U. S. DEPARTMENT OF THE INTERIOR

U. S. GEOLOGICAL SURVEY

DESCRIPTIONS OF SEISMIC ARRAY COMPONENTS: PART 3. SOFTWARE MODULES FOR DATA CONVERSION

Compiled by

W. H. K. Lee
MS 977, 345 Middlefield Road
Menlo Park, CA 94025

Open-File Report 92-598
August, 1992

INTRODUCTION

In the summer of 1990, funding was available to design and implement two portable seismic arrays for the volcano program. The approach was based on Lee et al. (1989). Several contracts were awarded to commercial companies to design and implement various components needed to build the portable arrays. The purpose of this report to present the software modules for data conversion purposes – DR2ST/ST2DR, GRM2ST/ST2GRM, PCEQ2ST/PCQ2ST, SEGY2ST, IO_RSDS, and IO_WRSDS, and two miscellaneous modules – PLT_PMTN, and SPR_PLFL – in detail as submitted by the contractors. Source code on PC-DOS/MS-DOS diskette for this report is presented in U. S. Geological Survey Open-File Report 92-598-B.

DR2ST/ST2DR (p. 4 - 24)

DR2ST/ST2DR are two computer programs for converting data from DR-100 (GEOS) data format (Borcherdt et. al.) to the SUDS data format (Ward, 1989; Banfill, 1992) and vice versa, respectively.

GRM2ST/ST2GRM (p. 25 - 46)

GRM2ST/ST2GRM are two computer programs for converting data from CUSP's GRM data format (Allan Walter, personal communication, 1992) to the SUDS data format (Ward, 1989; Banfill, 1992) and vice versa, respectively.

PCEQ2ST/PCQ2ST (p. 47 - 62)

PCEQ2ST/PCQ2ST are two computer programs for converting data from the PCEQ data format (Valdes, 1989) and from the PC-Quake data format (Tottingham et al., 1989) to the SUDS data format (Ward, 1989; Banfill, 1992), respectively.

SEGY2ST (p. 63 - 69)

SEGY2ST is a computer program for converting data from the SEGY data format (Barry et al., 1975) to the SUDS data format (Ward, 1989; Banfill, 1992).

MISC (p. 70 - 106)

The MISC directory contains four source code modules that may be added to the PITSA program package (Scherbaum and Johnson, 1992) in order to (1) read and write data files in the SUDS format (Ward, 1989; Banfill, 1992): IO_RDSDS.C and IO_WRSDS.C, (2) plot particle motion: PLT_PMTN, and (3) perform polarization filtering: SPR_PLFL.C.

REFERENCES

Banfill, R., (1992). SUDS: Seismic Unified Data System, version 1.31, *Small Systems Support, Big Water, Utah.*

Barry, K. M., D. A. Cavers, and C. W. Kneale, (1975). Recommended standards for digital tape format, *Geophysics, 40*, 344-352.

Borcherdt R.D., J. B. Fletcher, E. G. Jensen, G. L. Maxwell, J. R. Vanschaack, (1985). A general earthquake-observation system (GEOS) *Bull. Seism. Soc. Am., 75*, 1783-1825.

Lee, W. H. K., D. M. Tottingham, and J. O. Ellis (1989). Design and implementation of a PC-based seismic data acquisition, processing, and analysis system, *IASPEI Software Library, 1*, 21-46.

Scherbaum, F., and J. Johnson, (1992). "Programmable Interactive Toolbox for Seismological Analysis (PITSA)", *IASPEI Software Library, 5*, in preparation.

Tottingham, D. M., W. H. K. Lee, and J. A. Rogers, (1989). User manual for MDETECT, *IASPEI Software Library, 1*, 49-88.

Valdes, C. M., (1989). User manual for PCEQ, *IASPEI Software Library, 1*, 175-201.

Ward, P. L. (1989). SUDS: Seismic Unified Data System, *U. S. Geol. Surv. Open-file Report 89-188.*

DR2ST - DR100 (GEOS) to SUDS data converter version 1.00
ST2DR - SUDS to DR100 file converter version 1.00

Sun 16-Aug-1992 21:27, RB

These two programs convert data files between DR100 and SUDS format.

DR100 refers to the DR100 format as defined by the USGS for use with its GEOS instrument.

These programs were written using Microsoft C 6.00AX and the makefile provided is for the PWB.

Additional libraries required:
HSUDS.LIB - SUDS data file library version 1.31.
Available from:
Small Systems Support
2 Boston Harbor Place
Big Water, UT 84741-0205
(801) 675-5827 Voice
(801) 675-3730 FAX

Once built, these programs offer command line help by typing only the name of the executable (e.g., ST2DR) and pressing return.

DR2ST takes an "ordered arrival list" (.OAR) file as input and processes the DR100 "trigger files" contained in that list into a single SUDS data file. The user should avail his/her self to a program such as ORDARR (OAR.EXE, contained in USGS OFR 89-172) to manage these ordered arrival lists. This program allows the user to easily extract a subset of the data based on various criteria and automates the production of .OAR files.

ST2DR takes a SUDS data file and generates DR100 trigger files (1 per channel per event).

