwords-- > 0) *t++ = *f++;
}

int fdconttape;
int connum,blocksz;

int initconttape()
{
   connum=5120/bufszw;
   blocksz=connum*bufszw*sizeof(short);
```

```
    fdconttape=open(conttap,O_WRONLY);
    if(fdconttape<=2) {
        st_error(0,0,"Unable to open tape %s for writing.\n",conttap);
        conttape=0;
        return(0);
    }
    conbuf=(short *)malloc(blocksz);
    if(conbuf==0) {
        st_error(0,0,"Unable to malloc(%d) for continuous tape.\n",blocksz);
        close(fdconttape);
        conttape=0;
        return(0);
    }
    fprintf(stderr,"Writing tape: %d buffers or %d bytes per block.\n",
        connum,blocksz);
    return(1);
}

int number=0;

writecont(lpabuf)
    short *lpabuf;
{
    int ret;

    copyw(lpabuf,conbuf+number*bufszw,bufszw);
    if(++number>=connum) {
        ret=write(fdconttape,conbuf,blocksz);
        if(ret<blocksz) {
            close(fdconttape);
            conttape=0;
            fprintf(stderr,
                "Continuous tape is finished. Wrote %d blocks of %d bytes.\n",
                number,blocksz);
            return(0);
        }
        number=0;
    }
}
```

```
/*  diag.c  --  diagnostic printouts for online.c  */

#include "eqdetector.h"

#define  NLINE     16
#define  MINN(x,y)  (x < y ? x : y)

wrt_params() {
   int i, j, k;
   long now, time();
   char *ctime();
   double spb = bufsz/(double)(nchan*rate);  /* secs per buf */
   now = time(0);

   fprintf(stderr,"Eqdetector: network %s: begin at local time: %s",
      network,ctime(&now));

   if(!prt_parm && !prt_diag){
      if(pseudo_a2d)
         fprintf(stderr,"data from pseudo a2d file %s, not a2d\n", pseudo_a2d);
      return; }

   fprintf(stderr,"        before    after    write aperture    sweep  rate\n");
   fprintf(stderr,"psecs %8g %8g %8g %8g %8g  1/%D\n",
      before, after, before+after, aperture, sweep, rate);
   fprintf(stderr,"samps %8D %8D %8D %8D %8D %5d\n",
      ns_before, ns_after, ns_write, ns_aperture, ns_sweep, 1);
   fprintf(stderr,"bufs  %8d %8d %8d %8.3g %8.3g\n",
      nb_before, nb_after, nb_before+nb_after,(nsw_aperture*sweep)/spb, sweep/spb);
   fprintf(stderr,"asecs %8g %8g %8g %8g %8g\n\n",nb_before*spb, nb_after*spb,
      (nb_before+nb_after)*spb,nsw_aperture*sweep, sweep);

   fprintf(stderr,
      "nsta decim const1/const2 thresh const3/const4 wav_inc bufsz nbuf nchan sec/buf\n")
   fprintf(stderr,"%4d %5d %6d/%-6d %6d %6d/%-6d %6d %5d %4d %5d %7g\n\n",
      nstations,decim,const1,const2,threshold,const3,const4,wav_inc,bufsz,
      nbuf,nchan,spb);

   wrt_snet(stanames, subnet, nstations);

   fprintf(stderr,"Triggering computation will be done on these stations:\n");
   fprintf(stderr,"\n");

   for(j=0; j<nlook; j+=NLINE){
      k = MINN(j+NLINE, nlook);
      for(i=j; i<k; i++)fprintf(stderr,"%4d ",        i ); fprintf(stderr,"\n");
      for(i=j; i<k; i++)fprintf(stderr,"%4d ",looksta[i]); fprintf(stderr,"\n");
      for(i=j; i<k; i++)fprintf(stderr,"%4s ", stanames[ looksta[i] ]);
      fprintf(stderr,"\n\n"); }

   if(cach_file)fprintf(stderr,"cache is on disk file %s\n",cach_file);
         else fprintf(stderr,"cache is in core\n");
   if(tape_file)fprintf(stderr,"output to tape: %s\n",           tape_file);
   if(disk_file)fprintf(stderr,"output to disk: %s<date+time>\n",disk_file);
   if(diffsam  )fprintf(stderr,"samples differenced before computation\n");
   if(dyn_flag )fprintf(stderr,"dynamic flags enabled\n");
   if(pseudo_a2d)fprintf(stderr,"data from pseudo a2d file %s, not a2d\n",
      pseudo_a2d);
   fprintf(stderr,"\n");
}

wrt_snet(staname, subnet, nsta) char staname[][NSZ]; int *subnet; {
   int i, j, k, nsub=0, lensub=0, thresh, st_num;

   for(j=0; j<nsta; j+=NLINE){
      k = MINN(j+NLINE, nsta);
      for(i=j; i<k; i++)fprintf(stderr,"%4d ",i);             fprintf(stderr,"\n");
      for(i=j; i<k; i++)fprintf(stderr,"%4s ",staname[i]);    fprintf(stderr,"\n");
      fprintf(stderr,"\n"); }

   while(    (thresh  = subnet[lensub++]) != -1 ) {
      fprintf(stderr,"\nsubnet %d: %3d ", nsub++, thresh);
      while( (st_num  = subnet[lensub++]) != -1 )
         fprintf(stderr," %s", staname[st_num]); }

   fprintf(stderr,"\n\nsubnet vec:");
   for(i=0; i<lensub; i++)fprintf(stderr," %d",subnet[i]); fprintf(stderr,"\n\n");
}

#define PRT(mes,var,fmt) {fprintf(stderr,"%s:",mes);            \
                          for(i=j; i<k; i++){           \
                               st_num = looksta[i];     \
                               fprintf(stderr,fmt ,var); }     \
                          fprintf(stderr,"\n"); }

wrt_diags(force) {
   int i, j, k, st_num;

 /*  fprintf(stderr,"\n");*/

   for(j=0; j<nlook; j+=NLINE-1){
      k = MINN(j+NLINE-1, nlook);
                          PRT("nam",          stanames[st_num]," %4s");
                          PRT("num",                 st_num, "%5d");
      if(prt_st_av || force)PRT("sta",            stas[st_num], "%5d");
      if(prt_st_av || force)PRT("lta",            ltas[st_num], "%5d");
      if(prt_st_abs_av || force)PRT("staa",   abs_stas[st_num], "%5d");
      if(prt_st_abs_av || force)PRT("ltaa",   abs_ltas[st_num], "%5d");
      if(prt_etas || force)PRT("eta",             etas[st_num], "%5d");
      if(prt_strg || force)PRT("trg",      sta_trigger[st_num], "%5d");
      fprintf(stderr,"\n"); }
}

perr(message) char *message; {
   st_error(0,0,"online: error on statement \"%s\"\n",message);
   exit(1);
}
```

```
/*                      d y n f l a g . c
 *
 *  Subroutine dynflag(name_of_flag) for controlling the calling
 *  program dynamically through use of a designated named file.
 *
 *  The flag is signaled either by creation or touching of a file.
 *
 *  James W. Herriot    September 1, 1986    Beijing, China
 */

#include "eqdetector.h"
#include <sys/types.h>
#include <sys/stat.h>

#define  MAXDICT   16
#define  MAXLEN    16
#define  eq        !strcmp

ck_flags() {
   static save_st_av, save_st_abs_av, save_etas, save_strg, save_diag, notfirstime;

   if(!notfirstime){                        /* remember initial state of flags */
      save_diag = prt_diag;
      save_st_av = prt_st_av;
      save_st_abs_av = prt_st_abs_av;
      save_etas = prt_etas;
      save_strg = prt_strg;
      notfirstime = 1; }

   if(!save_st_av) prt_st_av = dynflag("st_av");   /* if not init set then dyn */
   if(!save_st_abs_av) prt_st_abs_av = dynflag("st_abs_av");
   if(!save_etas) prt_etas = dynflag("etas");
   if(!save_strg) prt_strg = dynflag("strg");

   if( dynflag("all") ) prt_st_av = prt_st_abs_av = prt_etas = prt_strg = 1;

   if( !save_diag ){
       if(prt_st_av || prt_st_abs_av || prt_etas || prt_strg) prt_diag = 1;
       else                                                   prt_diag = 0; }

   if( dynflag("trigger") )fake_trig = 1;
   if( dynflag("harikari") )
      st_error(0,0,"\n(simulated) LPA read failure: please restart online program");
}

char statdict[MAXDICT][MAXLEN];
long stattime[MAXDICT];

lookup(name) {
   int i;

   for(i=0; i<MAXDICT && statdict[i][0]; i++)
      if( eq(name, statdict[i]) ) return(i);

   if(i == MAXDICT){ fprintf(stderr,"stat dictionary overflow\n"); exit(1); }

   strcpy(statdict[i], name);
   stattime[i] = 0;

   return(i);
}

long gettime(name) char *name; { return( stattime[lookup(name)] ); }
puttime(name, t)   char *name; long t; { stattime[lookup(name)] = t; }

long filtime(name) char *name; {
   int fd;
   struct stat statb;

   fd = open(name, 0);
   if(fd < 0) return(0);

   fstat(fd, &statb);
   close(fd);
   return(statb.st_mtime);
}

dynflag(name) char *name; {
   int fd;
   long t, f;

   t = gettime(name);     /* rtns last filtime or 0 1st time     */
   f = filtime(name);     /* rtns mod time or 0 if nonexistant   */
   puttime(name, f);
   /* fprintf(stderr,"dynflag(%s) t=%D f=%D\n", name, t, f); */

   return( f > t);        /* file must exist and be new or newer */
}

#ifdef DEBUG
main(argc, argv) char **argv; {
   int s, i;

   if(argc < 3){
      fprintf(stderr,"Usage: dynflag sleeptime names\n");
      exit(1); }

   s = atof(argv[1]);

   while(1) {
      for(i=2; i<argc; i++)
         if(dynflag(argv[i])) fprintf(stderr,"Got dynflag: %s\n", argv[i]);
      sleep(s); }
}
#endif
```

```
/*  eqdetector.c  --  Online seismic processing system with LPA or CDD Data Sampler
 *
 *                      ---  J. Harriot  and  D. Fong  ---
 *                      USGS, Menlo Park, CA, August, 1982
 *                      ---  PLWard, August 1987, June, July, 1988
 *
 *  Program features:
 *
 *    1. DMA from LPA (buffer size and number of bufs set dynamically).
 *    2. Cache in core or on disk file (raw or cooked) as desired.
 *    3  Automatic triggering and all i/o in one real-time process.
 *    4. Final output on disk or tape or both as desired.
 *    5. Output is compatable with Ping and Pong programs.
 *    6. Most parameters dynamically set from command line.
 *    7. Many seismically important arguments are in seconds (not bufs).
 *    8. When executed with no args, options are documented.
 */

#define GROUPID    2
#define USERID     2
#define DETECTDIR  "/usr/eqdetector/detected"
#define DETECTLOG  "detect.log"
#define PERMISSIONS 0133                /* umask */

