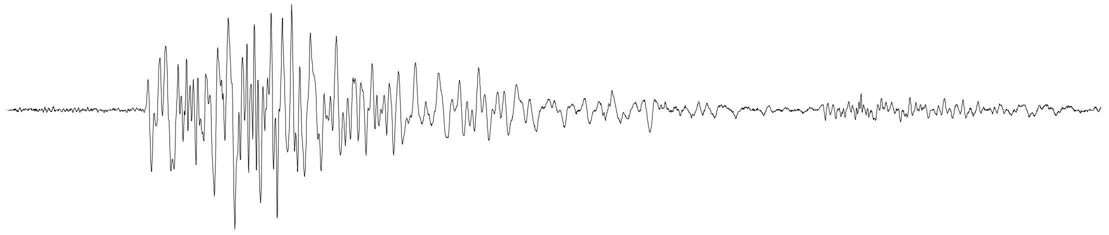
PC-SUDS Utilities



A Collection of Tools for Routine Processing of Seismic Data
Stored in the Seismic Unified Data System
for DOS (PC-SUDS)

Version 2.60 (Y2K) – August 1999

Robert Banfill
Banfill Software Engineering

Contents

CHAPTER 1

Getting Started

This document describes version 2.6 of the PC-SUDS utilities. This is the first year 2000 ready version of the software. The PC-SUDS utilities are a collection of 16-bit DOS programs designed to perform routine processing of seismic data that is stored in the Seismic Unified Data System for DOS (PC-SUDS, or in this document, just SUDS) format.

This document is arranged as follows:

- Chapter 1 describes the system requirements, installation, and the SUDSUTIL.INI file. This chapter should be read by all.
- Chapter 2 presents the PC-SUDS primer. This should be read by anyone not familiar with PC-SUDS.
- Chapters 3 through 7 describe the various software tools by functional group. These chapters should be referred to as needed.
  - Chapter 3 – Tools used in the earthquake location process.
  - Chapter 4 – Filtering tools.
  - Chapter 5 – Display and plotting tools.
  - Chapter 6 – Spectral analysis tools.
  - Chapter 7 – File management and other tools.
- Chapter 8 describes the PC-SUDS I/O library. This library greatly simplifies accessing SUDS data and can be called from both C or FORTRAN.
- The appendices describe the minimum data requirement for data files, the most commonly used PC-SUDS structures, and the contents of a typical PC-SUDS data file.

The user of this software is assumed to have a working knowledge of MS-DOS and Intel x86 based computers as well as the components of the IASPEI software library.

These software tools have been developed with high volume batch style processing in mind. Most of the programs in this package are controlled entirely from the DOS command line. The reason for this is to allow the programs to be called from within DOS batch programs (also called “batch files"). These batch programs are used automate routine processing. Example batch programs are presented throughout the text.

The reader should note that throughout this document a distinction is made between a command line *switch* and a command line *argument*. A *switch* is used to modify the action of the program and an *argument* is what the program is to act upon.

System Requirements

The use of this software requires a personal computer that meets the following minimum criteria:

- Is running MS-DOS 3.2 or later.
- Has 640kB or more RAM.
- Has a hard disk with 20 MB available space.
- Has one of the following display adapters with the appropriate monitor attached:
    - Hercules Graphics Card (HGC.)
    - IBM Color Graphics Adapter (CGA) or compatible.
    - IBM Enhanced Graphics Adapter (EGA) or compatible.
    - IBM Video Graphics Array (VGA) or compatible.
    - VESA compliant[1] SuperVGA display adapter.
- Is attached to one of the following output devices:
    - A Hewlett-Packard Laserjet or compatible printer with > 1 MB memory. Preferably a Laserjet III or later supporting HP-PCL level 5 data compression features.
    - Any POSTSCRIPT output device.
    - Any A–size Hewlett-Packard Graphics Language (HP-GL) output device.
- If you wish to use ReSpec to compute response spectra, a 80286 or later processor and a DSP32C-PC/AT coprocessor board[2] with ≥ 1 MB memory is required.

The following features are not required but are strongly recommended:

- An *i*486DX, *i*486DX/*n*, Pentium, Pentium *II*, or Pentium *III* processor. The faster, the better.
- Eight or more MB memory.

[1]Complying with the Video Electronics Standards Association (VESA) SuperVGA standard version 1.0 (VS891001).

[2]Available from Symmetric Research, http://www.symres.com, (702) 341-9325.

- MS-DOS 6.22 or later with all device drivers and terminate-and-stay-resident (TSR's) software loaded into upper memory blocks (UMB's).
- A Microsoft mouse or compatible pointing device.

The performance of these programs can be greatly enhanced by installing disk caching software (e.g., SmartDrive that comes with DOS 5.0 or later) on your system. The PC-SUDS utilities do not use extended or expanded memory, so if your system has such memory, generally its best use is for disk caching. You should allocate as much memory as possible to the cache. You may also want to set aside some extended memory as a virtual disk (RAMDRIVE.SYS) for storage of plot files. This can increase plotting performance.

Because these programs use only conventional memory (i.e., the lower 640 KB), it is important that TSR programs and device driver such as network software be loaded into upper memory. At a minimum, you should run MS-DOS 6.x and use the MemMaker utility to optimize the use of conventional memory in your system.

SCSI Devices

If you will be using the SCSITape program, we recommend the following hardware and software:

- Adaptec AHA-152x or AHA-1542CF VL-Bus host adapter. If the computer has a PCI local bus, one of Adaptec's newer PCI host adapters is recommended.
- Adaptec's EZ-SCSI 3.1 software. This software provides the ASPI manager and disk drivers for hard disks, MO disks, and CD-ROM.
- Any Hewlett-Packard DDS or DDS-2 tape drive. We recommend that you use a HP 35470A, 35480A, or C1533A tape drive. Other drives may work, but have not be tested with this software. Please see the SCSITape section in chapter 7 for more information.
- The PC-SUDS utilities work well with almost any SCSI disk drives configured as DOS volumes. We have successfully used a variety drives from Hewlett-Packard, Seagate, Conner, and Quantum. We have also successfully used various SCSI CD-ROMS and magneto-optical drives (both 3½" and 5¼").

Local Area Networks

Many of the utilities in this package are designed to work together on several computers attached to a local area network (LAN). There are many commercial network operating systems (NOS) available that use many different mediums. We have tested and use the following networking products with PC-SUDS.

- Artisoft's Lantastic 6.x, ethernet. For small installations (up to about 10 machines) we have found Lantastic to be a very stable and easy to manage network. We particularly like it's configurablity. However, if you need to attach to another network such as a building wide network, this might not be the best choice. We like Lantastic for small, self-contained networks.
- Microsoft peer-to-peer (SMB) networks. This is the NOS that is used in Windows for Workgroups (WFW) and Windows95. We have had good luck with this type of NOS. The nice features of this NOS include; NDIS adapter support, DOS only client and server software is available and easy to use, and many protocols are available (NetBEUI, SPX/IPX, TCP/IP) making connections to other networks pretty straight-forward. This is our NOS of choice when connecting to an existing UNIX (TCP/IP), NetWare, or LAN Manager network.
- FTP software's PC-TCP. This is not so much a NOS as it is the tools needed to connect to a TCP/IP network. We have used this package to connect to large UNIX networks.

Installation

Insert the diskette marked "PC-SUDS Utilities Disk #1" into a floppy disk drive. Log into that drive and type the following command:

`INSTALL` *drive:*

Where *drive:* is the hard disk that you wish to install the programs on (e.g., C:). The utility software will be copied to a directory named \PCSUDS on this drive. The installation batch program will prompt you as to whether or not to install the library, conversion programs and sample data files. The \PCSUDS directory should be added to the PATH (i.e., the list of directories specified with the DOS PATH command, usually in your AUTOEXEC.BAT file) so that you may execute the program from any drive and directory. This may be accomplished by loading AUTOEXEC.BAT file into your favorite text editor and adding (assuming that you installed on C:) ";C:\PCSUDS" to the end of your PATH statement. Then save the file and reset the system.

The SUDSUTIL.INI file

Many of the PC-SUDS utilities look for a file named SUDSUTIL.INI (hereafter called the “INI file”) in their home directory (i.e., the directory where the executable file is located). This file contains initialization information for the programs.

The INI file is an ASCII text file that is made up of sections. Each section begins with a section header of the form: [*name*] where *name* is the name of that section. Each section contains entries of the form: *keyword = entry*. Comments are delimited by a semicolon (;) and blank lines are ignored. This file uses the same structure and conventions found in the INI files used in Microsoft Windows.

Each program that accesses the INI file has its own section. They will each search for their section and read and act upon entries only in that section. The entries in each program’s section are covered in the chapter for that program.

All of the utility programs that generate graphics use the 2.0 version of the PlotX graphics library. This library is used because of the device independent vector graphics it offers and its high resolution (maximum device resolution, not just a screen dump), high performance hardcopy ability. Programs that use the library will search the INI file for the [2.X_PLOTX] section as well as their own. The [2.X_PLOTX] section contains entries that describe your graphics hardware. The INI file provided on the diskette expects your system to have a HP Laserjet III connected to LPT1:. The following is a copy of the [2.X_PLOTX] section provided on the diskette and explanation of each entry:

```
[2.X_PLOTX]
; Entries for PlotX graphics library 2.0 or later

; The following entries are a complete list of modes supported by the library.
; Uncomment a SINGLE display mode that your display adapter and monitor is
; capable of displaying.  If no entry is uncommented, the mode with the best
; resolution up to VGA is automatically chosen.
;DisplayMode = -3     ; Auto, Max resolution <= 640x480, << Default >>
;DisplayMode = -2     ; Auto, Max colors with resolution <= 640x480
;DisplayMode = 4      ; CGA, 320x200, 4 colors
;DisplayMode = 5      ; CGA, 320x200, 4 grays
;DisplayMode = 6      ; CGA, 640x200, black and white
;DisplayMode = 8      ; HGC, 720x348, Hercules monochrome (req. MSHERC.COM)
;DisplayMode = 13     ; EGA, 320x200, 16 colors
;DisplayMode = 14     ; EGA, 640x200, 16 colors
;DisplayMode = 15     ; EGA, 640x350, black and white
;DisplayMode = 16     ; EGA, 640x350, 16 colors
;DisplayMode = 17     ; VGA, 640x480, black and white
;DisplayMode = 18     ; VGA, 640x480, 16 colors
;DisplayMode = 19     ; VGA, MCGA, 320x200, 256 colors
;DisplayMode = 64     ; OVGA, 640x400, 1 of 16 colors, Olivetti only
; The following eight modes require the VESA SuperVGA BIOS extension.  You
; may need to load a TSR to enable VESA support before using these modes.
; The adapter must be compliant with version 1.0 of the VESA standard
; (VS891001).
;DisplayMode = 256    ; SVGA, 640x400, 256 colors, (VESA mode 100h)
;DisplayMode = 257    ; SVGA, 640x480, 256 colors, (VESA mode 101h)
; << WARNING! >> DO NOT attempt to set the following modes without ensuring
; that your monitor can safely handle the resolution.  You may physically
; damage your monitor otherwise.  Consult your hardware documentation for
; display adapter and monitor specifications before proceeding.
;DisplayMode = 258    ; SVGA, 800x600, 16 colors, (VESA mode 102h)
;DisplayMode = 259    ; SVGA, 800x600, 256 colors, (VESA mode 103h)
;DisplayMode = 260    ; SVGA, 1024x768, 16 colors, (VESA mode 104h)
;DisplayMode = 261    ; SVGA, 1024x768, 256 colors, (VESA mode 105h)
;DisplayMode = 262    ; SVGA, 1280x1024, 16 colors, (VESA mode 106h)
;DisplayMode = 263    ; SVGA, 1280x1024, 256 colors, (VESA mode 107h)

; Command used to produce hardcopy (process .PLX files)
HardCopy = LJPLOT /Plpt1
;HardCopy = PSPLOT /Plpt1
;HardCopy = HPPLOT /Plpt1 /2

; Prompt before saving the current plot to the queue
PromptBeforeSaving = No

; Plot queue directory (home for .PLX files)
PlotQueue = .\

; Directory for temporary files. Should be on a RAM disk if possible
TempDir = .\
```

The DisplayMode entry specifies the graphics mode that the display adapter will be placed in when displaying graphics on your screen. If no entry is made, the best resolution mode up to 640×480 is automatically selected by default. Note that if you have a Hercules Graphics Card or compatible, you must run MSHERC.COM before using the programs that use the graphics library.

If your display adapter is VESA compliant, you may use the SuperVGA modes. Run VESA.EXE to see a list of VESA modes supported by your display adapter. Note that your monitor must by capable of displaying the mode that you select. Damage to your monitor may result if you select a mode beyond its capabilities. “Uncomment” the best DisplayMode entry that your display adapter and monitor can handle.

The PromptBeforeSaving entry is used to specify whether or not you should be prompted to save a plot when it is closed. This entry should be left set to NO.

The HardCopy entry is used to specify the command used to generate hardcopy plots on your system. Each program that generates hardcopy will create a binary plot file (hereafter called a PLX file) in the directory specified with the TempDir entry named PLOT*nnnn*.PLX, where *nnnn* is the number of the plot. This file is then processed to produce hardcopy.

Four programs are provided to process these PLX files: LJPLOT.EXE, PSPLOT.EXE, HPPLOT.EXE and VIEW.EXE.

LJPLOT prints to HP Laserjet or compatible printers. This program exploits the data compression features of the HP-PCL/5 printer control language to increase printing performance. The data compression features can be disabled when using older (earlier than II*p*) Laserjet printers. A help screen that describes the use of command line switches for this and the other print programs is available by simply typing the program name followed by a question mark on the command line (i.e., LJPLOT ?).

PSPLOT generates a POSTSCRIPT page description for the plot contained in the PLX file. Output from this program conforms to the Encapsulated POSTSCRIPT (EPS) conventions but does contain the bit-mapped screen image (TIFF) data. This EPS data is suitable for import into most desktop publishing and word processing programs.

HPPLOT generates HP Graphics Language (HP-GL) output. This output may be sent to a HP or compatible plotter or used for import to desktop publishing and word processing programs. This program can also generate HP-GL/2 output for use with HP Laserjet III or later laser printers.

VIEW is used to display the contents of a PLX file on the screen. The plot is displayed exactly as it would appear on paper.

The PlotQueue entry specifies the directory used to store PLX files before processing.

The TempDir entry specifies the directory where PLX files will be created. This directory should be on a RAM disk larger than 1 MB if possible or alternatively on your fastest hard disk with disk caching. Once the PLX file is created it is either processed immediately and deleted (depending on the particular program) or moved to the directory specified as the plot queue.

CHAPTER 2

A PC-SUDS Primer

This chapter presents a practical overview of the Seismic Unified Data System for DOS (PC-SUDS) as it is now commonly used. It is intended to familiarize you with some of the underlying concepts of PC-SUDS.

We begin by introducing PC-SUDS at the most basic level and progress through more advanced topics. This chapter is written for both the user of PC-SUDS based software tools as well as the application programmer that wishes to access PC-SUDS data from within their programs.

For the past several years myself and several others have been developing a series of software tools for processing seismic waveform data stored in the PC-SUDS. Most of this software is now part of the IASPEI Software Library. We have gained much experience regarding the use of PC-SUDS and below I try to pass some of the more important bits along to you.

What is PC-SUDS?

At the most basic level PC-SUDS is a method of storing digital seismic waveform data. Although PC-SUDS was designed to store seismic data, it is not limited to that application. PC-SUDS is designed to allow flexibility not provided by most traditional blocks-following-header type formats used to store seismic data. PC-SUDS can be easily extended to accommodate future needs without compromising older software tools' ability to read the data. This backward compatibility is very important as many gigabytes of data are now stored in PC-SUDS.

PC-SUDS has features that allow data files to be independent of hardware data representation (i.e., byte order, alignment, and floating point representation) although these features are only partially implemented currently. PC-SUDS is currently available only on personal computers running MS-DOS, although, conversions to other formats and platforms are available.

A Little History

SUDS version 1 was designed by Peter Ward of the U.S. Geological Survey (USGS) in Menlo Park, CA and is defined in USGS Open-file report 89-188. The open-file report describes SUDS version 1 as implemented on Sun-3 computers. Dean Tottingham of the TottCo Consulting Group, ported and implemented SUDS version 1 on the MS-DOS platform. This implementation has been used extensively in the IASPEI software library and become know as PC-SUDS. Robert Banfill of Banfill Software Engineering began maintaining the DOS implementation of SUDS at version 1.20. The current version of PC-SUDS is version 1.45.

Version 1.45 differs from the original version only slightly, and care has been taken to not disturb existing data by maintaining backward compatibility. The following is a summary of the changes:

- The I/O library was rebuilt to allow for unlimited data lengths and to provide language independence. Currently, the library is callable from C and FORTRAN and is easily adapted to most other languages.
- Two new structures were added: SUDS_INSTRUMENT and SUDS_CHANSET. These structures are used by most of the PC-SUDS utilities.

SUDS 2.x and PC-SUDS

SUDS version 2.x (SUDS-2) is currently under development by Peter Ward and many others. SUDS-2 has been developed with very broad goals. It is a format for seismic data, a relational database design and implementation, a table driven programming system, and a machine independent environment for data and software.

Although PC-SUDS is now a subset of SUDS-2, it should be thought of as a distinct entity. SUDS-2 provides the ability to read and write PC-SUDS data so that user may migrate to SUDS-2 over time. In the meantime, many software tools exist that work with PC-SUDS data only. SUDS-2 has been developed primarily on the UNIX platform but can be used on several others including MS-DOS.

For more information about SUDS-2 please contact Peter Ward (ward@andreas.wr.usgs.gov).

PC-SUDS as a File Format

PC-SUDS was originally designed to be much more than “yet another file format” (YAFF) even though that is by far its most common use. As a file format, PC-SUDS is quite flexible. A PC-SUDS file might contain no waveforms, one waveform from a single channel, many waveforms from a single channel, or many waveforms from many channels. These waveforms may also be grouped into sets, for example, the three channels from a three-component sensor can be grouped together as a channel set.

PC-SUDS data files may be concatenated using the DOS COPY command. This can be quite useful. For example, you might have records from many stations, each in its own file. You could join all of these files into one “event” file by concatenating each file to the event file as follows:

```
COPY /B station1+station2+station3 event
```

The /B (binary file) switch is very important, without it the combined file will likely be unreadable. If you see an error message saying “input file is out of sync” or “bad machine type”, you probably forgot the /B switch. For additional information about using the COPY command to concatenate files please refer to the DOS help system of your DOS user reference.

SUDS Data File Structure

PC-SUDS is based on self-identifying structures (which may or may not have data following them) stored as a linked list. These structures are typically stored in files on disk or tape. Data of any type may be stored in PC-SUDS files and there are no limits on the length of these data. PC-SUDS was designed to be easily extensible by allowing users to add new structures to meet their specific needs without compromising the ability of others to access the portions of the data that they recognize. Additionally, new fields may be added to the end of existing structures without interfering with ability of older software the read the structure. This allows data to be shared among users with a minimum of problems associated with the data format.

An I/O library is provided in the PC-SUDS Utilities for users who wish to develop software that will access PC-SUDS data files. The library manages the tag structures and allows the user to read, write or seek structures and data in the file without dealing with the tags, although an understanding of the file structure is helpful when using the library. The tag is a 12 byte structure that identifies the structure that immediately follows it, its length and the length of any data that may follow it.

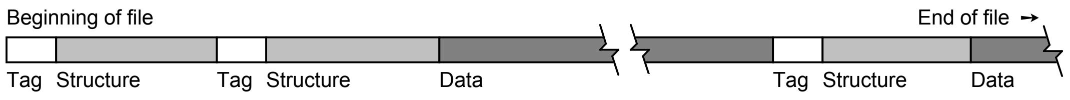


Figure 1 – PC-SUDS file structure

Using C syntax, the tag structure has the following form:

```
typedef struct _SUDS_STRUCTTAG {
   char  sync;
   char  machine;
   short struct_type;
   long  struct_length;
   long  data_length;
} SUDS_STRUCTTAG;
```

The `sync` element must always contain the character ‘S’. This is used to insure the file is not out of sync. The `machine` element identifies the hardware representation (byte order and floating point representation) of the data in the file, this must contain the character ‘6’ for Intel x86 based machines. The `struct_type` element indicates the type of PC-SUDS structure of `struct_length` (in bytes) that immediately follows the tag in the file. If any data follows the PC-SUDS structure, it is `data_length` bytes in length.

It important to note that you should make no assumptions about the order in which structures appear in the file. You should also make no assumptions about which type of structures are in the file. If you find a structure that you don’t recognize, you should simply move on to the next structure looking for structures that you do recognize. Further, you should make no assumptions about the length of any PC-SUDS structure. Additional fields may have been added to the end of an existing structure. The I/O library will take care of this for you, but if you are accessing the file directly, use the `struct_length` field in the tag struct instead of the sizeof operator to determine the structure length.

The IASPEI Library and PC-SUDS

IASPEI adopted PC-SUDS in 1989 as the data file format to be used by programs in the IASPEI Software Library. All of the software in the library that processes waveform data use PC-SUDS and require a minimum number of structures to be present in the file to perform their particular task. Please refer to appendices A and C for more about the minimum data requirement and a look into a typical data file.

CHAPTER 3

Earthquake Location Tools

This chapter covers programs used in the process of earthquake location. The software described includes:

- Trigger – This program performs off-line event detection for small to moderate sized microearthquake networks.
- AutoPick – This program performs automatic P phase and coda duration picking.
- XTRHY71 & PRT2SUD – These programs extract and insert phase information from/to SUDS files for use with Hypo71PC.
- SUDSPick – This is an interactive multi-component phase picker that can be used to pick P, S, and coda. This program can also be used to re-pick phases after AutoPick has been used for the preliminary solution.
- HypoMap – This program generates epicenter maps. It can be setup to display solutions in real-time and can also produce hardcopy.

These programs along with Hypo71PC or some other location program can be used to implement a fully automatic earthquake location system. A typical system might use AutoPick to pick P phases and coda duration and then use Hypo71PC to locate and estimate coda magnitude. The analyst can then use this preliminary solution when re-picking with SUDSPick to rotate the horizontal components into the radial and tangential orientation before picking the S phase. Once the P phases are adjusted (if needed) and S phases are picked, the event is then relocated with Hypo71PC or some other program to get a refined solution.

Trigger

Trigger is an off-line seismic network event trigger. “Off-line” means that the seismic data has been collected and processed to some degree before Trigger is run against it. Typically, data from several acquisition systems will have been merged before Trigger is used to analyze it. See the Merge program elsewhere in this document.

Trigger implements a straight-forward STA/LTA microearthquake network event detection algorithm as described in Lee and Stewart (1981)[3].

The purpose of the Trigger program is to be the event detection component of a near-real-time seismic network.

There were several goals we designed the Trigger program:

- It should be modular. By this we mean that the Trigger program should perform only event detection, it should not try to perform any other processing tasks. Various components of the PC-SUDS Utilities and other programs as needed will be used to perform all other processing tasks. It should integrate cleanly with this other software.
- It should be dependable. This means not only that the trigger algorithm must be reliable, but the program must run for long periods of time without errors or system instabilities.
- It should be as efficient as possible. This means that it must be able to process data at 6 or more times the rate that data is being collected so that system resources are available for other processing tasks such as archival, phase picking, hypocenter location, and plotting.
- It should be configurable. Any adjustments that are required in operation cannot require rebuilding the program. These adjustments are made via an ASCII initialization file that is editable by the user at run-time.

Implementation overview

A typical observatory will have several seismic data acquisition systems. For example, there might be a local earthquake network made up of analog radio telemetered stations, digital radio telemetered stations, and dial-up telemetered stations. In most cases, each of these methods of telemetry require a separate acquisition system.

In conventional systems, event detection occurs during data acquisition. This is a problem because in order to detect events over the entire network, the data from each of these systems must be merged into a single stream of data and then analyzed for events.

In this system, each of data acquisition systems generate data continuously. These data are then merged into a single data stream. Before merging, the data from each system may require various preprocessing such as demultiplexing and time correction. These continuous data may also be archived to a high capacity storage device as well.

[3]Lee, W. H. K., and Stewart, S. W. (1981). "Principles and Applications of Microearthquake Networks." Academic Press, New York, New York.

The Trigger program is used to analyze this merged data in near-real-time and detect events so that they may be processed further. Typically, this further processing consists of determination of phase arrivals, hypocenter location, and plotting of a record-section.

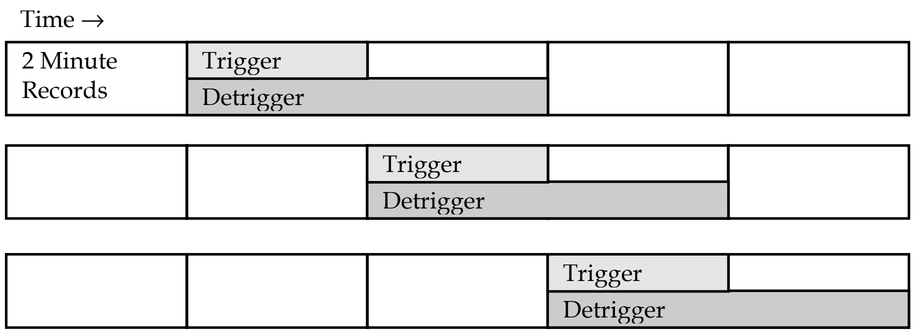
All of this processing must occur in near-real-time, that is, the system as a whole must process data faster than it is being recorded, but it need not do everything instantaneously. This implies that various latencies will introduced along the processing path. These latencies will manifest themselves as delays between the time an event occurs and when the system triggers, locates, and reports the event. As implemented, these latencies do not exceed 2 or 3 minutes.

Using Trigger

Trigger is used to analyze multi-channel seismic records and detect events. The program is geared toward processing continuous data that is broken into records 2 minutes in length.

In order to achieve continuous coverage of the input data, the program triggers on only the first half of the input record, if an event is detected, it then detriggers on the entire remaining record. This means that you must save the current record and prepend it to the next record and repeat this process over and over. The example PROC.BAT file below demonstrates a method of doing this.

Trigger saves the state of the detector between invocations of the program. The figure below shows how continuous coverage works. If a large event occurs, the program can look for detriggers for more than the next record. You specify the maximum possible event length in the .INI file.



When an event occurs, an event file is prepared for picking and the input record is set aside for later use. This event file is constrained to be rather short so that the phase picking software will be able to successfully process it.

When an event is longer than the input data record length, all files from the trigger time through the detrigger time are saved.

Command line syntax

To see a help screen, type TRIGGER ?.

```
Trigger - Version 1.00

TRIGGER Filespec

Switches:
    /C  - Continue previous event. (No)
    /Dstn - Dump debug info for station name stn. (Off)

Arguments:
   Filespec - The file to process.

Exit Codes:
   0   - No event detected.
   1   - Event detected within frame, no FMP file created.
   2   - Event detected within frame, FMP file created.
   3   - Event still in progress, call next frame with /C switch.
   4   - Multiframe event has ended, FMP file created.
   255 - Fatal error.

[]=Optional, ()=Default. Switches are not case sensitive and may appear
anywhere after the command in the command line.
```

Trigger expects one argument: the file specification of the PC-SUDS data file to detect events in. The /C switch tells the program that an event is still in progress and it will only look for detriggers this time. This switch is used after a exit code of 3 was returned.

Trigger returns exit codes to indicate status. Use the IF ERRORLEVEL construct in your batch program to test the exit codes. Note that IF ERRORLEVEL tests whether the exit code is *greater than or equal to* the given value so you must test the value in reverse order. The following batch program fragment demonstrates how to do this. Please see Appendix B for a complete example.

```
@ECHO OFF
IF NOT EXIST %1 GOTO END

IF EXIST PREVIOUS.FIL GOTO JOIN
MOVE %1 PREVIOUS.FIL
GOTO CLEANUP

:JOIN
SUDSJOIN PREVIOUS.FIL %1 CURRENT.FIL
MOVE %1 PREVIOUS.FIL

IF EXIST C:\PCSUDS\HOLD.TRG TRIGGER CURRENT.FIL /C
IF NOT EXIST C:\PCSUDS\HOLD.TRG TRIGGER CURRENT.FIL
IF ERRORLEVEL 255 GOTO ERROR
IF ERRORLEVEL 4 GOTO PICK2
IF ERRORLEVEL 3 GOTO HOLD
IF ERRORLEVEL 2 GOTO PICK2
IF ERRORLEVEL 1 GOTO PICK1
GOTO CLEANUP
```

```
:HOLD
ECHO "Hold, multifragment event in progress!" > C:\PCSUDS\HOLD.TRG
GOTO END

:ERROR
ECHO Trigger had a fatal error!
PAUSE
GOTO END

:PICK1
IF EXIST C:\PCSUDS\HOLD.TRG DEL C:\PCSUDS\HOLD.TRG
REM - Pick P, S, and F (coda) phases here

GOTO LOCATE

:PICK2
IF EXIST C:\PCSUDS\HOLD.TRG DEL C:\PCSUDS\HOLD.TRG
REM - Pick P, and S phase only here, Trigger provides F (coda)

GOTO LOCATE

o
o
o

:END
```

The event detection algorithm

The program first detects and confirms triggers on each channel. Then it analyzes these channel triggers and determines in an event trigger has occurred. The detection algorithm is based on Lee and Stewart (1981). The algorithm is implemented using integer arithmetic for performance reasons.

We start with the raw signal in a large array of integer samples: x_i. We then condition the signal by computing the rectified first difference.

$$dx_i = \left| x_i - x_{i-1} \right|$$

The first test is whether the average amplitude of the signal exceeds the value given in the Critical_Amplitude entry in Trigger.INI.

$$avg\_amp = \frac{1}{N} \cdot \sum_{i=1}^{N} dx_i$$

If avg_amp does not exceed the value given for Critical_Amplitude, the channel is disqualified at this point and no further processing is attempted.

Next we compute the short-term average and long-term average using a recursive approximation.

$$sta_i = sta_{i-1} + \frac{(dx_i - sta_{i-1})}{sta\_window} \qquad lta_i = lta_{i-1} + \frac{(sta_i - lta_{i-1})}{lta\_window}$$

The values of *sta_window* and *lta_window* are read from the Trigger.INI at startup, see below.

We then compute α and β as follows.

$$\alpha_i = \frac{dx_i}{lta_i} \qquad \beta_i = \frac{sta_i}{lta_i}$$

First we see if α_i exceeds the value of Critical_Alpha. If it does, an alpha trigger is declared. We then save the time at that point and see if β_i exceeds Critical_Beta. If it does, a beta trigger is declared and counted. When the cumulative number of beta triggers exceeds Trigger_Confirmation_Count, a channel trigger is declared. If a value is given for LTA_Freeze and when the cumulative number of beta triggers reaches that value, the lta is held at its current value.

If the channel has triggered, the remaining signal is analyzed for detrigger.

$$\gamma_i = \frac{sta_i}{lta_i}$$

When γ_i is less than or equal to Critical_Gamma, a gamma detrigger is declared and counted. When the cumulative number of gamma detriggers reaches Event_Continuation_Count, the channel is detriggered.

Once all channels have been analyzed, the program analyzes the channel trigger information to see if an event trigger is called for. Critical_Nu channels must trigger within the Trigger_Time_Limit before an event is declared. When an event is declared, the trigger time is taken from the earliest channel trigger within the window.

An event is detriggered at the point were less than Critical_Mu channels remain triggered. The detrigger time is the time at that point.

Trigger.INI

Trigger looks for a file named Trigger.INI in its home directory at startup. If this file is found, the entries in the [TRIGGER] section are read. The entries allow you to control the setting of the various parameters described above.

Below is the contents of a sample Trigger.INI file. The settings in this sample are the built-in defaults used by the program if the .INI file is not found or a particular entry is not defined or valid.

```
; TRIGGER.INI
; This file contains initialization data for the TRIGGER program.

; This file is arranged in sections.  Each section is marked with a
; "Section Header" inside square brackets (e.g., [TRIGGER]). Each program
; will look for its section and then read the entries following its
; section header.  None of these entries are case-sensitive.  Comments
; are delimited by a pound-sign (#) or a semi-colon (;).  Blank lines
; are ignored.

;---------------------------------------------------------------------
[TRIGGER]

; Average rectified first difference of waveform must exceed this value.
Critical_Amplitude = 15          ; Counts.

; --- Length of STA and LTA windows ---
STA_Window = 16                  ; Samples, will round down to next power of 2.
LTA_Window = 64                  ; STA windows, will round down to next power of 2.

; --- Channel trigger settings ---
Critical_Alpha = 10              ; Alpha (x[i]/LTA) trigger threshold.
Critical_Beta = 4                ; Beta (STA/LTA) trigger threshold.
Trigger_Confirmation_Count = 30  ; Cumulative beta triggers needed to
                                 ;  trigger a channel.
LTA_Freeze = 4                   ; LTA is frozen when cumulative beta
                                 ;  triggers reaches this value.

Critical_Gamma = 2               ; Gamma (STA/LTA) detrigger threshold.
Event_Continuation_Count = 30    ; Consecutive gamma detriggers needed to
                                 ;  detrigger a channel.
; --- Event trigger settings ---
Critical_Nu = 10                 ; Number of channel triggers inside of the
                                 ;  trigger window needed to trigger event.
Trigger_Time_Limit = 15.0        ; Trigger window in seconds.
Critical_Mu = 3                  ; Detrigger event when less than this
                                 ;  many channels remain triggered.
; --- Record lengths ---
Pre_Event_Length = 10.0          ; Seconds.
Post_Event_Length = 10.0         ; Seconds.
Min_Event_Length = 40.0          ; Seconds, this applies only to pick file.
Max_Event_Length = 80.0          ; Seconds, this applies only to pick file.

Max_Save_Length = 600.0          ; Seconds, the maximum time that an event
                                 ;  run without forced detrigger.
; --- Paths ---
Event_Data_Path = .\EVENTS       ; When an event is declared, input data
                                 ;  will be copied to this directory.
Pick_File_Extension = EVT        ; Event file for Pick will have this ext.
FMP_File_Extension = FMP         ; Coda will be saved to a SUDS file with
                                 ;  this extension.
```

The entries down through Event trigger settings are described in the previous section. Here we will cover only the parts of the file labeled Record lengths and Paths.