Source files included:

DR2ST	C	15948	03-09-92	5:14p
ST2DR	C	13480	07-21-92	7:47p
DR1HEAD	H	6001	07-19-92	3:54p <- DR100 header definitions
DR2ST	MAK	2329	07-07-92	4:42p
ST2DR	MAK	1733	07-21-92	7:47p

```
// DR2ST - DR100 to SUDS data file converter

// Mon 06-Jan-1992 12:17, RB

char version[] = "1.00";

#include <stdio.h>
#include <conio.h>
#include <stdlib.h>
#include <string.h>
#include <malloc.h>
#include <dos.h>
#include <ctype.h>

#include <suds.h>
#include "dr1head.h"

int read_dr_header( void );
int get_arrival( void );
int open_oar( void );
int open_suds( void );
int write_suds( void );
int suds_sti( void );
void close_files( void );
void s_upper( char * );

char curfile[_MAX_PATH], outfile[_MAX_PATH], oarfile[_MAX_PATH];
FILE *dr1fil, *stfil, *oarfil;
char stn[5], comp, buf[72];
int arrivals, k, eof_flag = 0, verbose = 1;
char comps[6][34];

extern char *progname;

//*************************************************************************
main( int argc, char *argv[] ) {
   register i, j;
   int ierr, first = 1;
   char buf[_MAX_PATH], *p;

   oarfile[0] = '\0';
   outfile[0] = '\0';

   printf( "DR2ST - DR100 to SUDS data file converter - Version %.4s\n", version );
   printf( "Copyright (c) Robert Banfill 1992. All rights reserved.\n\n" );

   // Help
   if( argv[1][0] == '?' || argv[1][1] == '?' ) {
      printf( "Usage: DR2ST [switches] [inputfile] [outputfile]\n\n" );
      printf( "Switches:\n" );
      printf( "   /Q = Quiet, minimum status messages (Verbose).\n\n" );
      printf( "   inputfile  = Ordered Arrival List (UNDEFINE.OAR)\n" );
      printf( "   outputfile = SUDS output file (Hex IST of 1st arrival with .DMX exte
nsion).\n\n" );
      printf( "() indicates default, [] indicates optional arguments.\n" );
      exit( 1 );
   }

   progname = argv[0];

   // Parse the command-line
   for( i=1; i<argc; i++ ) {
```

```
      if( argv[i][0] == '/' || argv[i][0] == '-' ) {
         switch( argv[i][1] ) {
            case 'q':
            case 'Q':
               verbose = 0;
               break;
         }
      }
      else if( oarfile[0] == '\0' ) {
         strcpy( oarfile, argv[i] );
         if( ( p = strchr( oarfile, '.' ) ) == NULL )
            strcat( oarfile, ".OAR" );
      }
      else if( outfile[0] == '\0' ) {
         strcpy( outfile, argv[i] );
         if( ( p = strchr( outfile, '.' ) ) == NULL )
            strcat( outfile, ".DMX" );
      }
      else
         fprintf( stderr, "WARNING: Unrecognized argument: %s\n", argv[i] );
   }

   if( oarfile[0] == '\0' )
      strcpy( oarfile, "UNDEFINE.OAR" );

   _fullpath( buf, oarfile, _MAX_PATH );
   strcpy( oarfile, buf );

   s_upper( oarfile );

   if( verbose )
      printf( "Input file: %s\n\n", oarfile );

   // Open the Ordered arrival list
   if( open_oar( ) != 0 )
      exit( 1 );

   printf( "Processing %d arrivals\n", arrivals );
   if( verbose )
      printf( "\n" );

   for( i=1; i<=arrivals; i++ ) {

      // Get an arrival
      ierr = get_arrival( );
      if( ierr < 0 )
         exit( 1 );
      else if( ierr > 0 )
         break;

      // Process each component
      for( j=1; j<=k; j++ ) {
         strcpy( curfile, comps[j-1] );

         if( verbose )
            printf( "   %s", curfile );
         else
            printf( "%s                              \r", curfile );

         // Open the current DR100 file and read the header
         if( read_dr_header( ) != 0 )
            exit( 1 );
```

```
         if( first ) {
            // Open the SUDS output file after reading first header (for time)
            open_suds( );
            first = 0;
         }

         // Output station related data to SUDS file
         if( suds_sti( ) != 0 )
            exit( 1 );

         // Output trace to SUDS file
         if( write_suds( ) != 0 )
            exit( 1 );
      }
   }

   // Close SUDS and OAR file
   close_files( );

   exit( 0 );
}

//*****************************************************************************
int get_arrival( void ) {
   register i;
   char newname[34], *p, *q, hr[3];
   struct find_t fi;

   // Clear out component array
   memset( comps, '\0', sizeof( comps ) );
   memset( hr, '\0', 3 );

   // Trim off trailing crap
   if( ( p = strchr( buf, ' ' ) ) != NULL )
      *p = '\0';
   else
      buf[32] = '\0';

   if( verbose )
      printf( "Processing arrival: %s\n", buf );

   // Build new name
   strcpy( newname, buf );
   p = strrchr( newname, '\\' )+4;
   strncpy( hr, p, 2 );
   *p = 'A'+atoi( hr );
   p++;
   strcpy( p, p+1 );

   // Decompose into components
   p = strchr( newname, '.' )-1;
   switch( *p ) {
      case 'A':
         for( i=0; i<=2; i++ ) {
            strcpy( comps[i], newname );
            q = strchr( comps[i], '.' )-1;
            *q = '0'+i+1;
         }
         k = 3;
         break;
      case 'V':
         for( i=0; i<=2; i++ ) {
            strcpy( comps[i], newname );