#include <stdio.h>
#include <sys/file.h>
#include <signal.h>
#include "eqdetector.h"

char *progname;

die(n)
   int n;
{
   fprintf(stderr,"%s: Die called with arg=%d\n",progname,n);
   exit(n);
}

/*------------------ set_defaults & param_init ----------------------*/

static char *eqdot = "mx.";         /* if no disk or tape use this    */
static char *onsta = "online.sta"; /* default network input file     */

set_defaults() {

   nchan     =   16 ;              /* number of a2d channels         */
   before    =   20.;              /* seconds of data save bef trig  */
   after     =   60.;              /* seconds of data save aft trig  */
   aperture  =   20.;              /* secs for coincident statn trig */
   sweep     =    3.;              /* secs of data for short term av */
   decim     =    1 ;              /* decimation for trigger calcing */
   bufsz     = 2048 ;              /* buffer size for a2d reads, shorts */
   const1    =    2 ;              /* numerator of long-term weight  */
   const2    =    1 ;              /* denominator of long-term weight*/
   threshold =    5 ;              /* minimum eta for a trigger      */
   const3    =    7 ;              /* weigted average numerator      */
   const4    =    8 ;              /* weigted average denominator    */
   wav_inc   =    1 ;              /* weigted average increment      */
   diffsam   =    0 ;              /* don't diff samps bef computatn */
   dyn_flag  =    0 ;              /* dynamic flags are not enabled  */
#ifdef LPA
   rate      =   50 ;              /* sampling rate in samples/sec   */
   nbuf      =    2 ;              /* num of bufs for a2d to use     */
#endif LPA
#ifndef LPA
   rate      =  100 ;              /* sampling rate in samples/sec   */
   nbuf      =    1 ;              /* num of bufs for a2d to use     */
#endif
}

param_init() {
   long nhunk   = (long)nchan    * decim;       /* getsweep sz words */

   bufsz        = (bufsz/nhunk) * nhunk;          /* want multiple   */
   bufszb       =  bufsz        * sizeof(short); /* in bytes        */

   ns_before    = before        * rate;           /* num samples ... */
   ns_after     = after         * rate;
   ns_write     = ns_before     + ns_after;
   ns_aperture  = (aperture     * rate) / decim;
   ns_sweep     = (sweep        * rate) / decim;  /* Note: decim div */

   nb_before    = (ns_before    * nchan + bufsz - 1) / bufsz;
   nb_after     = (ns_after     * nchan + bufsz - 1) / bufsz;

   nsw_aperture = aperture      / sweep;       /* aperture of sweeps */
}

#ifdef LPA       /* for 44unix, version 7 */
onquit(){signal(SIGQUIT,onquit); fake_trig=1; fprintf(stderr,"\nFake trigger\n");}
#endif LPA
#ifndef LPA      /* for unix version 4.2bsd */
onquit(){fake_trig=1; fprintf(stderr,"fake trigger\n");}
#endif

/*----------------------------- main ---------------------------------*/
main(argc,argv)
   int argc;
   char **argv;
{
   FILE *tempfile,*efopen();

   progname=argv[0];
 /*  umask(PERMISSIONS);
   chdir(DETECTDIR);
   if(setgid(GROUPID))st_error(0,0,"unable to setgid");
   if(setuid(USERID ))st_error(0,0,"unable to setuid");
*/
   set_defaults();                          /* set all parameters to defaults */
   proc_args(argc,argv);          /* change parameters given in command line */

   /* Disconnect standard input from terminal and attach to /dev/null */
/*   close(fileno(stdin));
   open("/dev/null", O_RDONLY);
*/
   /* Divert standard output and standard error to a logfile */
/*   tempfile=efopen(DETECTLOG,"a+");
   fclose(stdout);
   fclose(stderr);
   dup(fileno(tempfile));
   dup(fileno(tempfile));
   fclose(tempfile);

*/   /* Increase priority where initial priority is unknown */
/*   if(nice(-40)!=0) st_error(0,0,"Unable to set nice=-20");
   if(nice( 16)!=0) st_error(0,0,"Unable to set nice=-4");
*/
   if(!disk_file && !tape_file)disk_file = eqdot;        /* final output */
   head_init();                             /* read in online.sta net file  */
   param_init();                            /* derived parameters calulated */
```

```
   wrt_params();                        /* display values of parameters */
   if(exit_early)exit(0);               /* for testing online startup   */
   if(fake_ok)signal(SIGQUIT, onquit);  /* fake trigger on cntl-\ key   */

   cach_init(cach_file, nb_before, nb_after, bufsz);       /* init cache */
   a2dopen(rate, 0, 1, nchan, nbuf, bufsz);                /* open a2d   */

#ifdef LPA
   if(!not_rtp && !pseudo_a2d && rtp(1) == 0)    /* real time process */
          fprintf(stderr,"online process %d set to run RTP\n"    ,getpid());
      else fprintf(stderr,"online process %d NOT set to run RTP\n",getpid());
#endif LPA
   eqdetection();
}

/* eqdetection: calculate running averages and trigger on suspected events */
eqdetection()
{
   int    i,s,j,samp,diff;
   short  *psamp;
   static int lastsamp[MAX_STA],thissamp;

   for(sweepnum=0; 1; sweepnum++){
      for(s=0; s<nstations; s++)st_sum[s]=st_abs_sum[s]=0;

      if(shortcut && trig){                        /* speed short cut    */
         for(i=0; i<ns_sweep; i++)getsweep();      /* ignore a2d data    */
         continue;                                 /* don't call trigger */
      }
      for(i=0; i<ns_sweep; i++){                   /* for each sample    */
         psamp = getsweep();                       /* get 1 sample/stat  */

       /*for(s=0;s<16;s++)fprintf(stderr,"%d ",psamp[s]&0xF);
         fprintf(stderr,"\n"); */  /* print headers to show a2d working */

         for(j=0; j<nlook; j++){                   /* just some stations */
            s    = looksta[j];                     /* get station number */
            samp = psamp[s] - A2D_ZERO;            /* get a2d sample     */

            if(diffsam){                           /* use differences?   */
               thissamp   =samp;                   /* save curr sample   */
               samp       =thissamp-lastsamp[s];   /* diff with last     */
               lastsamp[s]=thissamp;               /* remember last samp */
            }
            diff           = samp - ltas[s];       /* diff for abs func  */
            st_sum[s]     += samp;                 /* add to reg sums    */
            st_abs_sum[s] += Abs(diff);            /* and rectified sums */
         }
      }
      if( dyn_flag  ) ck_flags();                  /* ck dynamic flags   */
      if( trigger() ) trig_on();                   /* may trigger        */
   }
}

/* trigger: done once after each ns_sweep=sweep+rate/decim              */

trigger() {                  /* calc etas, station and master triggers */
   register i, s;            /* return: 1=master trigger, 0=no trigger */
   int eta, sta, abs_sta, new_lta, new_abs_lta, thresh;
   static firsttime=1;

   if(firsttime){
      for(i=0; i< nlook; i++) {
         s = looksta[i];
         ltas[s]=st_sum[s]/ns_sweep;
         abs_ltas[s]=st_abs_sum[s]/ns_sweep; }
      firsttime=0;
      if(prt_diag)wrt_diags(0);
      return(0);
   }

   if(fake_trig) { fake_trig=0; return(1); }          /* one-time fake trig */

   for(i=0; i< nstations; i++) {
      s = looksta[i];
      sta=st_sum[s]/ns_sweep;
      abs_sta=st_abs_sum[s]/ns_sweep;

      eta = abs_sta -
            (const1 * abs_ltas[s]) / const2 -
            abs(ltas[s]-sta) - threshold;

      new_lta = (sta  +  const3 * ltas[s]) / const4;       /* weighted avg   */
      if(new_lta == ltas[s]){                              /* if no change   */
         if(sta > new_lta)new_lta++;                       /* favor sta...   */
         if(sta < new_lta)new_lta--;                       /* over long avg  */
      }
      new_abs_lta=(abs_sta+const3*abs_ltas[s])/const4;     /* weighted avg   */
      if(new_abs_lta == abs_ltas[s]){                      /* if no change   */
         if(abs_sta > new_abs_lta)new_abs_lta+=wav_inc;     /* favor sta...   */
         if(abs_sta < new_abs_lta)new_abs_lta-=wav_inc;     /* over long avg  */
      }
      ltas[s] = new_lta;
      abs_ltas[s] = new_abs_lta;

      if(sta_trigger[s])sta_trigger[s]--;                  /* age trigger    */
      if(eta > 0)sta_trigger[s] = nsw_aperture;            /* set statn trig */

      etas[s]=eta; stas[s]=sta; abs_stas[s]=abs_sta;       /* save for diagn */
   }

   if(prt_diag)wrt_diags(0);
   if(log_file)fflush(stderr);

   /* loop thru subnet vector looking for a particular subnet
    * whose number of triggered stations is >= to thresh for that net.
    * Structure of subnet where thresh=threshhold and sn=station num is:
    * <thresh> <sn> <sn> ... -1 <thresh> <sn> <sn> ... -1 ... -1 -1
    * ie, thresh-sn groups separated by -1 all terminated by -1 -1
    */

   i=0;
   while(    (thresh  = subnet[i++])  != -1 )
      while( (s       = subnet[i++])  != -1 )
         if( (thresh -= sta_trigger[s] >  0 )  <= 0)return(1);

   return(0);
}
```

```
/* getsweep.c -- a2d read routines for stdio-like access
 *
 * The following routines are called from online.c:
 *      open and ioctl for SCSI atod or for lpa call
 *      a2dopen(rate, ichan, chani, nchan, nbuf, bufsz)
 *           Eg: 50L,     0,     1,    16,    2,  2048
 *           Opens a2d.  Locks in core so no mallocs after this point.
 *      getsweep()
 *           Returns pointer to a vector of samples of one time slice.
 *           If decim != 1 time slices will be skipped accordingly.
 *      a2dread()
 *           This routine is only called by getsweep or a getsweep macro.
 *           Lparead reads one buffer from a2d and calls putcach.
 */
# include "eqdetector.h"
# include <stdio.h>
# include <sys/types.h>
# include <sys/stat.h>
# include <sys/timeb.h>

#ifdef LPA
# include <whoami.h>
# include "lpa.h"
# define BSIZE 1
#endif LPA

#ifndef LPA
# define BSIZE 4096
# include <sys/file.h>
# include <sys/ioctl.h>
# include <sys/types.h>
static char A2D[] =   "/dev/rst1";