The entries; Pre_Event_Length and Post_Event_Length are the amount of signal that will be saved before and after the trigger and detrigger times.

The Min_Event_Length and Max_Event_Length entries are the minimum and maximum length allowed for the event file generated when an event occurs.

The Event_Data_Path entry is the name of the directory where a copy of the input data file will be placed when an event occurs. Note that if the event is longer than the record length of the input data, all data files containing the event will accumulate in this directory.

When an event is declared, Trigger generates an event file according to the length parameters above and names it with the same filename as the input file with the extension given in the Pick_File_Extension entry. This file is used to pick phases for location. Note that the input file is copied to the directory specified above as well.

If the event is longer than the Max_Event_Length entry above, the detrigger times detected by Trigger are stored to a file name with the same name as the input file and the extension given as FMP_File_Extension. This is needed because most phase picking software can only deal with relatively short records and may not be able to pick F phases (coda) on larger events. In this case, the detrigger times are used to compute the F minus P interval (FMP) so that the location software can estimate coda magnitude.

AutoPick

AutoPick is an automatic P-phase arrival and coda duration picking program for use with SUDS data files. The program can be constrained to pick arrivals on vertical components only if component information is available in the input file. Coda duration is only picked when the P-phase pick weight meets or exceeds a user specified value.

Phase arrival data is stored in the SUDS data file using SUDS_FEATURE structures. Location programs can read data directly from and store solutions directly (using SUDS_ORIGIN structures) to the SUDS data files or these data can be extracted, used and then appended to the SUDS data files.

The program is designed to run without user interaction although a screen display mode is available to help you tune the picking algorithm. The algorithm used is loosely based on Rex Allen's paper; Automatic Earthquake Recognition and Timing from Single Traces[4]. A detailed description of the actual algorithm is presented later in this section.

Command line syntax

AutoPick is controlled from the DOS command line and by settings in the [AutoPick] section of the .INI file. The program returns an exit code of 0 to indicate successful execution and 1 to indicate that an error occurred during processing. If an error does occur, a verbose message will be written to stdout (the screen) describing the error.

Help may be obtained by typing AutoPick ? and pressing return at the command line. The following screen will be displayed:

```
AUTOPICK - Version 2.40

Usage: AUTOPICK [switches] InputFilespec [switches]

Switches:
   /A - Pick all traces (verticals only)
   /D - Display waveforms on screen

[] = optional, () = default.
Arguments are not case sensitive.
```

Arguments

[4]BSSA, Vol. 68, No. 5, pages' 1521-1532, Oct 1978.

AutoPick expects a single SUDS file specification as the only argument on the command line. The program will update phase arrival data in existing SUDS_FEATURE structures in the input file and create and append new structures as required. Phase information can be extracted into a Hypo71PC compatible phase list by using the XTRHY71 utility

Switches

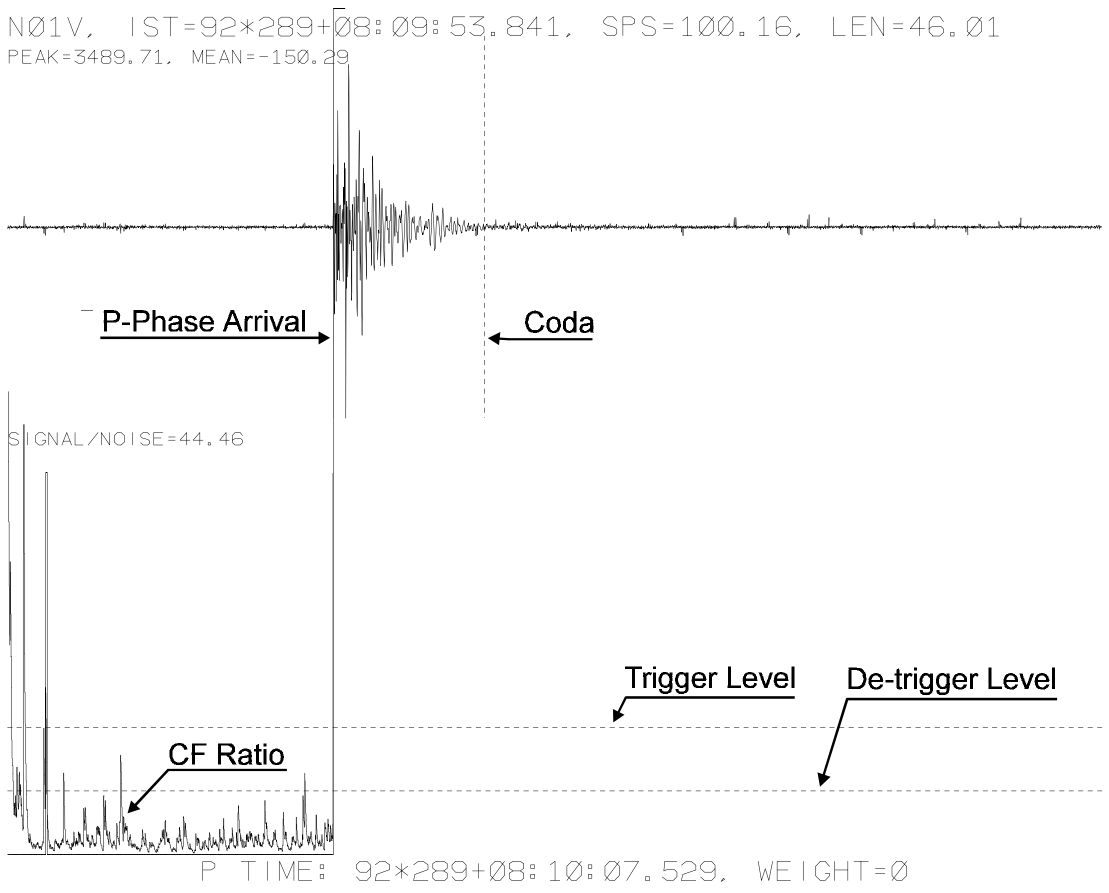
/A - Pick all traces

By default, the program will pick only vertical components. This switch specifies that you wish for the program to pick all available waveforms in the input file.

The program recognizes vertical components by examining the component descriptor contained in the SUDS_STATIDENT sub-structure used in SUDS_DESCRIPTRACE and other structures. If the program finds an upper or lower case 'v', 'u' or 'z' as the component descriptor the waveform is considered a vertical component and is therefore picked. If no component descriptor is available in the data file (component descriptor = '_'), you must use this switch to cause the program to pick all traces.

/D - Display waveform while picking

By default, the program simply displays a line on the screen that contains the P-phase arrival time, station and weight for each pick. This switch causes the program to display the following screen while each waveform is being picked. The information on this screen is useful while tuning the picking algorithm to suit your particular needs.



The algorithm

In this section we will cover each step of the picking algorithm in detail. The following equations are very much the same as those presented in Rex's paper.

We start with the raw waveform read from the SUDS data file and convert it to reals in a large array *N*.

First we filter and remove any DC offset using the following equation:

$$R_i = C_1 \cdot R_{i-1} + \left(N_i - N_{i-1}\right)$$

where:

$$C_1 = 1.0 - \left[\frac{CornerFrequency}{SamplingRate \cdot 0.5}\right]$$

The value of *CornerFrequency* is taken from the High_Pass entry in the .INI file and *SamplingRate* is taken from the SUDS data expressed as samples per second. This equation acts as a single pole high-pass filter.

Next we compute the weighted derivative with the following equation:

$$\Delta r_i = C_2 \cdot \left(N_i - N_{i-1}\right)$$

The value of C_2 is taken from the D_Weight entry in the .INI file.

Next we compute the characteristic function. The short-term-average or α and long-term-average or β are computed from the result of this function. This particular function is chosen because it contains quantities descriptive of both amplitude and frequency of the waveform, it varies rapidly with changes in either of these parameters and is always positive.

$$E_i = R_i^{\,2} + \Delta r_i^{\,2}$$

Next we compute α and β using the following equations:

$$\alpha_i = \alpha_{i-1} + C_3 \cdot (E_i - \alpha_{i-1})$$

$$\beta_i = \beta_{i-1} + C_4 \cdot (E_i - \beta_{i-1})$$

where:

$$C_n = \left[\frac{2.0}{Seconds \cdot SamplingRate}\right]$$

The value of *Seconds* is taken from the Short_Term_Avg entry in the .INI file for α and the Long_Term_Avg entry for β. *SamplingRate* is again taken from the SUDS data and is expressed as samples per second.

As each sample is processed the ratio of α/β is examined. If this ratio exceeds the Trigger_Ratio value specified in the .INI file, an event is declared. Once an event is declared the ratio must drop below the Detrigger_Ratio specified in the .INI for the event to end. If LTA_hold is specified in the .INI file, the value of β is held constant for the duration of the event.

If the event length is less than the Minimum_Length required as specified in the .INI file, the event is ignored. Once an event lasts the minimum length required, the pick is declared at the trigger point.

Once a pick is declared, the mean absolute amplitude of *n* seconds on either side of the pick is computed where *n* is the value specified in the Weighting_Window entry in the .INI file. The signal (the window after the pick) to noise (the window before the pick) ratio is computed and used to classify the weight of the pick using Weight_*n* entry values from the .INI file.

If the pick weight is less than or equal to the value given as Coda_Weight in the .INI file, the window after the pick is moved down the waveform. When the signal to noise ratio drops below the value given as Coda_Ratio in the .INI file, this point is marked as the coda duration and F-P time is computed.

INI file entries

The following is an example [AutoPick] section of SUDSUTIL.INI. This represents all of the entries recognized by the program.

```
[AUTOPICK]
; Entries for AUTOPICK 1.00

; Length of time before picking in seconds
Settle_Time = 1.5

; Corner frequency (Hz) of single pole High-Pass filter before picking
High_Pass = 2.0

; Weight used while computing derivative for characteristic function
D_Weight = 1.0

; Length for short-term average computation in seconds
Short_Term_Avg = 0.1

; Length for long-term average computation in seconds
Long_Term_Avg = 4.0

; "Trigger" ratio, event has started
Trigger_Ratio = 8.0

; "Detrigger" ratio, event has ended
Detrigger_Ratio = 1.5

; Freeze LTA during event (Yes/No)
LTA_Hold = Yes

; Minimum length of event in seconds
Minimum_Length = 0.4

; Weighting window (seconds) and weighting criteria (signal/noise ratio)
Weighting_Window = 2.5 ; Length of window before (noise) and after (signal)
                       ; pick to compute absolute amplitude
Weight_0 = 20.0        ; Ratio >= n weight = 0
Weight_1 = 15.0        ; Ratio >= n weight = 1
Weight_2 = 10.0        ; Ratio >= n weight = 2
Weight_3 = 5.0         ; Ratio >= n weight = 3, ratio < n weight = 4

; Compute Coda duration for picks with weight >= n
Coda_Weight = 2

; Signal to noise ratio at Coda (F-P)
Coda_Ratio = 2.0
```

XTRHY71 & PRT2SUD

These two programs provide an interface to Hypo71PC. XTRHY71 extracts a Hypo71PC phase list from a SUDS data file and PRT2SUD reads a .PRT output file from Hypo71PC and writes a SUDS_ORIGIN structure back into the SUDS data file. Writing the hypocenter solution back into the data file is useful when re-picking with SUDSPick.

Using XTRHY71

XTRHY71 reads a SUDS data file, searching for SUDS_FEATURE structures, and writes a ASCII phase list to be read by Hypo71PC. To see the following help screen, type XTRHY71 and press return:

```
XTRHY71 - Version 1.00 - Extract phases from SUDS in HYPO71PC format.

Usage: XTRHY71 InputFilespec [OutputFilespec]

  If no output file is specified, the output filespec will
  be the same as the input filespec with an extension of ".PHA"

[] = optional, () = default.
Arguments are not case sensitive.
```

The program attaches a Hypo71 instruction card to the end of the phase list. This card sets KNST=0 to indicate that S arrivals are not to be used and INST=0 to indicate that Hypo71 should calculate focal depth. See the Hypo71PC documentation for more details about these settings.

Running Hypo71PC

Typically, you would create the input file for Hypo71 in advance. The phase list would be concatenated to a copy of this template input file and used to run Hypo71. Hypo71 would then create a “printer output file” or .PRT file which would then be read by PRT2SUD to insert the hypocenter solution back into your SUDS file. This process can be easily accomplished with a simple batch program as follows:

```
@ECHO OFF
REM - GOHYPO.BAT

IF NOT EXIST %1.PRT GOTO NOPRT
IF NOT EXIST C:\SUDS\HYPO71\NETWORK.HDR GOTO NOHDR

COPY C:\SUDS\HYPO71\NETWORK.HDR+%1.PRT HYPO71PC.PRT

HYPO71PC < C:\SUDS\HYPO71\RETURNS.TXT

IF NOT EXIST HYPO71PC.PRT GOTO END
```

```
COPY HYPO71PC.PRT %1.PRT
GOTO END

:NOPRT
ECHO No input PRT filename!
PAUSE
GOTO END

:NOHDR
ECHO C:\SUDS\HYPO71\NETWORK.HDR does not exist!
PAUSE

:END
ECHO y | DEL HYPO71PC.*
```

This batch program is called with the filename (no extension, .PHA is assumed) and if successful, a PRT file with the same name and a .PRT extension will be created. The batch program above makes several assumptions; first, a file named NETWORK.HDR must exist in the C:\SUDS\HYPO71 directory; second, a small text file named RETURNS.TXT containing three carriage return/line feeds must exist in this same directory. NETWORK.HDR should contain all of the control and station cards for Hypo71 and be ready to have the phase list attached to the bottom. See the Hypo71 documentation for detailed instructions.

Using PRT2SUD

This program simply reads the hypocenter solution line (card) from a PRT file and creates a SUDS_ORIGIN structure and appends it to the end of a SUDS data file. To see a help screen type PRT2SUD and press return:

```
PRT2SUD - Version 1.00 - Read PRT file, output SUDS_ORIGIN.

Usage: PRT2SUD PRTfilespec SUDSfilespec

Arguments are not case sensitive.
```

SUDSPick will use this information along with the station location to allow auto-rotation of the horizontal components into radial and tangential orientation. When re-picking with SUDSPick, you can adjust the P arrivals and, with auto-rotation, pick the S arrivals (see below).

SUDSPick

SUDSPick 2.x is an interactive, multi-component phase picking program. The program can be used to pick P and S phases and coda duration. The program reads existing phase information from the SUDS data file and allows you to update these picks. It offers user defined Butterworth filters and allows rotation of horizontal components. Given station and hypocenter locations, the program can automatically rotate the horizontal components into the radial and tangential directions relative to the hypocenter.

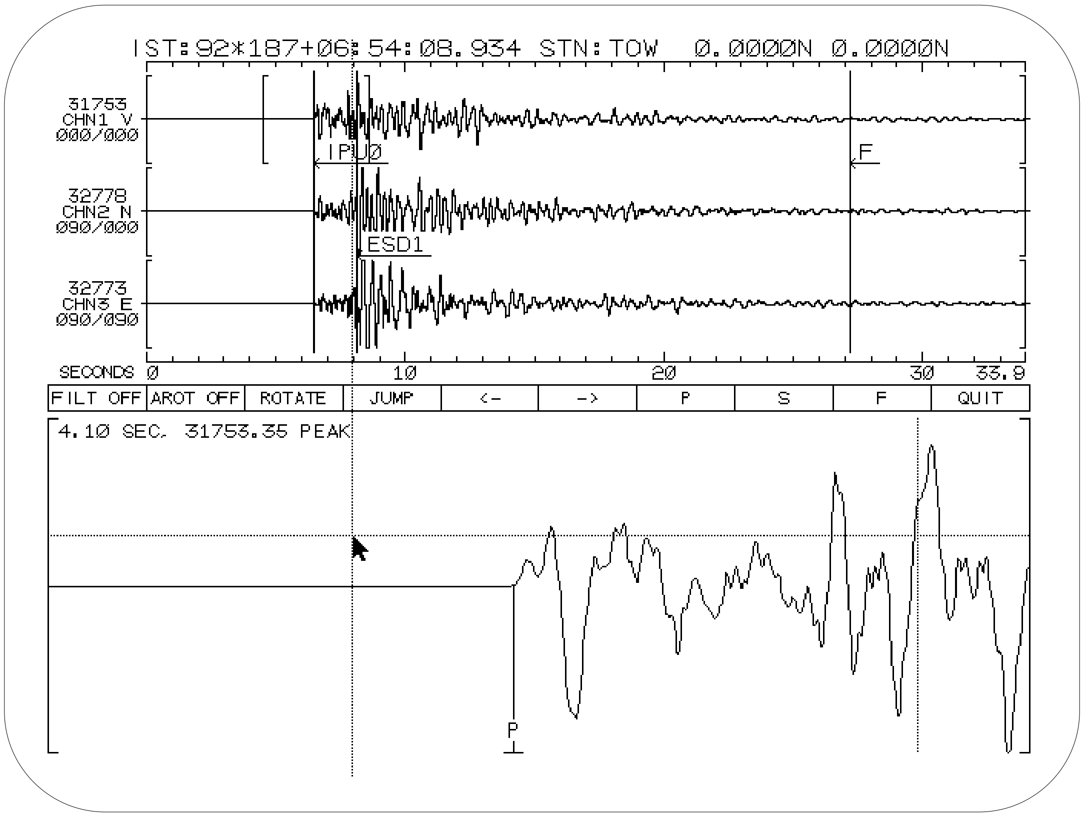
This is the second major version of the SUDSPick program. The previous version (1.x) is now obsolete and its section; [SUDSPick] in SUDSUTIL.INI can safely be removed. Also this new version uses the 2.x version of the PlotX graphics library. None of the current utilities use the now obsolete PlotX 1.x library so the [PLOTX] section of the .INI file can also safely be removed.

Using SUDSPick

SUDSPick expects the file specification of the input SUDS data file to be passed on the command line.

In order for SUDSPick to deal with three component data as such, SUDS_CHANSET structures must be in the input file that define each three component set. Please refer to the SUDS Library chapter later in this manual for more information about these structures.

Once SUDSPick has indexed the input file you should see the following screen:



The program is controlled either by the mouse or keyboard. The crosshair cursor is moved about on the screen and key or button presses invoke various commands. The screen is divided vertically into three regions; the upper region that contains the waveforms in their entirety, the menu bar that can be pointed to invoke commands and the lower portion that displays an adjustable window onto one of the waveforms in the upper portion.

Because most users have a mouse, we will cover mouse based operation and add keystroke equivalents as needed.

Probably the most common operation is moving the window displayed in the lower portion of the screen. This is accomplished by several means as summarized below. Please refer to the figure above as necessary.

Window movement

Move the mouse cursor into the upper portion of the display. Place the cursor on a waveform at the desired position of the left side of the window. Press the left mouse button. Then move the cursor to the desired position of the right side of the window and again press the mouse button.

You may move the cursor using the arrow keys on the numeric key pad and press the spacebar to set the window.

You may adjust the left or right side of the window by moving the cursor to the desired position and pressing the L or R keys respectively.

You may move the window to the previous or next waveform in the set without moving its position in time by pressing the Home or End keys.

You may move the crosshair into the upper portion of the display without moving the window by dragging it with the right mouse button.

Once the window is adjusted to the position of interest phases are picked in the lower portion of the display.

Picking phases

Move the cursor into the lower portion of the screen. Position the crosshair at the position of the pick and then press the mouse button or spacebar (note that you can drag the crosshair by holding down the mouse button).

You may pick P, S or F (coda) by moving the cursor onto the menu bar and pressing the mouse button or by pressing the P, S or F keys. Once you have selected which phase you are picking you will be prompted for it's weight. Note the uncertainty bars displayed in the lower portion of the screen. The size of these bars is controlled by settings in the .INI file (see below). A weight of 0 always means the uncertainty is within the sampling period. The first bar is the 1 weight uncertainty, the second is for a weight of 2 and so on.

After picking a weight you will be prompted for the direction of first motion and then the onset. These prompts again may be answered by selecting menu items with the mouse, by pressing the first letter key or pressing return to leave them unchanged and continuing.

A pick may be cleared by moving the crosshair onto the waveform in the upper or lower portion of the display and pressing C. You will be prompted for which pick to clear; P, S or F. Press the P, S, F or Escape key in response.

Navigating the SUDS data file

You may move to the previous or next set of waveforms in the SUDS data file by moving the cursor to the <- or -> buttons on the menu bar and pressing the mouse button. You may also do this by pressing the PgUp or PgDn key as well.

You may jump to any set in the data file by moving the cursor to the Jump button on the menu bar and pressing the mouse button. You will be presented with a table of available sets in the data file. Simply choose the desired set by moving the cursor onto it and pressing the mouse button or spacebar. This table may contain several pages, these are represented by the <- and -> cells. The <<

cell allows you to return to the main menu bar without change (equivalent to pressing Escape).

Filters

The .INI file contains parameters for high and low pass Butterworth filters (see below). These filters may be applied to the waveforms at any time. You may toggle these filters on and off by moving the cursor to the FILT button on the menu bar and pressing the mouse button or by pressing the F2 function key.

Rotation

Given that you are working with orthogonal three component data with valid channel sets defined and component orientations supplied in the SUDS data file, the program can rotate the horizontal components into any arbitrary azimuth. This is accomplished by choosing the ROTATE button from the menu bar or pressing the F3 function key. A dialog box will prompt you for the desired radial azimuth and then rotate the second component into this azimuth. The third component is rotated as well and is assumed to be perpendicular to the second. The orientation data to the right of the waveforms in the upper portion of the screen will reflect this rotation.

If the data file contains a hypocenter location (in a SUDS_ORIGIN structure) and the station location (in the SUDS_STATIONCOMP structure), the program can automatically compute the radial azimuth relative to the hypocenter and rotate the horizontals into the radial and tangential directions. If a station location is available for this set, it will be displayed in the upper right corner of the screen. You can check for the presence of a hypocenter location by pressing the O key. Auto-rotation is enabled by choosing the AROT button on the menu bar or by pressing the F4 function key.

INI file entries

The following is an example [2.X_SUDSPick] section of SUDSUTIL.INI. This represents all of the entries recognized by the program.

```
[2.X_SUDSPICK]
; Entries for SUDSPICK 2.0 or later.

; Maximum number of samples to read per trace.
Maximum_Samples = 16000

; Baseline length in seconds.
Baseline = 1.0

; Filter(s) below On or Off by default, can be toggled on menu.
Filter = Off

; Length of cosine taper at start of waveform when filtering in seconds
```

```
Taper = 3.0

; Low pass filter corner frequency in Hz.
Low_Pass_Corner = 10.0
; Low pass filter poles (1 pole = 6 dB per octave, value will be rounded
; up to the nearest factor of 2, range: 2-10).
; Set to 0 to disable filter.
Low_Pass_Poles = 4

; High pass filter corner frequency in Hz.
High_Pass_Corner = 0.1
; High pass filter poles (1 pole = 6 dB per octave, value will be rounded
; up to the nearest factor of 2, range: 2-10).
; Set to 0 to disable filter.
High_Pass_Poles = 0

; 75% (1 weight) timing uncertainty in seconds.
1_Weight_Uncertainty = 0.05

; 50% (2 weight) uncertainty = 75% (1 weight) uncertainty times this value, etc.
Uncertainty_Multiplier = 2.0
```

HypoMap

HypoMap is a program that displays hypocenter information on a map. The map may also contain line data, station locations, and annotation. The program polls a directory for the appearance of .PRT output files from Hypo71PC and plots the hypocenters on the map as they appear.

The symbol plotted for each hypocenter is a circle whose size is proportional to magnitude or depth. The latest event is plotted as a different color than the other events. The program offers user-definable classes for hypocenters. Each class is specified as the symbol size and the lower limit of depth or magnitude.

The current version of the program uses a general orthographic projection. This allows you to view any location on the globe at any scale. Other projections will be added in later versions. Coastlines and international border line data is included in this package. A utility program (VPFGet) is provided to allow you to extract line data from the DCW[5]or other VPF format database.

Using HypoMap

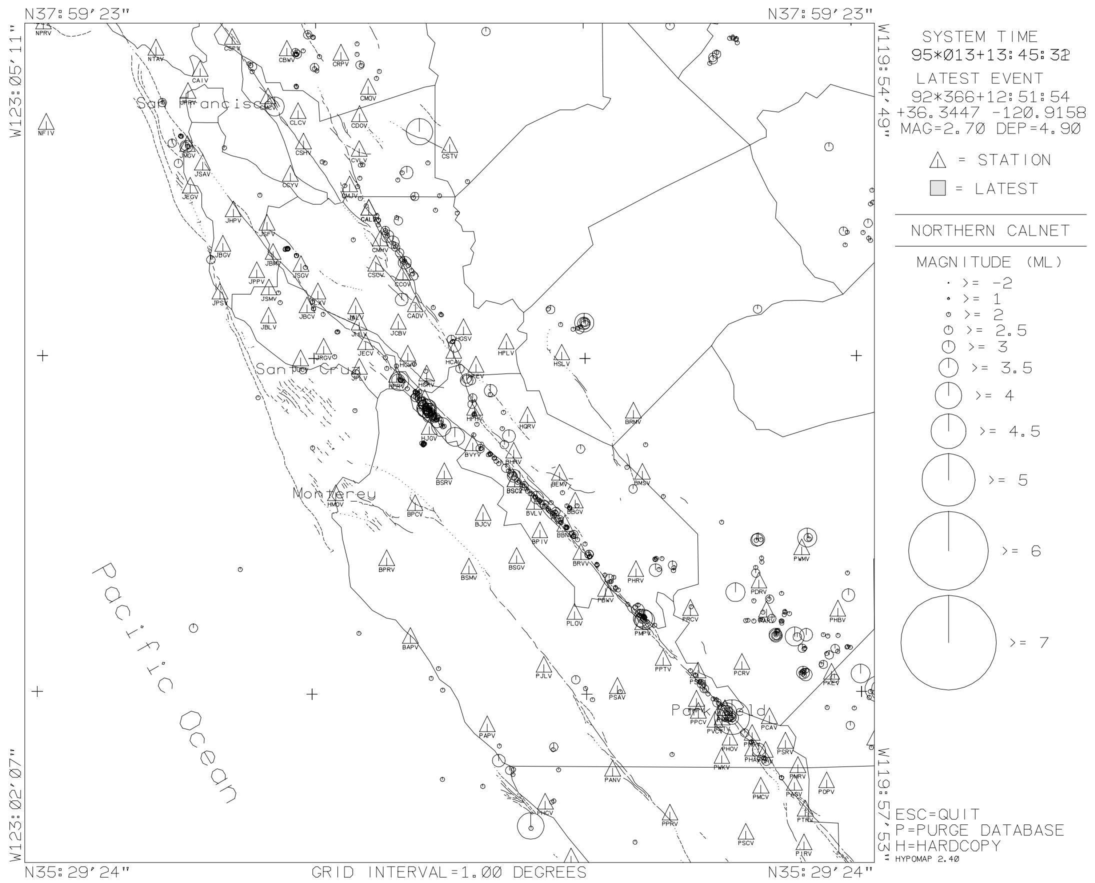
When HypoMap is executed, it looks for an initialization file named HypoMap.INI in it's home directory. Once the program has read the .INI file, it displays the map. Once the map is displayed, the program will look for printer output (.PRT) files from Hypo71PC in the directory specified in the .INI file. When a .PRT file is found, it will add the event in the .PRT file to the database and display it on the map.

The program can automatically purge the event database at a user specified interval. This provides a sliding time window on the events. For example, you might specify that only events for the last 72 hours be displayed, and that the database should purge every 6 hours.

When you exit the program, it will save the current event data to a file named HypoMap.EVN in it's home directory. On startup, it will read this file in and display the events found in this file. This allows you to stop the program at any time and then return to the same state and resume operation. If you wish for the program to store this data in another file, pass the desired file specification on the command line.

While the map is displayed, press escape to quit, h to produce hardcopy, or p to purge the database. By pressing h, you cause the program to generate a hardcopy plot of the map as it is currently displayed.

[5]Digital Chart of the World (4 CD-ROM's). Available from USGS Map Sales, Denver Federal Center, Box 25286, MS 306, Denver, CO 80225 (303) 236-7477.



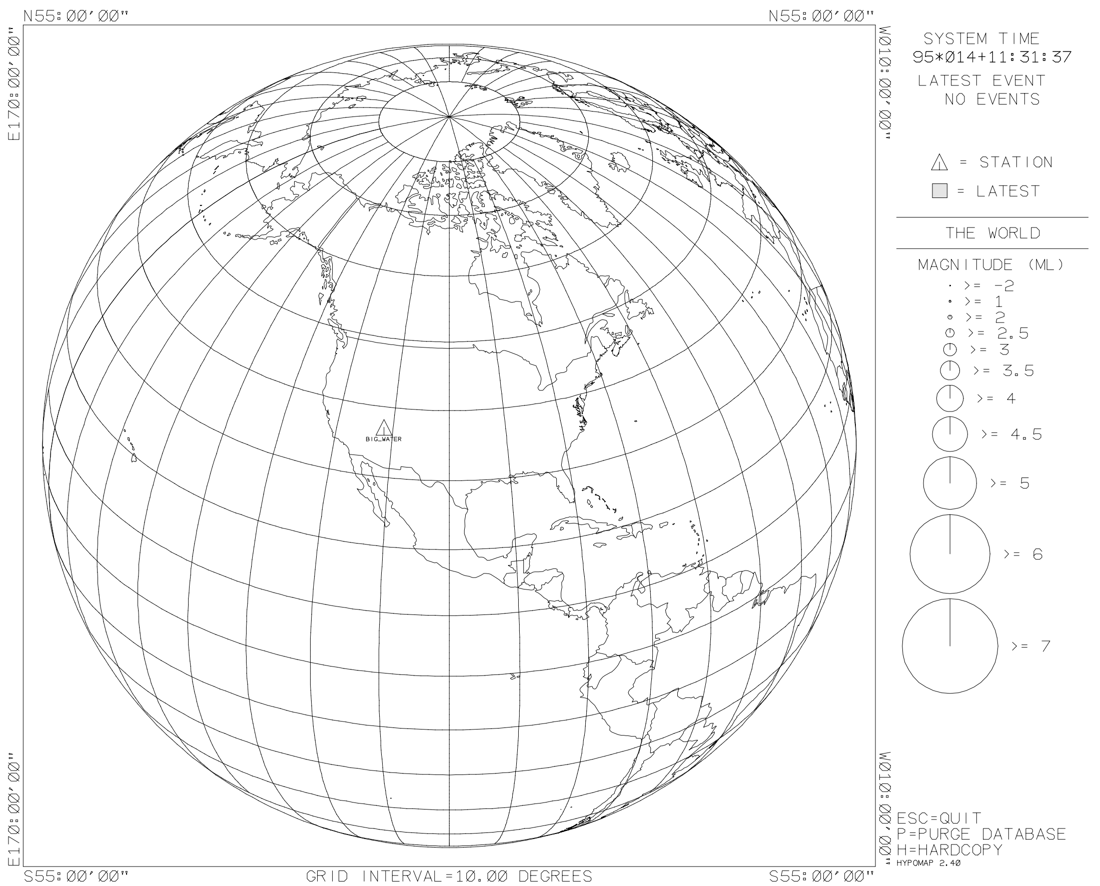
HypoMap.INI

This file is organized in sections that begin with a section header in square brackets followed by entries of the form: *keyword = entry*. This is the same format used in SUDSUTIL.INI.

The .INI file contains two required sections; [2.X_PLOTX] and [HYPOMAP]. Other sections may be added to this file for use by other programs and conversely, the HypoMap section may be added to SUDSUTIL.INI. You may specify that HypoMap should look in a different file passing that filename on the command line using the /I switch (e.g., /Ic:\suds\sudsutil.ini).

The PlotX section contains information about the display and printer on your system. For a detailed discussion of the entries in this section, please refer to chapter one of this document.

The HypoMap section contains all of the entries for control of HypoMap. These entries are covered in detail below



Date_Format = *DOY | Month/Day*

This entry is used to specify the format used to display date values. Valid entries are: DOY or Month/Day. The DOY or day-of-year format is "92*004+23:43:12.854". The Month/Day format is"01/04/92 23:43:12.854". DOY is the default value.

Station_File = *filespec*

This entry is used to specify the station file. This file contains the identifier and coordinates of the seismic stations to be plotted on the map. At each station, a small triangle symbol will be plotted with the station identifier below it. If no stations are to be plotted, comment out this entry be placing a semi-colon in the first column

The station file is a simple ASCII file containing one line for each station. Each line begins with a 4 or less character station identifier followed by, one or more spaces, the latitude (north is positive) in decimal degrees, one or more spaces, and the longitude (east is positive) in decimal degrees. There is no limit on the number of stations that may be plotted on the map.

The following is a few lines from the sample station file provided with the program:

```
NFIV 37.698333 -123.000000
NTAV 37.923833 -122.595000
CADV 37.163833 -121.625833
CAIV 37.861333 -122.429500
CALV 37.451167 -121.799167
CBWV 37.924167 -122.106667
CCYV 37.551667 -122.090833
CCOV 37.257667 -121.672500
```

Note that west longitude is negative.

See also: Station_Size, and Station_Color.

Line_Data_File = *filespec*

This entry is used to specify the geographic line data file. This file contains any type of line data to be plotted on the map. If no line data is to be plotted, comment out this entry by placing a semi-colon in the first column.

HypoMap can read two different line data formats: binary or ASCII. Binary format data is generated by the VPFGet program when extracting data from the DCW and is described in detail in the VPFGet.TXT file on disk. Binary line data files are typically given a ".MAP" filename extension. Binary line data files are smaller and plot faster than ASCII line data files.

ASCII format line data files are typically given a ".LIN" filename extension to distinguish them from binary line data files. These are simple ASCII files that contain one point per line and can be edited with your favorite text editor or other software. Each line begins with a U or P to indicate pen position (up or down) followed by, one or more spaces, the latitude (north is positive) in decimal degrees, one or more spaces, and the longitude (east is positive) in decimal degrees. There is no limit on the number of points that may be plotted.