```

```
            q = strchr( comps[i], '.' )-1;
            *q = '0'+i+4;
         }
         k = 3;
         break;
      case 'D':
         for( i=0; i<=2; i++ ) {
            strcpy( comps[i], newname );
            q = strchr( comps[i], '.' )-1;
            *q = '0'+i+7;
         }
         k = 3;
         break;
      case 'G':
         for( i=0; i<=5; i++ ) {
            strcpy( comps[i], newname );
            q = strchr( comps[i], '.' )-1;
            *q = '0'+i+1;
         }
         k = 6;
         break;
      default:
         strcpy( comps[0], newname );
         k = 1;
         break;
   }

   if( !eof_flag ) {
      // Read next "Arrival filespec" from the OAR file
      if( fgets( buf, 73, oarfil ) == NULL ) {
         if( feof( oarfil ) )
            eof_flag = 1;
         else {
            fprintf( stderr, "\nERROR: Read error in: %s\n", oarfile );
            return( -1 );
         }
      }
   }
   else
      return( 1 );

   // Check that component files exist
   for( i=0; i<k; i++ ) {
      if( _dos_findfirst( comps[i], _A_NORMAL, &fi ) != 0 ) {
         fprintf( stderr, "WARNING: Missing component: %s\n", comps[i] );
         comps[i][0] = '\0';
      }
   }

   return( 0 );
}

//*****************************************************************
int open_oar( void ) {
   register i;

   // Open the Ordered Arrival List
   if( ( oarfil = fopen( oarfile, "r" ) ) == NULL ) {
      fprintf( stderr, "\nERROR: Unable to open: %s\n", oarfile );
      return( -1 );
   }

   // Read .OAR header data (line beginning with "|")
```

```
   arrivals = 0;
   do {
      if( fgets( buf, 73, oarfil ) == NULL ) {
         if( feof( oarfil ) ) {
            fprintf( stderr, "\nERROR: Premature end-of-file: %s\n", oarfile );
            return( -1 );
         }
         else {
            fprintf( stderr, "\nERROR: Read error in: %s\n", oarfile );
            return( -1 );
         }
      }
      if( strncmp( buf, "|Included Arrivals =", 20 ) == 0 )
         arrivals = atoi( &buf[21] );
   } while( buf[0] == '|' );

   // Check that we have some arrivals
   if( arrivals == 0 ) {
      fprintf( stderr, "\nERROR: No arrivals in: %s\n", oarfile );
      return( -1 );
   }

   // Stream pointer at first DR100 filespec
   return( 0 );
}


//*************************************************************************
int read_dr_header( void ) {

   // Open the DR100 input file
   if( ( dr1fil = fopen( curfile, "rb" ) ) == NULL ) {
      fprintf( stderr, "\nERROR: Unable to open: %s\n", curfile );
      return( -1 );
   }

   // Read in the integer header
   if( fread( &ihdr, 1, sizeof( ihdr ), dr1fil ) != sizeof( ihdr ) ) {
      fprintf( stderr, "\nERROR: Reading integer header: %s\n", curfile );
      return( -1 );
   }

   // Skip any additional integer "Bullshit blocks"
   if( ihdr.ex_int != 0 ) {
      if( fseek( dr1fil, (long)ihdr.ex_int*512L, SEEK_CUR ) != 0 ) {
         fprintf( stderr, "\nERROR: Unable to seek record: %s\n", curfile );
         return( -1 );
      }
   }

   // Skip any additional ASCII "Bullshit blocks"
   if( ihdr.ex_asc != 0 ) {
      if( fseek( dr1fil, (long)ihdr.ex_asc*512L, SEEK_CUR ) != 0 ) {
         fprintf( stderr, "\nERROR: Unable to seek record: %s\n", curfile );
         return( -1 );
      }
   }

   // Read in the real header
   if( fread( &rhdr, 1, sizeof( rhdr ), dr1fil ) != sizeof( rhdr ) ) {
      fprintf( stderr, "\nERROR: Reading real header: %s\n", curfile );
      return( -1 );
   }
```

```
   // Skip any additional real "Bullshit blocks"
   if( (int)rhdr.ex_flt != 0 ) {
      if( fseek( drlfil, (long)rhdr.ex_flt*512L, SEEK_CUR ) != 0 ) {
         fprintf( stderr, "\nERROR: Unable to seek record: %s\n", curfile );
         return( -1 );
      }
   }

   // input file left open, stream pointer positioned at first data sample
   return( 0 );
}

//*************************************************************************
int open_suds( void ) {
   int mon, day;
   char buf[_MAX_PATH];
   SUDS_DETECTOR sd;

   // If we dont have an explicit filespec, use hex IST as filename
   if( outfile[0] == '\0' ) {
      mnday( ihdr.day, 1, &mon, &day );
      sprintf( outfile, "%s.DMX", list_mstime( make_mstime( ihdr.yr+1900,
                     mon, day, ihdr.hr, ihdr.min, (double)ihdr.sec ), 9 ) );
   }

   _fullpath( buf, outfile, _MAX_PATH );
   strcpy( outfile, buf );

   s_upper( outfile );

   // Open output file
   stfil = st_open( outfile, "w+b" );

   // DETECTOR struct
   st_init( DETECTOR, &sd );
   strcpy( sd.net_node_id, "DR2ST" );
   sscanf( version, "%f", &sd.versionnum );
   st_put( &sd, DETECTOR, sizeof( SUDS_DETECTOR ), stfil );
   st_flush( stfil );

   return( 0 );
}

//*************************************************************************
int write_suds( void ) {
   int mon, day, blk;
   long bytes;
   double sec;
   char _huge *ptr;
   char _huge *ptr1;