#include <signal.h>
int intr=0;
onintr() {
   intr=1;
}

#endif


# define NBPW      sizeof(short)            /* number of bytes per word */

static double bef_time, aft_time, now_time;
static double clock() {
   struct timeb tp;
   ftime( &tp );
   return( tp.time + tp.millitm * .001 );
}

       int  prt_a2d;    /* diagnostic switch set in args.c */
       int  prt_tim;    /* diagnostic switch to print time */
       int  decim;      /* decimation factor for getsweep    */
       int  fake_ok;    /* allow fake trigger from signal  */
       char *pseudo_a2d;/* data from file instead of a2d   */
       int  prt_mall;   /* print out mallocs               */


static char *a2dbuff;   /* entire contiguous a2d data area  */
static int   a2dfd;     /* file descriptor for a2d          */
static int   a2dbufsz;  /* size of one a2d buffer in bytes  */
static char  a2dbufn;   /* idx of current buff (0..nchan-1) */
static char *a2dbed;    /* pointer to current buffer        */
       int   a2dsize;   /* size of sample vector  in bytes  */
       char *a2dptr;    /* pointer to next sample vector    */
       int   a2dcount;  /* num bytes still valid in cur buf */

a2dopen(scrate,ichan,chani,nchan,nbuf,bufsz)
   int ichan,chani,nchan,nbuf,bufsz;
   long scrate;
{
   a2dsize   = nchan * (decim>1 ? decim : 1) * NBPW;
   a2dbufsz  = (bufsz/nchan)*nchan * NBPW;

   if(prt_mall) fprintf(stderr,"a2dopen: malloc(%u)\n", a2dbufsz*nbuf);
   a2dbuff = (char *)malloc(a2dbufsz*nbuf);

#ifdef LPA
   if(a2dbuff == 0 || (long)a2dbufsz * nbuf > 65535L){  /* malloc err */
      fprintf(stderr,"**********   a2d buffers are too large.   **********\n");
      fprintf(stderr,"Need to re-adjust parameters or use disk-file cache.\n\n");
#endif LPA
#ifndef LPA
   if(a2dbuff == 0){                              /* malloc err */
#endif
       st_error(0,0,"a2dopen: malloc(%u)\n", a2dbufsz*nbuf);
   }
   if(prt_mall) fprintf(stderr,"a2dopen: malloc(%u) = %u\n",
                        a2dbufsz*nbuf, a2dbuff);

   if(pseudo_a2d) {
      if(prt_a2d) fprintf(stderr,"pseudo_a2d to be used. Lpa not opened.\n");
      a2dfd = open_pseudo();
      return;
   }

#ifdef LPA
   if( (a2dfd = adopen(a2dbuff, a2dbufsz/NBPW, nbuf,
                       scrate*nchan, ichan,chani,nchan)) < 0  )
         st_error(0,0,"online/a2dopen: cannot open a2d");
   if(prt_a2d) {
      fprintf(stderr,"a2d: scrate ichan chani nchan nbuf bufsz  size bufsz\n");
      fprintf(stderr,"     %6D %5d %5d %5d %4d %5d  %4d %5d\n\n",
            scrate,ichan,chani,nchan,nbuf,bufsz, a2dsize,a2dbufsz);
   }
}
#endif LPA
#ifndef LPA
   signal(SIGINT,onintr);
   a2dfd = atod_open(A2D,1000/rate,nchan);
   if(prt_a2d){
      fprintf(stderr,"a2d: scrate ichan chani nchan nbuf bufsz  size bufsz\n");
      fprintf(stderr,"     %6D %5d %5d %5d %4d %5d  %4d %5d\n\n",
            scrate,ichan,chani,nchan,nbuf,bufsz, a2dsize,a2dbufsz); }
}
#endif

short *getsweep() {
   short *sp;
   char *a2dread();
register i;

   if(a2dcount <= 0)a2dread();

   a2dcount -= a2dsize;
   sp        = (short *)a2dptr;
   a2dptr   += a2dsize;
   return (sp);
}
```

```
char *a2dread() {
   int r;

  /* if(prt_a2d)fprintf(stderr,"a2d"); */                          /**** before ****/
   if(prt_tim){
     now_time=clock();
     if(aft_time!=0)fprintf(stderr,"calc time = %-6.5g/%-6.5g = %-6.3g\n",
         now_time-aft_time,    now_time-bef_time,
        (now_time-aft_time) / (now_time-bef_time)  );
     bef_time=now_time; }


#ifdef LPA
   if(pseudo_a2d) r = get_pseudo(&a2dbufn);        /* get data from a file */
   else           r = read(a2dfd,&a2dbufn,BSIZE);  /***** read the a2d *****/
   if(r != BSIZE && fake_ok) r = read(a2dfd,&a2dbufn,BSIZE);  /* try again */
#endif
#ifndef LPA
   if(intr){
      fprintf(stderr,"\nOnline program interrupted by Cntr-C.\n");
      if(stderr!=stdout)
         fprintf(stdout,"\nOnline program interrupted by Cntr-C.\n");
      close(a2dfd);
      exit(0);
   }
   if(pseudo_a2d) r = get_pseudo(a2dbuff);          /* get data from a file */
   else           r = read(a2dfd,a2dbuff,BSIZE);    /***** read the a2d *****/
   if(r==EOF && !pseudo_a2d) {
      if(atod_geterr(a2dfd) == 4) {                  /* a2d buffer is full */
         st_error(0,0," AtoD buffer overflow. Resetting.");
         atod_rezero(a2dfd);
      }
      r = read(a2dfd,a2dbuff,BSIZE);                /***** read the a2d *****/
   }
   if(r != BSIZE && fake_ok) r = read(a2dfd,a2dbuff,BSIZE);    /* try again */
#endif

   if(r != BSIZE)st_error(0,0,"\n A2D read failure: please restart online program");

   if(prt_tim)aft_time=clock();
#ifdef LPA
   if(prt_a2d)fprintf(stderr,"(%d) ", a2dbufn);              /**** after ****/
   a2dcount = a2dbufsz;
   a2dbed   = a2dptr = a2dbuff + a2dbufn * a2dbufsz;
#endif
#ifndef LPA
 /*  if(prt_a2d)fprintf(stderr,"(%d) ", r);   */               /**** after ****/
   a2dcount = a2dbufsz;
   a2dbed   = a2dptr = a2dbuff;
#endif

fflush(stderr);


   putcach(a2dbed);   /* put cache is in cache.c */

   return a2dbed;
}

open_pseudo()
{
   int i, n, r, fd;
   char *h, buf[2048];

 /*  n = sizeof_h();
NEED TO FIX THIS UP TO WORK 7/27/88 PLN
*/
   h = n < 2048 ? buf : (char *)malloc(n);   /* try to avoid malloc to save space */
   if(h == 0) st_error(0,0,"Can't malloc(%d) for for reading a2d header.", n);

   fd = open(pseudo_a2d, 0);
   if(fd < 0) st_error(0,0,"Can't open %s as pseudo a2d file", pseudo_a2d);

   r = read(fd, h, n);
   if(r != n) st_error(0,0,"Error in reading pseudo a2d file header. r=%d", r);
   free(h);

   if(prt_a2d) fprintf(stderr,"open_pseudo() called n=%d\n", n);

   return(fd);
}

get_pseudo(n) int *n; {
   int r;

   r = read(a2dfd, a2dbuff, a2dbufsz);

   if(prt_a2d) fprintf(stderr,"get_pseudo: %d = read(%d, %u, %u)\n",
                                    r, a2dfd, a2dbuff, a2dbufsz);

   if(r != a2dbufsz) st_error(0,0,"End of file reached on %s pseudo a2d file",
                              pseudo_a2d);
   *n = 0;
   return(1);
}
```

```
#define MSK4 0xF                                /* 4 least significant bits */

gettim_(tbuf,yr,dy,hr,mn,sec)
   short *tbuf;
   int *yr,*dy,*hr,*min;
   float *sec;
{
   *yr=tbuf[1]&MSK4;
   *dy=(tbuf[2] & MSK4)*100+(tbuf[3] & MSK4)*10+(tbuf[4] & MSK4);
   *hr=(tbuf[5] & MSK4)*10+(tbuf[6] & MSK4);
   *mn=(tbuf[7] & MSK4)*10+(tbuf[8] & MSK4);
   *sec=(tbuf[9] & MSK4)*10.0+(tbuf[10] & MSK4)*1.0+(tbuf[11] & MSK4)*0.1+
      (tbuf[12] & MSK4)*0.01+(tbuf[13] & MSK4)*0.001;
}
```

137

```
#
/*
 * adopen() --
 * open lpa for a / d input.
 * for now,
 * exits on error?
 */
# include <whoami.h>
# include <sys/param.h>
# include "lpa.h"
# include <sgtty.h>

# include "key.h"

struct lpastty sg;

static char A2D[] =     "/dev/ad0";
static char MCODE[] =   "/kmcl1/dmast.dat";
static char DMDT[] =    "/kmcl1/dmdt.dat";

int adopen(buf,bufsiz,nbuf,rate,ichan,chani,nchan)
    short *buf;                 /*ptr to buffers*/
    short bufsiz;               /*buf size in words*/
    short nbuf;                 /*#buffers*/
    long rate;                  /*sample rate in hz*/
    short ichan,chani,nchan;    /*scan sequence*/
{
    extern char *malloc();
    char junk[2048];
    reg int nread;
    int ffd,fd;
    /*expand preparatory to locking in core*/
    free(malloc(2048));
    stakup(10);
    if( (ffd = open(A2D,2))<0 )
        st_error(0,0,"can't open lpa file %s",A2D);
    if( (fd = open(MCODE,0))<0 )
        st_error(0,0,"can't open microcode %s",MCODE);
    if( (nread = read(fd,junk,sizo junk))<0 )
        st_error(0,0,"read error on microde %s",MCODE);
    if( write(ffd,junk,nread)!=nread )
        st_error(0,0,"write error on microcode %s",MCODE);
    close(fd);
    if( (fd = open(DMDT,0))<0 )
        st_error(0,0,"can't open table %s",DMDT);
    if( (nread = read(fd,junk,sizo junk))<0 )
        st_error(0,0,"read error on table %s",DMDT);
    if( write(ffd,junk,nread)!=nread )
        st_error(0,0,"write error on table %s",DMDT);
    close(fd);
    sg.lbufvad = buf;
    sg.lsrate = -1000000L/rate;
    sg.lbufdesc = (nbuf-1)<<13 | bufsiz;
    if( nchan<=0 )
        nchan = 1;
    if( nchan>1 && chani<=0 )
        chani = 1;
    if( chani!=0 )
        /*
         *sequential mode
         */
        sg.lmochan = L_MCHN | nchan;
    else
        sg.lmochan = ichan;
    sg.lcinc = chani<<8 | ichan;
    sg.lclkbits = ENACTR|RlM|MFIE|MRI;
    /*start it up*/
    if( ioctl(ffd,SETLPAP,&sg)<0 )
        st_error(0,0,"init error on lpa file %s",A2D);
    return ffd;
}
static int stakup(n)
    int n;
{
    int junk[10];
    if( --n>=0 )
        return stakup(n);
}
```

```
/*  out.c  --  final output routines for online.c  */

#include <stdio.h>
#include "eqdetector.h"

static FILE *d_fd, *t_fd;                       /* disk and tape file descriptors */