Comments may be added to these files by placing a semi-colon in the first column. Two keywords are also recognized in ASCII line data files: color and line. Color is used to set the color of subsequent lines. Line is used to specify the line type used for subsequent lines.

The following is a few lines from the line data file provided with the program:

```
; State of California, political boundaries
Color = 2 ; Green
Line = 0  ; Solid
U 32.733167 -114.734667
D 32.744333 -114.694333
D 32.738667 -114.654333
D 32.738667 -114.591333
```

```
D 32.761667 -114.551167
D 32.830333 -114.522500
D 32.876167 -114.476667
```

See also: Color and Line type codes

Annotation_File = *filespec*

This entry specifies the file that contains text annotation to be plotted on the map. The annotation may be of any size, orientation, or color. If no annotation is to be plotted, comment out this entry be placing a semi-colon in the first column.

This is a simple ASCII file that contains one line for each string of annotation. Each line consists of the latitude (north is positive) in decimal degrees, the longitude (east is positive) in decimal degrees, the size of the text, the orientation, the color of the text, and the text itself. Each of these values must be separated by one or more spaces. Size of text is inches from baseline to top-of-character when printed on paper, and the orientation angle (degrees) is 0 for left to right increasing counter-clockwise. Comments may be added to this file by placing a semi-colon in the first column.

Below is the partial contents of the sample annotation file provided with the program:

```
;Lat      Long     Size Angle Color Text...
;-----------------------------------------------------------------------------
+37.7600 -122.3900  0.1  0.0    3   San Francisco
+36.9700 -122.0000  0.1  0.0    3   Santa Cruz
+36.6000 -121.9000  0.1  0.0    3   Monterey
+35.9500 -120.5000  0.1  0.0    3   Parkfield

+36.0000 -122.5000  0.2 -60.0   1   Pacific Ocean
+36.2500 -120.7500  0.1 -48.0   4   San Andreas Fault
```

See also: Color and Line type codes

Event_Dir = *path*

This entry is used to specify the directory that the program will poll for hypocenter solutions. Printer output (.PRT) files from Hypo71PC should be copied to this directory for plotting on the map. Note that the files should be copied, because once HypoMap reads the file, they are deleted.

Event_Mask = *wildcard*

This entry specifies the wildcard mask used while polling the event directory. If the printer output files from Hypo71 will have an extension other .PRT, use this entry to specify the new extension. The default value is *.PRT.

Origin_Latitude = *degrees* and Origin_Longitude = *degrees*

These entries are used to specify the latitude and longitude of the center of the map. Origin_Latitude specifies the latitude in decimal degrees with north being positive. Origin_Longitude specifies the longitude in decimal degrees with east being positive.

Extent = *degrees*

This entry specifies the distance in decimal degrees from the center of the map to the top or bottom edge.

Grid_Type = *type* and Grid_Ticks = *minutes*

These entries control the appearance to a grid plotted on the map.

Grid_Types are: None, Ticks, and Grid. None means no grid is plotted. Ticks plots a small cross at each intersection. Grid plots a dotted line for each line in the grid.

Grid_Ticks specifies in minutes the interval used for the grid.

Oldest_Event = *hours*

This entry specifies the oldest event that will be retained in the database after a purge. This time is compared to the most recent origin time in the database. The database may be purged automatically (see Purge_Interval below), or by pressing the p key while the map is displayed.

Purge_interval = *hours*

This entry specifies the interval at which the database will be purged automatically. If you do not want the database to purge automatically, set hours=0. The database may be purged at any time by pressing the p key. When the database is purged, all events with an origin time later than the latest event origin time minus the Oldest_Event time specified above, are removed from the database.

Title = *text*

This entry specifies the text string to be displayed as the map title. Up to 24 characters may be specified. The string is plotted above the legend on the right side of the map.

Legend_Type = *Magnitude | Depth*

This entry specifies whether this event symbol size will be proportional to event magnitude or focal depth.

Legend_Heading = *text*

This entry specifies the string used to label the legend. This string should describe the Legend_Heading unit above. For example, you might wish to label the legend with “Magnitude (ML)”, if the unit is local magnitude. Up to 24 characters may be specified.

Legend_Class = *value, size*

Up to 24 of these entries are used to specify the legend classes for the map. Each entry specifies a lower limit of magnitude or depth and a symbol size in inches. The first entry should have a lower limit of 0 and the smallest size that will be plotted. Events are classed according to the ranges specified.

The following classes are provided with the program:

```
Legend_Class =   0.0,   0.01
Legend_Class =   2.0,   0.03
Legend_Class =   2.5,   0.08
Legend_Class =   3.0,   0.12
Legend_Class =   3.5,   0.18
Legend_Class =   4.0,   0.25
Legend_Class =   4.5,   0.33
Legend_Class =   5.0,   0.5
Legend_Class =   6.0,   0.75
Legend_Class =   7.0,   0.9
```

The smallest circle to be plotted is 0.01 inches. A magnitude 3.2 event will be plotted with 0.12 inch circle on the map. Any event greater than or equal to 7.0 will be plotted at 0.9 inches.

Latest_Event_Color = *color*

This entry specifies the color used when plotting the latest event. This should be a bright color so that the latest event will be emphasized on the map. A small square of this color is displayed in the legend to inform the viewer that this is the color of the latest event. The latest event origin time, coordinates, magnitude, and depth are also displayed above the legend.

See also: Color and Line type codes

Station_Color = *color*, Station_Size = *size*

These entry are used to specify the attributes of the station symbol used on the map. Station_Color specifies the color code. Station_Size specifies the size of the symbol in inches.

See also: Color and Line type codes

Color and Line type Codes

The following color codes are used anywhere color is specified. If you are using a 256 color mode, these are only the first 16 colors.

0 = Black
1 = Blue
2 = Green
3 = Cyan
4 = Red
5 = Magenta
6 = Brown
7 = White
8 = Gray
9 = Light Blue
10 = Light Green
11 = Light Cyan
12 = Light Red
13 = Light Magenta
14 = Yellow
15 = Bright White

Line types 0 through 7 are available. 0 indicates a solid line. 1 through 6 are various dash-dot lines. 7 indicates a dotted line.

CHAPTER 4

Filter Tools

This chapter covers programs used to filter digital waveform data. The programs described here include:

- SUDSFilt – This program implements Butterworth high and low pass filters.
- DecFilt – This program is a filter-decimator. It uses 4 fixed FIR filter stages set at 80% of Nyquist of the output sampling rate. These filter-decimate stages are concatenated to achieve various integer decimation factors.

SUDSFilt

SUDSFilt is used to apply causal Butterworth digital high and low pass filters to waveforms contained in SUDS data files. The filter routines used in this implementation are based on filters written by Keith McCamy.

The program implements two independent filters, one or both may be applied to the waveforms in the input SUDS file. A second SUDS file is produced containing the filtered waveforms.

Command line syntax

SUDSFilt is controlled entirely from the DOS command line. The program returns an exit code of 0 to indicate successful execution and 1 to indicate that an error occurred during processing. If an error does occur, a verbose message will be written to stdout (the screen) describing the error.

Help may be obtained by typing SUDSFilt ? and pressing return at the command line. The following screen will be displayed:

```
SUDSFILT - Version 2.40

Usage: SUDSFILT [switches] inputfile outputfile [switches]

Switches:
   /A    = Append data to output file, create if needed. (OVERWRITE)
   /F    = Promote data type to floats on output. (OFF)
   /Bn   = Baseline, subtract mean of first n samples from trace. (200)
   /Tn   = Taper, length of cosine taper applied to trace in samples. (200)
   /Sn   = Scaler, all samples will be multiplied by n. (1.0)
   /Lp,f = Low pass filter, p = poles, f = corner freq. (NO FILTER)
   /Hp,f = High pass filter, p = poles, f = corner freq. (NO FILTER)

Arguments:
   input and output files are PC-SUDS data files.

[] = optional, () = default.

The filters are implemented as Recursive (IIR) Butterworth digital filters.
Poles value will be rounded up to the nearest factor of 2, 1 pole = 6 dB per
octave slope.  Corner freq. value is specified in hertz.

Arguments may be placed in any order and are not case sensitive.
```

Arguments

SUDSFilt expects two SUDS file specifications as arguments. The first argument specifies the input data file. The input file is accessed in read-only mode and the contents of this file will not be modified under any circumstances. The second argument specifies the output data file. This file will be created if it does not exist. Filtered data may be appended to this file if it exists by using the /A (append mode) switch, otherwise the file will be overwritten.

Switches

The following switches may appear any position within the command line. If a switch appears more than once on the command line, the last occurrence will be used. None of these switches are case sensitive.

/A - Append data to output file

By default the program will overwrite any existing output file. By specifying the /A switch you will cause the program to open any existing data file in append mode and add the filtered data to the end of that file.

/F - Promote output data to floats

This switch specifies that the sample data should be written to the output data file in floating point format. By default, if the input data is 16 bit integers, it will be converted to floats, filtered, scaled (see /s below) and then rounded back into 16 bit integers. This switch simply preserves the data as floats.

/B*n* - Baseline (DC offset) removal

This switch is used to specify the length (*n* samples) starting from the first sample of each waveform that are to be averaged to establish the baseline or zero level. This value is then subtracted from each sample in the waveform before any filtering is performed. By default, a 200 sample mean is computed and subtracted. If you wish for no baseline removal to be performed, specify /B0 on the command line.

/T*n* - Apply taper before filtering

This switch is used to specify the length (*n* samples) starting from the first sample of waveform that a simple cosine taper will be applied to before filtering is performed. By default, the first 200 samples of the waveform will be tapered.

/S*n* - Scaler

This switch is used to provide a real scaler to be applied to each sample after filtering. Each filtered sample is simply multiplied by *n* before being written to the output file. By default, *n* = 1.0.

/L*p, f* - Low-pass filter

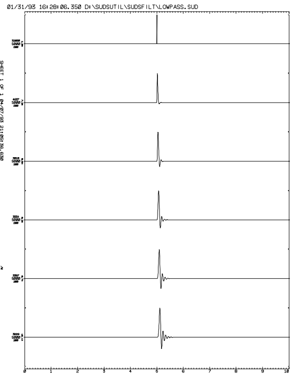
This switch is used to low-pass filter the input waveform(s). *p* = the number of poles for the filter and *f* is the corner frequency expressed in Hz. The value of *p* must be a factor of 2 between 2 and 10 inclusive. 1 pole causes a roll-off of 6 dB per octave above the corner frequency.

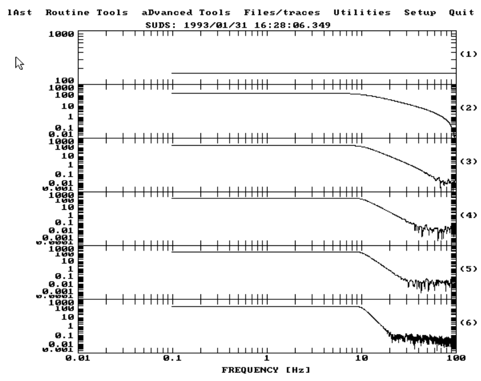
/H*p, f* - High-pass filter

This switch is used to high-pass filter the input waveform(s). *p* = the number of poles for the filter and *f* is the corner frequency expressed in Hz. The value of *p* must be a factor of 2 between 2 and 10 inclusive. 1 pole causes a roll-off of 6 dB per octave below the corner frequency.

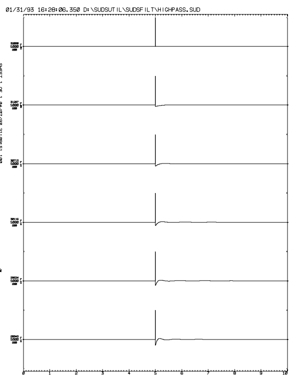
Examples

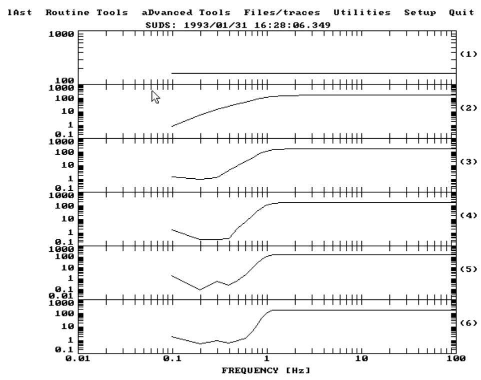
The following plots show the impulse response of various filter settings in the time and frequency domains.



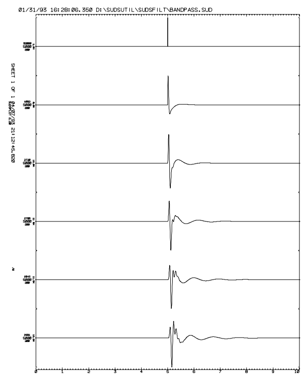


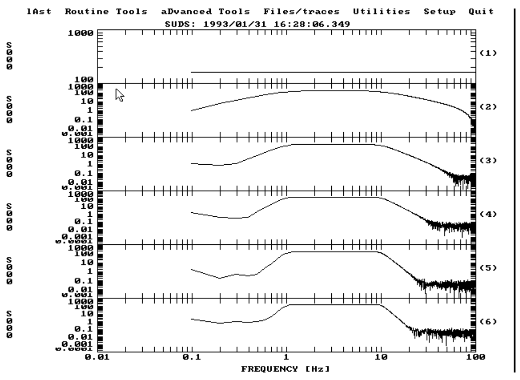
The plot above left is a single full scale positive spike. The uppermost waveform is unfiltered. The second through the bottom waveforms are low-pass (high-cut) filtered at 10 Hz with 2,4,6,8 and 10 poles respectively. The plot above right is the amplitude spectra of the corresponding waveforms on the left plotted to log-log scale.





Again the plot above left is a single full scale positive spike and the uppermost waveform is unfiltered. The second through the bottom waveforms are high-pass (low-cut) filtered at 1 Hz with 2 through 10 poles respectively. The plot above right is the amplitude spectra of the corresponding waveforms on the left plotted to log-log scale.




lAst Routine Tools aDvanced Tools Files/traces Utilities Setup Quit
SUDS: 1993/01/31 16:28:06.349
FREQUENCY [Hz]
(1)
(2)
(3)
(4)
(5)
(6)

Again the plot above left is a single full scale positive spike and the uppermost waveform is unfiltered. The second through the bottom waveforms are band-pass filtered with the lower corner at 1 Hz, the upper corner at 10 Hz with 2 through 10 poles respectively. The plot above right is the amplitude spectra of the corresponding waveforms on the left plotted to log-log scale.

DecFilt

DecFilt is a general purpose filter/decimation program. The program uses 4 FIR low pass filter/decimate stages with factors of 2:1, 3:1, 4:1 and 5:1. These stages are used in sequence to achieve the desired output decimation factor.

Command line syntax

DecFilt is controlled entirely from the command line. The program returns an exit code of 0 to indicate successful execution and 1 to indicate that an error occurred during processing. If an error does occur, a verbose message will be written to stderr (the screen) describing the error.

Help may be obtained by typing DecFilt ? and pressing return at the command line. The following screen will be displayed:

```
DECFILT - Version 2.40

Usage: DECFILT [switches] InFilespec OutFilespec Factor [switches]

Switches:
   /A  = Append data to output file if it exists.
   /Bn = Baseline and taper length in samples. (200)

Type: DECFILT ? for a list of available decimation factors.

[] = optional, () = default.  Arguments are not case sensitive.
```

Arguments

The program expects three arguments: the input filespec, the output filespec and the desired decimation factor. The output file will contain all waveforms contained in the input file decimated by the specified factor.

If the specified factor is greater than 5, the program will determine the best combination of filter/decimate stages to apply to the data. Below is a list of achievable factors up to 50:1 and the stages that will be applied:

```
  2:1 = 2:1
  3:1 = 3:1
  4:1 = 4:1
  5:1 = 5:1
  6:1 = 3:1 + 2:1
  8:1 = 4:1 + 2:1
  9:1 = 3:1 + 3:1
 10:1 = 5:1 + 2:1
 12:1 = 4:1 + 3:1
 15:1 = 5:1 + 3:1
 16:1 = 4:1 + 4:1
 18:1 = 3:1 + 3:1 + 2:1
 20:1 = 5:1 + 4:1
 24:1 = 4:1 + 3:1 + 2:1
 25:1 = 5:1 + 5:1
```

```
27:1 = 3:1 + 3:1 + 3:1
30:1 = 5:1 + 3:1 + 2:1
32:1 = 4:1 + 4:1 + 2:1
36:1 = 4:1 + 3:1 + 3:1
40:1 = 5:1 + 4:1 + 2:1
45:1 = 5:1 + 3:1 + 3:1
48:1 = 4:1 + 4:1 + 3:1
50:1 = 5:1 + 5:1 + 2:1
```

The following table shows the design parameters of the low pass FIR filter used in each of the four stages.

These filters where designed using the McClellan-Parks optimal equirriple FIR filter design method. The coefficients where computed with MATLAB-386 using the B = REMEZ(N, F, M) function in the Signal Processing Toolbox. This function implements the Remez Exchange Algorithm. The values of N were arrived at empirically using the IMPULSE program to view the response spectra.

These filters have been designed to have a passband at 80% of the Nyquist frequency and a stopband at the Nyquist frequency after decimation. Stopband attenuation is greater than -100dB.

Factor	Passband	Stopband	N
2:1	0.4	0.5	95
3:1	0.2666	0.3333	151
4:1	0.2	0.25	189
5:1	0.16	0.2	235

The passband and stopband values are normalized frequency units where 0 is DC and 1 is equal to the Nyquist frequency.

Switches

Switches may appear at any point on the command line. These switches are not case sensitive.

/A - Append data to output file

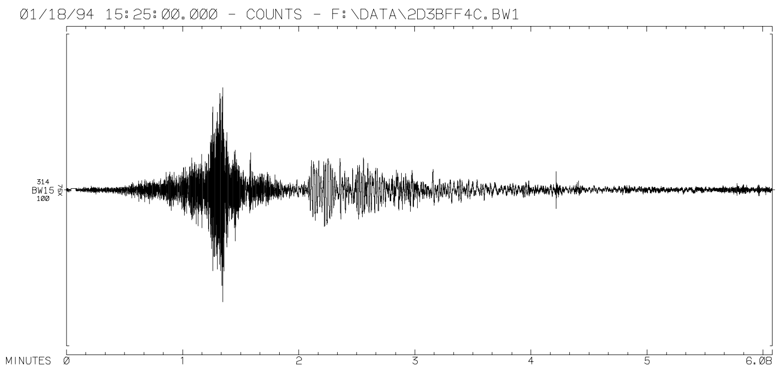
By default the program will overwrite any existing output file. By specifying the /A switch you will cause the program to open any existing data file in append mode and add the filtered data to the end of that file.

/B*n* - Baseline (DC offset) removal

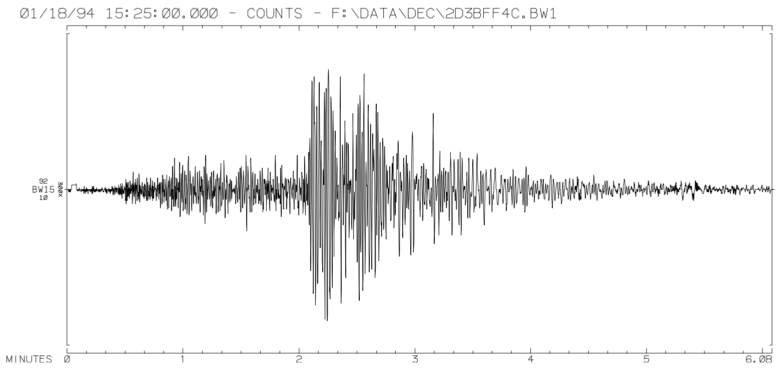
This switch is used to specify the length (*n* samples) starting from the first sample of each waveform that are to be averaged to establish the baseline or zero level. This value is then subtracted from each sample in the waveform before any filter/decimation is performed. By default, a 200 sample mean is computed and subtracted. If you wish for no baseline removal to be performed, specify /B0 on the command line.

Examples

The following plots show a very long event recorded at a distance of about 800 km with a lot of local high frequency noise obscuring the longer period event.



This is a six minute record of a 5.2 aftershock of the Northridge, California earthquake recorded in Big Water, UT at 100 sps. Note the high frequency energy peaked at about 1:20.



This plot shows the same record after filter/decimation to 10 sps (5:1+2:1). The effective corner frequency of the low pass filter is 4 Hz (80% of Nyquist). Note that the high frequency energy at 1:20 is almost completely gone and the record size has been reduced from 36000 samples to 3600 samples.

CHAPTER 5

Waveform Plotting Tools

This chapter covers programs that plot digital timeseries' data to the display and paper. The programs described here include:

- SUDSPlot – A general purpose waveform plotting program.
- SUDSDrum – A program that generates "pseudo smoked drum" plots from continuous waveform data.

SUDSPlot

SUDSPlot is a general purpose timeseries' plotting program. The purpose of this program is to produce high quality plots in a timely manner. The program offers many options while plotting, these include:

- True record-sections with phases and coda duration marked and hypocenter solution data annotated.
- Selective display of waveforms based on pick weight.
- Complete control of amplitude scaling.
- Baseline (DC offset) removal.
- Control of the starting point and length of the plotting window.
- Display the plot on screen, printed hardcopy or both.
- A "windowing" decimation algorithm (minimum and maximum preserved) is used to speed plotting without distorting the visual character of the waveform.
- Extensive control of plot annotation.
- Various amplitude scaling and units.

The program reads the "printer output" file from Hypo71PC to obtain station and hypocenter solution data. If this file is available, the program plots the waveforms ordered by epicentral distance and optionally plots the waveforms against range on the *y* axis to create a record-section. If no station or hypocenter data is available, the program simply plots the data found in the SUDS file.

Command line syntax

Like most of the SUDS utilities, SUDSPlot is controlled from the DOS command line and/or by setting entries in the .INI file. The program returns an exit code of 0 to indicate successful execution and 1 to indicate that an error occurred during processing. If an error does occur, a verbose message will be written to the screen and the log file describing the error. Help may be obtained by typing SUDSPlot ? and pressing return at the command line. The following screen will be displayed:

```
SUDSPLOT - Version 2.40

Usage: SUDSPLOT [switches] SUDSfile [hypofile] [switches]
Switches:
   /Pn - Plot mode; 1=screen only, (2)=screen & hardcopy, 3=hardcopy only.
   /R  - Plot traces against range axis, record section. (OFF, reqs. .PRT file)
   /An - Plot phase arrivals with weight <= n. (5, requires .PRT file)
   /Bn - Baseline, subtract n sample average from trace. (OFF)
   /Jn - Jump, start plotting n seconds into trace. (0)
   /Ln - Length, plot n seconds of trace. (ALL)
   /Mn - Magnify amplitude up to n times where peak = full scale. (1X)
   /MFn- Fixed, magnify amplitude n times. (1X)
   /ABSstn - Plot stn with ABS amplitude, up to 12 may be specified
   /Tn - Plot n traces per page. (AUTO)
   /Wn - Window about first IST to quit plotting based on time axis. (120.0)
   /X1 - No decimation. (Window decimation, min & max preserved)
   /Xn - Simple decimation, plot every nth sample. (OFF)

   SUDSfile - SUDS 1.x data file.
   hypofile - HYPO71x output file. (same name as SUDSfile w/.PRT ext.)

()=default, []=optional, arguments are not case sensitive.
```

Arguments

SUDSPlot expects a single SUDS input data file specification as the first argument. The second argument is the file specification of a Hypo71 “printer output” file, also called a PRT file because it is conventional to name this file with a .PRT extension. This argument is optional. If a PRT file is not specified, SUDSPlot will look in the same directory as the input file for a PRT file with the same name as the input file with a .PRT extension.

If no PRT file is found, certain program options will not be available.

Switches

The following switches may appear at any position within the command line. If a switch appears more than once in the command line, the last occurrence will be used. None of these switches are case sensitive. Many of the options specified on the command line will override settings in the .INI file (see .INI file entries below).

/P*n* - Plotting mode

This switch is used to specify whether the plot will be displayed on the screen and/or printed. The following modes are available:

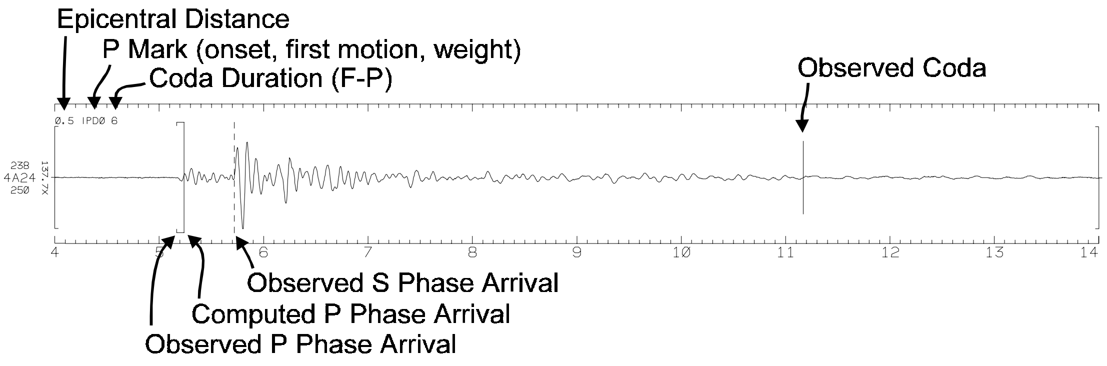
n = 1	Plots are displayed on the screen only. The program will pause when the plot is complete (see the Pause entry in the .INI file).
n = 2	This is the default. Plots are displayed on the screen and then printed. The program does not pause when the plot is completed on screen.
n = 3	Plots are generated without screen display. Status information is displayed on screen while processing. This mode is useful in batch programs.

/R - Plot waveforms against range axis (record-section)

This switch requires a PRT file. If no PRT file was found, this switch is ignored. If a PRT file was found, the waveforms are arranged along a range axis on the page. The range axis starts with the nearest station to the epicenter and ends with the furthest. The units along this axis are kilometers. All other switches will effect the plot as usual while plotting against the range axis.

/A*n* - Plot phase arrivals

This switch requires a PRT file. If no PRT file was found, this switch is ignored. If a PRT file was found, only waveforms with pick weight less than or equal to *n* are plotted in the order that they appear in the PRT file and are marked as in the figure below, all others are ignored. If no *n* is specified, all waveforms are plotted in the order that they appear in the PRT file and marked as in the figure below.



/B*n* - Baseline (DC offset) removal

This switch is used to specify the length (*n* samples) starting from the first sample that are to be averaged to establish the baseline or zero level. This value is then subtracted from each sample in the waveform. By default, no baseline is computed or removed.

/J*n* - Jump
/L*n* - Length

These switches are used to control the plotting window. /J*n* is used to specify how many seconds into the waveform plotting should begin and /L*n* is used to specify how many seconds from that point plotting should end. By default, plotting starts with the earliest initial sample time (IST) of all waveforms and ends with the latest last sample time (LST) of all waveforms.

/M*n* - Amplitude magnification
/Mf*n* - Fixed amplitude magnification

By default, the program will plot waveforms with absolute amplitude, i.e., the maximum possible sample value according to the data type (e.g., 32767 for 16 bit data) is equal to full scale. The /M switch (with no value for *n*) causes the program to magnify the waveform to where the peak sample in the displayed portion of the waveform is equal to full scale. If a value for *n* is specified, the waveforms are magnified up to but not more than *n* times. This prevents "dead" or very low amplitude waveforms from being magnified to full scale.

If the /MF*n* switch is specified, the program will simply magnify amplitude *n* times. No clipping is performed so you should exercise care when using this option.

/ABS*stn* - Plot *stn* at absolute amplitude

This option allows you to specify up to 12 station/component identifiers to be plotted at absolute amplitude regardless of the amplitude settings specified with the /M option. This is useful when Irig time code is recorded on a channel and you are using the /MF switch to specify a fixed amplitude magnification. If there are several stations that should not be magnified, specify /ABS*stn* for each one individually.

/T*n* - Number of waveforms (traces) per sheet

By default the program will determine the optimum number of waveforms to place on one sheet. This number specified with *n* must range from 1 to 32 inclusive. As many sheets as needed will be plotted.

/W*n* - Alignment window

If each waveform in the input file does not share the same IST and LST (this is common with data recorded on portable recorders), the program aligns each waveform in time before plotting. By default the program will plot starting with the earliest IST and ending with the latest LST. The /W*n* switch is used to specify a window about the earliest IST that the IST of each waveform must be within in order to be aligned. If a the IST for a waveform is outside this window, it is aligned with the earliest IST and annotated with its IST to denoted that it is not aligned with the other waveforms. The program uses a 120 (±60) second exclude window by default.

/X*n* - Decimation

By default, the program uses a "windowing" decimation algorithm to enhance plotting performance. The program determines the optimum decimation factor (the number of samples to skip or "window") based on the total number of samples, the sampling rate and the resolution of the output device and then plots only the minimum and maximum value within this window. This preserves the visual character of the waveform while offering better performance.

If you specify the /X*n* switch the program will perform simple decimation with a factor of *n* (i.e., it will plot every n^{th} sample). If you don't want the program to perform any decimation, simply specify /X1.

/D*f* - Date/time format

This switch is used to specify the format used to display date/time values. If *f* = "M" (the default), all date/time values are displayed in month and day format (e.g., 9/28/92 14:06:24.713). If *f* = "J", all date/time values will be display in "Julian day" or day-of-year format (e.g., 92*272+14:06:24.713).

INI file entries

SUDSPlot searches the INI file for the [SUDSPlot] section at startup. The entries should be thought of as defining the default behavior of the program. Many of the entries in this section are equivalent to the command line switches above. Here we will only cover those entries that are not. Command line switches that correspond to these entries will override the settings in the .INI file.

The following is an example [SUDSPlot] section.

```
[SUDSPLOT]
; Entries for SUDSPLOT 2.30 or later.
; Most of these may be overridden on the command line.

; Maximum and fixed amplitude magnification, /Mn, /MFn (0<=n)
; The following two entries are mutually exclusive, only one of these will take
; affect, if both are specified, only the later occurring one takes affect.
Max_Magnification = 32
;Fixed_Magnification = 8

; Station/components to plot with absolute amplitude regardless of
; the previous two entries or their corresponding switches.
; A total of 12 may be specified, see also the /ABS command line switch.
ABS_Amplitude = IRIG
ABS_Amplitude = IRG1
ABS_Amplitude = IRG2

; Plotting mode, /Pn (n=1=screen only, 2=screen & hardcopy, 3=hardcopy only)
Plot_Mode = 1

; Baseline length in samples, /Bn (0<=n)
Baseline = 200

; Traces per page, /Tn (1<=n<=32), automatically computed by default
;Traces_per_Page = 16  ; Force 16 traces per sheet

; Time alignment window, /Wn (0<n), +-60.0 by default.
Align_Window = 90.0

; Simple decimation, /Xn (0<=n), 1=no decimation, If this option is not
; specified, the program uses "windowing" decimation (min & max preserved)
Decimate = 0   ; "Windowing" decimation
;Decimate = 1   ; No decimation
;Decimate = 10  ; Simple decimation with a factor of 10 (plot every 10th sample)

; Jump and length for plot window in seconds, /Jn, /Ln (0<=n)
;Jump   = 10.0  ; Plot starts at earliest IST by default
;Length = 20.0  ; Plot ends at latest LST by default

; Exclude traces by weight criteria (if .PRT file available), /An (0<=n<=5)
Weight_Criteria = 5   ; Plot all arrivals in PRT file
;Weight_Criteria = 2   ; Plot arrivals with <=2 weight

; Plot against range axis, record-section (if .PRT file available)
;Range_Axis = Yes
```

```
; --- Amplitude unit entries ---
; If the following two entries are specified, they override the gain and
; digitizing constant stored in the input SUDS data file. Generally, you should
; not specify these unless you are sure that the data in the file is incorrect.
; RTP and Ref2SUDS generate correct data files.
;Amplifier_Gain  = 1            ; Gain as magnification
; Digitizing Constant...
;Counts_per_Volt = 409.6        ; 12 bit A/D, +-5V (RDAS)
;Counts_per_Volt = 6553.6       ; 16 bit A/D, +-5V (RDAS)
;Counts_per_Volt = 8737.866667  ; 16 bit A/D, +-3.75V (RefTek)
;Counts_per_Volt = 524288.0     ; 24 bit A/D, +-10V (RefTek)

; Annotate peak amplitude in ground motion units (GMU's), Volts or Counts.
; The figure is the abs peak amplitude of the displayed portion only.
;Amplitude_Unit  = GMU          ; Compute and annotate in ground motion units*
;Amplitude_Unit  = Volts        ; Compute and annotate in volts^
Amplitude_Unit  = Counts       ; Annotate in digital counts (default)
; * requires valid counts_per_volt, amplifier gain and volts_per_gmu values.
; ^ requires valid counts_per_volt and amplifier gain values.

; If "Amplitude_Unit = GMU" the following entry must be provided.
Volts_per_GMU   = 0.5          ; USGS L22, 0.5 volts per CM per second
;Volts_per_GMU   = 0.88         ; LDGO L22, 0.88 volts per CM per second
;Volts_per_GMU   = 0.005102     ; Kinemetrics FBA, 10V / 2G, 1G ~= 980 CM/SEC/SEC

; Text describing amplitude unit
;Annot_Amp_Text  = CM/SEC/SEC   ; Acceleration
;Annot_Amp_Text  = CM/SEC       ; Velocity
;Annot_Amp_Text  = CM           ; Displacement
;Annot_Amp_Text  = VOLTS        ; Voltage
Annot_Amp_Text  = COUNTS       ; Digital counts (default)

; ------------------------------

; Annotation control (Yes|No), defaults to Yes.
;Annot_Eventline    = No   ; Event line at top of sheet
;Annot_P_Remark     = No   ; Phase data in upper left of trace area
;Annot_SampleRate   = No   ; Sample rate below station name
;Annot_Amp_Mag      = No   ; Amplitude magnification
;Annot_Amp_Brackets = No   ; Amplitude axis brackets on ends of traces
;Annot_Sideline     = No   ; Sheet # and session time in upper left margin
;Annot_CL_args      = No   ; Command line args in lower left margin
```

The Amplitude_Unit and its associated entries control the annotation of the peak values. You should read the notes in the .INI file before using the settings.