   SUDS_DESCRIPTRACE _huge *dt;

   // Check if block size is defined
   if( ihdr.rec_bytes == ihdr.i_null )
      ihdr.rec_bytes = 512;

   // Allocate trace buffer
   bytes = ((long)(ihdr.num_recs-1)*(long)ihdr.rec_bytes)+
           (long)(ihdr.l_samp*2)+(long)sizeof( SUDS_DESCRIPTRACE );
   if( ( ptr = (char _huge *)halloc( bytes, sizeof(char) ) ) == NULL ) {
      fprintf( stderr, "\nERROR: Not enough memory!\n" );
      return( -1 );
```

```
   }
   dt = (SUDS_DESCRIPTRACE _huge *)ptr;

   // Init the struct
   st_init( DESCRIPTRACE, dt );

   // Stuff the header values
   strcpy( dt->dt_name.st_name, stn );
   dt->dt_name.component = comp;
   mnday( ihdr.day, 1, &mon, &day );
   sec = (double)(ihdr.sec)+((double)ihdr.msec*.001)+((double)ihdr.usec*0.000001);
   dt->begintime = (MS_TIME)make_mstime( ihdr.yr+1900, mon, day, ihdr.hr, ihdr.min, se
c );
   if( ihdr.d_type > 0 )
      dt->datatype = 'f';
   else
      dt->datatype = 'i';
   dt->length = (((long)(ihdr.num_recs-1)*(long)ihdr.rec_bytes)/2)+
                (long)ihdr.l_samp;
   dt->rate = rhdr.rate;
   dt->mindata = -32767;
   dt->maxdata = 32767;
   dt->avenoise = 0;
   if( ihdr.p_flag )
      dt->time_correct = -(MS_TIME)rhdr.clk_cor;
   else
      dt->time_correct = 0.0;
   dt->rate_correct = 0.0;

   if( verbose )
      printf( ", %ld samples, IST=%s\n", dt->length, list_mstime( dt->begintime, 4 ) )
;

   // Init a pointer to first sample
   ptr1 = ptr+sizeof(SUDS_DESCRIPTRACE);

   // Read in the full blocks of data
   for( blk=1; blk<=ihdr.num_recs-1; blk++ ) {
      if( fread( ptr1, sizeof(char), ihdr.rec_bytes, dr1fil ) != ihdr.rec_bytes ) {
         fprintf( stderr, "\nERROR: Reading sample data: %s\n", curfile );
         return( -1 );
      }
      ptr1+=ihdr.rec_bytes;
   }
   // Read in the last block
   if( fread( ptr1, sizeof(int), ihdr.l_samp, dr1fil ) != ihdr.l_samp ) {
      fprintf( stderr, "\nERROR: Reading sample data: %s\n", curfile );
      return( -1 );
   }

   // Write it and flush it
   st_put( dt, DESCRIPTRACE, bytes, stfil );
   st_flush( stfil );

   // Free memory
   hfree( ptr );

   // Close the current input file
   fclose( dr1fil );

   return( 0 );
}
```

```
//*********************************************************************
void close_files( void ) {

   st_flush( stfil );
   st_close( stfil );

   fclose( oarfil );

   if( verbose )
      printf( "Output written to: %s\n", outfile );

   return;
}

//*********************************************************************
int suds_sti( void ) {
   char *p;
   int mon, day;
   SUDS_STATIONCOMP sc;
   SUDS_INSTRUMENT in;

   // Extract station name from current input filename
   strnset( stn, '\0', 5 );
   p = strchr( curfile, '.' )+1;
   strncpy( stn, p, 3 );

   // Stuff a station component and instrument struct with header values
   st_init( STATIONCOMP, &sc );
   st_init( INSTRUMENT, &in );

   switch( ihdr.com_num ) {
      case 1:
      case 4:
      case 7:
         sc.sc_name.component = 'v';
         in.in_name.component = 'v';
         comp = 'v';
         break;
      case 2:
      case 5:
      case 8:
         sc.sc_name.component = 'n';
         in.in_name.component = 'n';
         comp = 'n';
         break;
      case 3:
      case 6:
      case 9:
         sc.sc_name.component = 'e';
         in.in_name.component = 'e';
         comp = 'e';
         break;
   }
   strcpy( sc.sc_name.st_name, stn );
   strcpy( in.in_name.st_name, stn );

   switch( ihdr.mot_type ) {
      case 1:
         sc.sensor_type = 'a';
         in.sens_type = 'a';
         break;
      case 2:
         sc.sensor_type = 'v';
```

```
            in.sens_type = 'v';
            break;
         case 3:
            sc.sensor_type = 'd';
            in.sens_type = 'd';
            break;
      }
      if( ihdr.d_type > 0 ) {
         sc.data_type = 'f';
         in.datatype = 'f';
      }
      else {
         sc.data_type = 'i';
         in.datatype = 'i';
      }

      sc.data_units = 'd';
      sc.clip_value = 32767.0;
      sc.channel = ihdr.com_num;

      if( ihdr.p_flag ) {
         sc.st_lat = (LONLAT)rhdr.lat;
         sc.st_long = (LONLAT)rhdr.lon;
         sc.elev = rhdr.elev;
         sc.azim = ihdr.sen_horz;
         sc.incid = ihdr.sen_vert;
         sc.con_mvolts = rhdr.dig_con;
         sc.clock_correct = rhdr.com_lag;
      }

      mnday( ihdr.day, 1, &mon, &day );
      sc.effective = (ST_TIME)make_mstime( ihdr.yr+1900, mon, day, ihdr.hr,
                     ihdr.min, (double)ihdr.sec );
      in.effective = sc.effective;

      in.in_serial = ihdr.ser_num;
      in.comps = ihdr.tot_com;
      in.channel = ihdr.com_num;
      in.void_samp = (long)ihdr.i_null;
      in.trig_num = ihdr.seq_num;
      if( isalpha( ihdr.study[0] ) || isdigit( ihdr.study[0] ) )
         strncpy( in.study, ihdr.study, 6 );
      if( ihdr.sen_ser_num == ihdr.i_null )
         in.sn_serial = 0;