#include <sys/time.h>
#define MSK4 0xF                                /* 4 least significant bits */

int striptime(tbuf,mstime,locdiff,timstr)
   short *tbuf,*locdiff;
   MS_TIME *mstime;
   char *timstr;
{
   int yr,dy,hr,mn,err,i;
   double sec;
   MS_TIME make_mstime();
   extern char *list_mstime();
   int diff;
   struct timeval tp;
   struct timezone tzp;
   struct tm *gt;

   diff=gettimeofday(&tp,&tzp);          /* calculate difference GMT-local time */
   if(diff== -1) {
      st_error(0,0,"Unable to get time of day from UNIX");
      return(0);
   }
   gt=localtime(&tp);
   diff=60*gt->tm_hour+gt->tm_min;
   gt=gmtime(&tp);
   diff=(60*gt->tm_hour+gt->tm_min)-diff;
   if(diff>720)diff-=1440;
   else if(diff<-720)diff+=1440;
   *locdiff=diff;

   i=tbuf[1]&MSK4;                              /* calculate the mstime          */
   yr=gt->tm_year+1900;
   if(yr%10!=i) {
      dy=1900+((gt->tm_year/10)*10)+i;
      if(dy-yr>1)dy-=10;
      else if(dy-yr<1)dy+=10;
      yr=dy;
   }
   dy=(tbuf[2] & MSK4)*100+(tbuf[3] & MSK4)*10+(tbuf[4] & MSK4);
   hr=(tbuf[5] & MSK4)*10+(tbuf[6] & MSK4);
   mn=(tbuf[7] & MSK4)*10+(tbuf[8] & MSK4);
   sec=(tbuf[9] & MSK4)*10.0+(tbuf[10] & MSK4)*1.0+(tbuf[11] & MSK4)*0.1+
      (tbuf[12] & MSK4)*0.01+(tbuf[13] & MSK4)*0.001;
   *mstime=make_mstime(yr,0,dy,hr,mn,sec);
   strcpy(timstr,list_mstime(*mstime,1));

   if((err=tbuf[14]&MSK4))                      /* check magic and error bits    */
      st_error(0,0,"Clock error bits at time %s=%d",timstr,err);
   if((i=tbuf[15]&MSK4) != 14)
      st_error(0,0,"Data error: Magic time bits at time %s=%d",timstr,i);
}

newfile()
{
   char   full_name[100], *unit_num;
   int    i,j;
   long   time(),timep;
   char   *ctime(),timestring[14];
   struct tm *gmtime(), *stmp;
   struct muxdata md;
   struct trigsetting ts;
   struct triggers tr;
   extern short *getcach();
   extern int current;

   event_num++;
   wrt_num=0;

   if(no_head)return;          /* dont write header */

   st_init(MUXDATA,&md);
   strncpy(md.netname,network,4);
   striptime(getcach(current),&md.begintime,&md.loctime,timestring);
   fprintf(stderr,"Event detected at %s\n",timestring);
   md.numchans=nstations;
   md.dig_rate=rate;
   md.typedata='r';

   if(disk_file){
      sprintf(full_name,"%s%s",disk_file,timestring);
      d_fd=st_open(full_name,"w");
   }

   if(tape_file){
      if(xchg_tape && event_num > xchg_tape){
         unit_num = tape_file + strlen(tape_file) - 1;
         *unit_num = *unit_num == '1' ? '3' : '1';
         /*????? event_num = 0; What about disk_file naming ?????*/
         fprintf(stderr,"changing tape output to %s\n", tape_file);
      }
      if((t_fd=fopen(tape_file,"w")) != NULL)
         st_error(die,0,"Unable to open tape file %s",tape_file);
   }

   if(prt_trig) {
      st_init(TRIGSETTING,&ts);
      strncpy(ts.netwname,network,4);
      ts.beginttime=md.begintime;
      ts.const1=const1;
      ts.const2=const2;
      ts.threshold=threshold;
      ts.const3=const3;
      ts.const4=const4;
      ts.wav_inc=wav_inc;
      ts.sweep=sweep;
      ts.aperture=aperture;
      st_put(&ts,TRIGSETTING,sizeof(struct trigsetting),d_fd);
      for(j=0; j<nlook; j++){
         i=looksta[j];
         st_init(TRIGGERS,&tr);
         strncpy(tr.tr_name.network,stids[i].network,4);
         strncpy(tr.tr_name.st_name,stids[i].st_name,5);
         tr.tr_name.component=stids[i].component;
         tr.tr_name.inst_type=stids[i].inst_type;
         tr.sta=stas[i];
         tr.lta=ltas[i];
         tr.abs_sta=abs_stas[i];
         tr.abs_lta=abs_ltas[i];
         tr.trig_value=etas[i];
         tr.num_triggers=sta_trigger[i];
         tr.trig_time=md.begintime;
         st_put(&tr,TRIGGERS,sizeof(struct triggers),d_fd);
      }
```

```
    }

    st_put(&md,MUXDATA,sizeof(struct muxdata),d_fd);
    if(prt_vbse)wrt_diags(1);
}

wrtfile(buf) short *buf; {
    if(disk_file){
        if(fwrite(buf,1,bufszb,d_fd) != bufszb)
            st_error(0,0,"Unable to write full data buffer to disk");
        if(prt_out)fprintf(stderr,"disk");
    }
    if(tape_file){
        if(fwrite(buf,1,bufszb,t_fd) != bufszb)
            st_error(0,0,"Unable to write full data buffer to tape");
        if(prt_out)fprintf(stderr,"tape");
    }
    if(prt_out)fprintf(stderr,"(%d) ", wrt_num);
    wrt_num++;
    rec_num++;
}

endfile()
{
    if(disk_file)
        if(fclose(d_fd)==EOF)st_error(die,0,"Unable to close disk file");

    if(tape_file)
        if(fclose(t_fd)==EOF)st_error(die,0,"Unable to close tape file");
}

head_init()
{
    int i, s, stamask[MAX_STA];

    nstations = getstats(stat_file, stanames);
    getsubnets(subnet_file, subnet);
    if(nstations > MAX_STA)st_error(0,0,"Too many stations %d", nstations);

    for(i=0; i<nstations; i++)stamask[i] = 0;
    i=0;
    while( subnet[i++] != -1)
        while( (s=subnet[i++]) != -1) stamask[s]++;
    for(nlook=s=0; s<nstations; s++)if(stamask[s])looksta[nlook++] = s;
}

int getstats(stats_file,staname)        /* read stationcomp structure information */
    char *stats_file, staname[][NSZ];
{
    FILE *statfile;
    struct stationcomp *sc_ptr;
    int i,j,maxstat,st_type,st_len;

    maxstat= -1;
    statfile=st_open(stats_file,"r");
    i= -1;
    while(st_get(&sc_ptr,&st_type,&st_len,statfile)!=EOF) {
        if(st_type==STATIONCOMP) {
            if(sc_ptr->channel==(short)NODATA)
                st_error(die,0,"Channel number not assigned for station %s",
                    sc_ptr->sc_name.st_name);
            if(sc_ptr->channel>=MAX_STA)
                st_error(die,0,"Channel number too large. %d>=%d",
                    sc_ptr->channel,MAX_STA);
            j=sc_ptr->channel-1;
            if(j>maxstat)maxstat=j;
            if(j==1)strncpy(network,sc_ptr->sc_name.network,4);
            strncpy(staname[j],sc_ptr->sc_name.st_name,5);
            strncpy(stids[j].network,sc_ptr->sc_name.network,4);
            strncpy(stids[j].st_name,sc_ptr->sc_name.st_name,5);
            stids[j].component=sc_ptr->sc_name.component;
            stids[j].inst_type=sc_ptr->sc_name.inst_type;
            staname[j][4]='\0';
        }
        st_free(sc_ptr);
    }
    st_close(statfile);
    return(maxstat+1);
}

#define  EQ(strg)    ( strcmp(s,strg)       == 0 )
#define  GET_TOK     ( fscanf(fp, "%s", s) == 1 )
static char *rdaccess  = "r";

getsubnets(fname, subnet)
    char *fname;
    int  *subnet;
{
    char s[128];
    int  net=1, nsta, nsub;
    FILE *fp, *fopen();

    nsub=0;
    if((fp = fopen(fname, rdaccess)) == 0 )
        st_error(die,0,"Unable to open subnet file %s",fname);
    while( GET_TOK ) {
        if( EQ("/*") )while( GET_TOK && !EQ("*/") );    /* skip comments     */
        else if(*s>='0' && *s<='9') {                    /* read subnets      */
            if(nsub)subnet[nsub++]= -1;
            subnet[nsub++]=atoi(s);
        }
        else subnet[nsub++] = snam2snum(s, stanames, nstations);
    }
    subnet[nsub++] = -1;
    subnet[nsub++] = -1;
    fclose(fp);
}

snam2snum(s, staname, nsta)
    char *s, staname[][NSZ];
    int nsta;
{
    int i;
    for(i=0; i<nsta; i++)if( EQ(staname[i]) )return(i);
    fprintf(stderr,"online:read net: unknown station name %s\n", s);
    exit(1);
}
```

```
while
   echo restarting online system
do
   online -v -o online.log
done
```

101

```
/*  online.h  --  Online seismic processing system with LPA
 *
 *                      ---  J. Herriot  and  D. Fong  ---
 *                      USGS, Menlo Park, CA, August, 1982
 *
 *  Program features:
 *
 *    1. DMA from LPA (buffer size and number of bufs set dynamically).
 *    2. Cache in core or on disk file (raw or cooked) as desired.
 *    3  Automatic triggering and all i/o in one real-time process.
 *    4. Final output on disk or tape or both as desired.
 *    5. Output is compatable with Ping and Pong programs.
 *    6. Most parameters dynamically set from command line.
 *    7. Many seismically important arguments are in seconds (not bufs).
 *    8. When executed with no args, options are documented.
 */

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

#define MAX_STA     32              /* maximum number of stations           */
#define MAX_NET     196             /* maximum size of subnet vector        */
#define NSZ         5               /* nbytes in station name incl null     */

#ifdef  LPA
#define A2D_ZERO    (4096/2)        /* logical zero for lpa                 */
#endif  LPA
#ifndef LPA
#define A2D_ZERO    0               /* logical zero for lpa                 */
#endif

#define Abs(n)      (n<0 ? -n : n)  /* macro for absolute value             */

/*----- settable parameters either by command args or by default ----------*/

int     bufsz;                      /* buffer size for a2d reads            */
int     nbuf;                       /* number of buffers for a2d to use     */
int     const1;                     /* numerator for trigger const          */
int     const2;                     /* denominator for trigger const        */
int     threshold;                  /* threshold for trigger                */
int     const3;                     /* weigted average numerator            */
int     const4;                     /* weigted average denominator          */
int     wav_inc;                    /* weigted average increment            */
float   before;                     /* seconds of data to save before trigger*/
float   after;                      /* seconds of data to save after trigger */
float   aparture;                   /* secondss for coincident station trig  */
float   sweep;                      /* secs of data for short term average   */
long    rate;                       /* sampling rate in samples/sec         */
int     decim;                      /* decimation factor for averages       */
int     nstations;                  /* number of stations in network        */
int     nchan;                      /* num channels from a2d, dflt nstations */
int     xchg_tape;                  /* exchange tape units after event      */
int     shortcut;                   /* suppress trigger computations        */
int     diffsam;                    /* difference samples before computation */
int     dyn_flag;                   /* enable use of dynamic flags          */