The annotation control settings turn on or off various plot annotation.

Plot annotation

Across the top (looking at the page in portrait orientation) is a line of text that contains the earliest IST on the plot, the amplitude unit and the fully qualified file specification of the input file.

If a PRT file was found, a second line of text is printed. This line contains the event origin time, location and other hypocenter solution information.

Printed vertically down the left margin is the sheet number, the time that the sheet was printed and the version number of SUDSPlot used to produce the plot.

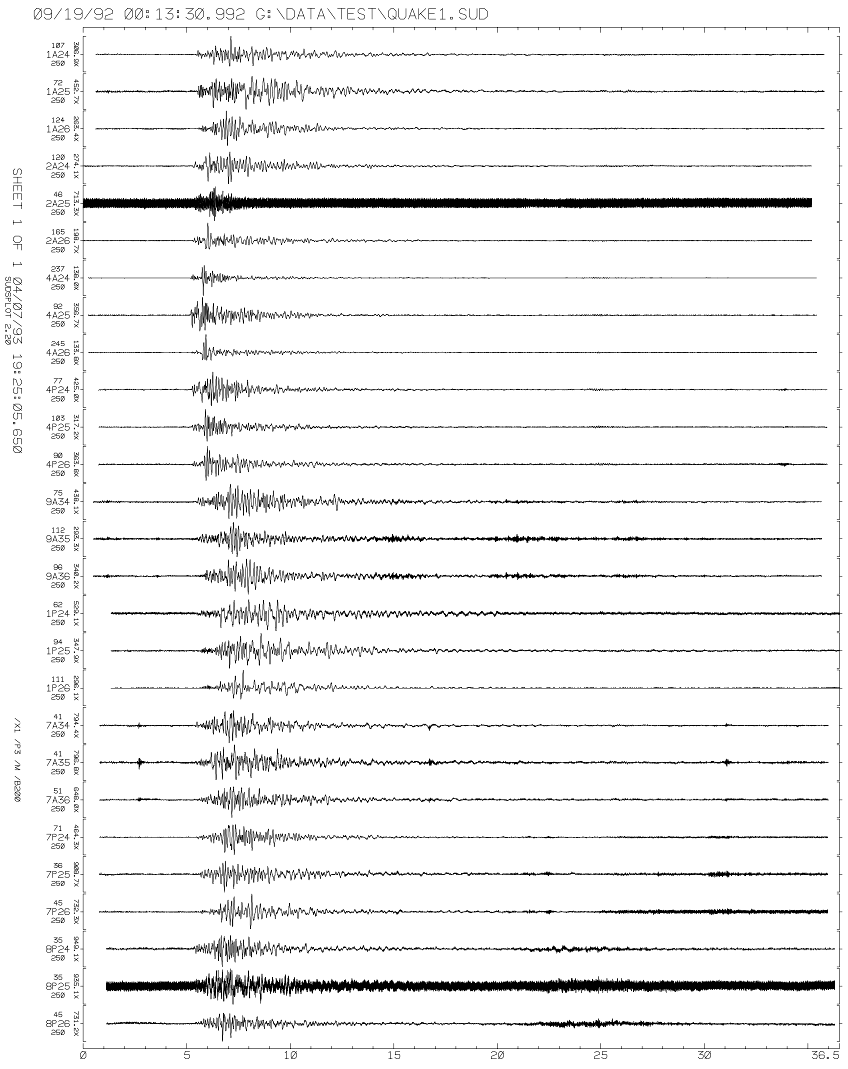
Each waveform is annotated with the station/component identifier to the left of the left amplitude axis. Above the station/component identifier is the peak absolute sample value in the waveform and below is the sampling rate in Hz. Rotated 90° and centered on the amplitude axis is the amplitude magnification factor applied to the waveform.

The values along the time axis are seconds relative to the IST annotated at the top of the plot. This is true even when the /J and /L switches are being used.

Other annotation is added to the plot as various circumstances require (see switches above).

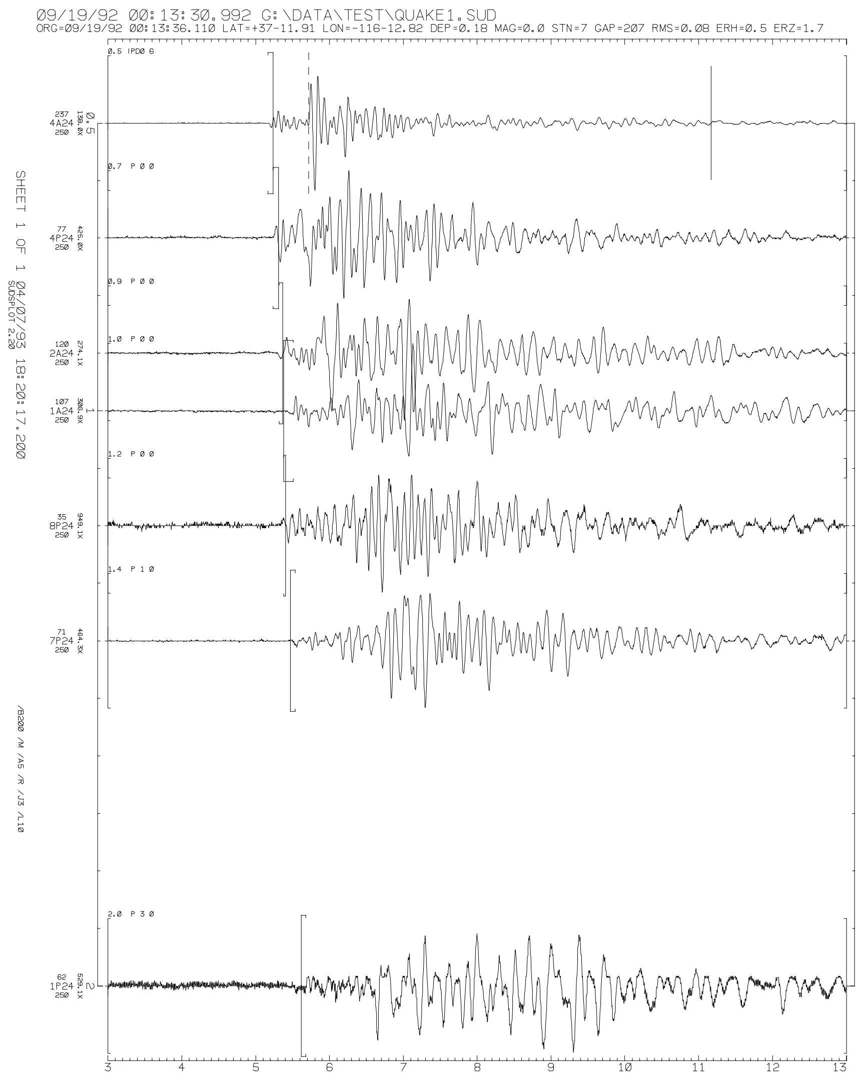
Examples

The following plots demonstrate many of the options available when using SUDSPlot. Below each plot is a brief description of the displayed data and the command line used to invoke SUDSPlot.



The plot above shows the raw data from seven portable instruments. The only options used to create this plot was baseline removal and amplitude magnification. The command line used to invoke SUDSPlot was as follows:

```
SUDSPLOT /B200 /M QUAKE1.SUD
```



The plot above demonstrates the range plotting and phase marking features of SUDSPlot. The same data file from the previous plot is used after phase picking with SUDSPick and location with Hypo71PC. The following command line was used to invoke SUDSPlot:

```
SUDSPLOT QUAKE1.SUD QUAKE1.PRT /B200 /M /A5 /R /J3 /L10
```

SUDSDrum

SUDSDrum produces a "pseudo smoked drum" plot from continuous waveform data. The length of each line, the number of lines and the starting time can be controlled. A windowing decimation algorithm is used increase plotting performance without distorting the visual character of the waveform.

The input data is processed in "chunks" so that there is no memory limitation imposed on the length of the waveform data. The length is only limited by available disk space.

An additional utility program (DECIMATE) is provided to perform windowing decimation and concatenation of the data files used with SUDSDrum.

Command line syntax

Like most of the SUDS utilities, SUDSDrum is controlled entirely from the DOS command line. The program returns an exit code of 0 to indicate successful execution and 1 to indicate that an error occurred during processing. If an error does occur, a verbose message will be written to the screen describing the error.

Help may be obtained by typing SUDSDrum ? and pressing return at the command line. The following screen will be displayed:

```
SUDSDRUM - Version 2.40

Usage: SUDSDRUM [switches] InputFilespec StationID [switches]

Switches:
   /Sdatetime = Start time, "MM,DD,YY,HH,MM,SS.SSS" or "YY,JJJ,HH,MM,SS.SSS"
       depending on /D switch, commas may be replaced with any non-digit.
   /Ln = Length of single trace in seconds. (900, 15 min)
   /Tn = Number of traces on each sheet. (96)
   /Mn = Amplitude magnification multiplier. (1)
   /C  = Force amplitude unit to counts (volts if data is avail.)
   /X  = Perform no decimation while plotting. (DECIMATE)
   /Pn = Plot mode: 1=screen only, (2)=screen & hardcopy, 3=hardcopy only.
   /Df = Date format: f=('J')=Julian day or day-of-year, f='M'=month/day.

[] = optional, () = default.
Arguments are not case sensitive.
```

Arguments

SUDSDrum expects two arguments on the command line. First, the input SUDS data file specification and second, the station/component identifier for the waveform you wish to process. Only waveforms from one station/component in the input file can be processed at each invocation of the program. To process several station/components from a single file you must invoke the program for each one.

The records that make up the continuous waveform will generally be contained in several SUDS data files. For example, when recording continuous data with a RefTek 72 series instrument, the data is broken into segments of a specified length. When this raw data is processed by QuickLook, each of these segments will be output in an individual SUDS file. These SUDS files should be combined into a single SUDS file before processing with SUDSDrum.

As a specific example, lets say that we have a 72A-02 recording a single channel (we'll say channel 1) continuously at 20 samples per second on stream 2 with a record length of 21600 seconds (6 hours). The station identifier will be "BW". The 4 SUDS files created by QuickLook for each 24 hour period will have a hex time (trigger time) filename and an extension of BW2 (station BW, stream 2). These files could be combined using one of the following commands:

```
COPY /B *.BW2 DAYnnn.SUD
COPY /B file1.BW2+file2.BW2+file3.BW2+file4.BW2 DAYnnn.SUD
```

The /B (binary file) switch is very important, without it the combined file (DAY*nnn*.SUD) will likely be unreadable. If you see an error message saying "input file is out of sync" or "bad machine type", you probably forgot the /B switch. For additional information about the COPY command please refer to the DOS help system of your DOS user reference.

This combined SUDS data file would contain several 6 hour segments from stream 2 at station BW. SUDSDrum will index and analyze each segment found for a given stream and station in the SUDS file. These segments may overlap in time or have holes between them. If they overlap, the waveforms will be superimposed on the plot and if there is no data for some time period, no trace will be plotted for that time period.

Switches

/P*n* - Plotting mode

This switch is used to specify whether the plot will be displayed on the screen and/or printed. The following modes are available:

n = 1	Plots are displayed on the screen only. The program will pause when the plot is complete.
n = 2	This is the default. Plots are displayed on the screen and then printed. The program does not pause when the plot is completed on screen.

n = 3	Plots are generated without screen display. Status information is displayed on screen while processing. This mode is useful in batch programs.

/S*date/time* - Start time

This switch allows you to specify the start time of the plot. The *date/time* value may be specified using two different formats: month/day or day-of-year. By default day-of-year notation is used.

Note that if you wish to specify the start time in the month/day format, you must specify the /Dm switch on the command line *before* the /S switch. If the /Dm switch follows the /S switch, only the date/time values annotated on the plot will be affected.

/D*f* - Date/time format

This switch is used to specify the format used for all date/time input and output. *f* = "M" = MM/DD/YY,HH:MM:SS.SSS and *f* = "J" = YY*DDD+HH:MM:SS.SSS, where:

YY	=	Year - Century (e.g., 93, not 1993).
DDD	=	Day-of-year or "Julian day" (1-366).
MM	=	Month (1-12).
DD	=	Day of month (1-31).
HH	=	Hour of day (0-23).
MM	=	Minute of hour (0-59).
SS.SSS	=	Second of minute (0-59.999).

By default the program uses the day-of-year (J, "Julian day") format. When specifying a date/time value the individual numbers may be separated by any non-digit. Below are several example command lines using the different date/time formats:

```
SUDSDRUM FILE.SUD STN /I93,37,12:00:1.234
SUDSDRUM FILE.SUD STN /DM /I2,8,93,12:00:1.234
```

/L*n* - Trace length

This switch is used to specify the length of each trace on the plot in seconds. This parameter is analogous to the rotation speed on a smoked drum recorder. The default line length is 900 seconds (15 minutes).

/T*n* - Number of traces

This switch is used to specify the number of traces that will appear on the plot. This parameter is analogous to the translation rate on a smoked drum recorder. The default number of traces is 96.

/M*n* - Amplitude magnification

By default, each trace is plotted so that full scale is equal to the space available on the plot for the trace (1× magnification). If a magnification factor is specified with this switch, the amplitude of all waveforms will be magnified *n* times. This parameter is analogous to the gain setting on a smoked drum recorder.

Note that no clipping is performed as the waveforms are plotted, much like a smoked drum recorder.

/C - Force amplitude unit to digital counts

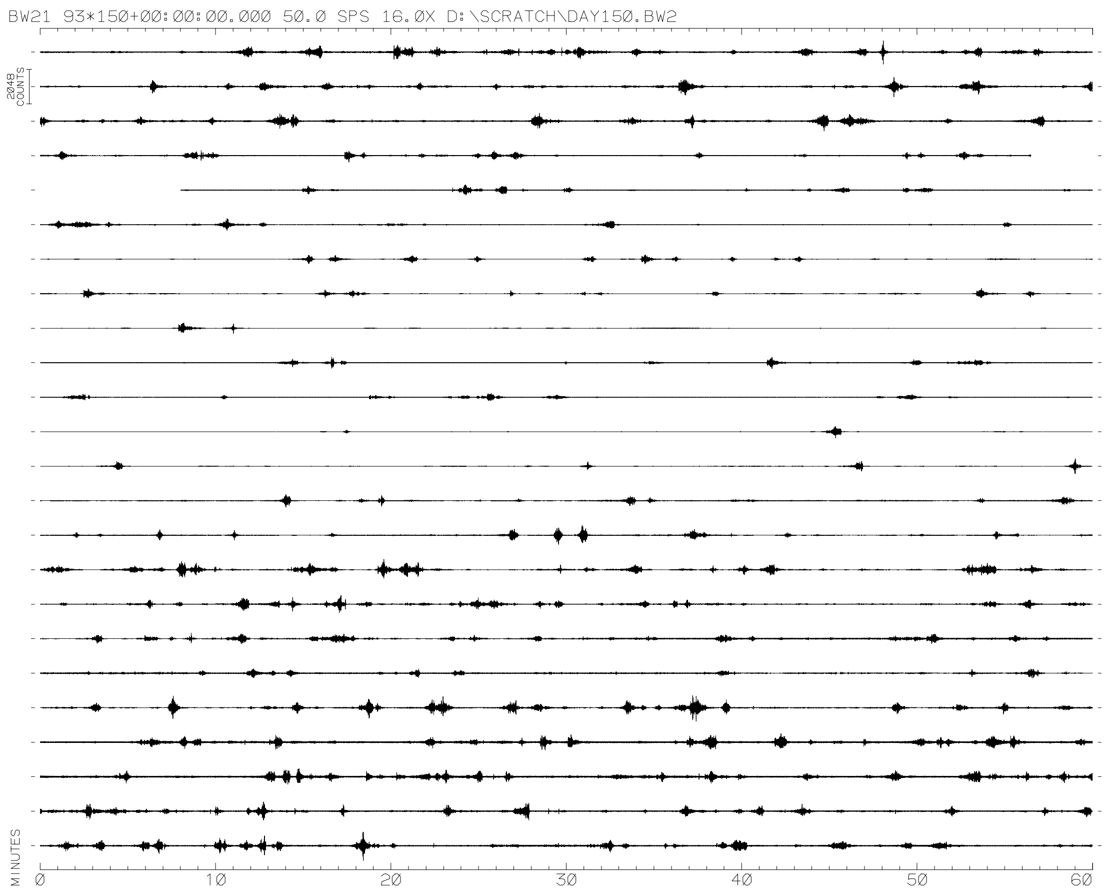
If the input file contains SUDS_INSTRUMENT structures with valid gain and digitizing constant values, this information is used to annotate the amplitude scale in volts. By using this switch you will force the amplitude scale to be annotated in digital counts even if this information is available.

Examples

The plot below is twenty-four hours of single channel continuous data recorded at 50 samples per second. Twenty-seven SUDS data files containing 3600 seconds (1 hour) of data were combined into one file named DAY150.BW2 (see the discussion about combining SUDS files in Arguments above). The /S switch was used to specify the start time at 93*150+00:00:00 UTC, the /M switch was used to specify magnification of 16 and /C forced the amplitude scale (upper left corner) to be in digital counts. Each line is one hour long and there are twenty-four lines, these are the default settings.

The annotation across the top of the plot contains the station/component identifier, plot start time, sampling rate, amplitude magnification and the input file specification. The time unit used on the time axis is annotated in the lower left corner.

Note that data acquisition was off for twelve minutes between 3:56 and 4:08.



The command line used to produce the plot above was:

```
SUDSDRUM DAY150.BW2 BW21 /S93,150,0,0,0 /T12 /L3600 /M16 /C
```

CHAPTER 6

Spectral Analysis Tools

This chapter covers programs that perform spectral operations on digital timeseries data. The programs described here include:

- SUDSSpec – A flexible general purpose FFT program.
- ReSpec – A BAP-compatible response spectra program.

SUDSSpec

SUDSSpec is used to compute and display amplitude or power spectra of SUDS data. The program uses an external memory Fast Fourier Transform (FFT) to accommodate very large waveforms. FFT's of 2^5 to 2^{31} points can be computed given enough disk space. Eight different data windowing functions are provided and waveforms can be cut or padded as required.

The plots generated by this program display the windowed timeseries and the spectrum plotted log or linear on both the *x* and *y* axis.

Command line syntax

SUDSSpec is controlled entirely from the DOS command line. The program returns an exit code of 0 to indicate successful execution and 1 to indicate that an error occurred during processing. If an error does occur, a verbose message will be written to stderr (the screen) describing the error.

Help may be obtained by typing SUDSSpec ? and pressing return at the command line. The following screen will be displayed:

```
SUDSSPEC - Version 2.40

Usage: SUDSSPEC [switches] InputFilespec StationID [switches]

Switches:
   /Pn = Plot mode: (1)=screen only, 2=screen & hardcopy, 3=hardcopy only.
   /Wwin = win is type of window to apply before FFT, 0-7. (1,HANNING)
           Type SUDSSPEC /w? for a list of available windows.
   /Jn = n is number of seconds to jump into waveform. (0.0)
   /Ln = n is the log2 length of the FFT, will be zero padded if needed (10).
   /Sn = n is the scaler applied to each raw sample before FFT. (1.0)
   /Xs,u,min,max = X axis control, s=LOG or (LIN), u=(HERTZ), BINS or NORM.
   /Ys,u,min,max = Y axis control, s=(LOG) or LIN, u=AMPLITUDE or (POWER).
   /G[+|-] = Display grid at major ticks on X and Y axis. (NO)
   /N[+|-] = Use negative frequencies. (YES)
   /A[+|-] = Display all of timeseries or just window. (WINDOW)
   /B[+|-] = Baseline removal. (NO)
   /Ofilespec = Output FFT results to this ASCII file. (NONE)
   /Ifilespec = Alternate .INI filespec.

Strings need only be long enough to be unique e.g., LI instead of LINEAR.
Switches on the command line override settings in the .INI file.
[] = optional, () = default.  Arguments are not case sensitive.
```

Arguments

SUDSSpec expects two arguments on the command line. First, the input SUDS data file specification and second, the station/component identifier for the waveform you wish to process. Only waveforms from one station/component in the input file can be processed with each invocation of the program. To process several station/components from a single file you must invoke the program for each one.

Switches

/P*n* - Plotting mode

Use this switch to specify whether the plot will be displayed on the screen and/or printed. The following modes are available:

n = 1 This is the default. Plots are displayed on the screen only. The program will pause when the plot is complete (see the Pause entry in INI file entries below).

n = 2 Plots are displayed on the screen and then printed. The program will pause when the plot is complete (see the Pause entry in INI file entries below).

n = 3 Plots are generated without screen display. Status information is displayed on screen while processing. This mode is useful in batch programs.

/W*n* - Data window

One of eight different data windowing functions may be selected with this switch. To see a list of available data windowing functions, type SUDSSpec /W? on the command line. Window type may be specified by number or unique string and the default type can be set in the .INI file (see below).

Window Types:

0) Rectangle, also called a “square” window (i.e., no window).

1) Hann, also called a "Hanning" or "cosine" window.
2) Hamming.
3) Bartlett, also called a "triangular" window.
4) Welch.
5) Blackman.
6) Blackman–Harris 4 term.
7) Blackman–Harris 7 term.

The windowing functions are applied continuously over the sample data, that is that the rise from zero at the first sample to unity at the *n*/2th sample and then fall back to zero at the *n*th sample. The windowing functions are applied to only the sample data if zero padding is required.

/J*n* - Jump

This switch is used to specify the starting point in the waveform. Specify *n* as the number of seconds into the waveform to begin computation. The waveform will be zero padded or cut to length as needed (see /L below).

/L*n* - Length of FFT

This switch is used to specify the length of the FFT to compute. *n* is the power which 2 will be raised for the number of points in the FFT. The waveform will be zero padded or cut as required. The data windowing function (see /W above) will be applied beginning with the first real sample of data if zero padding is required.

/S*n* - Timeseries' scaler

This switch is used to specify a scaler to be applied to the raw timeseries' before computation of the FFT. Each raw sample is simply multiplied by *n* before any other operation is performed.

/X*s,u,min,max* - X axis control

This switch is used to specify the *x* axis scaling, unit, and range. Values of *s* are "Log" or "Linear" for logarithmic and linear scaling respectively. Values of *u* are "Hertz", "Bins" or "Normalized" for units in Hertz, FFT bin numbers or normalized frequency units (DC=0, Nyquist=1) respectively. Note that *s* and *u* need only be long enough to be unique. For example you may specify *s* as "Li" instead of typing out "Linear" and *u* as "H" instead of "Hertz". The default scaling and unit may be set in the .INI file.

min and *max* are used to specify the minimum and maximum values of the *x* axis. For example, you might specify /XLI,H,80,120 to view linear frequency from 80 to 120 hertz.

If you wish to change *max* but want the default values for everything else, simply type commas, leaving these settings blank, followed by *max* (e.g., /X,,,100 will use the default values for *s, u,* and *min* and set *max* to 100).

/Y*s,u,min,max* - Y axis control

This switch is used to specify the *y* axis scaling, unit, and range. Values of *s* are "Log", "Linear" or "dB" for logarithmic, linear or decibel scaling respectively. Values of *u* are "Amplitude" or "Power" for units amplitude or power respectively. Note that *s* and *u* need only be long enough to be unique. For example you may specify *s* as "Li" instead of typing out "Linear" and *u* as "A" instead of "Amplitude". The default scaling and unit may be set in the .INI file.

min and *max* are used to specify the minimum and maximum values of the *y* axis. For example, you might specify /XLO,A,1e-5,1e5 to view log amplitude from 10^{-5} to 10^{5}.

If you wish to change *max* but want the default values for everything else, simply type commas, leaving those settings blank, followed by *max* (e.g., /X,,,1e5 will use the default values for *s, u,* and *min* and set *max* to 10^{5}).

/G[+|-] - Display grid

Use this switch is used to turn on or off the display grid in the spectrum plot. The display grid is made up of fine dotted lines crossing the plot at major ticks on both the *x* and *y* axes. You may choose the default for this setting in the .INI file (see below). You may enable or disable the *x* and *y* grid individually in the .INI file.

/N[+|-] - Use negative frequencies

This switch is used to specify whether or not the negative frequencies in the complex results of the FFT are used. If they are used, each complex pair except the Nyquist point is multiplied by 2, because the positive and negative frequencies are symmetrical around the Nyquist point. If they are not used, the points from DC to the Nyquist point are simply used. You may choose the default for this setting in the .INI file (see below).

/A[+|-] - Display all of timeseries' or just the window

This switch is used to specify whether to display the entire input timeseries' with the window marked, or to display only the portion that is within the window. This switch effects only the display. You may choose the default for this setting in the .INI file (see below).

/B[+|-] - Baseline removal

This switch is used to specify whether the displayed timeseries has had the DC offset removed before plotting. Note that this switch effects only the display, the mean is always removed from the timeseries' before the FFT is computed. You may choose the default for this setting in the .INI file (see below).

/O*filespec* - FFT output file

This switch is used to specify the file specification of a file in which to dump the raw FFT output. If the file exists, it will be overwritten. This file specification may be set in the .INI file. The file has the following format:

```
C:\SUDSUTIL\SUDSSPEC\10LP.SUD ; Input filespec
S000 ; Station ID

16384 ; N, Number of data points
8193 ; N/2+1, Number of complex pairs that follow
1000.00 ; R, Sampling rate (SPS)
1 ; S, Scaler
0.375 ; MS, Mean-square of window function

Bin#     Real                   Imag                   Amplitude
0        +2.953285457055017e+006 +0.000000000000000e+000 +1.802542393222056e+002
1        -4.178970421644875e+004 +1.017834116509454e+007 +1.242483759000359e+003
2        -7.969418704816625e+006 -7.512636222991658e+006 +1.336942113810198e+003
o
o
o
8188     -2.561781996977516e+000 -1.206076948624104e+001 +1.505107091076366e-003
8189     +1.161944681499153e+001 +2.880716240033507e+000 +1.461330372936911e-003
8190     -7.439223348163068e+000 +9.022974209394306e+000 +1.427524011601486e-003
8191     -4.900198771072610e+000 -1.042489379551262e+001 +1.406143813064585e-003
8192     +1.145917624467984e+001 +0.000000000000000e+000 +6.994126125903222e-004
```

A header starts the file. This contains the input file specification and station identifier followed by the number of input samples, the sampling rate, scaler and mean-square of the window function applied to the data.

For each bin, the bin number, real and imaginary parts of the complex pair followed by the normalized spectrum value are printed to 15 digits. Bin 0 is DC and bin N/2 is the Nyquist point.

/D*f* - Date/time format

This switch is used to specify the format used to display date/time values. If *f* = "J" (the default), all date/time values will be display in "Julian day" or day-of-year format (e.g., 92*272+14:06:24.713). If *f* = "M", all date/time values are displayed in month and day format (e.g., 9/28/92 14:06:24.713).

INI file entries

SUDSSpec searches the INI file for the [SUDSSPEC] section at startup. The entries should be thought of as defining the default behavior of the program. Many of the entries in this section are equivalent to the command line switches above. Here we will only cover those entries that are not. Command line switches that correspond to these entries will override the settings in the .INI file.

The following is a sample [SUDSSPEC] section.

```
[SUDSSPEC]
; Entries for SUDSSPEC 1.10a or later
; () indicates default value

; FFT log2 length, 5-31
FFT_Length = 10        ; (2^10, 1024 points)

; Data Window function
;Window = 0 Rectangle  ; No window
Window = 1 Hann       ;(Cosine)
;Window = 2 Hamming    ; Cosine, doesn't quite taper to zero
;Window = 3 Bartlett   ; Triangular
;Window = 4 Welch      ; Parabola
;Window = 5 Blackman   ; Blackman 3 term
;Window = 6 BH4        ; Blackman-Harris 4 term
;Window = 7 BH7        ; Blackman-Harris 7 term

; Timeseries scaling factor
; Each raw sample will be multiplied by this number
Scaler = 1.0           ; (1.0)
;Scaler = 1.41e-6

; FFT normalization method
;Normalize = MS*N*R    ;(/ by mean-square of window func * number of points * sampling
rate)
Normalize = N         ; / by number of points, window normalized
;Normalize = SQRT(N)   ; / by square-root of number of points, window normalized

; Use negative frequencies?
Negative_Freq = Yes   ; (Yes)
;Negative_Freq = No

; Output FFT to this ASCII file (None)
;Output_Filespec = FFT.OUT

; Timeseries display options ------

;Timeseries_Display = All         ; Display the entire timeseries
Timeseries_Display = Window_Only  ; Display only the portion in the window

Baseline_Removal = Yes ; Subtract mean before plotting timeseries
;Baseline_Removal = No  ; (No)

Pause = 9999.0           ; Pause in seconds after plotting in modes 1 & 2 (300)

Date_Format = DOY      ; (Day-of-year)
;Date_Format = Month/Day

; X axis control ------
; Up to 72 character user label, comment out for default labeling
;X_Label = This is the X Axis User Label

X_Grid = Yes           ; (No) Grid lines on major tic's

X_Scale = Log          ; (Linear)
;X_Scale = Linear

X_Unit = Hertz         ; (Hertz)
;X_Unit = Bins
;X_Unit = Normalized

; These fix the range of the X axis, if commented, the range
; is automatically determined.  Must be specified in the
; units as indicated in the X_Unit entry above.
;X_Min = 10            ; (Automatic)
;X_Max = 100           ; (Automatic)

; Y axis control ------

; Up to 48 character user label, comment out for default labeling
;Y_Label = This is the Y Axis User Label

Y_Grid = Yes           ; (No) Grid lines on major tic's

Y_Scale = Log          ; (Log)
;Y_Scale = dB
;Y_Scale = Linear

Y_Unit = Amplitude    ; (Power)
;Y_Unit = Power

; These fix the range of the Y axis, if commented, the range
; is automatically determined.  Must be specified in the
; units as indicated in the Y_Unit entry above.
;Y_Min = 1e-3          ; (Automatic)
;Y_Max = 1e5           ; (Automatic)
```

The Normalize entry is used to specify the normalization method used to compute the spectrum. There are three options; *ms*n*r*, *n*, and sqrt(*n*), where *ms* = the mean-square of the window function, *n* = the number of input data points, and *r* is the sampling rate. Each complex point is divided by one of these values in the normalization step. If the normalization method is *n* or sqrt(*n*), the data window function is normalized, that is, the window function integrates to 1. If *ms*n*r* is specified, the window function is not normalized.

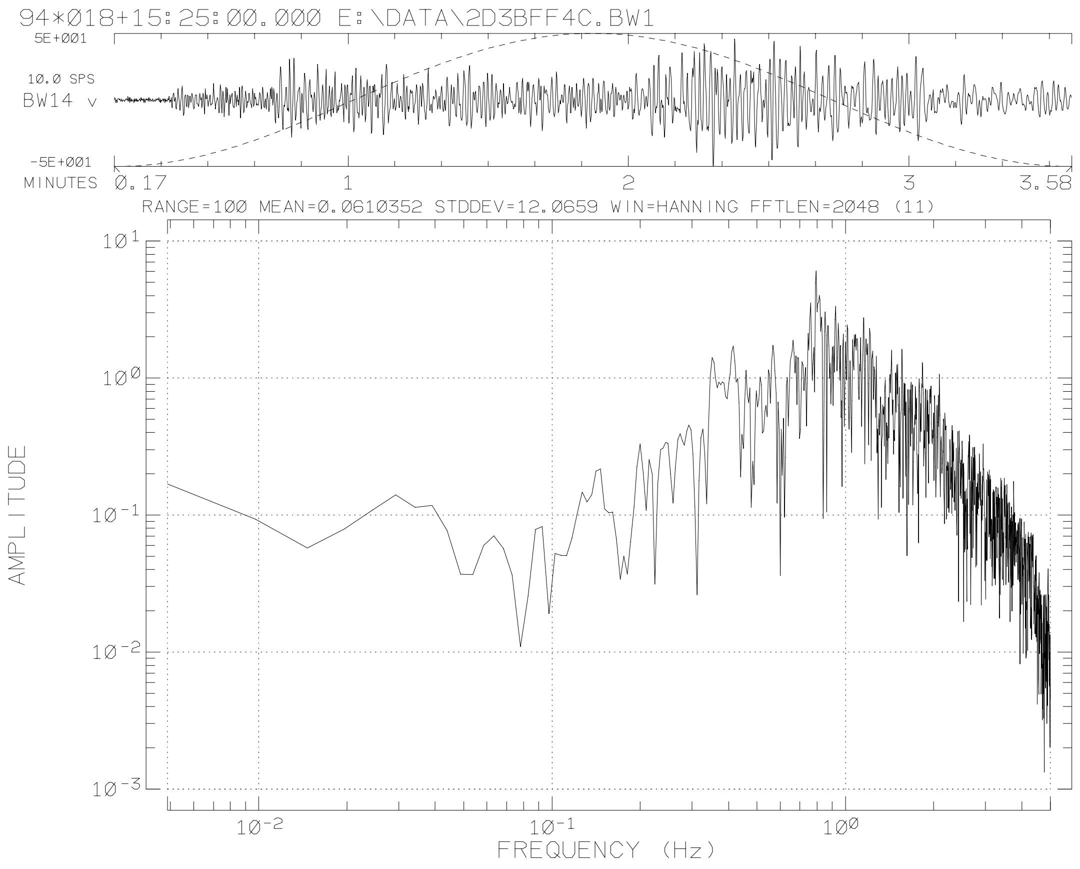
The Pause entry is used to specify a time-out value when the plot mode is 1 or 2 (a plot is displayed on the screen). When the time has expired, the program will exit.

The X_Label and Y_Label entries allow you to specify the labels plotted on the *x* and *y* axis respectively. If no entry is specified, default labeling is provided.

Plot annotation

Across the top of the plot is the initial sample time and the input file specification. To the left of the timeseries' from top to bottom is the positive peak sample value, the sampling rate in samples per second, the channel identifier followed by the component descriptor, and the negative peak sample value. The time axis is in the unit given on the left and is marked relative to the initial sample time.