      else
         in.sn_serial = ihdr.sen_ser_num;
      if( ihdr.pre_event == ihdr.i_null )
         in.pre_event = 0;
      else
         in.pre_event = (float)ihdr.pre_event*10.0;

      if( ihdr.p_flag ) {
         in.dig_con = rhdr.dig_con;
         in.aa_corner = rhdr.aa_corner;
         in.aa_poles = rhdr.aa_poles;
         in.nat_freq = rhdr.nat_freq;
         in.damping = rhdr.damp_coe;
         in.mot_con = rhdr.coil_con;
         in.gain = rhdr.gain;
         in.local_x = rhdr.loc_x;
         in.local_y = rhdr.loc_y;
         in.local_z = rhdr.loc_z;
      }
```

```
   // Write it and flush it
   st_put( &sc, STATIONCOMP, sizeof( SUDS_STATIONCOMP ), stfil );
   st_flush( stfil );
   st_put( &in, INSTRUMENT, sizeof( SUDS_INSTRUMENT ), stfil );
   st_flush( stfil );


   return( 0 );
}

//*****************************************************************************
long die( int in ) {
   exit(in);
}

//*****************************************************************************
void s_upper( char *buffer ) {
   char *p;
   for( p = buffer; *p; p++ )
      *p = toupper( *p );
}
```

```
// ST2DR - SUDS to DR100 Conversion

// R. Banfill 21-Jul-1992

char version[] = "1.00";

#include <stdio.h>
#include <conio.h>
#include <stdlib.h>
#include <string.h>
#include <malloc.h>
#include <ctype.h>
#include <search.h>
#include <math.h>

#include <suds.h>
#include "dr1head.h"

#define INULL -32768
#define RNULL -1e+30

typedef struct _DB {
   ST_TIME effective;    // Time stamp
   char  stn[5];         // sc Station ID
   char  study[6];       // in Study name
   char  comp;           // sc Component
   int   incid;          // sc Component incidence
   int   azim;           // sc Component azimuth
   float lat;            // sc Station latitude
   float lon;            // sc Station longitude
   float elev;           // sc Station elevation
   int   void_samp;      // in Void sample value
   int   gain;           // sc A/D gain (magnification)
   int   chan;           // sc Channel number
   float bcv;            // sc Bit count volts (mv/count)
   float samp_lag;       // sc Serial sampling lag
   int   i_serial;       // in Instrument serial number
   int   s_serial;       // in Sensor serial number
   int   n_comps;        // in Number of components recorded
   float pre_evn;        // in Pre-event length
   int   trig_num;       // in Trigger number
} DB;

int build_db( void );
int st2dr( void );
void s_upper( char * );
int lookup( ST_TIME time, char *stn, int add );
double x2db( double x );
void dump_db( char *file );
int compare ( DB *rec1, DB *rec2 );

DB *db;
DB *rec;
int nrecs;

char sudsfspec[_MAX_PATH], dr1fspec[_MAX_PATH];
FILE *sudsfil, *dr1fil;
int verbose = 1;
int debug = 0;
extern char *progname;

//---------------------------------------------------------------------
main( int argc, char *argv[] ) {
```

```
   register i;

   printf( "ST2DR - SUDS to DR100 data file converter - Version %.4s\n", version );
   printf( "Copyright (c) Robert Banfill 1992. All rights reserved.\n\n" );

   // Help
   if( argv[1][0] == '?' || argv[1][1] == '?' ) {
      printf( "Usage: ST2DR [switches] [inputfile]\n\n" );
      printf( "Switches:\n" );
      printf( "    /Q = Quiet, minimum status messages (Verbose).\n" );
      printf( "    /D = Debug, dump database to ST2DR.DB (Off).\n\n" );
      printf( "    inputfile  = SUDS 1.3x data file.\n\n" );
      printf( "() indicates default, [] indicates optional arguments.\n" );
      exit( 1 );
   }

   progname = argv[0];
   sudsfspec[0] = '\0';

   // Parse the command-line
   for( i=1; i<argc; i++ ) {
      if( argv[i][0] == '/' || argv[i][0] == '-' ) {
         switch( argv[i][1] ) {
            case 'q':
            case 'Q':
               verbose = 0;
               break;
            case 'd':
            case 'D':
               debug = 1;
               break;
            default:
               fprintf( stderr, "WARNING: Unrecognized command-line argument: %s\n", a
rgv[i] );
               break;
         }
      }
      else {
         _fullpath( sudsfspec, argv[i], _MAX_PATH );
         s_upper( sudsfspec );
      }
   }


   if( verbose )
      printf( "Input filespec: %s\n\n", sudsfspec );

   sudsfil = st_open( sudsfspec, "r+b" );

   if( ! build_db( ) )
      exit( 1 );

   if( ! st2dr( ) )
      exit( 1 );

   st_close( sudsfil );

   if( ! verbose )
      printf( "\n" );

   exit( 0 );
}
```

```
//-----------------------------------------------------------------------
int st2dr( void ) {
   SH_INT typ;
   LG_INT inp, len;
   CHAR _huge *ptr;

   st_rewind( sudsfil );


   while( ( inp = st_get( &(void _huge *)ptr, &typ, &len, sudsfil ) ) != EOF ) {
      if( typ == DESCRIPTRACE ) {
         if( ! write_dr( ptr ) )
            return( 0 );
      }
      st_free( ptr, inp );
   }
   return( 1 );
}

//-----------------------------------------------------------------------