/*---- file names -- also set by command arguments or by default ----------*/

char    *stat_file;                 /* stationcomp structure file           */
char    *subnet_file;               /* network and subnet file name         */
char    *cach_file;                 /* usually 0=core or "/dev/rrllb"       */
char    *disk_file;                 /* final output to disk file name       */
char    *tapa_file;                 /* final output to tapa file name       */
char    *pseudo_a2d;                /* data input from file, not a2d        */
char    *log_file;                  /* output file for diagnostics          */

int     conttape;                   /* if true write all data continuously
                                       to tape                              */

/*------- derived parameters calculated from parameters above -------------*/

int     bufszb;                     /* buffer size in bytes (cf bufsz)      */
long    ns_before;                  /* num of samples to save before trigger */
int     nb_before;                  /* num of buffers before trigger (cache) */
long    ns_after;                   /* num of samples to save after trigger  */
int     nb_after;                   /* num of buffers to save after trigger  */
long    ns_write;                   /* total num samples to write on trigger */
long    ns_aperture;                /* num samples for quasi-simul trigger   */
long    ns_sweep;                   /* num samples in one trigger sweep      */
int     nsw_aperture;               /* num of sweeps per aperture            */

/*------------ variables used in triggering calculations -------------------*/

long    st_sum      [MAX_STA];      /* short-term sums for one sweep         */
long    st_abs_sum  [MAX_STA];      /* rectified short-term sums             */
int     stas        [MAX_STA];      /* regular short-term averages           */
int     abs_stas    [MAX_STA];      /* rectified short-term averages         */
int     ltas        [MAX_STA];      /* regular long-term averages            */
int     abs_ltas    [MAX_STA];      /* rectified long-term averages          */
int     etas        [MAX_STA];      /* eta values for each station           */
int     sta_trigger[MAX_STA];       /* individual station trigger            */
int     subnet      [MAX_NET];      /* subnet configuration                  */
char    stanames    [MAX_STA][NSZ]; /* stations names per a2d chan           */
int     looksta     [MAX_STA];      /* sta in a subnet, s=looksta[i]         */
STATIDENT stids     [MAX_STA];      /* station component identifiers         */
int     nlook;                      /* num stations in looksta vector        */
char    network     [4];            /* network name read from STATIDENT for
                                       station on channel 1                 */

/*------------------- misc global variables -------------------------------*/

int     event_num;                  /* event number beg at 1 for output      */
long    rec_num;                    /* count of all non-header records       */
int     wrt_num;                    /* count of per-event non-hd records     */
int     trig;                       /* master trigger, ages to 0==off        */
int     nbuftrig;                   /* num of buffers to write on trigger    */
int     a2dsize;                    /* size of sample vector in bytes        */
char   *a2dptr;                     /* pointer to next sample vector         */
int     a2dcount;                   /* num bytes still valid in current buffer*/
long    sweepnum;                   /* global count of sweeps from start     */

/*---------- diagnostic switches to produce terminal output ---------------*/

int     prt_diag;                   /* print outs of misc diagnostics        */
int     prt_parm;                   /* print outs of initial params          */
int     prt_etas;                   /* print outs of eta values              */
int     prt_strg;                   /* print outs of stations triggers       */
int     prt_trig;                   /* print outs of master trigger          */
int     prt_vbse;                   /* print outs of triger verbose          */
int     prt_st_av;                  /* print outs of st_av and lt_av         */
int     prt_st_abs_av;              /* print outs of st_abs_av and lt_abs_av */
int     prt_cach;                   /* print outs of cache activity          */
int     prt_a2d;                    /* print outs of a2d activity            */
int     prt_out;                    /* print outs of output activity         */
int     prt_tim;                    /* print outs of time in a2d read        */
int     prt_mall;                   /* print outs malloc calls               */

/*---------- diagnostic switches for debugging the system -----------------*/

int     not_rtp;                    /* suppress RTP (real-time process)      */
int     fake_ok;                    /* allow fake trigger from signal        */
```

```
int     fake_trig;              /* fake trigger from signal SIGQUIT     */
int     no_head;                /* suppress write out of header         */
int     exit_early;             /* for testing program startup          */

/*----------------------- global declarations ------------------------------*/

short   *getsweep();            /* returns ptr to 1 vector of data      */
```

bc1

```
/*      HEADER structure of 11/34 seismic data - - - -
        These are the data included in that part of the common
        structure which is written at the begining of each triggered
        event on 11/34 tape.  It is of a fixed size independent
        of how many channels are being digitized.
*/
#define MAXSTA 32

struct header {
        short tchan;            /* total number of channels in data             */
        short rate;             /* number of samples per second per channel     */
        short bufsize;          /* Tape record size                             */
        char time[8];           /* ascii hr,min,and sec from 1134 realtime clock */
        short date[3];          /* Date (day, month, year) in binary            */
        long  s[MAXSTA];        /* Running sum of signal for each chan          */
        long  r[MAXSTA];        /* Running sum of rectified signal              */
        short sbrbr[MAXSTA];    /* Long term signal average                     */
        short rbrbr[MAXSTA];    /* Long term rectified signal average           */
        short eta[MAXSTA];      /* Trigger criteria: plus for triggered station */
        short strig[MAXSTA];    /* Trigger state for each station               */
        short strval[MAXSTA];   /* Set value for each station trigger           */
        short tri;              /* The master trigger                           */
        short trval;            /* The set value for the master trigger         */
        short decim;            /* Decimation factor for trigger routine        */
        char station[MAXSTA][4];/* Station list                                 */
        short subnet[196];      /* Subnet list (changed to 196 2/12/80          */
        short flag[100];        /* Odds and ends and intertask comunication     */
};
#define HEADSIZE sizeof(h)

/*      See the common block structure of 11/34 routines for details
        of most of the above.  Flag values that are of interest to
        us here are listed below.
flag[3] Event number on tape.
flag[4],flag[5] fraction for computing eta
flag[7] number of buffers to process in one trigger sweep
flag[8] length of disc delay "tank" in buffers.
flag[9] number of tape records written (figured out by "get34")
*/
```

```
# define reg            register
# define repeat         do
# define until(x)       while(!(x))
# define sizo           sizeof
# define slide          continue
# define nonneg         unsigned
# define LONG           long)(unsigned
# define bool           char
/*
# define new(x) ((x=malloc(sizo*x))==0)
# define move(x,y) smcopy(x,y,sizo(*y))
 */
```

```
#
/*
 * LPA driver header file
 *      Asa Romberger
 *      modified by WFJolitz, D. Fong, EJHaug
 *
 * method of usage:
 *      open
 *      write microcode
 *      write dedicated mode dispatch table
 *      stty to set parameters
                word0   buffer address
                word1   rate    (-10**6/freq_in_hz)
                word2   nbufs   15:13 #buffers-1
                        bufsz   12:0  buffer size in words
                word3   mode    15:15 1 for seq channel mode
                                      0 for sgl channel mode
                        chan     7:0  #channels for seq mode
                                      channel# for sgl channel mode
                word4   incr    15:8  channel incr for seq mode
                        ichan    7:0  start channel for seq mode
                                      ignored in sgl mode
                word5   clk             user requested clk params
 *
 *      read - 1 character indicating buffer index
 *              fill or empty buffer
 *
 *      gtty -- to get the real address of the buffers
                word0   hi order bits of buffer phys addr
                word1   lo order bits of buffer phys addr
                word2   unused
 *
# ifdef notdef
 * minor device number = DDCCCCCC where:
 *      DD      = 00 for analog input
 *              = 01 for analog input with second adc
 *      CCCCCC  = channel number
 *      CCCCCC  ignored in this version of the driver
 *              channel parameters passed in as above
# endif notdef
 *
 */
        /*CLOCK START COMMAND*/
#define CLOCK  1                /*mode -- clock start*/
#define CLOCKA (0<<4)           /*clock A*/
#define CLOCKB (1<<4)           /* clock b */
        /*clock status word*/
#define ENACTR 1                /*enable counter*/
#define R1M     (1<<1)          /*1 MHz rate*/
#define R100K   (2<<1)          /*100 KHz rate*/
#define R10K    (3<<1)          /*10 KHz rate*/
#define R1K     (4<<1)          /*1 KHz rate*/
#define R100    (5<<1)          /*100 Hz rate*/
#define REXT    (6<<1)          /*external rate (from st1 input)*/
#define R60     (7<<1)          /*line frequency rate*/
#define MFIE    0100            /*mode flag interrupt enable*/
#define MSI     (0<<8)          /*single interval mode*/
#define MRI     (1<<8)          /*repeat interval mode*/
#define MEET    (2<<8)          /*external event time mode*/
#define MEETZ   (3<<8)          /*external event time mode from zero base*/
#define ST1EC   020000          /*st1 enable counter*/
#define ST1IE   040000          /*st1 interrupt enable*/

#define GETLPAP   ('l'<<8)|'g'
#define SETLPAP   ('l'<<8)|'s'
```

```
CFLAGS = -g -f68881
LIBS= -lm -lsuntool -lsunwindow -lpixrect

all: cdd plotatod

cdd: cdd.o CDDatod.o
	$(CC) $(CFLAGS) -o cdd cdd.o CDDatod.o -lsuds -lm

plotatod: plotatod.o CDDatod.o
	$(CC) $(CFLAGS) -o plotatod plotatod.o CDDatod.o -lsuds $(LIBS)

install:
	cp plotatod cdd /usr/seis/bin

plotatod.o : /usr/include/seis/suds.h
plotatod.o : /usr/include/seis/seis.icon
```

```
#define WRTBUF  4096            /* size buffer to write data to CDDatod        */
char wrtbuf[WRTBUF];

#include <stdio.h>
#include <sys/file.h>
#include <sys/ioctl.h>
#include <sys/types.h>
#include <sys/mtio.h>
struct mtop  ctl;
struct mtget stat;

extern die();
extern int errno;

int atod_rezero(fd_atod)
   int fd_atod;
{
   int ioc;

   ctl.mt_op=5;
   ctl.mt_count=1;
   ioc=ioctl(fd_atod,MTIOCTOP,&ctl);
   if(ioc== -1)st_error(0,errno,
      "Unable to rezero CDD Data Sampler on file number %d",fd_atod);
   return(ioc);
}

int atod_open(device,ms_samp,numchan)
   char *device;
   int ms_samp,numchan;
{
   int fd_atod,ret,nummux,maxi;
   short*b;
   char  a[2];
   double sc;
   extern double get_mstime();
   int yr,mo,dy,hr,mn;

   if((fd_atod=open(device,O_RDWR))== -1) st_error(die,errno,
      "Unable to open CDD Data Sampler on filename %s",device);