Across the top of the spectrum is the range, the mean, and the standard deviation of the input data, the data window function used, and the length of the computed FFT.



The command used to create the plot above was as follows:

```
SUDSSPEC 2D3BFF4C.BW1 BW14 /L11 /YLO,A /XLO,H /J10 /P2
```

ReSpec

REPSEC is used to compute response spectra for SUDS data. The program uses the DSP32C-PC/AT digital signal processing boards from Symmetric Research for 25 Mflops performance. The algorithm used is based upon and equivalent to that used in BAP: Basic Strong-Motion Accelerogram Processing Software[6].

The program features four types of response spectra computation:

- Relative displacement response.
- Relative velocity response.
- Pseudo-velocity response.
- Pseudo-acceleration response.

Options are provided for baseline removal or to differentiate the waveform before the response computations. You have complete control over the damping values (up to 8 may be specified) and the period values at which the response is calculated.

Command line syntax

ReSpec can be controlled from the command line and by setting entries in the [ReSpec] section of the INI file. Certain aspects of ReSpec are controlled only from the INI file, although if an INI file is not found at startup, defaults are provided for all settings.

Help may be obtained by typing ReSpec ? and pressing return at the command line. The following screen will be displayed:

```
RESPEC - Version 2.40

Usage: RESPEC [switches] sudsfilespec stationname [switches]

Switches:
   /Pn    - Plot mode: 1=screen, 2=both, 3=hardcopy. (2)
   /D     - Differentiate waveform before computing response (NO)
   /Bn    - Remove baseline (DC offset) of n samples. (200)
   /Rn    - Response spectra type:
      /R0    - Relative displacement response.
      /R1    - Relative velocity response.
     (/R2)   - Pseudo-velocity response.
      /R3    - Pseudo-acceleration response.

   sudsfilespec - SUDS 1.4x data file
   stationname  - Station/component identifier for waveform to process.
```

[6]Version 1.0, USGS Open-File Report 92-196A.

```
[] = optional, () = default value.
Returns exit code 1 if an error occurred, otherwise 0.
Switches & args are not case-sensitive. Switches may appear in any order.
```

Arguments

ReSpec expects two arguments on the command line. First, the input SUDS data file specification and second, the station/component identifier for the waveform you wish to process. Only one waveform from the input file can be processed at each invocation of the program. To process several waveforms from a single file you must invoke the program for each one.

Switches

/P*n* - Plotting mode

This switch is used to specify whether the plot will be displayed on the screen and/or printed. The following modes are available:

$n = 1$	Plots are displayed on the screen only. The program will pause when the plot is complete.
$n = 2$	This is the default. Plots are displayed on the screen and then printed. The program pauses when the plot is completed on screen.
$n = 3$	Plots are generated without screen display. Status information is displayed on screen while processing. This mode is useful in batch programs.

/D - Differentiate waveform before processing

This switch specifies that the waveform is velocity and must be differentiated to acceleration before the response spectra computations. The follow expression is used to differentiate the waveform:

$$a_i = \frac{(v_{i+1} - v_i)}{\Delta t}$$

/B*n* - Baseline (DC offset) removal

This switch is used to specify the length in samples starting from the first sample that is to be averaged to establish the baseline or zero level. This value is then subtracted from each sample in the waveform before the response computations are performed. By default, a 200 sample mean is computed and subtracted.

/R*n* - Response spectra type

This switch is used to specify the type of spectra that will be computed.

n = 0	Relative displacement response.
n = 1	Relative velocity response.
n = 2	Pseudo-velocity response.
n = 3	Pseudo-acceleration response.

INI file entries

ReSpec searches the INI file for the [ReSpec] section at startup. Many of the entries in this section are equivalent to the command line switches above. Here we will only cover those entries that are not.

The following is an example [ReSpec] section.

```
[RESPEC]
; This section contains entries for RESPEC 1.00 or later.

; Number of samples used to compute mean for subtraction from timeseries
; See /B command line switch
Baseline = 300

; DSP32C coprocessor board base address (hex)
DSP_base_io = 280

; Hardcopy mode
Hardcopy = Immediate   ; Print plot immediately
;Hardcopy = Deferred    ; Save PLX file to queue directory, process later

; Differentiation of timeseries before response computations
; See /D command line switch
Differentiate = No
;Differentiate = Yes

; Type of spectra
; See /S command line switch
;Spectra = 0      ; Relative displacement
;Spectra = 1      ; Relative velocity
Spectra = 2      ; Pseudo-velocity
;Spectra = 3      ; Pseudo-acceleration

; Damping values as fractions of critical damping
; One curve will be plotted for each value, max entries = 8
Damping_Values = 0.0, 0.02, 0.05, 0.1, 0.2
;Damping_Values = 0.0, 0.025, 0.05, 0.1, 0.2, 0.3, 0.4, 0.5

; Period values, points computed along each curve, max entries = 1024
; Values must be separated by commas (and optionally spaces)
Period_Values = 0.040,  0.0425, 0.045,  0.0475,
        0.050,  0.055,  0.060,  0.065,  0.070,
        0.075,  0.080,  0.085,  0.090,  0.095,
        0.100,  0.110,  0.120,  0.130,  0.140,
        0.150,  0.160,  0.170,  0.180,  0.190,
        0.200,  0.220,  0.240,  0.260,  0.280,
 0.300,  0.320,  0.340,  0.360,  0.380,
 0.400,  0.420,  0.440,  0.460,  0.480,
 0.500,  0.550,  0.600,  0.650,  0.700,
 0.750,  0.800,  0.850,  0.900,  0.950,
 1.000,  1.100,  1.200,  1.300,  1.400,
 1.500,  1.600,  1.700,  1.800,  1.900,
 2.000,  2.200,  2.400,  2.600,  2.800,
 3.000,  3.200,  3.400,  3.600,  3.800,
 4.000,  4.200,  4.400,  4.600,  4.800,
 5.000,  5.500,  6.000,  6.500,  7.000,
 7.500,  8.000,  8.500,  9.000,  9.500,
10.000, 11.000, 12.000, 13.000, 14.000,
15.000, 17.500, 20.000, 25.000, 30.000,
35.000, 40.000, 45.000, 50.000
```

The DSP_Base_IO entry is used to specify the base address where the DSP32C signal processing board can be found. The value is a hexadecimal number. This entry must match the switch settings on the board.

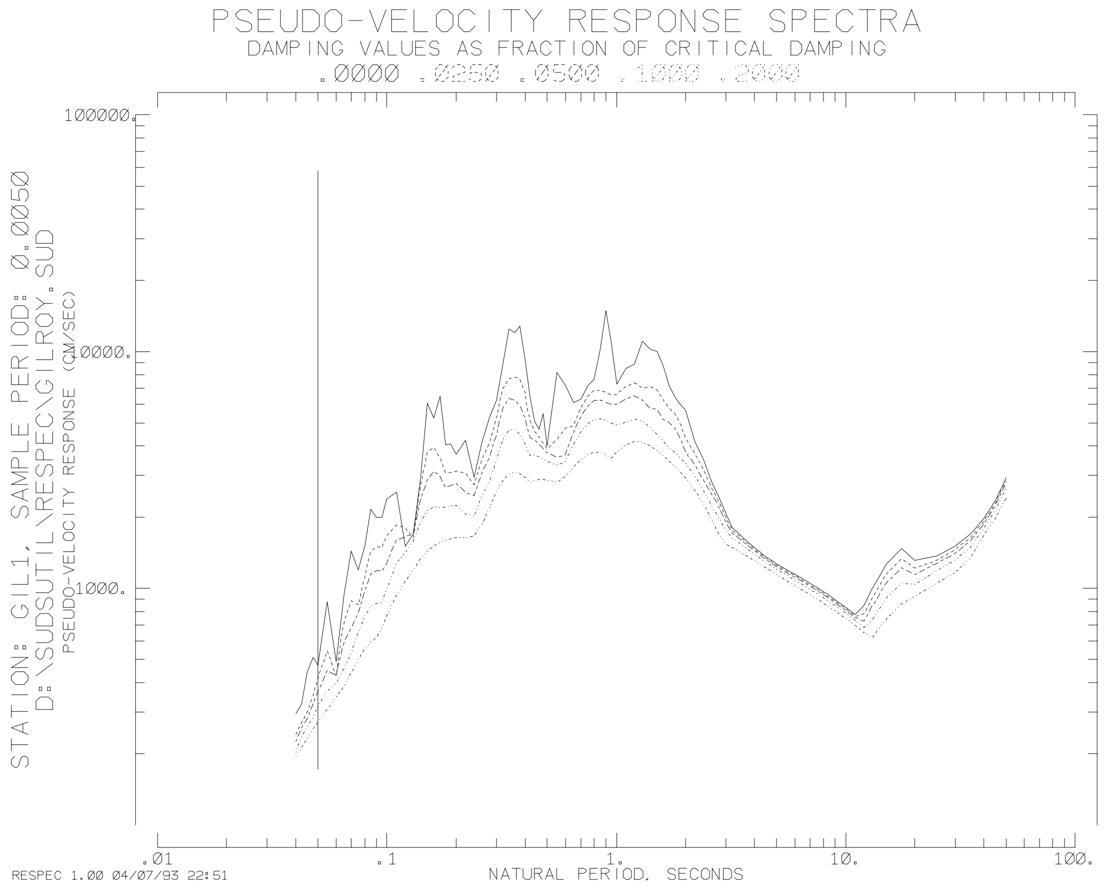
The Hardcopy entry is used to specify whether to print the plot immediately or save it to the plot queue for printing at a later time. See Getting Started for details about the graphics library.

The Damping_Values entry is used to specify up to eight damping values for which a response curve will be computed. These values must be separated by commas and can wrap to the next line. These values are expressed as fractions of critical damping.

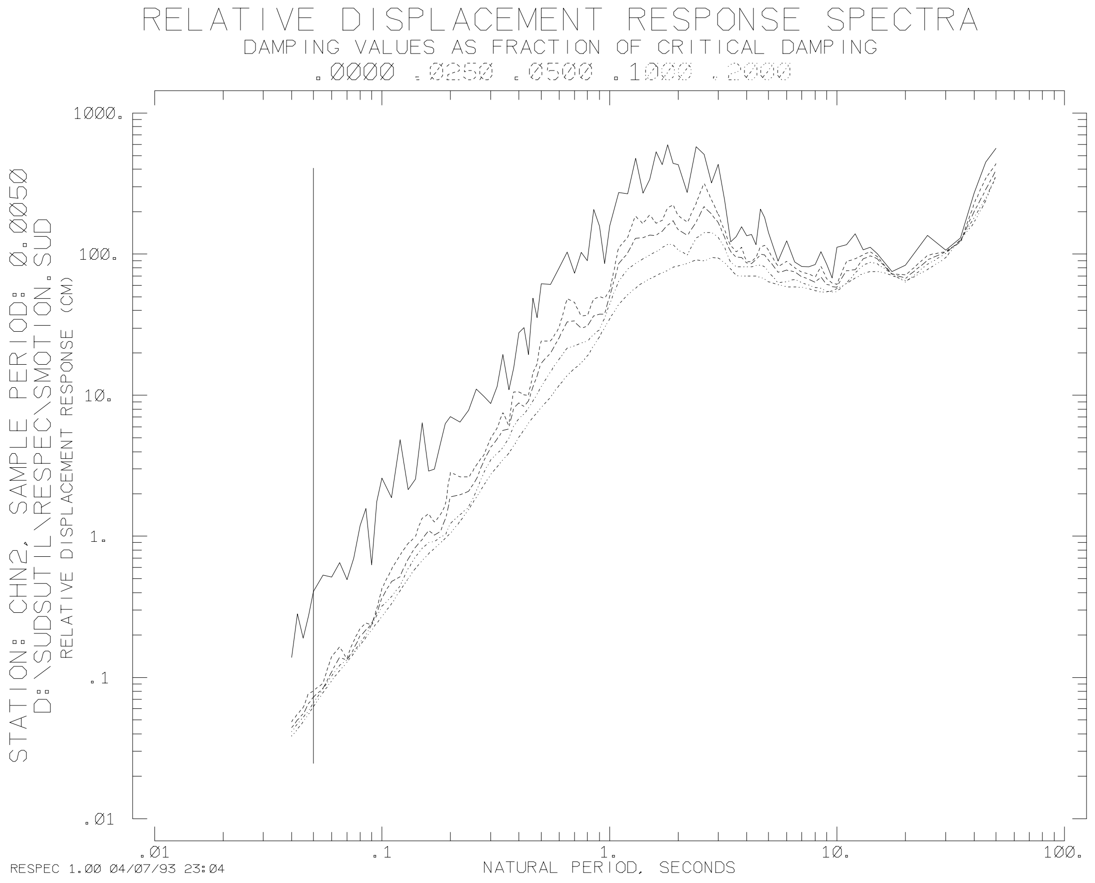
The Period_Values entry is used to specify up to 1024 period values used to build each response curve. These values must be separated by commas and can wrap to the next line. Each value represents a single point along the curve at which the response will be computed.

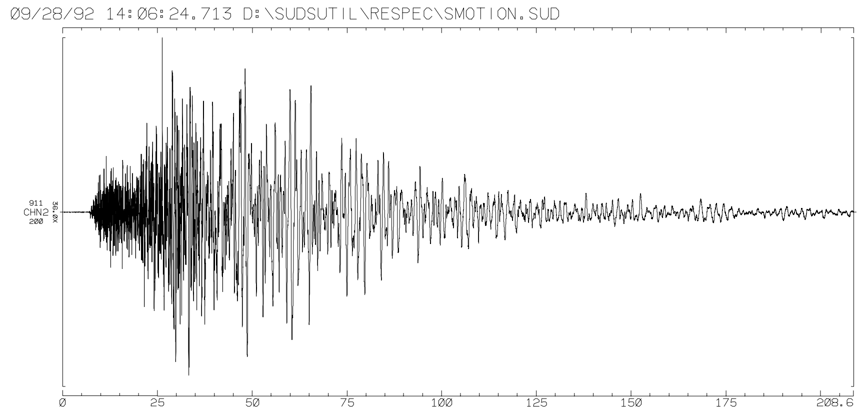
Examples

The plots show various response spectra computed by ReSpec.



The plot above shows pseudo-velocity response spectra for the five default damping values.


RELATIVE DISPLACEMENT RESPONSE SPECTRA
DAMPING VALUES AS FRACTION OF CRITICAL DAMPING
.0000 .0250 .0500 .1000 .2000
STATION: CHN2, SAMPLE PERIOD: 0.0050
D:\SUDSUTIL\RESPEC\SMOTION.SUD
RELATIVE DISPLACEMENT RESPONSE (CM)
1000.
100.
10.
1.
.1
.01
.01
.1
1.
10.
100.
NATURAL PERIOD, SECONDS
RESPEC 1.00 04/07/93 23:04


09/28/92 14:06:24.713 D:\SUDSUTIL\RESPEC\SMOTION.SUD
911
CHN2
200
0
25
50
75
100
125
150
175
208.6

CHAPTER 7

File Management Tools

This chapter covers various file management programs and other software that does not fit into the previous categories. The programs described here include:

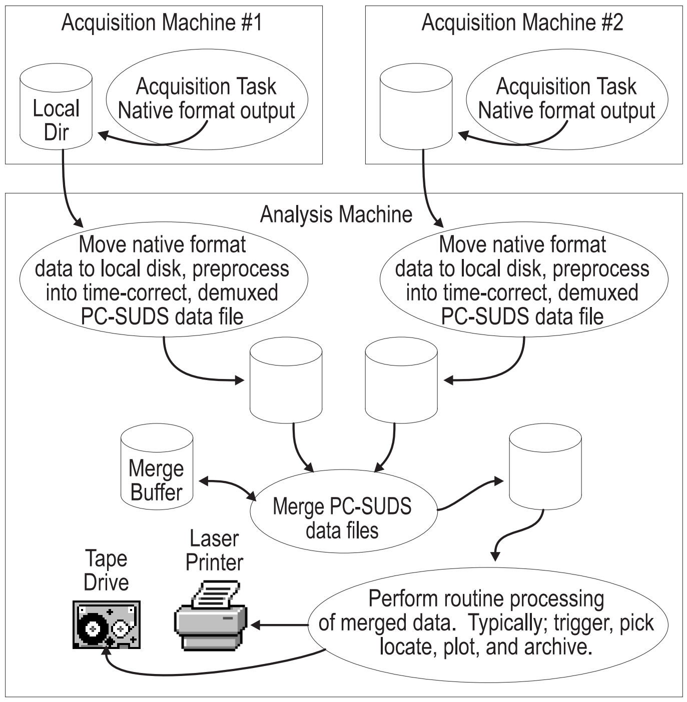
- Merge – An automated processing and data merging tool.
- SUDSMan – A tool for managing automatic data file processing.
- SUDSTrim – A tool for trimming and cutting SUDS data files.
- SUDSJoin – A tool for joining SUDS data files.
- Irig – An IRIG-E time code cracker.
- Demux – A SUDS file demultiplexer.

Merge

Merge is used to manage data from up to eight sources and associate these data before processing. With moderate sized seismic networks that have more than one acquisition systems there is the problem of merging the data from each acquisition system before processing of the data can occur. Merge allows you to retrieve data from each acquisition system, preprocess it into a PC-SUDS data file as required, merge the data, and then process the data as a complete network record.

Merge can work with data that is acquired continuously, triggered, or a combination of the two.

As an example, we will discuss a hypothetical system that has two data acquisition systems. The following figure is a block diagram of this system.


Acquisition Machine #1
Local Dir
Acquisition Task Native format output
Acquisition Machine #2
Acquisition Task Native format output
Analysis Machine
Move native format data to local disk, preprocess into time-correct, demuxed PC-SUDS data file
Move native format data to local disk, preprocess into time-correct, demuxed PC-SUDS data file
Merge Buffer
Merge PC-SUDS data files
Tape Drive
Laser Printer
Perform routine processing of merged data. Typically; trigger, pick locate, plot, and archive.

In the above figure, the small cylinders represent a directory on a disk, the ellipses represent a task, and the lines show the flow of data through the system.

The program polls each data source (Acquisition machines #1 and #2 in the figure) for the appearance of new data files. These files are in the native format of the recording system. Typically, the Merge machine polls these data source machines across a Local Area Network (LAN). When a new data file is found, the program moves the file to a working directory on the local disk and spawns a preprocessing batch program to process the file. The resulting file from this preprocess batch program should be a time-corrected, demultiplexed, PC-SUDS data file in the working directory.

After the program polls the sources and preprocesses at least one data file, it then analyzes the new file and the data in the merge buffer. This new data is either merged with other data of the same time span or it is simply added to the merge buffer so that other data may merge with it. In either case, a copy of the file (merged or not) is placed in a processing directory and a processing batch file is spawned against it.

This processing batch program may perform any sort of processing on the data file. Merge passes information to the process batch program to let it know what the status of the current file is so that it can behave accordingly.

Command line syntax

Merge currently accepts no command line arguments. It is controlled entirely from it's initialization (.INI) file. The run the Merge program, simply type "merge" at the command prompt.

The program returns an exit code of 0 to indicate successful execution and 1 to indicate that an error has occurred.

The MERGE.INI file

Merge looks for a file named "MERGE.INI" in it's home directory (C:\PCSUDS) at startup. This file has the same structure and conventions as SUDSUTIL.INI. The settings in the various sections in the INI file are used to control the Merge program.

Below is an example INI file for the Merge program. After the text of the file is a detailed discussion of each section and entry.

```
; MERGE.INI, 01-04-96, RB

; This file contain initialization information for MERGE 1.0 or later.

; This file is arranged in sections.  Each section is marked with a
; "Section Header" inside square brackets (e.g., [SOURCE]). The program
; will look for each section and then read the entries following the
; section header.  None of these entries are case-sensitive.  Comments
; are delimited by a semi-colon (;).  Blank lines and extra spaces
; are ignored.

; This file contains two sections:
;    [SOURCE] - These sections describe a data source with its
;               associated preprocess batch file.
;    [MERGE]  - This section describes the behavior of the program.

;-----------------------------------------------------------------------
[SOURCE]
; Data source #1

; Source data directory
Path = F:\DATA

; Process file with this extension (may be wildcard)
Extension = SN*

; Sort files by timestamp or filename
SortBy = Timestamp
;SortBy = Filename

; Process all except the last n files
MinimumFiles = 0

; Process last file(s) after this much time has past (seconds).
LastFileTimeOut = 0.0
```

```
; Batch file called to preprocess files before moving to the merge directory
PreProcessBatchFile = C:\PCSUDS\PREPROC1.BAT

;----------------------------------------------------------------------
[SOURCE]
; Data source #2

; Source data directory
Path = G:\DATA

; Process file with this extension (may be wildcard)
Extension = WV?

; Sort files by timestamp or filename
SortBy = Timestamp
;SortBy = Filename

; Process all except the last n files
MinimumFiles = 1

; Process last file(s) after this much time has past (seconds).
LastFileTimeOut = 310.0

; Batch file called to preprocess files before moving to the merge directory
PreProcessBatchFile = C:\PCSUDS\PREPROC2.BAT

;----------------------------------------------------------------------
[MERGE]

; Date format used for display
DateFormat = DayOfYear
;DateFormat = MonthDay

; Check source(s) for new files at this interval
CheckInterval = 5.0        ; Seconds

; Working directory entries -------------------------------------------

; Working directory, each preprocess should put files in here
WorkingPath = C:\MERGE

; Merge files with these extensions in the working directory
MergeExtensions = SN0, SN1, SN2, SN3, DMX

; Merge parameters ----------------------------------------------------

; Merge window
MergeWindow = 30.0         ; Seconds

; Merge files go stale (marked for purge) after this much time
;StaleFile = 10.0 Seconds
StaleFile = 30.0 Minutes
;StaleFile = 6.0 Hours
;StaleFile = 2.0 Days

; Once n records have merged, record is considered complete (purge)
MergeCompleteCount = 5     ; Number of records merged

; Purge merge buffer at this interval
PurgeInterval = 10.0 Seconds
```

```
;PurgeInterval = 10.0 Minutes
;PurgeInterval = 6.0 Hours
;PurgeInterval = 7.0 Days

; Merge buffer directory, partially merge files are stored here
MergeBuffer = C:\MERGE\BUFFER

; Merged filename extension
MergedExtension = MRG

; Post process entries ------------------------------------------------

; Merged files are moved to this directory for processing
ProcessPath = C:\PROCESS

; Batch file called to process merged files
ProcessBatchFile = C:\PCSUDS\PROCESS.BAT
```

The [SOURCE] sections

Lets assume, for the propose of demonstration, that in the example system shown in the figure, the analysis machine sees the local disk on acquisition machine #1 as network drive F:, and the local disk on acquisition machine #2 as network drive G:. Further, that the data acquisition systems on these machines both output data files that are two minutes long into a directory on their local disk named C:\DATA.

A [SOURCE] section is used to describe a source of data. Up to eight sources of data can be managed simultaneously by Merge. A separate [SOURCE] section is used to describe each data source. Each [SOURCE] section may contain five entries:

- Path = *path*

  This is the pathname to the source of data. In our example, the path to the data on acquisition machine #1 is F:\DATA. This is a mandatory entry. At least one [SOURCE] section with a valid path must be specified in the INI file.

- Extension = *extension*

  This is the filename name extension used in a wildcard mask (*.*extension*) when Merge looks at a directory listing of the source path. This entry defaults to "*". Any DOS legal filename extension may be specified, including wildcards, e.g., "SN?".

- SortBy = *Timestamp* or *Filename*

  This entry specifies whether the source directory is sorted by the file timestamp or the filename. Merge needs to process these file in chronological order from oldest to newest. Some acquisition software does not name output files so that they sort chronologically and those that do sometimes use names that wrap around at some interval (day, month,…). On these types of systems, Merge will need to sort these files based on the timestamp that the operating system placed on them when they were created to avoid problems with out-of-order processing. The entry defaults to *filename*.

- MinimumFiles = *n*

  Some acquisition software holds the current output file open and writes to it in real-time, e.g., XRTP. When the file is complete, it then closes the file. Some systems also perform post-processing to these files as well. This entry specifies how many files, counting from the newest file back, are to be considered off-limits. By specifying *n* as 1, you're telling Merge that the very newest file cannot be processed until a newer file appears or the current file times-out (see LastFileTimeOut below).

  Merge checks permissions on the file before it attempts to move it. If the acquisition software uses the file sharing features of DOS 3.x or later to deny read/write access to the file until it is finished with it, this entry may be set to 0. This entry defaults to 2.

- LastFileTimeOut = *seconds*

  This entry specifies the amount of time in seconds that must pass before the last file may be processed. This entry is only meaningful when the MinimumFiles entry is set greater than 0. This entry should be set just longer than the maximum record length the acquisition software will create. For example, if the MaximumLength parameter in XRTP.INP is set to 300.0 seconds, LastFileTimeOut should be set to about 310.0 seconds. This setting is especially important when the acquisition software will be operating in a triggered mode. It is less important for continuous systems because there will always be a new file created to cause the current file to be processed.

- PreProcessBatchFile = *filespec*

  This is the file specification of the batch program that will preprocess the data from this source to prepare it for merging. This preprocessing typically involves; format conversion, time-correction, and demultiplexing. This is a mandatory entry. For an example, see Preprocess batch programs below.

  This batch program will be passed three arguments: the fully qualified pathname of the data file to be processed, the filename, and the extension. These file specification components are accessed in the batch program as the replaceable parameters; %1, %2, and %3, respectively.

  The resultant file from this batch program must reside in the directory represented by %1. The filename and extension may be changed as required but bear in mind that the filename must be unique and the extension used must be listed in the MergeExtensions entry in the [MERGE] section (see below).

The [MERGE] section

The [MERGE] section contains various parameters for the Merge program itself.

- DateFormat = *DayOfYear|MonthDay*

  This entry specifies how dates are displayed, either year and day-of-year, or month-day-year may be used. This entry defaults to day-of-year notation.

- CheckInterval = *n*

  This is the interval, in seconds, at which the program will poll the data source(s) for new data. This entry defaults to 10.0.

- WorkingPath = *path*

  This entry specifies the working directory for Merge. This is the directory were preprocessed data files from each source will be placed in preparation for merging. This is a mandatory entry, we recommend that this be a directory on the local disk named C:\MERGE.

- MergeExtensions = *ext1, ext2…*

  This entry specifies the extensions used on data files in the working directory to be merged. Up to 32 extensions may be specified. The files in the working directory with this extension will be merged in to the merge buffer and then processed.

  Some acquisition systems used native formats that contain more than one stream in the data file. After conversion, there will be one PC-SUDS data file with a different extension for each stream. Simply list each of these extensions for each source here. This is a mandatory entry.

- MergeWindow = *n*

  This entry specifies the time window in seconds used as criteria for the merge operation. The initial sample time (IST) from the current record is compared with IST's from records in the merge buffer, if the current records IST is within ± the merge window of any record in the buffer, the current record is merged into the buffer, otherwise it is added to the buffer. This entry defaults to 10.0.

- StaleFile = *n units*

  This entry specifies the amount of time that a record is allowed to reside in the merge buffer. *n* can be specified with *units* as Seconds, Minutes, Hours, or Days. The program purges the merge buffer periodically for both stale files and completed records. See the PurgeInterval entry below for a detailed description of the purge operation. This entry defaults to 10 minutes.

- MergeCompleteCount = *n*

  This entry specifies that when *n* records have been merged together, that record is complete and marked for purge (see below). This entry defaults to 2.

- PurgeInterval = *n units*

  This entry specifies the interval at which the program will purge the merge buffer. *n* may be specified with *units* as Seconds, Minutes, Hours, or Days. A record in the buffer may be purged for one of two reasons. First, the record may be stale, that is, it may have resided in the buffer for too long. Second, the record may be complete, that is, data for the records time span may have been gathered from every source. In either case, the purged record is moved to the process directory and is processed by the process batch program (see below). This entry defaults to 10 minutes.

  The process batch program is passed status as the fourth argument. The first three arguments are the components of the file specification. When a purged record is processed, this fourth argument will contain "STALE" or "COMPLETE" to indicate the status of the record. See the ProcessBatchFile entry below.

- MergeBuffer = *path*

  This entry specifies the directory where partially merged data is held. Merge also creates an index file in this directory named MERGE.NDX. This is a mandatory entry, we recommend that this be a subdirectory of the working directory, e.g., C:\MERGE\BUFFER.

  This directory should be left to be managed by the Merge program only. Your batch programs should not read from or write to this directory.

- MergeExtension = *extension*

This entry specifies the filename extension that merged data records will be given. This entry defaults to “MRG”.

- ProcessPath = *path*

  This entry specifies the directory where merged records will be copied to, and purged records will be moved to for final processing. This is a mandatory entry, we recommend that this directory be on your local disk and named C:\PROCESS, however, you may have a need for this to be on a network drive in you installation.

- ProcessBatchFile = *filespec*

  This is the file specification of the batch program that performs the final processing of merged and purged data records. This processing typically involves; triggering, phase picking, hypocenter determination, plotting, and data archival. This is a mandatory entry. For an example, see Process batch programs below.

  This batch program will be passed four arguments: the fully qualified pathname of the data file to be processed, the filename, the extension, and the record status. The file specification components are accessed in the batch program as the replaceable parameters; %1, %2, and %3, respectively. %4 will contain one of the following status indicators:

  MERGED:*n* – The current file contains *n* merged records. If this is the first record for this time-span, *n* while be equal to 1. Each time a record is merged, the merged record will be processed with *n* equal to the total records contained in the file.

  There may or may not be any processing necessary when this status is indicated. If you data sources provide continuous data, you might run triggering software each time a new record appears.

  COMPLETE – The current file is complete, that is, it contains the number of records specified in the MergeCompleteCount entry (see above). Note if the complete count is equal to 5, and there are 4 records in the buffer, when the fifth record comes in, it will be processed with status MERGED:5 and then marked for purge. It will then be purged with complete status.

  Final first-order processing should be performed when this status is indicated.

  STALE – The current file has resided in the buffer for longer than allowed by the StaleFile entry (see above). The record is not complete. The processing performed when this status is indicated might simply be the same as for complete status, or it might save these records aside for an analyst to review.

Batch programs used by Merge

The following sections present several example batch programs for use by Merge. These may be used as a starting point for your particular system. These files are provided on the distribution diskette and can be found in the \PCSUDS directory after installation.

Preprocess batch programs

The following text is the contents of PREPROC1.BAT. This is the preprocessing batch program for data source #1 in our example system. The program converts the input file (a PASSCAL image file in this case) into PC-SUDS format and then moves the resulting files into the working directory.

```
@ECHO OFF

REM PREPROC1.BAT, Thu 04-Jan-1996 13:35, RB

REM This is a preprocess batch file that is called by MERGE every time it
REM moves a file from a data source.  The file should be processed into
REM a time-corrected, demuxed PC-SUDS data file and then moved
REM to the working directory (see MERGE.INI).

REM The file specification is passed as the first three arguments to this
REM batch file.  These are the; pathname, filename, and extension
REM respectively.

REM Assert that a complete and valid filespec was passed in.
IF "%1" == "" GOTO ASSERT
IF "%2" == "" GOTO ASSERT
IF "%3" == "" GOTO ASSERT

REM Assert that the file exists
IF EXIST %1\%2.%3 GOTO START
ECHO ERROR: File does not exist: %1\%2.%3
PAUSE > NUL
GOTO END

:ASSERT
ECHO ERROR: Incomplete or invalid filespec passed was to PREPROC1.BAT.
ECHO Argument 1: %1
ECHO Argument 2: %2
ECHO Argument 3: %3
PAUSE > NUL
GOTO END

:START
REM ------------------------------------------------------------------
REM All user processing starts here.

REM - Convert to PC-SUDS
REF2SUDS %1\%2.%3 %1
DEL %1\%2.%3

:END
```

The batch program simply runs the conversion program, REF2SUDS in this case. The conversion program generates output files in the directory specified as %1. The original file, represented by %1\%2.%3 is then deleted. This is a simple example, you should see that almost any amount of processing may be performed here.