int write_dr( char _huge *ptr ) {
   register i;
   char comp, buf[_MAX_PATH];
   int j, *iptr, yr, mn, dy, hr, mi, jd, found;
   float *fptr;
   double sc;
   SH_INT _huge *data;
   SUDS_DESCRIPTRACE _huge *dt;

   dt = (SUDS_DESCRIPTRACE _huge *)ptr;
   data = (SH_INT _huge *)(dt+1);

   // Build DR100 filename
   decode_mstime( dt->begintime, &yr, &mn, &dy, &hr, &mi, &sc );
   jd = yrday( mn, dy, isleap( yr, 0 ) );
   sprintf( dr1fspec, "%3.3d%c%2.2d%c%c.%.3s", jd, 'A'+hr, mi,
            'A'+((int)sc/3), dt->dt_name.st_name[3], dt->dt_name.st_name );

   if( verbose )
      printf( "%s -> %s\n", dt->dt_name.st_name, dr1fspec );
   else
      printf( "\r%s", dr1fspec );

   // Clear headers
   iptr = &ihdr.ex_int;
   for( i=0; i<=255; i++ ) {
      *iptr = INULL;
      iptr++;
   }
   fptr = &rhdr.ex_flt;
   for( i=0; i<=127; i++ ) {
      *fptr = RNULL;
      fptr++;
   }

   // Stuff the DR100 headers
   ihdr.inst_type = 2;
   ihdr.rec_sys = 4;
   ihdr.loc_num = 1;

   ihdr.ex_int = 0;
   ihdr.ex_asc = 0;
   ihdr.i_null = INULL;
```

```
   ihdr.d_type = INULL;
   ihdr.p_flag = 1;

   ihdr.yr = yr-1900;
   ihdr.day = jd;
   ihdr.hr = hr;
   ihdr.min = mi;
   ihdr.sec = (int)sc;
   ihdr.msec = (int)((sc-(int)sc)*1000.0);
   ihdr.usec = 0;

   ihdr.num_recs = (int)((dt->length+255L) / 256L);
   ihdr.l_samp = (int)(dt->length % 256L);
   ihdr.rec_bytes = 512;
   if( dt->length <= 32768 )
      ihdr.samps = (int)dt->length;

   strcpy( ihdr.filename, dr1fspec );

   rhdr.ex_flt = 0.0;
   rhdr.r_null = RNULL;

   rhdr.rate = dt->rate;

   if( lookup( (ST_TIME)dt->begintime, dt->dt_name.st_name, 0 ) ) {
      ihdr.sen_vert = rec->incid;
      ihdr.sen_horz = rec->azim;
      ihdr.com_num = rec->chan;
      ihdr.tot_com = rec->n_comps;
      ihdr.sen_ser_num = rec->s_serial;
      ihdr.ser_num = rec->i_serial;
      strcpy( ihdr.study, rec->study );
      ihdr.pre_event = (int)(rec->pre_evn*10.0);
      ihdr.seq_num = rec->trig_num;
      ihdr.mot_type = 2;

      rhdr.com_lag = rec->samp_lag;
      rhdr.lat = rec->lat;
      rhdr.lon = rec->lon;
      rhdr.elev = rec->elev;
      rhdr.dig_con = 1/(rec->bcv*(float)rec->gain*0.001);
      rhdr.gain = (float)x2db( rec->gain );
   }
   else {
      ihdr.com_num = dt->dt_name.st_name[3]-'0';
      ihdr.tot_com = 6;
      ihdr.mot_type = 2;
      switch( ihdr.com_num ) {
         case 1:
         case 4:
         case 7:
            ihdr.sen_vert = 0;
            ihdr.sen_horz = 0;
            break;
         case 2:
         case 5:
         case 8:
            ihdr.sen_vert = 90;
            ihdr.sen_horz = 0;
            break;
         case 3:
         case 6:
         case 9:
```

```
            ihdr.sen_vert = 90;
            ihdr.sen_horz = 90;
            break;
      }
      rhdr.com_lag = 0;
      rhdr.dig_con = 3276.8;
   }

   // Open the file
   if( ( dr1fil = fopen( dr1fspec, "w+b" ) ) == NULL ) {
      fprintf( stderr, "\nERROR: unable to open: %s\n", dr1fspec );
      return( 0 );
   }

   // Write the integer header
   if( fwrite( &ihdr, 2, 256, dr1fil ) != 256 ) {
      fprintf( stderr, "\nERROR: write error to: %s\n", dr1fspec );
      return( 0 );
   }

   // Write the real header
   if( fwrite( &rhdr, 4, 128, dr1fil ) != 128 ) {
      fprintf( stderr, "\nERROR: write error to: %s\n", dr1fspec );
      return( 0 );
   }

   // Write the full sample records
   for( i=1; i<ihdr.num_recs; i++ ) {
      if( fwrite( data, 2, 256, dr1fil ) != 256 ) {
         fprintf( stderr, "\nERROR: write error to: %s\n", dr1fspec );
         return( 0 );
      }
      data += 256;
   }

   // Write last samples
   if( fwrite( data, 2, ihdr.l_samp, dr1fil ) != ihdr.l_samp ) {
      fprintf( stderr, "\nERROR: write error to: %s\n", dr1fspec );
      return( 0 );
   }

   // Fill out the last block
   j = INULL;
   for( i=ihdr.l_samp; i<256; i++ ) {
      if( fwrite( &j, 2, 1, dr1fil ) != 1 ) {
         fprintf( stderr, "\nERROR: write error to: %s\n", dr1fspec );
         return( 0 );
      }
   }

   // Close the file
   fclose( dr1fil );

   return( 1 );
}

//-----------------------------------------------------------------
int build_db( void ) {
   SH_INT typ;
   LG_INT inp, len;
   CHAR _huge *ptr;
   SUDS_STATIONCOMP *sc;
   SUDS_INSTRUMENT *in;
```

```
   if( verbose )
      printf( "Building database..." );

   // Initialize database
   db = NULL;
   nrecs = 0;

   // Process all SC and IN structures
   while( ( inp = st_get( &(void _huge *)ptr, &typ, &len, sudsfil ) ) != EOF ) {
      switch( typ ) {
         case STATIONCOMP:
            sc = (SUDS_STATIONCOMP *)ptr;
            lookup( sc->effective, sc->sc_name.st_name, 1 );

            rec->effective = sc->effective;
            strcpy( rec->stn, sc->sc_name.st_name );
            rec->comp      = sc->sc_name.component;
            rec->azim      = sc->azim;
            rec->incid     = sc->incid;