   /* Calculate number of multiplexers (groups of 16 channels) */
   nummux=(numchan+15)/16;

   /* check ms_samp is within limits of atod: aggregate rate 25,600 samps per sec */
   maxi=(nummux*10)/16;
   if(ms_samp < maxi) st_error(die,0,
      "%d milliseconds per sample too small. Must be at least %d for %d channels",
      ms_samp,maxi,nummux*16);

   /* test byte order *b=780 on SUN3/50 and *b is 3075 on DEC 11/70  */
   a[0]=3;
   a[1]=12;
   b=(short *)&a[0];
   if(*b==3075)sprintf(wrtbuf,"MSIG1 \n");
   else sprintf(wrtbuf,"LSIG1 \n");

   /* set atod for twos complement arithmetic */
   sprintf(wrtbuf,"%s2S COMP \n",wrtbuf);

   /* set year counter in atod */
   decode_mstime(get_mstime(),&yr,&mo,&dy,&hr,&mn,&sc);
   sprintf(wrtbuf,"%sYEAR %d\n",wrtbuf,yr);

   sprintf(wrtbuf,"%sINITATOD %d %d\n",wrtbuf,ms_samp,nummux);

   atod_rezero(fd_atod);
   ret=write(fd_atod,wrtbuf,WRTBUF);
   if(ret!=WRTBUF) st_error(die,errno,
      "Unable to initialize CDD Data Sampler on file %s. Write returns %d",
      device,ret);
   return(fd_atod);
}

int atod_settime(fd_atod,day,hour,min,sec)
   int fd_atod,day,hour,min,sec;
{
   int ret,err;

   if(day<0) {
      sprintf(wrtbuf,"EXTERNAL RTC\n");
      ret=write(fd_atod,wrtbuf,WRTBUF);
      return(atod_geterr(fd_atod));
   }
   ret=0;
   if(day<0  || day > 366) {ret++; fprintf(stderr,"Incorrect day of %d\n",day);}
   if(hour<0 || hour > 23) {ret++; fprintf(stderr,"Incorrect hour of %d\n",hour);}
   if(min<0  || min  > 59) {ret++; fprintf(stderr,"Incorrect minute of %d\n",min);}
   if(sec<0  || sec  > 59) {ret++; fprintf(stderr,"Incorrect second of %d\n",sec);}
   if(ret==0) {
      sprintf(wrtbuf,"INTERNAL RTC\nTIME %d %d %d %d\n",day,hour,min,sec);
      ret=write(fd_atod,wrtbuf,WRTBUF);
      return(atod_geterr(fd_atod));
   }
   return(1);
}

int atod_incrtime(fd_atod,msec)
   int fd_atod,msec;
{
   int ret,err;

   if(msec<0)sprintf(wrtbuf,"RETARD %f\n",-msec);
   else sprintf(wrtbuf,"ADVANCE %f\n",msec);
   ret=write(fd_atod,wrtbuf,WRTBUF);
   return(atod_geterr(fd_atod));
}

int atod_geterr(fd_atod)
   int fd_atod;
{
   int ioc;

   ioc=ioctl(fd_atod,MTIOCGET,&stat);
   if(ioc== -1) st_error(die,errno,
      "Unable to get error from CDD Data Sampler on file number %d",fd_atod);
   return(stat.mt_erreg);
}
```

```
/* Exercise the Cutler Digital Design Data Sampler
 */
#define DEVICE  "/dev/rst1"
#define WRTBUF  4096            /* When writing to Atod, must send full buffer */

#define NUMPRN  WRTBUF/2
#define LOW4BITS 0xF
char wrtbuf[WRTBUF];
#include <stdio.h>
#include <sys/time.h>
extern int errno;
extern char *progname;

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

char menu[]=
"\no open(type msec/sample and number of channels)\nr rezero atod\ng get data\nh get head

main(argc,argv)
   int argc;
   char **argv;
{
   int dev,ioc,i,j,err,mssamp,numchans,day,hr,min,sec;
   char input[100];
   char buf[WRTBUF];
   short *ibuf;
   struct timeval tp;
   struct timezone tzp;
   struct tm *tmm,*gmtime();

   progname=argv[0];
   dev= -1;

   while(1) {
      fprintf(stderr,"%s",menu);
      gets(input);
      fprintf(stderr,"\n");
      switch(input[0]) {
         case 'o':if(dev!=0)close(dev);
                  ioc=sscanf(&input[2],"%d %d",&mssamp,&numchans);
                  if(ioc!=2) {
                     mssamp=10;
                     numchans=16;
                  }
                  dev=atod_open(DEVICE,mssamp,numchans);
                  fprintf(stderr,
                     "Open the CDD Data Sampler on file descriptor %d \n",dev);
                  fprintf(stderr," for %d msec/channel and %d channels.\n",
                     mssamp,numchans);
                  break;
         case 'r':atod_rezero(dev);
                  fprintf(stderr,"Rezero atod on file descriptor %d\n",dev);
                  break;
         case 'g':
                  ioc=read(dev,buf,WRTBUF);
                  if(ioc== -1) {
                     st_error(0,0,"Error %d on dev %d: ",errno,dev);
                     err=atod_geterr(dev);
                     if(err==4)fprintf(stderr,"Atod buffer overflowed.\n");
                     else fprintf(stderr,"Error register is %d\n",err);
                  }
                  fprintf(stderr,"Getdata returns %d bytes\n",ioc);
                  ibuf=(short *)buf;
                  if(ioc>0)for(i=0,j=0;i<NUMPRN;i++) {
                     fprintf(stderr,"%6d",ibuf[i]/16);
                     if(++j>=16) {
                        fprintf(stderr,"\n");
                        j=0;
                     }
                  }
                  break;
         case 'h':
                  ioc=read(dev,buf,WRTBUF);
                  if(ioc== -1) {
                     st_error(0,0,"Error %d on dev %d: ",errno,dev);
                     err=atod_geterr(dev);
                     if(err==4)fprintf(stderr,"Atod buffer overflowed.\n");
                     else fprintf(stderr,"Error register is %d\n",err);
                  }
                  fprintf(stderr,"Getheader returns %d bytes\n",ioc);
                  ibuf=(short *)buf;
                  j=0;
                  if(ioc>0)for(i=0;i<NUMPRN;i++) {
                     fprintf(stderr,"%2d ",ibuf[i] & LOW4BITS);
                     j++; if(j==16){fprintf(stderr,"\n");j=0;}
                  }
                  break;
         case 's':
                  fprintf(stderr,"Send: ");
                  gets(wrtbuf);
                  i=strlen(wrtbuf);
                  wrtbuf[i]='\n';
                  wrtbuf[++i]='\0';
                  for(i=0;i<strlen(wrtbuf);i++) {
                     if(wrtbuf[i]>='a' && wrtbuf[i]<='z')wrtbuf[i]=wrtbuf[i]-'a'+'A';
                     fprintf(stderr,"%d ",wrtbuf[i]);
                  }
                  fprintf(stderr,"%d (%s)",wrtbuf[i],wrtbuf);
                  fflush(stderr);
                  ioc=write(dev,wrtbuf,WRTBUF);
                  if(ioc== -1) {
                     fprintf(stderr,"\nError %d on dev %d: ",errno,dev);
                     perror("doit.put");
                  }
                  fprintf(stderr,"\nPut (%s) returns %d\n",wrtbuf,ioc);
                  break;
         case 't':sscanf(&input[2],"%d %d %d %d",&day,&hr,&min,&sec);
                  fprintf(stderr,"Set time to %d %d %d %d\n",day,hr,min,sec);
                  if(atod_settime(dev,day,hr,min,sec)==4)
                     fprintf(stderr,"Atod buffer overflow\n");
                  break;
         case 'T':if(gettimeofday(&tp,&tzp)== -1) {
                     fprintf(stderr,"Unable to read system clock");
                     break;
                  }
                  tmm=gmtime(&tp.tv_sec);
                  fprintf(stderr,"Set time to %d %d %d %d\n",
                     tmm->tm_yday+1,tmm->tm_hour,tmm->tm_min,tmm->tm_sec);
                  if(atod_settime(
                     dev,tmm->tm_yday+1,tmm->tm_hour,tmm->tm_min,tmm->tm_sec)==4)
                        fprintf(stderr,"Atod buffer overflow\n");
                  sec=tp.tv_usec/1000;
                  fprintf(stderr,"Change time by %d milliseconds\n",sec);
                  if(atod_incrtime(dev,sec)==4)
                     fprintf(stderr,"Atod buffer overflow\n");
                  break;
         case 'c':sscanf(&input[2],"%d",&sec);
                  fprintf(stderr,"Change time by %d milliseconds\n",sec);
                  if(atod_incrtime(dev,sec)==4)
                     fprintf(stderr,"Atod buffer overflow\n");
```

```
                break;
        case 'q':die(0);
        default: fprintf(stderr,"Unknown option %c\n",input[0]);
      }
    }
}
```

```
/* plotatod: plot output of atod in a SUN window
   Peter Ward, USGS, Menlo Park, Ca hacked up 8/87 and 7/88
 */
#define MAX_STA     128         /* maximum number of stations                    */
#define SHOWWIDTH     4         /* number of strips in x direction to be plotted*/
#define DECIM         2         /* default decimation factor                     */
#define NUMGET     4096         /* blocksize for reading atod                    */
#define BEGINTRACE   90         /* begin traces this many pixels from left       */
#define XZERO         0         /* upper left x position of frame                */
#define YZERO        65         /* upper left y position of frame                */
#define XSIZE      1152         /* x width of frame                              */
#define YSIZE       820         /* y width of frame                              */
#define PERNOISE    100         /* default percent noise will be of window       */
#define GAINSWEEPS 1000         /* number of atod sweeps used to autoscale       */
#define NUMSTATS     16         /* default number of stations                    */
#define MS_SAMPLE    10         /* default milliseconds per sample               */
#define TICMARK      20         /* length of time tic mark in pixels             */

static char *menuitem[]={
   "Auto gain",
   "Double gain",
   "Halve gain",
   "Double timebase",
   "Halve timebase",
   "Exit"
   };
int menutot=6;

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

#define LOW4BITS 0xF

#include <stdio.h>
#include <sys/file.h>
#include <signal.h>
#include <sys/types.h>