Here are a few tips about batch programming under DOS:

- Don't make any assumptions about what is the current drive or directory, always use fully qualified file specifications.
- Be very careful about any other dependencies that your batch program might have, such as, network drive mappings, virtual disks, temporary directories, and so on. Be deliberate about such matters.
- Clean up after yourself. Don't leave intermediate files lying around in whatever happens to be the current directory. These batch programs are executed over and over again, don't find yourself tripping over trash from the last invocation. A common mistake is to leave a specifically name temporary file lying around. On the next invocation, you rename the current file to the same name and the RENAME command fails because the file already exists. Be careful and tidy. If you don't lookout for yourself in these matters, you can rest assured that no one is…

As another example, lets say that we are acquiring data with XRTP (from the IASPEI library) and we want the data to be time-aligned to the next even minute boundary (the acquisition system is generating two minute records, continuously). Below we'll leave off the preamble portion of the file (everything down to the :START label).

```
REM ------------------------------------------------------------------
REM All user processing starts here.

REM - Demultiplex the record
DEMUX %1\%2.%3 %1\%2.S01
DELETE %1\%2.%3

REM - Correct timing
IRIG %1\%2.S01

REM - Time align the record -------------------------------

REM - See if we have the previous record
IF EXIST %1\SOURCE1.PRE GOTO ALIGN
GOTO SAVE_PRE

:ALIGN
REM - Rename to SOURCE1.CUR
IF EXIST %1\SOURCE1.CUR DELETE %1\SOURCE1.CUR
REN %1\%2.S01 %1\SOURCE1.CUR

REM - Join the previous and current records
SUDSJOIN %1\SOURCE1.PRE %1\SOURCE1.CUR %1\SOURCE1.S01
```

```
REM - Trim to next even minute boundary
SUDSTRIM %1\SOURCE1.JOI %1\%2.S01 /M2

REM - Save current records for next time
:SAVE_PRE
MOVE %1\SOURCE1.CUR %1\SOURCE1.PRE

:END
```

Process batch programs

The following text is the contents of PROCESS.BAT. This batch program is called to process merged records. The three components of the file specification (path, name, and extension) are passed as the first the arguments to the batch program. A record may be in one of three states as indicated by the contents of the fourth argument. The record status will be; MERGED:*n*, COMPLETE, or STALE. The specific processing that will be performed is determined by checking this status. See the ProcessBatchFile entry in the [MERGE] section above for more about status.

```
@ECHO OFF

REM PROCESS.BAT, Thu 04-Jan-1996 13:35, RB

REM This is the process batch file that is called by MERGE every time it
REM adds a record to the merge buffer, merges a record into the merge
REM buffer, or purges the merge buffer.

REM The file specification is passed as the first three arguments to this
REM batch file.  These are the pathname, filename, and extension
REM respectively.

REM The fourth argument is a status indication passed as a string as follows:

REM  "%4" = "MERGED:n" = This file made up of n merged records.  Note
REM                      that for a complete record, the file is
REM                      processed at the final merge and then again
REM                      when purged as a complete file.
REM  "%4" = "COMPLETE" = This file is being purge because it is
REM                      complete. It contains records >= the
REM                      MergeCompleteCount entry in MERGE.INI.
REM  "%4" = "STALE"    = This file is being purged because it has been
REM                      in the merge buffer longer than allowed by the
REM                      StaleFile entry in MERGE.INI.

REM Assert that a complete and valid filespec was passed in.
IF "%1" == "" GOTO ASSERT
IF "%2" == "" GOTO ASSERT
IF "%3" == "" GOTO ASSERT

REM Assert that the file exists
IF EXIST %1\%2.%3 GOTO START
ECHO ERROR: File does not exist: %1\%2.%3
PAUSE > NUL
GOTO END
```

```
:ASSERT
ECHO ERROR: Incomplete or invalid filespec passed was to PROCESS.BAT.
ECHO Argument 1: %1
ECHO Argument 2: %2
ECHO Argument 3: %3
PAUSE > NUL
GOTO END

:START

ECHO.
ECHO %4 - %1\%2.%3
ECHO.

REM - If this file was purged as a complete record...
IF "%4" == "COMPLETE" GOTO COMPLETE
REM - If this file was purged as an incomplete record...
IF "%4" == "STALE" GOTO STALEFILE

REM ------------------------------------------------------------------
REM - Perform routine processing for each merged record here...
ECHO Process merged record: %1\%2.%3

GOTO END

REM ------------------------------------------------------------------
REM - Process complete records here...
:COMPLETE
ECHO Process complete record: %1\%2.%3
GOTO END

REM ------------------------------------------------------------------
REM - Handle stale files here...
:STALEFILE
ECHO Handle stale file: %1\%2.%3

GOTO END
:END
```

SUDSTrim

SUDSTrim is used to combine data from several files into one file and cut or pad each waveform so that they share the same initial sample time (IST) and length. Typically data for a particular event are stored in several files. This is especially true for data recorded on portable instruments. These individual files may be combined by using the DOS copy command with the /B (binary) switch but the data will not be padded or cut. SUDSTrim performs the same function as copy except that additional processing is performed to align and trim the data.

Command line syntax

SUDSTrim is controlled entirely from the DOS command line. The program returns an exit code of 0 to indicate successful execution and 1 to indicate that an error occurred during processing. If an error does occur, a verbose message will be written to stdout (the screen) describing the error.

Help may be obtained by typing SUDSTrim ? and pressing return at the command line. The following screen will be displayed:

```
SUDSTRIM - Version 2.40

Usage: SUDSTRIM [switches] inputfile(s)... outputfile [switches]

Switches:
   /MIN  = Only data common to all traces written to output file.
  (/MAX) = All trace data written, short traces padded.
   /Idatetime = Absolute IST, "MM,DD,YY,HH,MM,SS.SSS" or "YY,JJJ,HH,MM,SS.SSS"
       depending on /D switch, commas may be replaced with any non-digit.
   /Jn   = Jump, IST = earliest IST + n seconds.
   /Ln   = Absolute length, n seconds relative to IST.

   /Df   = Date format: (f=M=month/day), f=J=Julian day or day-of-year.
   /Pn   = Padding sample value = n. (0)
   /Bn   = Baseline, mean value of n seconds subtracted from all samples. (0)
   /Tn   = Taper n seconds, implies /P0 and /Bn if not explicitly specified.(0)
   /N    = No warning if output file exists. (WARN)

   inputfile(s) may be wildcarded, outputfile must be last filename
   on command line.  Switches may appear at any position.

[] = optional, () = default.
Arguments are not case sensitive.
```

Arguments

SUDSTrim accepts two or more SUDS file specifications as arguments. DOS wildcards are acceptable. At least two file specifications are required: an input file and an output file. More than one input file may be specified, the last file specification is always taken as the output file. If the output file exists, you will be warned that SUDSTrim will overwrite the file (see the description of the /N switch below).

The files are processed in the order that they appear on the command line. If the input file specification is a wildcard, the files are processed in the order that they appear in the directory. The order of structures in the files is maintained as they are copied and processed into the output file.

Switches

Switches may appear at any point on the command line. These switches are not case sensitive.

/MAX - Copy all data

This is the default behavior of the program. The earliest IST and the latest last sample time (LST) is found. All data is then padded as necessary as it is copied to the output file. When a waveform is padded it may be tapered or padded with a specified value, see /T, /B and /P switches below.

/MIN - Copy only data common to all waveforms

This switch will cause the program to cut all waveforms to the latest IST and the earliest LST. This has the effect that only data common to all waveforms is copied to the output file. A baseline (DC offset) may be removed from all waveforms during this process (see /B switch below).

/I*date/time* - Absolute IST

This switch allows you to specify the IST that will be used in processing. The *date/time* value may be specified using two different formats: month/day or day-of-year. Which format is used depends on the /D switch. By default month/day notation is used.

All waveforms will be padded or cut as necessary and any taper, baseline or padding value as specified with the /T, /B or /P switches respectively will be applied.

/J*n* - Jump

This switch allows you to specify the IST relative to the earliest IST in the input file. Specify *n* as the number of seconds to add to the earliest IST.

All waveforms will be padded or cut as necessary and any taper, baseline or padding value as specified with the /T, /B or /P switches respectively will be applied.

/L*n* - Length

This switch is used to specify the length of waveforms to be output in seconds. The IST is established using the /MAX, /MIN or /I switches. The output data will be padded or cut to n seconds in length. Any taper, baseline or padding value as specified with the /T, /B or /P switches respectively will be applied as required.

/D*f* - Date/time format

This switch is used to specify the format used for all date/time input and output. *f* = "M" = MM/DD/YY,HH:MM:SS.SSS and *f* = "J" = YY*DDD+HH:MM:SS.SSS, where:

YY	=	Year - Century (e.g., 93, not 1993).
DDD	=	Day-of-year or "Julian day" (1-366).
MM	=	Month (1-12).
DD	=	Day of month (1-31).
HH	=	Hour of day (0-23).
MM	=	Minute of hour (0-59).
SS.SSS	=	Second of minute (0-59.999).

By default the program uses the day-of-year (J, "Julian day") format. When specifying a date/time value the individual numbers may be separated by any non-digit. Below are several example command lines using the different date/time formats:

```
SUDSTRIM FILE1.SUD FILE2.SUD /I2,8,93,12:00:1.234
SUDSTRIM FILE1.SUD FILE2.SUD /DJ /I93,37,12:00:1.234
```

/P*n* - Padding sample value

This switch is used to specify the value given to padding samples that may be added to the output data. By default the data is padded with zeros. If the taper switch (see /T switch below) is used the padding value is implied to be zero and this switch is ignored.

/B*n* - Baseline (DC offset) removal

This switch is used to specify the length (*n* seconds) of the waveform used to compute a mean sample value that will be subtracted from each sample in the waveform as it is processed. By default no baseline is computed. If the taper switch (see /T switch below) is used this switch is implied with the same length as the taper.

/T*n* - Taper

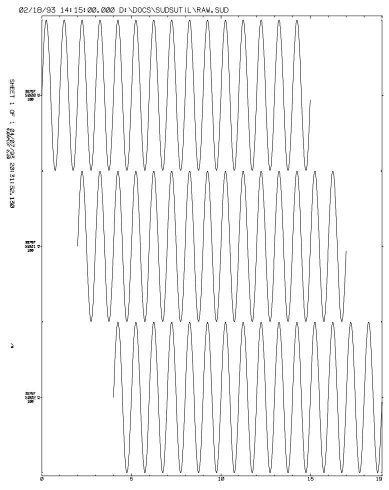
This switch is used to specify the length (*n* seconds) of a cosine taper to be applied wherever a waveform is padded. By default no taper is applied. The use of this switch implies a padding sample value (see /P switch above) of zero and a baseline (see /B switch above) of *n* seconds. A different baseline length may be specified by using the /B switch although the padding sample value will be zero regardless of the /P switch setting.

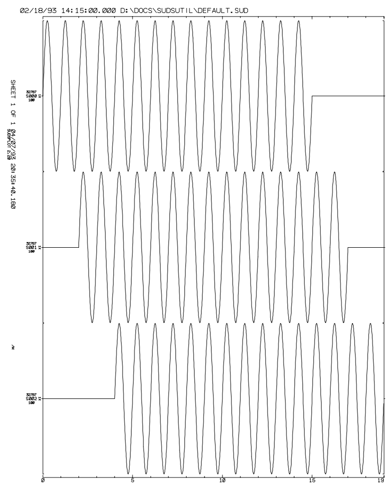
/N - No warning if output file exists

This switch simply tells the program not to prompt you with a warning of the output file already exists. The file is simply overwritten.

Examples

The plots on the following pages demonstrate the use of the various command line switches. Following each plot is an explanation and the command line used to create the displayed data. SUDSPlot version 2.20 was used to create the plots.

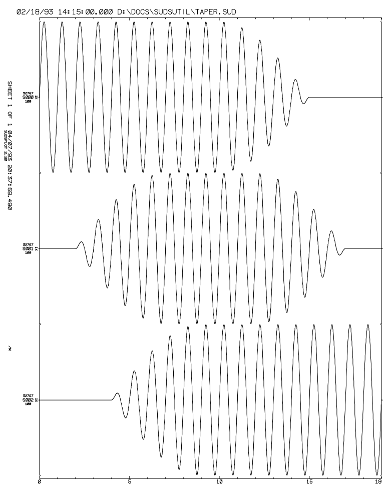


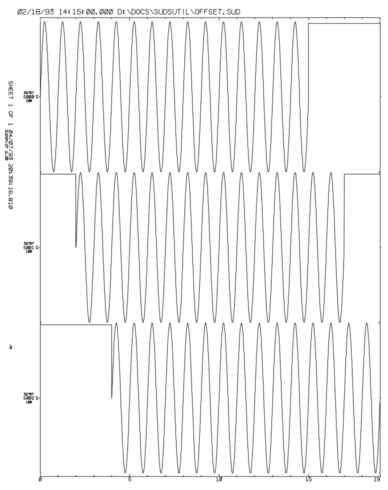


The plot above left is the raw data that will be trimmed. It consists of three 1 Hz sine waves that are 15 seconds long. Each waveform is offset +2 seconds with respect to the previous waveform.

This plot above right shows the default behavior of SUDSTrim. Waveforms are padded from the earliest IST to the latest LST with zeros. No taper or baseline is applied. The data was processed using the following command:

```
SUDSTRIM RAW.SUD DEFAULT.SUD
```



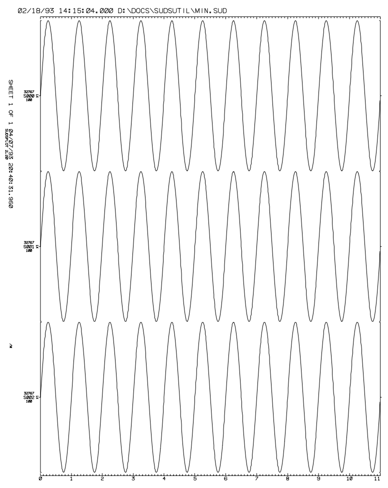


The plot above left shows the use of the /T switch. A 5 second taper and baseline are applied and the maximum amount of data is taken. The data was processed using the following command:

```
SUDSTRIM RAW.SUD TAPER.SUD /T5
```

The plot above right shows the use of the /P switch. Waveforms are padded with a sample value of 32000 instead of 0. The data was processed using the following command:

```
SUDSTRIM RAW.SUD OFFSET.SUD /P32000
```



The plot above shows the use of the /MIN switch. Only data common to all waveforms is copied. The data was processed using the following command:

```
SUDSTRIM RAW.SUD MIN.SUD /MIN
```

SUDSJoin

SUDSJoin was written to deal with a very specific problem: to rejoin a continuous waveform that is in two pieces in two files. This situation arises when acquiring long events with RTP or XDETECT. For example, if the record length setting is 60 seconds and a large event re-triggers the software immediately at the end of 60 seconds, the waveform will be stored in two different files. The purpose here is put the pieces back together again.

Command line syntax

SUDSJoin is controlled entirely from the command line. The program returns an exit code of 0 to indicate successful execution and 1 to indicate that an error occurred during processing. If an error does occur, a verbose message will be written to stderr (the screen) describing the error.

Help may be obtained by typing SUDSJoin ? and pressing return at the command line. The following screen will be displayed:

```
SUDSJOIN - Version 2.40

Usage: SUDSJOIN [switches] inputfile1 inputfile2 outputfile [switches]

Switches:
   /Wn  = Padding window, maximum padding when joining traces = n seconds. (10)

[] = optional, () = default.
Arguments are not case sensitive.
```

Arguments

The program expects three SUDS file specifications as arguments: the two files to be joined and the output file. The entire contents of the first input file will be copied to the output file. Any waveform(s) in the second input file that match up with waveforms in the first input file will be joined in the output file.

If the two waveforms to be joined overlap, all of the first waveform is used and the front of the second waveform is cut. If the waveforms to be joined do not overlap but the LST of the first waveform is within 10 seconds of the IST of the second waveform, the first waveform is padded with zeros and the waveforms are joined. If the IST of the second waveform is greater than 10 seconds later than the LST of the first waveform or is earlier than the IST of the first waveform, the first waveform is simply copied to the output file and a warning is issued. The width of this window can be controlled by using the /W switch (see below).

The program matches up overlapping waveforms based on time. The “join” sample in the second waveform is determined by subtracting the IST of the second waveform from the LST of the first waveform. The nearest sample to the absolute time is used. If the join sample in the two waveforms does not contain the same sample value, a warning is posted to stdout informing you of the mismatch. This warning has the form:

```
WARNING: sample mismatch: iiii:nnnn, iiii:nnnn
```

where: *iiii* is the sample index and *nnnn* is the sample value in the first and then second files respectively.

The program writes an ASCII text log file named SUDSJoin.LOG in the current directory. This file contains all of the information that was displayed on the screen during execution.

Switches

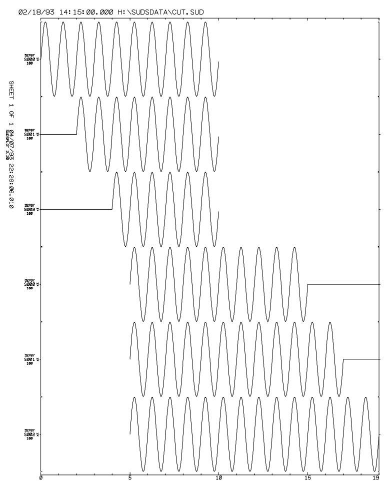
Switches may appear at any point on the command line. These switches are not case sensitive.

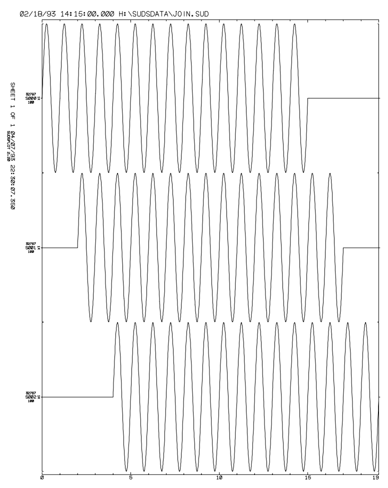
/W*n* - Exclude window length

This switch is used to specify the maximum number of seconds that the IST of the second waveform is allowed to follow the LST of the first waveform. By default this is set at 10 seconds. This prevents you from accidentally joining two waveforms that might have days or weeks between them and generating a truly huge data file.

Examples

The plots on the following page demonstrate usage of SUDSJoin.





The first three waveforms in the plot above right are from the first file CUT1.SUD and the second three are from the second file CUT2.SUD. As the plot shows, these two files contain waveforms that overlap by 5 seconds.

The left plot shows the two files: CUT1 and CUT2 after joining by SUDSJoin. The following command was used to join the files:

```
SUDSJOIN CUT1.SUD CUT2.SUD JOIN.SUD
```

SUDSMan

SUDSMan (the SUDS file manager) is designed to work across a local area network (LAN) to provide near real-time automated routine processing of seismic data. SUDSMan runs on the "off-line" machine that will be used to perform the analysis and it monitors a second computer (the "on-line" machine) for the creation of data files. When a file appears it is moved across the network and a DOS batch program (PROC.BAT) is called to process it. Once the file has been processed, SUDSMan returns to monitoring the on-line machine.

This batch program can be modified to perform any type on processing of the data. The program keeps a detailed log of all activity with time stamps on each event.

Command line syntax

SUDSMan can be controlled entirely from the command line or by setting entries in the [SUDSMan] of the INI file.

Help may be obtained by typing SUDSMan ? and pressing return at the command line. The following screen will be displayed:

```
SUDSMAN - Version 2.40

Usage: SUDSMAN [switches] source_dir dest_dir

Switches:

   /M=mask File name mask for source directory (*.WVM).
   /B=batchfile File name of batch file to execute for each file copied.
   /I=n    Seconds to wait before a file is inactive.
   /W=n    Seconds between checks for new files.
   /V      Verbose mode.
   /S      Safety, Copy and process first file only.
              Source file will not be deleted.

Arguments are not case sensitive.
All of these settings can be made in the .INI file.
Command-line setting override settings in the .INI file.
```

Arguments

SUDSMan accepts two command line arguments: the source directory specification and the destination directory specification. These should be fully qualified path specifications (including the drive letter). The program will monitor the source directory for the appearance of data files and when one appears it will be moved to the destination directory where PROC.BAT will be called to process it.

Switches

/M*mask* - Source filename mask

mask should be the filename mask used when monitoring the source directory. In most cases this will be a wildcard filename (e.g., *.WVM). By default, the program will only look for files that match the *.WVM wildcard.

/I*n* -Inactive time-out

When the program checks the source directory and more than one file is present, it simply moves and processes the oldest file. If only one file is present, the program cannot safely move the file until it is sure that whatever program that is generating it is finished and the file is closed. Because of this the program will wait *n* seconds after the appearance of the file before it moves and processes the file.

The value for *n* should be slightly longer than the longest possible record length set for the acquisition software.

/W*n* - Wait time

When the program checks for files in the source directory and none are present, the program will wait *n* seconds before checking again. This "sleeping" will continue until a data file appears or you interrupt the program.

/V - Verbose mode

This switch causes the program to display more comprehensive status messages to the display while running.

/S - Safety on

This switch sets safety mode on. In safety mode the program functions exactly as usual except the source file is copied instead of moved. The purpose of this mode is to allow you to debug your PROC.BAT file safely.

INI file entries

SUDSMan searches the INI file for the [SUDSMan] section at startup. The entries in this section are equivalent to the command line arguments and switches above. Anything specified on the command line will override the corresponding entry in the INI file.

Below is an example of the [SUDSMan] section of the INI file.

```
[SUDSMAN]
```

```
; This section contains entries for SUDSMAN 1.01 or later.

; Run in verbose mode
Verbose

; Source directory mask
Mask=*.PRE

; Source and destination directories
Source=G:\XDETECT
Destination=D:\PROC

; Time in seconds after which a file is to be considered inactive
Inactive=120

; Time in seconds to wait between checks for new files
Wait=10
```

PROC.BAT

PROC.BAT is called each time that a file is moved from the source to the destination directory. The filename without the extension is passed as the only command line argument.

This batch program should perform whatever processing you wish without any user interaction.

The following example PROC.BAT is provided on the distribution diskette as a starting point, you will probably need to edit it to meet your particular needs.

```
@ECHO OFF
REM Proc.bat

REM Make sure we have a filename as first argument
IF "%1" == "" GOTO END

REM Archive the data
COPY %1.WVM F:\ARCHIVE

REM Correct the timing
IRIG %1.WVM

REM Demux the file on RAM disk D:
COPY %1.WVM D:\%1.WVM
DEMUX D:\%1.WVM C:%1.DMX

REM Pick phases
AUTOPICK %1.DMX
IF ERRORLEVEL 1 GOTO ERROR
XTRHY71 %1.DMX

REM Locate the event
CALL GOHYPO %1

REM Plot a record section
SUDSPLOT /B200 /M /A /R %1.DMX %1.PRT
IF ERRORLEVEL 1 GOTO ERROR
```

```
REM Save the phase list and hypocenter solution
COPY %1.PHA F:\ARCHIVE
COPY %1.PRT F:\ARCHIVE

REM Clean up
ECHO Y | DEL D:\*.*
ECHO Y | DEL *.*

REM Done
GOTO END

:ERROR
ECHO An error occurred while processing %1
PAUSE

:END
```

This example batch program could be used if you are using XDETECT or RTP and they are recording Irig-E time code on a channel named Irig. This example also expects the Hypo71PC is set up and is invoked by calling a batch program named GOHYPO.BAT that accepts the filename as a command line argument.

The batch program would then archive, correct timing, demultiplex, pick, locate and plot a record section of each event with no user interaction.

SCSITape

When processing large amounts of seismic data in near real-time, we have a need to be able to quickly archive rather large data files onto tape as they are created. With most off-the-shelf backup software, it is difficult to backup a single file very quickly. SCSITape was designed to backup a few files as quickly as possible and yet provided a simple directory system that allows the user to restore files easily.

Although SCSITape was specifically designed to drive DDS and DDS-2 tape drives, it should work with most other SCSI tape drives (e.g., Exabyte or QIC). SCSITape communicates with the SCSI bus through the ASPI interface. ASPI is a host adapter independent method of communicating with SCSI devices, that is, you should be able to use any ASPI compliant host adapter to communicate with your tape drive.

The software was written and tested using Adaptec AHA-152x and AHA-1542 host adapters, Hewlett Packard 35470A, 35480A and C1533A DDS and DDS-2 tape drives, and Adaptec's EZ-SCSI 3.1 ASPI manager software.

SCSITape requires no extended or expanded memory and can be run in the background in a DOS virtual machine under Windows 3.x running in 386 Enhanced mode or Windows95 when multitasking is required.

Installation

To install SCSITape, copy SCSITAPE.EXE to directory on your hard drive that is on your search path (i.e., a directory that is included in your PATH= statement in AUTOEXEC.BAT). The program will create a file named SCSITAPE.DIR in its home directory as needed. The contents of this file are described in directory section below.

You must have a SCSI host adapter installed on your system with the SCSI bus properly terminated and attached to at least one tape drive. The ASPI manager software must be installed and operating correctly before SCSITape can function.

If you have more than one host adapter in your system, the ASPI manager may need to be loaded once for each adapter. These adapters will be numbered from 0 through *n*, up to 4 can be installed. SCSITape will use host adapter 0 unless otherwise instructed. To use an adapter other than zero, either specify the host on the command line, or set an environment variable named SCSI_HOST=*n*, where *n* is the number of the adapter that you wish to use.

By default, SCSITape will use the first sequential access (tape) device that it finds on the first (adapter 0) SCSI bus. If you have more than one sequential access device on the bus, you may set an environment variable named

SCSI_TAPE=*n*, where *n* is the SCSI device unit ID on the default host of the device that you wish to use. You may also specify the ID number on the command line with the /D*[host:]id* switch, see below. Note that the /D switch overrides the SCSI_TAPE environment variable setting.

Using SCSITape

SCSITape was written to be a simple command line utility program. The program may be called from within batch programs and returns an exit code to indicate success (0) or failure (1). The program is controlled entirely from the DOS command line. Note that none of the commands, arguments, or switches are case sensitive.

The program offers help screens at several levels. To see general help for all commands, type: SCSITAPE ?, and press return. The following screen will be displayed:

```
SCSITape - Version 1.13

Usage: SCSITAPE command [arguments and switches]

Commands:
  F[ORMAT] [TapeName] [/U]
  B[ACKUP] Filespec [/Mn] [/R]
  R[ESTORE] Filespec|#FileNumber[:FileNumber] [/A] [/O] [/N]
  E[JECT]
  U[NLOAD]
  L[OAD]
  D[IRECTORY] [/R]

Global switches:
  /D[Host:]DeviceID - Set SCSI host and device, overrides SCSI_TAPE variable.

Type: 'SCSITAPE command ?' for detailed help.

Set the environment variables: SCSI_HOST=h and SCSI_TAPE=i, where h=host
adapter#, and i=SCSI device ID#.  Defaults to the first tape device on
host adapter# 0.

[]=Optional, ()=Default, |=Mutually exclusive.
```

The program accepts seven different commands. Commands are specified by typing SCSITAPE followed by the command verb followed by any arguments and/or switches on the DOS command line. Note that the command verb need only be long enough to be unique, that is, you may type B instead of BACKUP. The arguments and switches may appear in any order after the command. Below is a brief description of the commands and their purpose:

- Format – This command prepares the tape for use.
- Backup – This command copies files to the tape.
- Restore – This command copies files from the tape.
- Eject – This command ejects the tape from the drive.

- Unload – This command unloads the tape from the heads in the drive.
- Load – This command loads the tape onto the heads in the drive.
- Directory – This command validates the directory file on disk.

Below we will cover each command in more detail in their own section.

The only switch common to all commands is the /D*[host:]id* switch. This switch specifies the SCSI device ID number and optionally the host ID of the device to use, where *host* is the host ID number (0-3) and *id* is the device ID number (0-7) on that host's bus. If *host* is not specified, the setting of the SCSI_HOST environment variable is used. If this variable is not set, host number 0 is used. If this switch is not used to specify the device, the setting of the SCSI_TAPE environment variable is used. If this variable is not set, the first sequential access device on the SCSI bus is used.

The program creates or appends to a log file in the current directory during each session. This file is named SCSITAPE.LOG. This file will contain a summary of each SCSITAPE session. Below is a log file fragment showing the typical contents of SCSITAPE.LOG:

```
SCSITape - Version 1.13
Session time: 11/17/95 09:55:34
Command line: format SCSITAPE Source Files

Format tape...

SCSI target device:
  0:6 - Sequential access device, removable media:
          HP, HP35470A, Rev:1109, SCSI-2

Medium in drive: 90 meter cartridge
Directory filespec: C:\PCSUDS\SCSITAPE.DIR
Tape name:    SCSITAPE Source Files
Tape time:    11/17/95 09:44:42
Formatted on: HP, HP35470A, Rev:1109, SCSI-2
Program used: SCSITape version 1.13

------------------------------------------------
SCSITape - Version 1.13
Session time: 11/17/95 09:57:53
Command line: backup *.* /r

Backup file(s)...

SCSI target device:
  0:6 - Sequential access device, removable media:
          HP, HP35470A, Rev:1109, SCSI-2

Medium in drive: 90 meter cartridge
Directory filespec: C:\PCSUDS\SCSITAPE.DIR
Medium has not changed since last access.
Processed 116 files in 3 directories.
Transferred 5.36 MB at 16.48 MB/min.
Tape contains 116 files totaling 5.36 MB.

------------------------------------------------
```

Format Command

To see help for this command, type: SCSITAPE FORMAT ?, and press return. The following screen will be displayed:

```
SCSITape - Version 1.13

SCSITAPE F[ORMAT] [TapeName] [/U]

Switches:
   /U - Unconditional, don't check for existing format, force overwrite.

Arguments:
   TapeName - The volume name to give the tape. ("UNNAMED")

This command prepares a tape for use by SCSITape.  To specify a tape name
that contains spaces, enclose the name in quotes, e.g. "Tape #1".

[]=Optional, ()=Default. Switches are not case sensitive and may appear
anywhere after the command in the command line.
```

The format command prepares a tape for use by SCSITape. A new tape must be formatted before it can be used. This command will effectively erase a previously formatted tape that contains existing data and therefore should be used with care.

This command accepts one optional argument: the name to be given to the tape. If you do not specify a tape name, the tape will be given the name "UNNAMED".

Use the /U switch to tell the program to unconditionally format the tape. By default, if the tape is already formatted, the program will display the tape name and other information and prompt you as to whether or not it should be overwritten. Use this switch with care!

Backup Command

To see help for this command, type: SCSITAPE BACKUP ?, and press return. The following screen will be displayed:

```
SCSITape - Version 1.13

SCSITAPE B[ACKUP] Filespec [/Mn] [/R]

Switches:
   /Mn - Minimum number of files that must exist before backup. (1)
         If n > 1, files are deleted after backup.
   /R  - Recurse subdirectories. (No)

Arguments:
   Filespec - The file(s) to backup to tape.  May be any DOS legal file
              specification including wildcards.

This command copies the specified file(s) to the tape and adds them to
the tape directory.  You may press ESCAPE to abort the backup operation.
```

```
[]=Optional, ()=Default. Switches are not case sensitive and may appear
anywhere after the command in the command line.
```

Use the Backup command to copy files onto the tape. The file specification may be any DOS legal partial or fully qualified file specification with wildcards. If the file specification is a wildcard, all files that match the wildcard pattern will be backed up.

As files are backed up, they are assigned a file number. This file number and the file specification for each file are written to the directory file (see below) and displayed on the console.

The /M*n* switch allows you to specify the minimum number of files that must exist before actually backing them up. The purpose of this switch is to allow you to call SCSITape at each iteration of your processing batch file, that is each time a data file is created, but have it only perform the backup operation every *n*th time. This can greatly increase performance because the overhead involving tape positioning and so forth is incurred only once. If you specify the minimum number greater than one, the files will be deleted after they are successfully backed up to the tape.

The /R switch causes the program to recurse subdirectories below the specified directory. For example, if you specify C:\*.* as the input file specification the /R switch will cause the program to backup all files in C:\ and then backup all files in all directories under C:\ (that is, the entire disk). Note that if you specify C:\YOURDIR\*.*, only the directories under C:\YOURDIR will be recursed.

The program leaves the tape positioned at the end of data so that on the next call to the program, as little tape positioning as possible will be needed.

Note that if you press the Escape key to abort the backup, the program will not stop until the current file is completely copied to the tape and the directory on the tape is updated. So please be patient. Stopping the program with Control-Break will leave the current file incomplete and destroy the tape directory requiring a directory rebuild command to fix the tape.

Restore Command

To see help for this command, type: SCSITAPE RESTORE ?, and press return. The following screen will be displayed:

```
SCSITape - Version 1.13

SCSITAPE R[ESTORE] Filespec|#FileNumber[:FileNumber] [/A] [/O] [/N]

Switches:
   /A - Restore all files on tape. (only specified files)
   /Opath - Restore file(s) to path. (to original path)
   /N - No directory will be used, must restore with file #'s only.

Arguments:
   Filespec - The file to restore, no wildcards allowed.
   #FileNumber[:FileNumber] - The number of the file to restore.  If the
      second file number is specified after the colon, the files starting
      from the first file number through the second file number (inclusive)
      will be restored.

This command restores the specified file(s) from the tape.  The files are
placed in their original location unless the /O switch is used to specify
a output pathname.  The /N switch can be used to restore files without using
the directory, restore using file numbers only.  You may press ESCAPE to abort
the restore operation.

[]=Optional, ()=Default. Switches are not case sensitive and may appear
anywhere after the command in the command line.
```

Use the restore command to retrieve files from the tape and place them back onto your hard drive. This command expects one or more arguments: the file(s) to restore from tape. The arguments may be specified as file names, in which case the file numbers will be looked up in the directory, or as file numbers by specifying a number sign followed by the file number, e.g., #23.

You may specify a range of files by following the first file number with a colon and the last file number. In this case the files numbered first though last (inclusive) will be restored.

You may specify as many file names and/or numbers on the command line as you like. As an example, the command:

```
SCSITAPE RESTORE file1 file2 #34:62 #123
```

will restore the files named file1 and file2, all files numbered 34 through 62, and the file numbered 123.

This command accepts three switches: /A, /O*path*, and /N.

The /A (restore all) switch causes the program to restore all files on the tape.

The /O*path* (output path) switch allows you to specify the output path. For example, by specifying /OC:\DATA, you cause the program to restore files to the C:\DATA directory regardless of where the originally were stored.

By default, the program restores files to their original locations without regard for the drive letter. This allows you to backup files from one drive and restore them to another. The directory structure will remain intact unless you use the /O switch to specify a particular output directory.

The /N switch allows you to restore files by file number without using the tape directory. This can be useful when a tape has been corrupted by a power failure during a backup for example. This switch should be used only as a last resort, try rebuilding the tape directory (see the Directory command below) before using this switch.

Unload, Load, and Eject Commands

These commands perform various media handling functions. None of these commands accept any arguments or switches.

The unload command tells the tape drive to unload the tape head mechanism. On helical scan drives, such as DDS or Exabyte, this can help extend the life of the tape. If you are going to leave a tape in the drive for a long time, but not use it regularly, it is a good idea to unload the heads.

The load command is simply the opposite of the unload command. The command loads the tape head mechanism and prepares the drive for use.

The eject command simply tells the drive to eject the tape. This causes the drive to rewind and unload the cartridge as if you pressed the eject button on the drive. Note that some tape drives, such as most QIC drives, do not support this function.