            rec->lat       = (float)sc->st_lat;
            rec->lon       = (float)sc->st_long;
            rec->elev      = sc->elev;
            rec->gain      = sc->atod_gain;
            rec->chan      = sc->channel;
            rec->bcv       = sc->con_mvolts;
            rec->samp_lag = sc->clock_correct;

            break;
         case INSTRUMENT:
            in = (SUDS_INSTRUMENT *)ptr;
            lookup( in->effective, in->in_name.st_name, 1 );

            rec->effective = sc->effective;
            strcpy( rec->stn, in->in_name.st_name );
            strcpy( rec->study, in->study );
            rec->void_samp = in->void_samp;
            rec->i_serial  = in->in_serial;
            rec->s_serial  = in->sn_serial;
            rec->n_comps   = in->comps;
            rec->pre_evn   = in->pre_event;
            rec->trig_num  = in->trig_num;

            break;
      }

      qsort( db, nrecs, sizeof( DB ), compare );

      st_free( ptr, inp );
   }

   if( debug )
      dump_db( "ST2DR.DB" );

   if( verbose )
      printf( "\r                    \r" );

   return( 1 );
}

//-------------------------------------------------------------------
int compare ( DB *rec1, DB *rec2 ) {
   // Compare routine for qsort
```

```
   return( rec1->effective-rec2->effective );
}

//---------------------------------------------------------------------
int lookup( ST_TIME time, char *stn, int add ) {
   register i;

   // Returns true if found, else false
   // rec points to current record

   // Lookup record
   for( i=0; i<nrecs; i++ ) {
      rec = db+i;
      if( strcmp( rec->stn, stn ) == 0 ) {
         if( rec->effective >= time ) {
            return( 1 );
         }
      }
   }
   // Add record if not found
   if( add ) {
      nrecs++;
      if( ( db = (DB *)realloc( (void *)db, sizeof(DB)*nrecs ) ) == NULL ) {
         fprintf( stderr, "\nERROR: cannot allocate memory!\n" );
         exit( 1 );
      }
      rec = db+(nrecs-1);
      rec->effective = -2147472000;
   }
   return( 0 );
}

//---------------------------------------------------------------------
LG_INT die( SH_INT in ) {
   // SUDS library fatal error handler
   exit( in );
}

//---------------------------------------------------------------------
void s_upper( char *buffer ) {
   char *p;
   for( p = buffer; *p; p++ )
      *p = toupper( *p );
}

//---------------------------------------------------------------------
void dump_db( char *file ) {
   register i;
   FILE *dmpfil;

   dmpfil = fopen( file, "w" );

   fprintf( dmpfil, "Database dump: %s\n\n", sudsfspec );

   for( i=0; i<nrecs; i++ ) {
      rec = db+i;
      fprintf( dmpfil, "Record #%d\n", i );
      fprintf( dmpfil, "   Time stamp                    %s\n", list_mstime( rec->effe
ctive, 4 ) );
      fprintf( dmpfil, "   Station ID                    %s\n", rec->stn );
      fprintf( dmpfil, "   Study name                    %s\n", rec->study );
      fprintf( dmpfil, "   Component                     %c\n", rec->comp );
      fprintf( dmpfil, "   Component incidence           %d\n", rec->incid );
```

```
      fprintf( dmpfil, "   Component azimuth              %d\n", rec->azim );
      fprintf( dmpfil, "   Station latitude               %f\n", rec->lat );
      fprintf( dmpfil, "   Station longitude              %f\n", rec->lon );
      fprintf( dmpfil, "   Station elevation              %f\n", rec->elev );
      fprintf( dmpfil, "   Void sample value              %d\n", rec->void_samp );
      fprintf( dmpfil, "   A/D gain (magnification)       %d\n", rec->gain );
      fprintf( dmpfil, "   Channel number                 %d\n", rec->chan );
      fprintf( dmpfil, "   Bit count volts (mv/count)     %f\n", rec->bcv );
      fprintf( dmpfil, "   Serial sampling lag            %f\n", rec->samp_lag );
      fprintf( dmpfil, "   Instrument serial number       %d\n", rec->i_serial );
      fprintf( dmpfil, "   Sensor serial number           %d\n", rec->s_serial );
      fprintf( dmpfil, "   Number of components recorded %d\n", rec->n_comps );
      fprintf( dmpfil, "   Pre-event length               %f\n", rec->pre_evn );
      fprintf( dmpfil, "   Trigger number                 %d\n\n", rec->trig_num );
   }

   fclose( dmpfil );

   return;
}

//---------------------------------------------------------------------
double x2db( double x ) {
   // Magnification to dB
   return( 20.0 * log10( x ) );
}
```

```
// DR100 Header defintions
// 18-Jul-1992, RB

// Integer header
struct DR1IHDR {
   int ex_int;          // 1   Number of extra 512 byte integer records
   int ex_asc;          // 2   Number of extra ASCII blocks
   int i_null;          // 3   "Undefined" integer value (INULL)
   int d_type;          // 4   Data type: >0 = Real,  <0 = Integer
                        //       abs = #bytes/sample except 1=R*4, INULL=I*2
   int p_flag;          // 5   Additional param flag, if 1 (*) params are defined