#include <suntool/sunview.h>
#include <suntool/canvas.h>
#include <suntool/panel.h>
#include <suntool/icon.h>
#include <pixrect/pixrect_hs.h>
#include <seis/suds.h>

struct rect framesize = {XZERO,YZERO,XSIZE,YSIZE}; /* size and position of frame */

static short seis_icon[]={
#include <seis/seis.icon>
};
DEFINE_ICON_FROM_IMAGE(my_icon,seis_icon);

char buf[2*NUMGET];
char decstr[][7]= {"","other ","3rd "};
int endit=0;

int intr=1;
onintr() {
   intr=0;
}

static Notify_value my_frame_destroyer(frame,status)
   Frame frame;
   Destroy_status status;
{
   onintr();
   if(endit) return(notify_next_destroy_func(frame,status));
   notify_veto_destroy(frame);
   return(NOTIFY_DONE);
}

int ms_samp=MS_SAMPLE;
int numstats=NUMSTATS;
double atof();
int dec,dev,ysize,yz,yincr,yzero,numstin,numchan;
int ptr= -1;
int maxptr= -1;
float gain=0.0;
float factor=1.0;
int first= -1;
int last= -1;
int print_ave=0;
int plotx=1;
int xfac=1;
long sta_offset[MAX_STA],stamax[MAX_STA],stamin[MAX_STA];
float sta_gain[MAX_STA],conmvolts[MAX_STA];
char statname[MAX_STA][5],temp[12];
short *ibuf;
struct pixfont *font;
Pixwin *pw;

main(argc,argv)
   int argc;
   char **argv;
{
   int i;

   signal(SIGINT,onintr);

   if(argc<3) {
      fprintf(stderr,"Usage: plotatod arguments stations device\n");
      exit(1);
   }
   dec=DECIM;
   if(dec<1)dec=1;
   endit=0;
   ptr= -1;
   maxptr= -1;
   intr=1;
   for(i=1;argv[i][0]=='-';i++) {
      switch(argv[i][1]) {
         case 'd': dec=atoi(argv[++i]);                  /* decimation           */
                   if(dec<1)dec=1; break;
         case 'f': first=atoi(argv[++i]); break;         /* first station in scan */
         case 'g': gain =atof(argv[++i]); break;         /* fixed gain           */
         case 'l': last =atoi(argv[++i]); break;         /* last station in scan  */
         case 'm': ms_samp=atoi(argv[++i]); break;       /* milliseconds per sample*/
         case 'n': numstats=atoi(argv[++i]); break;      /* number of stations    */
         case 'p': print_ave=1; break;                   /* print averages        */
         default: printf("plotatod: Unknown argument %s\n",argv[i]);
      }
   }
   plotit(argv[i],argv[i+1]);
}

autoscale()
{
   int i,j,ioc,num,samp;

   printf("Begin autoscaling by reading data for %d seconds.\n",
      GAINSWEEPS*ms_samp/1000);
```

```
    for(i=first-1;i<last;i++) {
       stamax[i]= -10000;
       stamin[i]=10000;
       sta_offset[i]=0;
    }
    if(numchan<=2) {                        /* determine scaling of traces */
       for(j=0,num=1;j<GAINSWEEPS;j++,num++){
          ioc=getsweep(ibuf,numstin,1);
          if(ioc==EOF) {
             printf("Only %d sweeps of data decimated by %d. \n",j,dec);
             j=GAINSWEEPS-1;
          }
          for(i=first-1;i<last;i++) {
             samp=ibuf[i];
             if(samp<stamin[i])stamin[i]=samp;
             if(samp>stamax[i])stamax[i]=samp;
             sta_offset[i]+=samp;
             if(j==GAINSWEEPS-1) {
                sta_offset[i]=sta_offset[i]/num;
                if(print_ave)printf("%5s min=%5d max=%5d ave=%5d\n",
                   statname[i],stamin[i],stamax[i],sta_offset[i]);
                /* calculate samples per pixel */
                if(gain!=0.0) {
                   sta_gain[i]=gain;
                   stsmin[i]=gain/(conmvolts[i]*yincr);
                }
                else  {
                   sta_gain[i]=((stamax[i]-stamin[i])*100.0/PERNOISE)*conmvolts[i];
                   stamin[i]=(stamax[i]-stamin[i])*100/(PERNOISE*yincr);
                }
                if(stsmin[i]<1)stamin[i]=1;
             }
             /*printf("%x ",samp&LOW4BITS);*/
          }
          /*printf("\n");*/
       }
    }
}

Notify_value decision();

plotit(stats,device)
    char *stats,*device;
{
    Frame frame;
    Canvas canvas;
    char frame_label[120];
    Icon icon;
    struct pr_prpos where;
    int i,j,k,s,numst,ioc,yy,yyl,swidth;
    int xcounts,xshow,dx,tempp;
    short x,x0,ex0,y[MAX_STA],y0[MAX_STA];
    extern char *malloc();
    char ttime[16];
    FILE *statfile;
    struct stationcomp *sc_ptr;
    int st_type,st_len,ms_tbase,pixpersec,tm;

    /* read stationcomp structure information */
    statfile=st_open(stats,"r");
    i= -1;
    while(st_get(&sc_ptr,&st_type,&st_len,statfile)!=EOF) {
       if(st_type==STATIONCOMP) {
          if(sc_ptr->channel==(short)NODATA)
             st_error(die,0,"Channel number not assigned for station %s",
                sc_ptr->sc_name.st_name);
          if(sc_ptr->channel>=MAX_STA)
             st_error(die,0,"Channel number too large. %d>=%d",
                sc_ptr->channel,MAX_STA);
          j=sc_ptr->channel-1;
          strncpy(statname[j],sc_ptr->sc_name.st_name,5);
          statname[j][4]='\0';
          if(sc_ptr->con_mvolts==atof(NODATA) || sc_ptr->con_mvolts ==0)
             st_error(die,0,"con_mvolts not set for station %s",
                sc_ptr->sc_name.st_name);
          conmvolts[j]=sc_ptr->con_mvolts;
       }
       st_free(sc_ptr);
    }
    st_close(statfile);

    dev=atod_open(device,ms_samp,numstats);

    ioc=read(dev,buf,NUMGET);
    if(ioc!=NUMGET) {
       perror("first read");
       return(1);
    }
    ibuf=(short *)buf;
    for(numchan=0,i=0;i<257;i+=16){
       if((j=ibuf[i]&LOW4BITS) > numchan) numchan=j;
    }
    numchan++;                                  /* number of multiplexers */
    numst=numstin=numchan*16;                   /* number of channels     */
    printf("Number of channels received from device %s is %d.\n",device,numst);
    if(first>last) { tempp=first; first=last; last=tempp; }
    if(last== -1 || last>numst)last=numst;
    if(first<=0)first=1;
    numst=last-first+1;
    printf("Plot channels %d through %d.\n",first,last);
    if(dec<4)sprintf(frame_label,
       "Input from %s for %d channels and plotting every %spoint.",
       device,numst,decstr[dec-1]);
    else sprintf(frame_label,
       "Input from %s for %d channels and plotting every %dth point.",
       device,numst,dec);
    yincr=(YSIZE-12)/numst;
    ysize=yincr*numst+12;
    framesize.r_height=ysize+25;

    font=pf_open("/usr/lib/fonts/fixedwidthfonts/screen.r.7");
    where.pr=(Pixrect *) icon_get(&my_icon,ICON_IMAGE);
    where.pos.x=2;
    where.pos.y=60;
    pf_text(where,PIX_SRC,font,device);
    icon_set(&my_icon,ICON_IMAGE,where.pr,0);

    frame=window_create(0,FRAME,
       FRAME_LABEL,       frame_label,
       FRAME_ICON,        &my_icon,
       FRAME_OPEN_RECT,   &framesize,
       0);
    canvas=window_create(frame,CANVAS,
       CANVAS_AUTO_SHRINK,        FALSE,
       CANVAS_WIDTH,              XSIZE,
       CANVAS_HEIGHT,             ysize,
       WIN_EVENT_PROC,            decision,
       0);

    (void) notify_interpose_destroy_func(frame,my_frame_destroyer);
```

```
pw=canvas_pixwin(canvas);
window_set(frame,WIN_SHOW,TRUE,0);

autoscale();

yy1=0;
swidth=(XSIZE-BEGINTRACE+SHOWWIDTH)/SHOWWIDTH;
yzero=yincr/2;
for(i=0;i<numst;i++) y0[i]=yzero+i * yincr;
dx=0;
yz=yincr/2;
pw_batch_on(pw);
yzero=0;
for(i=first-1;i<last;i++) {
   pw_vector(pw,0,yzero,BEGINTRACE,yzero,PIX_SRC,1);
   sprintf(temp,"%3d %s",i+1,statname[i]);
   pw_text(pw,1,yzero+yz+6,PIX_SRC,NULL,temp);
   sprintf(temp,"%3.0f",((float)sta_offset[i]*commvolts[i])/factor);
   pw_text(pw,65,yzero+yz+8,PIX_SRC,font,temp);
   sprintf(temp,"%3.0f",sta_gain[i]);
   pw_text(pw,65,yzero+yz-1,PIX_SRC,font,temp);
   yzero+=yincr;
}
printf("Now begin plotting data.\n");
pw_vector(pw,0,yzero,BEGINTRACE,yzero,PIX_SRC,1);
atod_rezero(dev);

ptr= -1;            /* reset getsweep */
maxptr= -1;

x=BEGINTRACE;
x0=BEGINTRACE;
ex0=BEGINTRACE;
xcounts=0;
xshow=BEGINTRACE+swidth;
while(intr && ((ioc=getsweep(ibuf,numstin,dec))!=EOF) ) {
   yzero=yz;
   if(x==BEGINTRACE) {       /* change time at beginning of trace */
      for(i=0,j=2;i<15;i++) {
         if(i==3 || i==8)ttime[i]=' ';
         else if(i==11)ttime[i]='.';
         else ttime[i]=(char)(ibuf[j++]&LOW4BITS)+'0';
      }
      ttime[15]='\0';
      sscanf(&ttime[12],"%d",&ms_tbase);
      pixpersec=xfac*(1000/ms_samp)/plotx;
      for(tm=BEGINTRACE+((1000-ms_tbase)*pixpersec)/1000;tm<xshow;tm+=pixpersec) {
         pw_vector(pw,tm,0,tm,TICMARK,PIX_SRC,1);
         pw_vector(pw,tm,YSIZE,tm,YSIZE-TICMARK,PIX_SRC,1);
      }
   }
   for(i=first-1;i<last;i++) {
      yy=yzero-(factor*(ibuf[i]-sta_offset[i])/ stamin[i]);
      pw_vector(pw,x0,y0[i],x,yy,PIX_SRC,1);
      y0[i]=yy;
      yzero+=yincr;
   }
   x0=x;
   if(++xcounts>=plotx) {xcounts=0; x+=dec*xfac;}
   if(x>=xshow) {
      pw_text(pw,BEGINTRACE,ysize-1,PIX_SRC,NULL,ttime);
      pw_show(pw);
      if(x>=XSIZE) {
         x=BEGINTRACE;
         x0=BEGINTRACE;
         xshow=BEGINTRACE+swidth;
         pw_writebackground(pw,BEGINTRACE,0,swidth+10,YSIZE,PIX_SRC);
      }
      else {
         xshow+=swidth;
         pw_writebackground(pw,x,0,swidth+10,YSIZE,PIX_SRC);
         pixpersec=xfac*(1000/ms_samp)/plotx;
         for(;tm<xshow;tm+=pixpersec) {
            pw_vector(pw,tm,0,tm,TICMARK,PIX_SRC,1);
            pw_vector(pw,tm,YSIZE,tm,YSIZE-TICMARK,PIX_SRC,1);
         }
      }
   }
   (void)notify_dispatch();
}
if(intr) {
   notify_dispatch();
   pw_batch_off(pw);
   endit=1;
   window_main_loop(frame);
}
close(dev);
}

static short *ibuffr;

int getsweep(buffer,numch,decim)
   short *buffer;
   int numch,decim;
{
   register i,ii;
   int got,next;

   ptr+=numch*decim-numch;
   for(i=0;i<numch;i++) {
      if(ptr<maxptr) {
         buffer[i]=ibuffr[ptr++];
      }
      else {
         got=read(dev,buf,NUMGET);
         if(intr==0)return(EOF);
         ibuffr=(short *)buf;
         if(got<NUMGET) {
            got=atod_geterr(dev);
            if(got==4) {
               printf("  AtoD buffer overflow. Resetting.\n");
              atod_rezero(dev);
            }
         }
         ptr=ptr-maxptr;
         maxptr=(got/2);
         buffer[i]=ibuffr[ptr++]/16;
      }
   }
   return(0);
}