Directory Command

To see help for this command, type: SCSITAPE DIRECTORY ?, and press return. The following screen will be displayed:

```
SCSITape - Version 1.13

SCSITAPE D[IRECTORY] [/R]

Switches:
   /R - Rebuild directory from tape.

Arguments:
   None.

This command synchronizes the directory on disk with the directory
stored on the tape, retrieving the directory from tape if needed.
The directory file is stored in SCSITAPE's home directory (i.e., the
directory where SCSITAPE.EXE is located) and is named SCSITAPE.DIR.
This is an ASCII text file containing the list of files on the tape.
Entries are the file number followed by the filespec and file size in
bytes.

The /R switch causes the program to analyze all data on the tape and
create a new directory.  The new directory is stored to the tape on
completion.  This operation can take an hour or more depending on the
size of the tape and how full it is.

[]=Optional, ()=Default. Switches are not case sensitive and may appear
anywhere after the command in the command line.
```

The directory command forces the program to validate the tape directory. This means that if a directory file exists in the home directory, it is checked against the directory on the tape to ensure that they are synchronized. If they are not, or the directory file does not exist on disk, the directory is restored from the tape.

When you invoke SCSITape, it determines whether the directory file on disk needs validation automatically. If the file exists on disk and the media has not changed since the last time it was accessed and the drive has not been reset or power cycled, the validation step is skipped in favor of performance. It is possible to fool the program into thinking that the directory is valid when it is not, particularly after an abnormal termination or power failure. So if in doubt, issue the directory command to force the program to validate the directory.

The /R switch tells the program to rebuild the directory from the tape. This means the program will analyze the tape and build a new directory file on disk. Upon completion, this new directory will be written to the tape. This process can take quite a long time to complete and so should be performed only when needed.

If a tape has been physically damaged, you may not be able to rebuild the directory. You can recover data from the tape by using the restore command with the /N switch to restore whatever undamaged data might be on the tape (see Restore above).

The directory file is always named SCSITAPE.DIR and is stored in the home directory, i.e., the directory where SCSITAPE.EXE is located. This file is a simple ASCII text file and may be viewed using a text editor or simply writing it the console with the DOS COPY, TYPE, or MORE commands.

The directory file contains a six line header containing information needed by the program. This is the tape identifier, tape name, and other data. This header is followed by a list of files on the tape in order. These entries consist of the file number followed by the fully qualified file specification and the size in bytes of each file on the tape, one per line. The file number is the number used as arguments to the restore command (see above).

The following is the first few lines of a directory:

```
SCSITAPE.EXE 1.13  <-Signature
951117094538  <-Tape identifier
\\BRAVO\C-DRIVE Complete Backup  <-Tape name
11/17/95 09:45:38  <-Format date
HP, HP35470A, Rev:1109, SCSI-2  <-Format device
-------------------------------------------------------------------------
1 C:\DATA\SUDS\2F6502D0.RT1 16216
2 C:\DATA\SUDS\2F6510E1.RT1 32538
3 C:\DATA\SUDS\2F651EF1.RT1 14832
4 C:\DATA\SUDS\2F652D02.RT1 12324
```

You should not edit the directory file because it will be out of sync with the tape and the program may not be able to tell this. This can cause undefined behavior from the software.

Glossary of SCSI Terms

SCSI terminology is littered with many acronyms. Some of these are quite obscure and can, at times, be very confusing. Below we have provided a short glossary of the more commonly used term as pertains to SCSI in the hope that we might help avoid and/or dispel some of the confusion.

ANSI	The American National Standards Institute.
ASPI	The Advanced SCSI Programming Interface. A software standard developed by Adaptec to allow application software to communicate with SCSI devices through a consistent set of interface routines, without regard to the particular host adapter hardware. The interface is typically implemented as an ASPI manager device driver but is sometimes implemented in firmware on the host adapter.
Byte	A unit of data storage consisting of 8 *bi*nary digi*ts* (bits). A byte is capable of storing 2^8 or 256 possible values. Typically we think of a byte as the amount of storage needed for a single alpha-numeric character.
DAT	Digital Audio Tape. This is a format used with a 4mm helical scan tape cartridge for storage of digital audio data. This format incorporates two levels on error detection and correction (C1 and C2).
DDS	Digital Data Storage. This is a format used to store digital data on 4mm helical scan tape cartridges. The cartridge has the same form factor as DAT, however, the data are stored on the medium in a different format. Because data storage applications are less forgiving than digital audio, a third level of error detection and correction (C3) and read–after-write verification is employed. 60 meter DDS tapes hold ~1.2 GB and 90 meter DDS tapes hold ~2.0 GB. Newer drives typically employ hardware data compression to achieve capacities up to ~4.0 GB.
DDS–2	An extension to the DDS format that allows longer tapes and higher densities. DDS–2 tape are 120 meters long with a native capacity of ~4 GB. Most DDS–2 drives employ hardware data compression to achieve capacities up to ~8.0 GB.
Exabyte	Exabyte is a company that builds high capacity tape drives. They pioneered the use of 8mm helical scan tape for data storage. This is the same form factor as 8mm video tape except that data grade tape is used. Exabyte also builds 4mm DDS and DDS-2 drives.
Gigabyte (GB)	A unit of data storage equal to 2^{30} or 1,073,741,824 bytes. A gigabyte is equal to 2^{10} or 1,024 megabytes, and 2^{20} or 1,048,576 kilobytes.
Host adapter	An interface board that plugs into the backplane of a computer and attaches it to the SCSI bus.
Initiator	Usually a host adapter. A SCSI device that issues commands and receives responses from targets on the SCSI bus.

Kilobyte (KB) ...	A unit of data storage equal to 2^{10} or 1,024 bytes.
Megabyte(MB)..	A unit of data storage equal to 2^{20} or 1,048,576 bytes. A megabyte is equal to 2^{10} or 1,024 kilobytes.
QIC	Quarter Inch Cartridge. As the name implies, these cartridges use ¼” tape. The data is stored in linear tracks in a serpentine format. Common tapes are QIC-40 and QIC-80. These tapes range in capacity from 40 MB to over 1 GB with the newer types.
SCSI	Small Computer Systems Interface, pronounced “scuzzy”. SCSI is an American National Standard interface used to connect disk drives, tape drives, CD-ROM’s, printers, scanners, and other peripheral devices to microcomputers.
SCSI bus	The physical cabling that connects SCSI devices. The bus can connect up to 8 devices (16 on wide SCSI). Typically, the bus consists of one initiator and one or more targets, although several initiators and/or targets may reside on the bus at one time. Each device occupies a unique address on the bus (0-7) called its SCSI device unit ID.
SCSI device.......	A device such as a host adapter, disk drive, or tape drive that conforms to the SCSI interface standard and occupies an address on the SCSI bus. The device may be an initiator (e.g., the host adapter), or a target. Some devices are capable of acting as both an initiator and a target.
SCSI-2..............	The second revision of the SCSI standard. This standard included expanded device types, wide SCSI (16 and 32 bits), and fast SCSI (>10Mbits/sec). This standard was adopted by ANSI in 1995. Most SCSI devices available today comply with this standard.
Target................	A disk drive, tape drive, CD-ROM, scanner, printer, or other peripheral device capable of responding to commands from initiators.
Terabyte (TB) ...	A unit of data storage equal to 2^{40} or 1,099,511,627,776 bytes. A terabyte is equal to 2^{10} or 1,024 gigabytes, 2^{20} or 1,048,576 megabytes, and 2^{30} or 1,073,741,824 kilobytes.
Termination.......	A physical requirement of the SCSI bus that each end of the bus must have a series of termination resistors installed. Most devices have terminators built into them that are enabled or disabled with switches, jumpers, or through software. Care must be taken so that only the devices at each physical end of the bus have termination installed or enabled in order for the bus function properly.

Irig and Demux

Irig-E time decoder

Irig is used to crack Irig-E timecode recorded as waveform data in a multiplexed or demutiplexed SUDS data file (e.g., files created by XDETECT or RTP). The program reads the Irig-E waveform, performs the required computations and determines the clock correction that should be applied to the initial sample time and the true sampling rate. These values are then written back into the SUDS file in a TIMECORRECTION structure. Status information concerning accuracy and satellite lock are displayed and saved in the SUDS structure. A log file named Irig.LOG is maintained in the current working directory containing all information written to the screen. This file is appended to at each invocation of the program.

Irig expects a SUDS data file specification as the first argument on the command line. By default the program will look for the time code in the file with a station/component identifier of "Irig". If the time code was recorded with a different name, that name must be passed as the second argument on the command line.

SUDS file demultiplexer

Demux is used to demultiplex SUDS data files created by XDETECT, MDETECT, TDETECT or RTP. The program expects the first argument on the command line to be the input SUDS data file specification and the second to be the output SUDS file specification.

The program simply processes the MUXDATA SUDS structures in the input file and outputs DESCRIPTRACE SUDS structures to the output file with any timing corrections (see above) applied in the process.

The program accepts two command line switches: /12 and /16. These switches force the data type descriptor in the SUDS DESCRIPTRACE structures in the output file to indicate 12 or 16 bit data respectively. These switches should not be required when using current versions and exist only to address compatibility problems with very early versions of the acquisition and utility software.

CHAPTER 8

PC- SUDS I/O library

This chapter describes version 1.45 of the PC-SUDS I/O library. This chapter was written for the programmer who wishes to develop applications that will process data stored in PC-SUDS. The reader is assumed to have a working knowledge of the C and/or FORTRAN programming language(s).

Note that the version number of the library is also the general version of PC-SUDS as a whole. The library defines PC-SUDS. That is, the library fully implements the data structures for a given version of PC-SUDS. If the library can read a data file, that is, by definition, a PC-SUDS data file. See appendix B for the definitions of the most commonly used structures.

Version 1.45

Version 1.45 differs from 1.44 only in changes need for year 2000 readiness.

The differences between this and previous versions involve the minor changes to the library code and the addition of the SUDS_CHANSET structure. At version 1.40, the library has been completely rebuilt providing simplified input/output (I/O) functions that are language independent and the ability to read/write very large (megabyte or greater) data objects from/to data files.

This implementation of SUDS is fully backward compatible with all previous versions of SUDS for DOS. Data files created with software using previous versions of the library should be readable with this library and visa versa.

The library code was written using Microsoft C/C++ version 7.0 and is callable from C or FORTRAN. The library should be callable from any of the Microsoft language products if you build the appropriate interface code. See SUDS.FI, SUDS.INC and the Mixed Language Programming Guide that came with our compiler for examples.

We have used the Microsoft specific keyword _huge in declaring data pointers in the library. If the you wish to use a non-Microsoft compiler, you must modify the library source code and recompile with your compiler. We have tried to keep the use of non-ANSI compliant features to a minimum.

Using the SUDS library

The SUDS library provides a convenient and consistent way for the application programmer to read and write SUDS structures and data from/to SUDS files.

Along with the source code, a ready-built library is provided on the diskette as SUDS144.LIB. This is a large model, emulator math package library for use with Microsoft C and/or FORTRAN versions 5 or later. If you are going to access the library from FORTRAN, you must link to the standard FORTRAN and C run-time libraries. See the Mixed Language Programming Guide that came with your compiler for details about this process.

The library functions are divided into three categories: I/O functions, utility functions and time functions.

The I/O functions use a composite structure (see SUDS structures below) to pass structures read or written back and forth between the functions and the application program. This structure contains three elements; the SUDS structure type identifier, the length of any data that follows the structure in bytes, and a union of all SUDS structure types. You will access the proper element of the union based on the type identifier. This technique may be a bit hard to understand at first if you have not had much experience dealing with structures, please refer to your language reference if needed. If you are working in C, see the end of SUDS.H for the definition of the suds structure typedef. If you are working in FORTRAN, see the end of SUDS.INC for the definition of /suds_struct/ record type. The example code demonstrates this technique clearly for both C and FORTRAN.

Currently, the library allows the user to have up to ten SUDS file's open concurrently. If you need to have more than ten files open at once you may increase the constant _SUDS_MAX_FILES in SUDSIO.C and rebuild the library. Note that you may need to increase the maximum number of open files allowed by DOS by modifying the FILES= entry in CONFIG.SYS.

All time values in the SUDS structures are stored as the number of seconds that have elapsed since 1/1/70 00:00:00.0. These values are stored either as an 8 byte real (type double in C, REAL*8 in FORTRAN) giving better than millisecond resolution, referred to MS_TIME (millisecond time) or as an 4 byte integer (type long in C, INTEGER*4 in FORTRAN) giving 1 second resolution, referred to as ST_TIME (stamp time). Storing date/time values in this way allows the user to perform date/time arithmetic in a straightforward way. It should be noted that the user is responsible for correcting for leap seconds, we intend to provide a diff_time function in a future version of the library that corrects for leap seconds. The time functions give the user the ability to encode, decode and display these values conveniently.

File I/O functions

The following is a summary of the file I/O functions provided in the library.

`suds_open( filespec, access )`

Opens a SUDS file for I/O, returns a file descriptor (fd).

`suds_close( fd )`

Closes a SUDS file specified by fd.

`suds_read( fd, suds )`

Reads the next structure in the file specified by fd.

`suds_read_data( fd, data, length )`

Reads length bytes of data that follows the previous structure into data from the file specified by fd.

`suds_write( fd, suds )`

Writes the next structure in the file specified by fd.

`suds_write_data( fd, data, length )`

Writes length bytes of data to follow the previous structure into the file specified by fd.

`suds_flush( fd )`

Flushes system I/O buffers containing data written to the file specified by fd to disk.

`suds_pos( fd )`

Returns the current position in the file specified by fd.

`suds_seek( fd, position )`

Returns to a previous position in the file specified by fd.

`suds_rewind( fd )`

Rewinds the file specified by fd, same as suds_seek(fd, 0).

Utility functions

The following is a summary of the utility functions provided in the library.

`suds_init( suds )`

Initializes all elements of a SUDS structure to default values.

`suds_get_err( )`

If any of the above functions return an error condition, this function returns a string containing a verbose error message.

Time functions

The following is a summary of the time manipulation functions provided in the library. These functions are all year 2000 ready.

`get_mstime( )`

Returns the current system date and time as MS_TIME.

`make_mstime( year, month, day, hour, minute, seconds )`

Returns MS_TIME for date and time specified as components parts.

`decode_mstime( mstime, year, month, day, hour, minute, seconds )`

Decodes MS_TIME into component parts.

`list_mstime( mstime, format )`

Returns a string containing an ASCII representation of MS_TIME in one of ten different formats.

`yrday( month, day, leap )`

Returns day-of-year for month and day.

`mnday( doy, leap, month, day )`

Returns month and day for day-of-year.

`isleap( year, cal )`

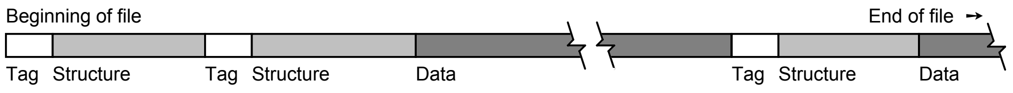
Returns whether or not year is a leap year.

SUDS file structure

SUDS files are made up of tagged structures on disk or tape forming a singly linked list. Each SUDS structure may be followed by any amount of data in any form. For a detailed description of the SUDS structures, see SUDS Structures below.

It is important to note that when using the library functions to access a SUDS file (the recommended method), the standard file I/O routines provided by whatever language you are using not be used to access the file. If you wish to access the SUDS file directly, use the SUDS library close function and then re-open with the open function provided by your language.

The library manages the tag structures and allows the user to read, write or seek structures and data in the file without dealing with the tags, although an understanding of the file structure is helpful when using the library. The tag is a 12 byte structure that identifies the structure that immediately follows it, its length and the length of any data that may follow it.



Using C syntax, the tag structure has the following form:

```
struct SUDS_STRUCTTAG {
   char sync;
   char machine;
   int struct_type;
   long struct_length;
   long data_length;
};
```

The sync element must always contain the character “S”. This is used to insure the file is not out of sync. The machine element identifies the hardware representation (byte order and floating point representation) of the data in the file, this must contain the character “6” for Intel x86 based machines.

The struct_type element indicates the type of SUDS structure of struct_length (in bytes) that immediately follows the tag in the file. If any data follows the SUDS structure, it is data_length bytes in length.

The structure type constants are defined in SUDS.H and SUDS.INC for C and FORTRAN respectively. If the tag indicates an unrecognized structure type, it is important that you simply ignore the structure and process all of the structures that are recognized. This insures the extensibility of SUDS by allowing users to define additional structures as needed, although the addition of a new structure type should not be taken lightly and should be coordinated with all users.

It is important to use the lengths for the structure and data provided in the tag when navigating the file because the version of the structure defined in the library may differ in length from the version of the structure in the file. For example, there may have been elements added to the end of the structure as currently defined.

Library function reference

If you will call the library functions from C, you must include SUDS.H in any module that contains calls to the library or references the structures.

If you will call the library functions from FORTRAN, you must include SUDS.FI outside any program unit in any source code module that contains program units with calls to the library functions. You must also include SUDS.INC inside any program unit that contains calls to the library functions or references the structures.

Constants used by the library

There are several constants defined in the include files; SUDS.H for C and SUDS.INC for FORTRAN, that simplify using the library. It is recommended that you use these constants as opposed to the numeric value that they represent because these values may change in future versions of the library, the names of the constants will not. The constants are divided into several categories:

Error indication:

TRUE The value of this constant is returned by most I/O functions to indicate successful execution.

FALSE The value of this constant is returned by most I/O functions to indicate that an error has occurred. You should call SUDS_GET_ERR to retrieve the error message.

File access:

SUDS_READONLY The value of this constant is passed to SUDS_OPEN to indicate that the file should be opened with read only access.

SUDS_READWRITE The value of this constant is passed to SUDS_OPEN to indicate that the file should be opened with read and write access.

SUDS_APPEND The value of this constant is passed to SUDS_OPEN to indicate that the file should be opened with read and write access and the current position set at the end-of-file.

SUDS_CREATE The value of this constant is passed to SUDS_OPEN to indicate the file should be created and opened with write access. The current position is set to the beginning-of-file. If the file exists, it will be opened and its contents will be erased.

End-of-file:

SUDS_EOF The value of this constant is returned by the read and seek functions when the end of file is reached.

Structure type:

There are 32 of these constants defined (see Structure Type Constants in appendix B) These constants let you specify structure types using a symbolic name. The example code demonstrates the use of these constants.

SUDS_OPEN, Open file

C prototype:

```
int suds_open( char *filespec, int access );
```

Typical C usage:

```
#include <suds.h>

int fd;

if( ( fd = suds_open( "TEST.SUD", SUDS_READWRITE ) ) == FALSE ) {
   fprintf( stderr, "%s\n", suds_get_err( ) );
   return( FALSE );
}
```

Typical FORTRAN usage:

```
INCLUDE 'suds.fi'

PROGRAM TEST

INCLUDE 'suds.inc'

INTEGER*2 fd
CHARACTER buf*256

fd = SUDS_OPEN( 'TEST.SUD'//CHAR(0), SUDS_READONLY )
IF( fd .EQ. FALSE ) THEN
   buf = SUDS_GET_ERR( )
   WRITE( *, '(A,A)' ) ' ', buf(:INDEX(buf,CHAR(0)))
   STOP
ENDIF
```

This function opens the file specified in filespec with access specified using one of the file access constants described in SUDS structure type constants below. Up to 10 files may be open at any given time.

If successful, a valid file descriptor is returned. This file descriptor must be used to identify this particular file in latter I/O function calls.

If an error occurs, FALSE is returned and you should call suds_get_err( ) to retrieve a verbose description of the error.

See also: SUDS_CLOSE.

SUDS_CLOSE, Close file

C prototype:

```
int suds_close( int fd );
```

Typical C usage:

```
#include <suds.h>

int fd;

if( suds_close( fd ) == FALSE ) {
   fprintf( stderr, "%s\n", suds_get_err( ) );
   return( FALSE );
}
```

Typical FORTRAN usage:

```
INCLUDE 'suds.fi'

PROGRAM TEST

INCLUDE 'suds.inc'

INTEGER*2 fd
CHARACTER buf*256

IF( SUDS_CLOSE( fd ) .EQ. FALSE ) THEN
   buf = SUDS_GET_ERR( )
   WRITE( *, '(A,A)' ) ' ', buf(:INDEX(buf,CHAR(0)))
   STOP
ENDIF
```

This function flushes all system I/O buffers for, and closes the file specified by the file descriptor fd.

The function returns TRUE if successful, FALSE if an error occurred.

See also: SUDS_OPEN.

SUDS_READ, Read next structure

C prototype:

```
int suds_read( int fd, SUDS *suds );
```

Typical C usage:

```
#include <suds.h>

int fd, ret_val;
SUDS suds;

while( ( ret_val = suds_read( fd, &suds ) ) != SUDS_EOF ) {
   if( ret_val == FALSE ) {
      fprintf( stderr, "%s\n", suds_get_err( ) );
      return( FALSE );
   }
   /* Process structures based on type*/
   switch( suds.type ) {
      case STATIONCOMP:
         printf( "Channel = %d", suds.sc.channel );
         break;
   }
}
```

Typical FORTRAN usage:

```
INCLUDE ‘suds.fi’

PROGRAM TEST

INCLUDE ‘suds.inc’

INTEGER*2 fd, ret_val
CHARACTER buf*256

RECORD /suds_structs/ suds

100 ret_val = SUDS_READ( fd, LOC(suds) )
IF( ret_val .EQ. FALSE ) THEN
   buf = SUDS_GET_ERR( )
   WRITE( *, ‘(A,A)’ ) ‘ ‘, buf(:INDEX(buf,CHAR(0)))
   STOP
ENDIF

! Process structure based type
SELECT CASE( suds.type )
   CASE( STATIONCOMP )
      WRITE( *, ‘(A,I2)’ ) ‘ Channel = ‘, suds.sc.channel
END SELECT

IF( ret_val .NE. SUDS_EOF ) GOTO 100
```

This function reads the next structure from the file specified by the file descriptor fd. If the structure read with a previous call to SUDS_READ had data following it that was not read using SUDS_READ_DATA, the data is skipped and the next available structure is read.

The structure is read into the suds composite structure, the type of the structure is placed in the suds.type element and if any data follows the structure, its length is placed in the suds.data_len element. If the suds.data_len element contains 0, no data follows the structure.

The function returns TRUE if successful, FALSE if an error occurred and SUDS_EOF of the end-of-file was encountered.

See also: SUDS_READ_DATA, SUDS_WRITE, SUDS_WRITE_DATA.

SUDS_READ_DATA, Read data following a structure

C prototype:

```
int suds_read_data( int fd, void _huge *data, long length );
```

Typical C usage:

```
#include <suds.h>

int fd, ret_val;
void _huge *data;
SUDS suds;

// Read a structure
while( ( ret_val = suds_read( fd, &suds ) ) != SUDS_EOF ) {
   if( ret_val == FALSE ) {
      fprintf( stderr, "%s\n", suds_get_err( ) );
      return( FALSE );
   }
   // Does data follow?
   if( suds.data_len != 0 ) {
      // Allocate memory for data
      if( ( data = halloc( suds.data_len, 1) ) == NULL ) {
         fprintf( stderr, "ERROR: Not enough memory!\n" );
         return( FALSE );
      }
      // Read in the data
      if( ( ret_val = suds_read_data( fd, data, suds.data_len ) ) != SUDS_EOF )
{
         if( ret_val == FALSE ) {
            fprintf( stderr, "%s\n", suds_get_err( ) );
            return( FALSE );
         }
         // Process the data here

      }
   }
}
```

Typical FORTRAN usage:

```
INCLUDE 'suds.fi'
```

```
PROGRAM TEST

INCLUDE ‘suds.inc’

INTEGER*2 fd, ret_val,  ierr
INTEGER*2 data [ALLOCATABLE, HUGE] (:)
CHARACTER buf*256
RECORD /suds_structs/ suds

100 ret_val = SUDS_READ( fd, LOC(suds) )
IF( ret_val .EQ. FALSE ) THEN
   buf = SUDS_GET_ERR( )
   WRITE( *, ‘(A,A)’ ) ‘ ‘, buf(:INDEX(buf,CHAR(0)))
   STOP
ENDIF
IF( ret_val .NE. SUDS_EOF .AND. suds.data_len .NE. 0 ) THEN
   ALLOCATE( data(suds.data_len), STAT=ierr )
   IF( ierr .NE. 0 ) THEN
      WRITE( *, ‘(A)’ ) ‘ ERROR: Not enough memory!’
      STOP
   ENDIF
   ret_val = SUDS_READ_DATA( fd, LOC(data), suds.data_len )
   IF( ret_val .EQ. FALSE ) THEN
      buf = SUDS_GET_ERR( )
      WRITE( *, ‘(A,A)’ ) ‘ ‘, buf(:INDEX(buf,CHAR(0)))
      STOP
   ENDIF

   ! Process data here

ENDIF
```

This function reads data following the structure previously read by SUDS_READ. You may read all data at once as in the above example or if the data is too large, you may read it in smaller pieces by calling SUDS_READ_DATA multiple times. It is your responsibility to read between 0 and suds.data_len bytes (inclusive) from the file. If you read more than suds.data_len bytes, the file will be out of sync and the next operation on the file will fail.

Note that in the above FORTRAN example the data array is allocatable and uses the dynamic memory allocation routines available in Microsoft FORTRAN 5.0 or later. There is no requirement to handle this array in this manner, you may simple declare the array with a fixed number of elements and then read only that many elements from the file.

The function returns TRUE if successful, FALSE if an error occurred and SUDS_EOF of the end-of-file was encountered.

See also: SUDS_READ, SUDS_WRITE, SUDS_WRITE_DATA.

SUDS_WRITE, Write structure

C prototype:

```
int suds_write( int fd, SUDS *suds );
```

Typical C usage:

```
#include <suds.h>

int fd;
SUDS suds;

suds.type = STATIONCOMP;
suds_init( &suds );

// Place values in suds.sc elements here

if( suds_write( fd, &suds ) == FALSE ) {
   fprintf( stderr, "%s\n", suds_get_err( ) );
   return( FALSE );
}
```

Typical FORTRAN usage:

```
INCLUDE ‘suds.fi’

PROGRAM TEST

INCLUDE ‘suds.inc’

INTEGER*2 fd
CHARACTER buf*256

RECORD /suds_structs/ suds

suds.type = STATIONCOMP
CALL SUDS_INIT( LOC(suds) )

! Place values in suds.sc elements here

IF( SUDS_WRITE( fd, LOC(suds) ) .EQ. FALSE ) THEN
   buf = SUDS_GET_ERR( )
   WRITE( *, ‘(A,A)’ ) ‘ ‘, buf(:INDEX(buf,CHAR(0)))
   STOP
ENDIF
```

This function writes the structure in suds at the current position in the file specified by the file descriptor fd. If any data will follow the structure, its length in bytes must be placed in the suds.data_len element before calling SUDS_WRITE and SUDS_WRITE_DATA must be called to write exactly suds.data_len bytes before any other calls are made.

If SUDS_OPEN was called with SUDS_READWRITE access, the current position is at the top of the file, a call to SUDS_WRITE will overwrite everything from the current position to the end-of-file, placing the current position at the end-of-file. Any structures or data that followed the current position before the call to SUDS_WRITE is lost.

If SUDS_OPEN was called with SUDS_APPEND access, and no intervening seeks or rewinds have occurred, SUDS_WRITE will append the structure to the end of the file.

The function returns TRUE if successful and FALSE if an error occurred.

See also: SUDS_OPEN, SUDS_INIT, SUDS_WRITE_DATA.

SUDS_WRITE_DATA, Write data following a structure

C prototype:

```
int suds_write_data( int fd, void _huge *data, long length );
```

Typical C usage:

```
#include <suds.h>

int fd;
long length;
void _huge *data;
SUDS suds;

suds.type = DESCRIPTRACE;
suds_init( &suds );
suds.data_len = length;

// Place values in suds.dt elements here

// Write the structure
if( suds_write( fd, &suds )  == FALSE ) {
   fprintf( stderr, "%s\n", suds_get_err( ) );
   return( FALSE );
}
// Write out the data
if( suds_write_data( fd, data, suds.data_len ) == FALSE ) {
   fprintf( stderr, "%s\n", suds_get_err( ) );
   return( FALSE );
}
```

Typical FORTRAN usage:

```
INCLUDE ‘suds.fi’

PROGRAM TEST

INCLUDE ‘suds.inc’

INTEGER*2 fd
INTEGER*4 length
INTEGER*2 data [ALLOCATABLE, HUGE] (:)
CHARACTER buf*256
RECORD /suds_structs/ suds
```

```
suds.type = DESCRIPTRACE
CALL SUDS_INIT( LOC(suds) )
suds.data_len = length;

! Place values in suds.dt elements here

! Write out the structure
IF( SUDS_WRITE( fd, LOC(suds) ) .EQ. FALSE ) THEN
   buf = SUDS_GET_ERR
   WRITE( *, '(A,A)' ) ' ', buf(:INDEX(buf,CHAR(0)))
   STOP
ENDIF
! Write out the data
IF( SUDS_WRITE_DATA( fd, LOC(data), suds.data_len )
   buf = SUDS_GET_ERR
   WRITE( *, '(A,A)' ) ' ', buf(:INDEX(buf,CHAR(0)))
   STOP
ENDIF
```

This function writes data to follow a structure. This function works in conjunction with SUDS_WRITE must be used with care. This function must be called only after a call to SUDS_WRITE where suds.data_len is > 0. You must write exactly suds.data_len bytes to the file before any other I/O call for the file is made.

You may call SUDS_WRITE_DATA multiple times to write very large data items, just remember that you must write a total of suds.data_len bytes.

Note that in the above FORTRAN example the data array is allocatable and uses the dynamic memory allocation routines available in Microsoft FORTRAN 5.0 or later. There is no requirement to handle this array in this manner, you may simple declare the array with a fixed number of elements and then read only that many elements from the file.

The function returns TRUE if successful and FALSE if an error occurred.

See also: SUDS_WRITE, SUDS_READ_DATA, SUDS_FLUSH.

SUDS_FLUSH, Flush I/O buffers associated with a file

C prototype:

```
int suds_flush( int fd );
```

Typical C usage:

```
#include <suds.h>

int fd;

if( suds_flush( fd ) == FALSE ) {
   fprintf( stderr, "%s\n", suds_get_err( ) );
   return( FALSE );
}
```

Typical FORTRAN usage:

```
INCLUDE ‘suds.fi’

PROGRAM TEST

INCLUDE ‘suds.inc’

INTEGER*2 fd
CHARACTER buf*256

IF( SUDS_FLUSH( fd ) .EQ. FALSE ) THEN
   buf = SUDS_GET_ERR( )
   WRITE( *, ‘(A,A)’ ) ‘ ‘, buf(:INDEX(buf,CHAR(0)))
   STOP
ENDIF
```

This function flushes all system I/O buffers associated with the file specified by fd to disk. If a file is to be left open after a call to SUDS_WRITE or SUDS_WRITE_DATA, this function should be called in case the file should be left open due to some catastrophic failure (such as power loss or system crash) thus stranding unwritten data in memory and leaving the file in an unstable state.

The function returns TRUE if successful and FALSE if an error occurred.

See also: SUDS_WRITE, SUDS_WRITE_DATA.

SUDS_POS, SUDS_SEEK, Get and set position in file

C prototype:

```
long suds_pos( int fd );
int suds_seek( int fd, long position );
```

Typical C usage:

```
#include <suds.h>

int fd;
long position;

// Save current position
position = suds_pos( fd );

// Additional reads and/or writes here

// Return to saved position
if( suds_seek( fd, position ) == FALSE ) {
   fprintf( stderr, "%s\n", suds_get_err( ) );
   return( FALSE );
}
```

Typical FORTRAN usage:

```
INCLUDE ‘suds.fi’

PROGRAM TEST

INCLUDE ‘suds.inc’

INTEGER*2 fd
INTEGER*4 position
CHARACTER buf*256

! Save current position
position = SUDS_POS( fd )

! Additional reads and/or writes here

! Return to saved position
IF( SUDS_SEEK( fd, position ) .EQ. FALSE ) THEN
   buf = SUDS_GET_ERR( )
   WRITE( *, ‘(A,A)’ ) ‘ ‘, buf(:INDEX(buf,CHAR(0)))
   STOP
ENDIF
```

SUDS_POS returns the current position in the file specified by fd. The position is the offset from the beginning of the file in bytes, where the first byte is 0.

SUDS_SEEK sets the current position in the file specified by fd to the position given in position. position must contain a value returned from a call to SUDS_POS, otherwise the file may be out of sync.

SUDS_SEEK returns TRUE if successful, FALSE if an error occurred and SUDS_EOF if the end-of-file was encountered.

See also: SUDS_REWIND, SUDS_OPEN.

SUDS_REWIND, Rewind file

C prototype:

```
int suds_rewind( int fd );
```

Typical C usage:

```
#include <suds.h>

int fd;

if( suds_rewind( fd ) == FALSE ) {
   fprintf( stderr, "%s\n", suds_get_err( ) );
   return( FALSE );
}
```

Typical FORTRAN usage:

```
INCLUDE 'suds.fi'

PROGRAM TEST

INCLUDE 'suds.inc'

INTEGER*2 fd
CHARACTER buf*256

IF( SUDS_REWIND( fd ) .EQ. FALSE ) THEN
   buf = SUDS_GET_ERR( )
   WRITE( *, '(A,A)' ) ' ', buf(:INDEX(buf,CHAR(0)))
   STOP
ENDIF
```

This function sets the current position in the file specified by fd to the beginning of the file.

The function returns TRUE if successful and FALSE if an error occurred.

See also: SUDS_POS, SUDS_SEEK.

SUDS_INIT, Initialize structure elements to default values

C prototype:

```
void suds_init( SUDS *suds );
```

Typical C usage:

```
#include <suds.h>

SUDS suds;

suds.type = DESCRIPTRACE;
suds_init( &suds );

Typical FORTRAN usage:

INCLUDE 'suds.fi'

PROGRAM TEST

INCLUDE 'suds.inc'

RECORD /suds_structs/ suds

suds.type = DESCRIPTRACE
CALL SUDS_INIT( LOC(suds) )
```

This function fills all elements of the structure type specified in suds.type with default (void) values and sets suds.data_len to 0. If data will follow the structure, you must set suds.data_len after the call to SUDS_INIT.

In general, the void value placed in each element is -32767 for types; int, long, float and double and '_' for type char.

The function has no return value.

See also: SUDS_WRITE, SUDS_WRITE_DATA.

SUDS_GET_ERR, Get verbose error message

C prototype:

```
char *suds_get_err( void );
```

This function is used to retrieve a verbose error message (a NULL terminated ASCII string) after one of the I/O functions return an error condition. The message will always be less than 128 characters in length.