Notify_value decision(canvas,event,arg)
   Canvas canvas;
   Event *event;
   caddr_t arg;
{
   Menu decide;
   int i,which;
```

```
    if(event_id(event)!=MS_RIGHT) return(NOTIFY_IGNORED);
    decide=menu_create(MENU_JUMP_AFTER_SELECTION,TRUE,0,0);
    for(i=0;i<menutot;i++) menu_set(decide,MENU_STRING_ITEM,menuitem[i],i+1,0);
    which=(int)menu_show(decide,canvas,canvas_window_event(canvas,event),0);
    switch(which){
       case 1:gain=0.0;
              autoscale();
              break;
       case 2:factor=factor*2.0;
              break;
       case 3:factor=factor*0.5;
              break;
       case 4:plotx=plotx/2;
              if(plotx==0) {
                 plotx=1;
                 xfac=xfac*2;
              }
              return;
       case 5:if(xfac!=1) xfac=xfac/2;
              else plotx=plotx*2;
              return;
       case 6:exit(0);
    }
    /* modify labels on left of plot */
    pw_writebackground(pw,0,0,BEGINTRACE,YSIZE,PIX_SRC);
    yzero=0;
    for(i=first-1;i<last;i++) {
       pw_vector(pw,0,yzero,BEGINTRACE,yzero,PIX_SRC,1);
       sprintf(temp,"%3d %s",i+1,statname[i]);
       pw_text(pw,1,yzero+yz+6,PIX_SRC,NULL,temp);
       sprintf(temp,"%3.0f",((float)sta_offset[i]*commvolts[i]));
       pw_text(pw,65,yzero+yz+8,PIX_SRC,font,temp);
       sprintf(temp,"%3.0f",sta_gain[i]/factor);
       pw_text(pw,65,yzero+yz-1,PIX_SRC,font,temp);
       yzero+=yincr;
    }
}
```

```
CC = cc -g -f68881
FC = f77 -g -f68881

demux: demux.o
	$(CC) -o demux demux.o -lsuds

olddemux: olddemux.o xdr_ahheader.o
	$(CC) -o olddemux olddemux.o xdr_ahheader.o

install:
	cp demux /usr/suds/bin
	cp olddemux /usr/seis/bin
```

```
/* demux.c: demultiplex mx.* files into suds streams
 */
#define  NOISE_SAMP     100     /* number of samples to calc noise of trace     */
#define  MAXINPUTBUF    4096    /* maximum size of input buffer in bytes        */
#define  MUXPOSITION    1       /* position of mux number in header             */
#define  LOW4BITS       0xF     /* bit mask for lower 4 bits                    */
#define  MAX_STA        128     /* maximum number of stations                   */

#include <stdio.h>
#include <sys/types.h>
#include <sys/stat.h>
#include <sys/time.h>
#define  MSK4 0xF                                /* 4 least significant bits */

#include "/usr/suds/include/suds.h"

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

int debug=0;
int remove=0;
int statsout=1;
int noisenum=NOISE_SAMP;

main(argc,argv)
   int argc;
   char **argv;
{
   int i,j,div,numsta;
   int first;
   char outfile[120];

   first=1;
   outfile[0]='\0';
   if(argc<3) {
      fprintf(stderr,"Usage: demux options stations datafiles\n");
      exit(1);
   }
   for(i=1;i<argc;i++) {
      if(argv[i][0]=='-') {
         switch(argv[i][1]) {
            case 'd': debug=1; break;           /* debug option                 */
            case 'o': strcpy(outfile,argv[i++]);/* specify output file          */
                      break;
            case 'n': noisenum=atoi(argv[i++]); /* number samps to sum for noise*/
                      break;
            case 'r': remove=1; break;          /* remove input file            */
            case 's': statsout=0; break;        /* no stations in outfile       */
            default: fprintf(stderr,"Unrecognized option %s ignored.\n",
                        argv[i]);
         }
      }
      else {
         if(first) { first=0; numsta=loadstations(argv[i]);}
         else {
            if(outfile[0]=='\0') {      /* change prefix to "st" for output     */
               strcpy(outfile,argv[i]);
               for(j=0,div=0;outfile[j]!='\0';j++) if(outfile[j]=='/') div=j;
               outfile[div]='s';
               outfile[div+1]='t';
            }
            demuxit(argv[i],outfile,numsta);
         }
      }
   }
}

struct stationcomp *stats[MAX_STA];

loadstations(statfile)
   char *statfile;
{
   int j,numstats,st_type,st_len;
   struct stationcomp *sc_ptr;
   FILE *statf;

   numstats=0;
   for(j=0;j<MAX_STA;j++)stats[j]=NULL;
   statf=st_open(statfile,"r");
   while(st_get(&sc_ptr,&st_type,&st_len,statf)!=EOF) {
      if(st_type==STATIONCOMP) {
         if(sc_ptr->channel==(short)NODATA)
            st_error(die,0,"Channel number not assigned for station %s",
               sc_ptr->sc_name.st_name);
         if(sc_ptr->channel>=MAX_STA)
            st_error(die,0,"Channel number too large. %d>=%d",
               sc_ptr->channel,MAX_STA);
         numstats++;
         j=sc_ptr->channel-1;
         stats[j]=sc_ptr;
      }
   }
   st_close(statf);
   for(j=0;j<numstats;j++) if(stats[j]==NULL)
      st_error(0,0,"Stationcomp structure for channel %d is missing",j);
   return(numstats);
}

demuxit(filein,fileout,numsta)
   char *filein, *fileout;
   int numsta;
{
   int i,j,mux,numin,numstats,yr,mo,dy,hr,mn,err,size,type,stlen;
   short *input,data,*out;
   char *ptr,*ptrout;
   float mintr,maxtr;
   double sec;
   struct muxdata *mx;
   struct descriptrace *dt;
   MS_TIME mstime,make_mstime();
   extern char *list_mstime();
   FILE *datain,*dataout;

   datain =st_open(filein,"r");
   dataout=st_open(fileout,"w");
new:
   numin=st_get(&ptr,&type,&stlen,datain);
   if(type!=MUXDATA) {
      st_put(ptr,type,stlen,dataout);
      goto new;
   }
   mx=(struct muxdata *)ptr;
   numstats=numsta;                     /* number of stations                   */
   size=mx->numsamps/numstats;          /* number of data samples per station   */
   if(numsta!=mx->numchans)
      fprintf(stderr,
         "WARNING: %d stations in stationfile but %d stations in muxdata structure",
```

```
      numsta,mx->numchans);
   if(mx->typedata!='r')
      st_error(die,0,
         "Expect datatype 'r': Can not interpret data type '%c'",mx->typedata);
   if((i=(numin-stlen)/2) != mx->numsamps)
      st_error(die,0,"%d samples read but %d samples expected",i,mx->numsamps);

   input=(short *)(ptr+stlen);
   st_create(DESCRIPTRACE,&dt,size*2);
   out=(short *)(dt+1);

   if(debug) {
      fprintf(stderr,"First header: ");
      for(i=0;i<16;i++)fprintf(stderr,"%x ",input[i]&LOW4BITS);
      fprintf(stderr,"\n");
   }

   i=input[1]&MSK4;                    /* calculate the mstime of first sample */
   decode_mstime(mx->begintime,&yr,&mo,&dy,&hr,&mn,&sec);
   dy= (input[2] & MSK4)*100 +(input[3] & MSK4)*10+(input[4] & MSK4);
   hr= (input[5] & MSK4)*10  +(input[6] & MSK4);
   mn= (input[7] & MSK4)*10  +(input[8] & MSK4);
   sec=(input[9] & MSK4)*10.0+(input[10] & MSK4)*1.0+(input[11] & MSK4)*0.1+
      (input[12] & MSK4)*0.01+(input[13] & MSK4)*0.001;
   mstime=make_mstime(yr,0,dy,hr,mn,sec);
   if(debug) fprintf(stderr,"Time first sample=%s in file %s\n",
      list_mstime(mstime,1),fileout);

   if((err=input[14]&MSK4))            /* check magic and error bits          */
      st_error(0,0,"Clock error bits at time %s=%d",list_mstime(mstime,1),err);
   if((i=input[15]&MSK4) != 14)
      st_error(0,0,"Data error: Magic time bits at time %s=%d",
         list_mstime(mstime,1),i);


   for(j=0;j<numstats;j++){
      mintr=32000.0;
      maxtr= -32000.0;
      st_init(DESCRIPTRACE,dt);
      for(i=0;i<size;i++) {
         out[i]=input[i*numstats+j]/16;
         if(out[i]<mintr)mintr=out[i];
         if(out[i]>maxtr)maxtr=out[i];
      }
      if(statsout)
         st_put(stats[j],STATIONCOMP,sizeof(struct stationcomp),dataout);
      strncpy(dt->dt_name.network,stats[j]->sc_name.network,4);
      strncpy(dt->dt_name.st_name,stats[j]->sc_name.st_name,4);
      dt->dt_name.component=stats[j]->sc_name.component;
      dt->dt_name.inst_type=stats[j]->sc_name.inst_type;
      dt->begintime=mstime;
      dt->datatype='s';
      dt->digi_by=0;
      dt->length=size;
      dt->rate=mx->dig_rate;
      dt->mindata=mintr;
      dt->maxdata=maxtr;
      for(i=0,maxtr=0.0;i<noisenum;i++) maxtr+=out[i];
      dt->avenoise=maxtr/noisenum;
      st_put(dt,DESCRIPTRACE,size*2+sizeof(struct descriptrace),dataout);
   }
   st_close(datain);
   st_close(dataout);
}
```