For examples of usage refer to any of the I/O function examples.

GET_MSTIME, Get current system date/time

C prototype:

```
double get_mstime( void );
```

Typical C usage:

```
#include <suds.h>

double mstime;

mstime = get_mstime( );
```

Typical FORTRAN usage:

```
INCLUDE 'suds.fi'

PROGRAM TEST

INCLUDE 'suds.inc'

REAL*8 mstime

mstime = GET_MSTIME( )
```

This function returns the current system date and time as MS_TIME. MS_TIME is the number of seconds that have elapsed since 1/1/70 00:00:00.0 stored as a double precision real number giving resolution better than a millisecond.

See also: MAKE_MSTIME, DECODE_MSTIME, LIST_MSTIME.

MAKE_MSTIME, Make MS_TIME for component parts

C prototype:

```
double make_mstime( int year, int month, int day, int hour,
                    int minute, double second );
```

Typical C usage:

```
#include <suds.h>

int year, month, day, hour, minute;
double mstime1, mstime2, second;

mstime1 = make_mstime( year, month, day, hour, minute, second );
mstime2 = make_mstime( 1992, 10, 20, 18, 32, 12.345 );
```

Typical FORTRAN usage:

```
INCLUDE 'suds.fi'

PROGRAM TEST

INCLUDE 'suds.inc'

INTEGER*2 year, day, hour, minute
REAL*8 mstime1, mstime2, second

mstime1 = MAKE_MSTIME( year, month, day, hour, minute, second )
mstime2 = MAKE_MSTIME( 1992, 10, 20, 18, 32, 12.345 )
```

This function returns the MS_TIME representation of the date and time specified as component parts. Note that year must include the century. This function performs the opposite operation of DECODE_MSTIME.

See also: GET_MSTIME, DECODE_MSTIME, LIST_MSTIME.

DECODE_MSTIME, Break MS_TIME into parts

C prototype:

```
int decode_mstime( double mstime, int *year, int *month, int *day,
                   int *hour, int *minute, double *second );
```

Typical C usage:

```
#include <suds.h>

int year, month, day, hour, minute;
double mstime, second;

if( decode_mstime( mstime, &year, &month, &day, &hour, &minute,
                   &second ) == FALSE ) {
   fprintf( stderr, "%s\n", suds_get_err( ) );
   return( FALSE );
}
```

Typical FORTRAN usage:

```
INCLUDE 'suds.fi'

PROGRAM TEST

INCLUDE 'suds.inc'

INTEGER*2 year, day, hour, minute
REAL*8 mstime, second
CHARACTER buf*256

IF( DECODE_MSTIME( mstime, year, month, day, hour, minute,
+                   second ) .EQ. FALSE ) THEN
   buf = SUDS_GET_ERR( )
   WRITE( *, '(A,A)' ) ' ', buf(:INDEX(buf,CHAR(0)))
   STOP
ENDIF
```

This function breaks the date/time specified in mstime down into its component parts. This function performs the opposite operation of MAKE_MSTIME.

The function returns TRUE if successful or FALSE if an error occurred.

See also: MAKE_MSTIME, GET_MSTIME, LIST_MSTIME.

LIST_MSTIME, Make ASCII string from MS_TIME

C prototype:

```
char *list_mstime( double mstime, int format );
```

Typical C usage:

```
#include <suds.h>

double mstime;

printf( "%s\n", list_mstime( mstime, 4 ) );
```

Typical FORTRAN usage:

```
INCLUDE ‘suds.fi’

PROGRAM TEST

INCLUDE ‘suds.inc’

REAL*8 mstime
CHARACTER*256 buf

! Note the string is NULL (CHAR(0)) terminated
buf = LIST_MSTIME( mstime, 4 )
WRITE( *, ‘(A,A)’ ) ‘ ‘, buf(:INDEX(buf,CHAR(0)))
```

This function returns a NULL terminated ASCII string containing the date and time specified in mstime in one of the following formats:

0	921025132958.900
1	921025132958
2	92 10 25 13 29 58.900
3	92 10 25 13 29 58
4	10/25/92 13:29 58.900
5	10/25/92 13:29 58
6	Oct 25, 1992 13:29 58.900 GMT
7	Oct 25, 1992 13:29 58 GMT
8	92 299 13:29:58.900
9	2AEAA156
10	92*299+13:29:58.900

Note that format 9 is simply the ST_TIME (32 bit integer seconds) representation of MS_TIME written as 8 hexadecimal digits. This representation is sometimes used for file naming.

See also: MAKE_MSTIME, GET_MSTIME, DECODE_MSTIME.

YRDAY, MNDAY, ISLEAP, Day-of-year to/from month, day

C prototypes:

```
int yrday( int month, int day, int leap );
int mnday( int doy, int leap, int *month, int *day );
int isleap( int year, in calendar );
```

Typical C usage:

```
#include <suds.h>

int year = 1992;
int month = 10;
int day = 25;
int doy;

// Get day-of-year from month and day
doy = yrday( month, day, isleap( year, 0 ) );

// Get month and day from day-of-year
mnday( doy, isleap( year, 0 ), &month, &day );
```

Typical FORTRAN usage:

```
INCLUDE 'suds.fi'

PROGRAM TEST

INCLUDE 'suds.inc'

INTEGER*2 year, month, day, doy, dummy

year = 1992
month = 10
day = 25

! Get day-of-year from month and day
doy = YRDAY( month, day, ISLEAP( year, 0 ) )

! Get month and day from day-of-year
dummy = MNDAY( doy, ISLEAP( year, 0 ), month, day )
```

YRDAY returns the day-of-year for the month, day and leap year indicator specified in month, day and the call to ISLEAP (see below.)

MNDAY returns the month and day for the day-of-year and leap year indicator specified in doy and the call to ISLEAP (see below.)

ISLEAP returns TRUE if year is a leap year, FALSE if not. calendar specifies the calendar in use; 0 = Gregorian or modern, 1 = Julian or old-style.

APPENDIX A

Minimum data requirement for PC-SUDS data files

This appendix describes the minimum elements that a PC-SUDS data file should contain to be considered a valid file by the PC-SUDS utilities and other components of the IASPEI software library,

A PC-SUDS data file containing digital waveform data should contain at least two structures for each waveform: SUDS_STATIONCOMP (SC) and SUDS_DESCRIPTRACE (DT) . The waveform data will follow the DT structure. The structure type declarations for these structures are presented below. Required elements are italicized. All unused elements should be initialized to "undefined" or "void" values. This is -32767 for both floating point and integer numbers, '_' (underscore) for characters, and "unknown" for strings. The I/O library provides the suds_init( ) function for this purpose.

The SUDS_STATIDENT Structure

Both the DT and SC structures have a nested structure named SUDS_STATIDENT (ST) that identifies the station / component. The ST structure is defined as follows:

```
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
} SUDS_STATIDENT;
```

ST.name: This element contains the name of the station. This can be up to four characters and should be NULL terminated. Typically, this is a three character station identifier with the fourth character as the component descriptor (v, n, or e) although this is not required. Each ST.name should be unique.

ST.component: This element contains a single character component descriptor. Most of the software recognizes v, n, and e for vertical, north, and east; u, t, and l for up, transverse, and longitudinal; and x, y, and z.

The SUDS_DESCRIPTRACE Structure

The DT structure is defined as follows:

```
typedef struct {
   SUDS_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 = 12 bit unsigned stored as short int, 0 to 4096,
                            // q = 12 bit signed stored as short int, -2048 to 2048,
                            // u = 16 bit unsigned stored as short, 0 to 65536
                            // i = 16 bit signed stored as short, -32767 to 32767,
                            // 2 = 24 bit signed integer stored as long,
                            // l = 32 bit signed integer stored as long,
                            // r = 12 bit data, 4 lsb time stored as short int,
                            // f = float (32 bit IEEE real),
                            // d = double (64 bit IEEE real),
                            // 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)
   LG_INT  numclip;         // number of clipped datapoints
   MS_TIME time_correct;    // time correction to be added to begintime
   FLOAT   rate_correct;    // rate correction to be added to rate
} SUDS_DESCRIPTRACE;
```

DT.name: This element must contain the elements described above for ST.

DT.begintime: This element contains the original uncorrected initial sample time (IST). This element should never be written to, any clock correction should be stored in DT.time_correct.

DT.datatype: This element contains the data type descriptor for the data the follows the structure.

DT.length: This element contains the number of samples that follow the structure. Note that this is the number of samples, not bytes.

DT.rate: This element contains the sampling rate in samples per seconds (Hz).

DT.mindata:
DT.maxdata:
DT.avenoise: These elements have been used by older software in the library to describe the minimum and maximum possible sample value and the zero level in the data. For this reason, you should interpret these fields as such to retain backward compatibility. For example, for 16 bit signed

data (DT.datatype=‘i’), DT.mindata would be -32767, DT.maxdata would be 32767, and DT.avenoise would be 0.

DT.time_correct: This element is used to store a correction to be applied to DT.begintime. This value is added to DT.begintime to get the corrected IST. If no clock correction is to be applied, DT.time_correct should be 0.0.

DT.rate_correct: This element is used to store a correction to be applied the DT.rate. This value is added to DT.rate to get the true sampling rate. If no correction is to be applied, DT.rate_correct should be 0.0.

The SUDS_STATIONCOMP Structure

The SC structure is defined as follows:

```
typedef struct {
   SUDS_STATIDENT sc_name;   // station component identification
   SH_INT  azim;             // component azimuth clockwise from north
   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, t=time code
   CHAR    data_type;        // see SUDS_DESCRIPTRACE.datatype
   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
                             // max_ground_motion=digital_sample*con_mvolts*
                             // max_gain
   SH_INT  channel;          // a2d channel number
   SH_INT  atod_gain;        // gain of analog to digital converter
   ST_TIME effective;        // date/time these values became effective
   FLOAT   clock_correct;    // clock correction in seconds.
   FLOAT   station_delay;    // seismological station delay.
} SUDS_STATIONCOMP;
```

SC.name: This element should be identical to its associated DT.name structure.

SC.data_type: This element should match DT.datatype.

SC.data_units: This element describes the unit of each sample. Typically this is ‘d’ (digital counts) for integer data types.

SC.channel: This element is the actual channel number on the instrument that the data came from.

SC.effective: This element should be equal to DT.begintime truncated to an integer.

APPENDIX B

The PC-SUDS structures

This appendix describes most of the currently defined PC-SUDS structures in C language syntax. The FORTRAN definitions are in SUDS.INC and should be referred to if necessary.

A typical PC-SUDS file will contain a SUDS_STATIONCOMP and SUDS_DESCRIPTRACE structure for each waveform and may contain other structures and data as needed (see appendix A). Most of the PC-SUDS utility software requires that a file contain at least these two structures for each waveform. Other commonly used structures include; SUDS_INSTRUMENT, SUDS_CHANSET, SUDS_FEATURE and SUDS_ORIGIN.

The order that the structures appear in the file is not significant and you should make no assumptions about structure order in your software.

Several sample SUDS files are provided on the diskette along with the SUD2ASC and ASC2SUD conversion programs. You may examine the contents of these or any SUDS file using the following command:

```
SUD2ASC filename
```

The contents of the file will be written to stdout (the screen) in verbose mode. This output may be written into a text file so that you may examine and/or edit it at your convenience. Simply type SUD2ASC or ASC2SUD at the command line to see a help screen.

Only commonly used SUDS structures are presented below. For a complete list of the SUDS structures see SUDS.H or SUDS.INC on the distribution diskette.

The SUDS composite structure

The following structure is used by the I/O library functions to pass structures read or written to/from the calling program. This structure is not written into the SUDS file. See SUDS_READ, SUDS_WRITE and SUDS_INIT for examples of use.

```
typedef struct _SUDS {
   int type;
   long data_len;
   union {
      SUDS_STATIDENT st;
      SUDS_ATODINFO ad;
      SUDS_CALIBRATION ca;
      SUDS_COMMENT co;
      SUDS_CHANSET cs;
      SUDS_DESCRIPTRACE dt;
      SUDS_DETECTOR de;
      SUDS_EQUIPMENT eq;
      SUDS_ERROR er;
      SUDS_EVENT ev;
      SUDS_EVENTSETTING es;
      SUDS_EVDESCR ed;
      SUDS_FEATURE fe;
      SUDS_FOCALMECH fo;
      SUDS_INSTRUMENT in;
      SUDS_LAYERS la;
      SUDS_LOCTRACE lo;
      SUDS_MOMENT mo;
      SUDS_MUXDATA mu;
      SUDS_ORIGIN or;
      SUDS_PROFILE pr;
      SUDS_RESIDUAL re;
      SUDS_SHOTGATHER sh;
      SUDS_STATIONCOMP sc;
      SUDS_TERMINATOR te;
      SUDS_TIMECORRECTION tc;
      SUDS_TRIGGERS tr;
      SUDS_TRIGSETTING ts;
      SUDS_VELMODEL vm;
   };
} SUDS;
```

Structure type constants

The following constants are used to identify structures read or written by the I/O library functions.

```
#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 EVENTSETTING    27
#define DETECTOR        28
#define ATODINFO        29
#define TIMECORRECTION  30
#define INSTRUMENT      31
#define CHANSET         32
```

Typedefs used in structure definitions

The following data types are used in the SUDS structure definitions. This was done for portability, clarity and to provide composite types.

```
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 unsigned char   BITS8;  // An 8 bit field
typedef unsigned short  BITS16; // A 16 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;
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 pos.
```

SUDS_STRUCTAG

This is the tag structure used to implement the linked list in SUDS files.

```
typedef struct {
   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
                              // ('6' = 80x86).
   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
} SUDS_STRUCTTAG;
```

SUDS_STATIDENT

This structure identifies a station and is usually nested with other SUDS structures.

```
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
} SUDS_STATIDENT;
```

SUDS_ATODINFO

This structure describes the A/D converter.

```
typedef struct {
   SH_INT  base_address;      // base I/O address of this device
   SH_INT  device_id;         // device identifier
   BITS16  device_flags;      // device flags
   SH_INT  extended_bufs;     // number of extended buffers used
   SH_INT  external_mux;      // AtoD external mux control word
   CHAR    timing_source;     // AtoD timing source: i=internal, e=external
   CHAR    trigger_source;    // AtoD trigger source: i=internal, e=external
} SUDS_ATODINFO;
```

SUDS_COMMENT

This structure is a comment tag to be followed by the formatted ASCII text of the comment.

```
typedef struct {
   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;
} SUDS_COMMENT;
```

SUDS_CHANSET

Associate station/components into sets.

```
typedef struct {
   SH_INT  type;             // Set type; 0=single channel(s), 1=orthogonal vector
   SH_INT  entries;          // Number of entries in set (these follow as data)
   STRING  network[4];       // Network name
   STRING  name[5];          // Set name
   ST_TIME active;           // Set is defined after this time
   ST_TIME inactive;         // Set is not defined after this time
} SUDS_CHANSET;

// Entries of this form follow SUDS_CHANSET struct.
typedef struct _CHANSETENTRY {
   LG_INT  inst_num;         // Instrument serial number
   SH_INT  stream_num;       // Stream of instrument
   SH_INT  chan_num;         // Channel of stream
   SUDS_STATIDENT st;        // Station/component identifier
} CHANSETENTRY;
```

SUDS_DESCRIPTRACE

This structure describes a waveform. The waveform data follows the structure.

```
typedef struct {
   SUDS_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 = 12 bit unsigned stored as short int, 0 to 4096,
                          // q = 12 bit signed stored as short int, -2048 to 2048,
                          // u = 16 bit unsigned stored as short, 0 to 65536
                          // i = 16 bit signed stored as short, -32767 to 32767,
                          // 2 = 24 bit signed integer stored as long,
                          // l = 32 bit signed integer stored as long,
                          // r = 12 bit data, 4 lsb time stored as short int,
                          // f = float (32 bit IEEE real),
                          // d = double (64 bit IEEE real),
                          // 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)
   LG_INT  numclip;          // number of clipped datapoints
   MS_TIME time_correct;     // time correction to be added to begintime
   FLOAT   rate_correct;     // rate correction to be added to rate
} SUDS_DESCRIPTRACE;
```

SUDS_DETECTOR

This structure contains information about the detector program being used.

```
typedef struct {
   CHAR    dalgorithm;       // triggering algorithm: x=xdetect, m=mdetect
```

```
                              // e=eqdetect
   CHAR    event_type;        // c=calibration, e=earthquake, E=explosion,
                              // f=free run, n=noise, etc.
   CHAR    net_node_id[10];   // network node identification
   FLOAT   versionnum;        // software version number
   LG_INT  event_number;      // unique event number assigned locally.
   LG_INT  spareL;            // spare
} SUDS_DETECTOR;
```

SUDS_FEATURE

This structure contains observed phase arrival time, amplitude, and period.

```
typedef struct {
   SUDS_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=ignore 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
   ST_TIME time_of_pick;      // time this pick was made
   SH_INT  pick_authority;    // organization processing the data
   SH_INT  pick_reader;       // person making this pick
} SUDS_FEATURE;
```

SUDS_INSTRUMENT

This structure contains instrument hardware settings, mainly PADS related.

```
typedef struct {
   SUDS_STATIDENT in_name;    // Station component identification
   SH_INT  in_serial;         // Instrument serial number
   SH_INT  comps;             // Number of components recorded by instrument
   SH_INT  channel;           // Actual channel number on recorder
   CHAR    sens_type;         // Sensor type; a=accel, v=vel, d=disp...
   CHAR    datatype;          // see SUDS_DESCRIPTRACE.datatype
   LG_INT  void_samp;         // Invalid or void sample value
   FLOAT   dig_con;           // Digitizing constant (counts / volt)
   FLOAT   aa_corner;         // Anti-alias filter corner frequency (Hz)
   FLOAT   aa_poles;          // Anti-alias filter poles
   FLOAT   nat_freq;          // Transducer natural frequency (Hz)
   FLOAT   damping;           // Transducer damping coeff.
   FLOAT   mot_con;           // Transducer motion constant (volts / GMU)
   FLOAT   gain;              // Amplifier gain (dB)
   FLOAT   local_x;           // Local coordinate X (meters)
   FLOAT   local_y;           // Local coordinate Y (meters)
   FLOAT   local_z;          // Local coordinate Z (meters)
   ST_TIME effective;        // Time these setting took effect
   FLOAT   pre_event;        // Pre-event length (IST+pre_event=trigger time)
   SH_INT  trig_num;         // Trigger number on instrument
   STRING  study[6];         // Study name, used to insure unique station names
   SH_INT  sn_serial;        // Sensor serial number
} SUDS_INSTRUMENT;
```

SUDS_MUXDATA

This structure contains a header for multiplexed data and is followed by the actual data.

```
typedef struct {
   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;         // see SUDS_DESCRIPTRACE.datatype
   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
   LG_INT  blocksize;        // number of demultiplexed samples per channel if
                             // data is partially demultiplexed, otherwise=0
} SUDS_MUXDATA;
```

SUDS_ORIGIN

This structure contains information about a specific solution for a given event.

```
typedef struct {
   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 program  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
} SUDS_ORIGIN;
```

SUDS_STATIONCOMP

This structure contains generic station component information.

```
typedef struct {
   SUDS_STATIDENT sc_name;   // station component identification
   SH_INT  azim;             // component azimuth clockwise from north
   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, t=time code
   CHAR    data_type;        // see SUDS_DESCRIPTRACE.datatype
   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
                             // max_ground_motion=digital_sample*con_mvolts*
                             // max_gain
   SH_INT  channel;          // a2d channel number
   SH_INT  atod_gain;        // gain of analog to digital converter
   ST_TIME effective;        // date/time these values became effective
   FLOAT   clock_correct;    // clock correction in seconds.
   FLOAT   station_delay;    // seismological station delay.
} SUDS_STATIONCOMP;
```

SUDS_TIMECORRECTION

This structure contains time correction information.

```
typedef struct {
   SUDS_STATIDENT tm_name;   // time trace station id used to determine
```

```
                              // correction.
   MS_TIME time_correct;      // time correction to be added to begintime
   FLOAT   rate_correct;      // rate correction to be added to rate
   CHAR    sync_code;         // synchronization code as follows:
                              //   0 = total failure,   1 = 1 second synch,
                              //   2 = 10 second synch, 3 = minute synch,
                              //   4, 5 = successful decode.
   CHAR    program;           // program used to decode time:
                              //   e = irige, c = irigc
   ST_TIME effective_time;    // time this correction was calculated
   SH_INT  spareM;
} SUDS_TIMECORRECTION;
```

SUDS_TRIGGERS

This structure contains earthquake detector trigger statistics.

```
typedef struct {
   SUDS_STATIDENT tr_name;    // station component identification
   SH_INT  sta;               // short term average
   SH_INT  lta;               // long term average; pre_lta for xdetect
   SH_INT  abs_sta;           // short term absolute average
   SH_INT  abs_lta;           // long term absolute average
   SH_INT  trig_value;        // value of trigger level (eta)
   SH_INT  num_triggers;      // number of times triggered during this event
   MS_TIME trig_time;         // time of first trigger
} SUDS_TRIGGERS;
```

SUDS_TRIGSETTING

This structure contains settings for earthquake trigger system.

```
typedef struct {
   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
   CHAR    algorithm;         // triggering algorithm: x=xdetect, m=mdetect
                              // e=eqdetect
   CHAR    spareJ;            // spare
   SH_INT  spareI;            // spare
} SUDS_TRIGSETTING;
```

APPENDIX C

A typical PC-SUDS data file

This appendix presents the contents of a typical PC-SUDS data file. The sample data has been abbreviated but all elements of the structures are there. This file contains a SUDS_INSTRUMENT structure in addition to the DT and SC structures described above to more fully describe each channel. It also contains a SUDS_CHANSET structure that associates the three channels into a set and several SUDS_FEATURE structures that contain phase picks.

This output was created by SUD2ASC. The SUD2ASC and ASC2SUD converters can be used together to edit PC-SUDS data files. Each structure starts with a single line header. This header starts with a dollar sign followed by the type number and the number of bytes of data that follows the structure. The next lines contain all elements of the structure followed by any data the follows the structure. Each structure and element is commented by the program. You may also add comments anywhere using a semi-colon as the delimiter.

```
$ 5 0 ; StationComp structure
unk                            ; network
BW14                           ; station name
v                              ; component
0                              ; instrument type
0                              ; component azimuth
0                              ; component incidence
-32767.000000                  ; latitude
-32767.000000                  ; longitude
-32767.000000                  ; elevation, meters
_                              ; enclosure
0                              ; annotated comment
_                              ; recorder type
_                              ; rock class
0                              ; rock type
_                              ; site condition
v                              ; sensor type
i                              ; data type
d                              ; data units
_                              ; polarity
0                              ; status
-32767.000000                  ; maximum gain
32767.000000                   ; clipping value
0.000000                       ; conversion to mvolts
4                              ; channel
8                              ; atod gain
```

```
01/18/94 15:25:00.000000       ; effective date
+0.000000                      ; clock correction
+0.000000                      ; station delay

$ 31 0 ; Instrument structure
unk                            ; network
BW14                           ; station name
v                              ; component
0                              ; instrument type
6149                           ; inst. serial number
3                              ; number of components
4                              ; channel number
v                              ; sensor type
i                              ; data type
-32767                         ; void sample value
8735.150391                    ; digitizing constant
-32767.000000                  ; AAF corner freq. (Hz)
-32767.000000                  ; AAF poles
-32767.000000                  ; trans natural freq. (Hz)
-32767.000000                  ; trans damping coeff.
-32767.000000                  ; trans motion constant
18.061800                      ; amplifier gain (dB)
-32767.000000                  ; local X coord. (meters)
-32767.000000                  ; local Y coord. (meters)
-32767.000000                  ; local Z coord. (meters)
01/18/94 15:25:00.000000       ; effective time
0.000000                       ; pre-event memory (secs)
1                              ; trigger number
Big W                          ; study ID
3784                           ; sensor serial number

$ 7 7300 ; DescripTrace structure
unk                            ; network
BW14                           ; station name
v                              ; component
0                              ; instrument type
01/18/94 15:25:00.000000       ; initial sample time
-32767                         ; local time diff
i                              ; data type
_                              ; data descriptor
0                              ; digitized by
0                              ; processed by
3650                           ; number of samples
10.000000                      ; samples per second
-32767.000000                  ; minimum data value
32767.000000                   ; maximum data value
0.000000                       ; average noise
-32767                         ; num clipped samples
+0.000000                      ; time correction
+0.000000                      ; rate correction
     0      0      0      0      0     -1      0      0     -1      0
     0      1      0      0      0     -1      1      0      1      1
     0      0      0      1      5      7      6      6      6      6
o
o
o
    -2     -2     -1     -1     -2     -1      0     -1      0      1
     1      1      1      1      1      1      0      1      1      0
     0      1      0      0      0      0     -1     -2     -1      0
```

```
$ 5 0 ; StationComp structure
unk                             ; network
BW15                            ; station name
n                               ; component
0                               ; instrument type
0                               ; component azimuth
90                              ; component incidence
-32767.000000                   ; latitude
-32767.000000                   ; longitude
-32767.000000                   ; elevation, meters
_                               ; enclosure
0                               ; annotated comment
_                               ; recorder type
_                               ; rock class
0                               ; rock type
_                               ; site condition
v                               ; sensor type
i                               ; data type
d                               ; data units
_                               ; polarity
0                               ; status
-32767.000000                   ; maximum gain
32767.000000                    ; clipping value
0.000000                        ; conversion to mvolts
5                               ; channel
8                               ; atod gain
01/18/94 15:25:00.000000        ; effective date
+0.000000                       ; clock correction
+0.000000                       ; station delay

$ 31 0 ; Instrument structure
unk                             ; network
BW15                            ; station name
n                               ; component
0                               ; instrument type
6149                            ; inst. serial number
3                               ; number of components
5                               ; channel number
v                               ; sensor type
i                               ; data type
-32767                          ; void sample value
8735.150391                     ; digitizing constant
-32767.000000                   ; AAF corner freq. (Hz)
-32767.000000                   ; AAF poles
-32767.000000                   ; trans natural freq. (Hz)
-32767.000000                   ; trans damping coeff.
-32767.000000                   ; trans motion constant
18.061800                       ; amplifier gain (dB)
-32767.000000                   ; local X coord. (meters)
-32767.000000                   ; local Y coord. (meters)
-32767.000000                   ; local Z coord. (meters)
01/18/94 15:25:00.000000        ; effective time
0.000000                        ; pre-event memory (secs)
1                               ; trigger number
Big W                           ; study ID
6231                            ; sensor serial number

$ 7 7300 ; DescripTrace structure
unk                             ; network
BW15                            ; station name
```

```
n                              ; component
0                              ; instrument type
01/18/94 15:25:00.000000       ; initial sample time
-32767                         ; local time diff
i                              ; data type
_                              ; data descriptor
0                              ; digitized by
0                              ; processed by
3650                           ; number of samples
10.000000                      ; samples per second
-32767.000000                  ; minimum data value
32767.000000                   ; maximum data value
0.000000                       ; average noise
-32767                         ; num clipped samples
+0.000000                      ; time correction
+0.000000                      ; rate correction
     0      0      1      0      1      1      0      0     -1      0
    -2      0      0     -2      0      1      0     -2      0      0
    -1      1      1      1      4      4      4      4      4      4
o
o
o
     2      1      2      2      3      1      0      0     -2     -1
    -1      0      2      1      3      2      1      0     -1     -2
    -3     -2     -1     -1      0      0      0      0      1      1

$ 5 0 ; StationComp structure
unk                            ; network
BW16                           ; station name
e                              ; component
0                              ; instrument type
90                             ; component azimuth
90                             ; component incidence
-32767.000000                  ; latitude
-32767.000000                  ; longitude
-32767.000000                  ; elevation, meters
_                              ; enclosure
0                              ; annotated comment
_                              ; recorder type
_                              ; rock class
0                              ; rock type
_                              ; site condition
v                              ; sensor type
i                              ; data type
d                              ; data units
_                              ; polarity
0                              ; status
-32767.000000                  ; maximum gain
32767.000000                   ; clipping value
0.000000                       ; conversion to mvolts
6                              ; channel
8                              ; atod gain
01/18/94 15:25:00.000000       ; effective date
+0.000000                      ; clock correction
+0.000000                      ; station delay

$ 31 0 ; Instrument structure
unk                            ; network
BW16                           ; station name
e                              ; component
```

```
0                              ; instrument type
6149                           ; inst. serial number
3                              ; number of components
6                              ; channel number
v                              ; sensor type
i                              ; data type
-32767                         ; void sample value
8735.150391                    ; digitizing constant
-32767.000000                  ; AAF corner freq. (Hz)
-32767.000000                  ; AAF poles
-32767.000000                  ; trans natural freq. (Hz)
-32767.000000                  ; trans damping coeff.
-32767.000000                  ; trans motion constant
18.061800                      ; amplifier gain (dB)
-32767.000000                  ; local X coord. (meters)
-32767.000000                  ; local Y coord. (meters)
-32767.000000                  ; local Z coord. (meters)
01/18/94 15:25:00.000000       ; effective time
0.000000                       ; pre-event memory (secs)
1                              ; trigger number
Big W                          ; study ID
5712                           ; sensor serial number

$ 7 7300 ; DescripTrace structure
unk                            ; network
BW16                           ; station name
e                              ; component
0                              ; instrument type
01/18/94 15:25:00.000000       ; initial sample time
-32767                         ; local time diff
i                              ; data type
_                              ; data descriptor
0                              ; digitized by
0                              ; processed by
3650                           ; number of samples
10.000000                      ; samples per second
-32767.000000                  ; minimum data value
32767.000000                   ; maximum data value
0.000000                       ; average noise
-32767                         ; num clipped samples
+0.000000                      ; time correction
+0.000000                      ; rate correction
     0      0      0      0      0     -1      0     -1      0      0
     1      0      0      4     -2     -1      0      0      0     -1
     1     -1      1      2      6      7      6      7      6      7
o
o
o
    -1      0      0     -1      0      1      1      0      2      3
     3      3      2      2      2      1     -1     -1     -1     -3
    -2     -2     -3     -3     -2      0     -1      0      2      1

$ 32 60 ; ChannelSet structure
1                              ; set type
3                              ; entries in set
                               ; network name
BW1                            ; set name
01/18/94 15:25:00.000000       ; active time
12/14/01 00:00:00.000000       ; inactive time
; Channel entry #1
```

```
6149                           ; Instrument number
1                              ; Stream number
4                              ; Channel number
unk                            ; Network ID
BW14                           ; Station ID
v                              ; Component ID
0                              ; Instrument type
; Channel entry #2
6149                           ; Instrument number
1                              ; Stream number
5                              ; Channel number
unk                            ; Network ID
BW15                           ; Station ID
n                              ; Component ID
0                              ; Instrument type
; Channel entry #3
6149                           ; Instrument number
1                              ; Stream number
6                              ; Channel number
unk                            ; Network ID
BW16                           ; Station ID
e                              ; Component ID
0                              ; Instrument type

$ 10 0 ; Feature structure
unk                            ; network
BW14                           ; station name
v                              ; component
0                              ; instrument type
50                             ; observed phase
i                              ; onset descriptor
d                              ; first motion
-32767                         ; signal/noise
i                              ; data source
1                              ; timing quality
_                              ; amplitude quality
_                              ; amplitude units
-32767                         ; gain range factor
01/18/94 15:25:21.610590       ; phase time
-32767.000000                  ; phase amplitude
-32767.000000                  ; phase period
10/21/94 12:12:49.000000       ; time of pick
0                              ; pick authority
-32767                         ; pick reader

$ 10 0 ; Feature structure
unk                            ; network
BW15                           ; station name
n                              ; component
0                              ; instrument type
100                            ; observed phase
e                              ; onset descriptor
n                              ; first motion
-32767                         ; signal/noise
i                              ; data source
2                              ; timing quality
_                              ; amplitude quality
_                              ; amplitude units
-32767                         ; gain range factor
01/18/94 15:27:04.785984       ; phase time
```

```
-32767.000000                  ; phase amplitude
-32767.000000                  ; phase period
10/21/94 12:12:49.000000       ; time of pick
0                              ; pick authority
-32767                         ; pick reader
```

APPENDIX D

Specifying Ground Motion Units in PC-SUDS Data Files

A typical PC-SUDS data contains digital waveform data stored as raw integer counts from an A/D converter. Typically, these are 12, 16, or 24 bit samples stored in signed integer words or double words. Most of the PC-SUDS utilities can deal with these data directly. PC-SUDS does provide data types for real samples, however most of the existing software tools do not support these types.

It is recommended that you store the waveform as raw samples in integer counts and derive ground motion units (GMU's) using constants provided in the SUDS structures as needed.

Appendix A describes the minimum data requirement for PC-SUDS data requirements. The SUDS_STATIONCOMP structure contains a field called `st.con_mvolts`. This field should contain the multiplier to convert the raw sample data to millivolts (mv per count) at unity gain. The `st.atod_gain` field contains the amplifier gain (magnification). Most data files will contain these two constants.

In order to compute GMU's, a SUDS_INSTRUMENT structure should be included in the data file for each waveform. The fields within this structure describe the instrument and sensor characteristics. The following fields should be provided.

The `in.sens_type` field is used to specify the ground motion unit measured by the sensor. Currently used values are:

Descriptor	Motion	Unit
'a'	Acceleration	cm/sec2
'v'	Velocity	cm/sec
'd'	Displacement	cm
'r'	Radial velocity	radians/sec

If you need to define other units, please let me know what they are so that I may coordinate their use among other PC-SUDS users.

The `in.dig_con` field should contain the digitizing constant (counts per volt). This is the counts per volt value of the A/D at unity gain.

The `in.gain` field should contain the amplifier gain specified as dB.

The `in.mot_con` field should contain the ground motion constant (volts per GMU). For example, if the sensor is a velocity transducer, this value should be volts per centimeter per second.

To convert raw samples to GMU's, multiply each sample by the following constant:

$$gmuConstant = \left(1 / \left(DigitizingConstant \times 10^{dBgain/20}\right)\right) / GroundMotionConstant$$

Index