   int rec_sys;         // 6   Recording system identification

   int dummy1[3];       // 7-9 Undefined

   int yr;              // 10  IST year
   int day;             // 11  IST julian day (day of year)
   int hr;              // 12  IST hour
   int min;             // 13  IST minute
   int sec;             // 14  IST second
   int msec;            // 15  IST millisecond
   int usec;            // 16  IST microsecond
   int tic_samp;        // 17  Sample index of first tickmark
   int tic_det;         // 18  Detection amplitude of tickmark
   int tic_num;         // 19  Number of tickmarks detected
   int ser_num;         // 20  Serial number of recording unit
   int seq_num;         // 21  Event sequence number

   int dummy2[5];       // 22-26 Undefined

   int fir_chn;         // 27  First active channel number recorded on unit
   int act_chn;         // 28  Actual channel as rcorded on unit
   int tot_chn;         // 29  Total number of channels recorded on unit
   int tot_com;         // 30  Total number of components recorded on unit
   int num_recs;        // 31  Number of data records that follow
   int l_samp;          // 32  Index of last data sample in last data record
   int rec_bytes;       // 33  Record size (bytes)
   int pb_prog;         // 34  Playback program: 1=RDGEOS, 2=AFTAPE
   int pb_ver_1;        // 35  Playback program major version number
   int pb_ver_2;        // 36  Playback program minor version number
   int inst_type;       // 37  Instrument type: 1=GEOS, 2=DR200
   int inst_ver_1;      // 38  Instrument major version number
   int inst_ver_2;      // 39  Instrument minor version number (GEOS software version)
   int sen_ser_num;     // 40  Sensor serial number
   int sen_vert;        // 41  (*) Vertical orientation (degrees)
   int sen_horz;        // 42  (*) Horizontal oreintation (degrees, CW from north)
  char sen_model[14];// 43-49 Sensor model number
   int loc_num;         // 50  Location number (GEOS)
   int exp_num;         // 51  Experiment or tape number (GEOS)
   int trig_type;       // 52  Trigger type
   int trig_sta;        // 53  Trigger STA (tenths of seconds) (GEOS)
   int trig_lta;        // 54  Trigger LTA (seconds) (GEOS)
   int trig_ratio;      // 55  Trigger ratio STA/LTA 2**  (GEOS)
   int trig_com;        // 56  Trigger component number (GEOS)
   int pre_event;       // 57  Pre-event memory size (tenths of seconds)
   int post_trig;       // 58  Post-trigger duration (seconds)

   int dummy3[42];      // 59-100 Undefined

   char history[214]; // 101-207 (*) Bugger processing history

   int dir_num_1;       // 208 Directory number (study ID number)
   int dir_num_2;       // 209 Subdirectory number (or tape-set number)
```

```
   char filename[14]; // 210-216 ASCII filename
   char study[6];     // 217-219 Study name

   int dummy4[32];    // 220-251 Undefined

   int clk_type;      // 252 Clock type (GEOS): 0=none, 1=WWVB,
                      //        2=external(master), 3=manual
   int evn_type;      // 253 Event type (GEOS): 0=continuous, 1=trigger,
                      //        2=preset, 3=calibration, 4=amplifier calibration,
                      //        5=sensor calibration
   int mot_type;      // 254 Motion type: 1=acceleration, 2=velocity,
                      //        3=displacement, 50=volumetric strain
   int com_num;       // 255 Component number: 1-3=acceleration vector,
                      //        4-6=velocity vector, 7-9=dispacement vector
                      //        1,4&7=vertical components, 2,3,5,6,8&9=horizontals
   int samps;         // 256 Total number of samples in event, valid only when
                      //        I031 (num_recs) <= 128
} ihdr;

// Real header
struct DR1RHDR {
   float ex_flt;      // 1   Number of extra 512 byte real records
   float r_null;      // 2   "Undefined" real value (RNULL)

   float dummy1[2];   // 3-4 Undefined

   float rate;        // 5   Sample rate (samples/second)
   float com_lag;     // 6   (*) Component serial sample lag

   float dummy2[32];  // 7-38 Undefined

   float t_type;      // 39  (*) Tranducer type
   float lat;         // 40  (*) Latitude (degrees)
   float loc_x;       // 41  Local coordinate X (meters)
   float lon;         // 42  (*) Longitude (degrees)
   float loc_y;       // 43  Local coordinate Y (meters)
   float elev;        // 44  (*) Elevation (meters)
   float loc_z;       // 45  Local coordinate Z (depth below surface, meters)
   float dig_con;     // 46  (*) Digitizing constant (counts/volts)
   float aa_corner;   // 47  (*) Anti-alias filter corner frequency (Hz)
   float aa_poles;    // 48  (*) Poles of AAF, roll-off = 6dB/pole
   float nat_freq;    // 49  (*) Transducer natural frequency (Hz)
   float damp_coe;    // 50  (*) Transducer damping coefficient
   float coil_con;    // 51  (*) Coil-constant (volts/motion unit)
   float gain;        // 52  (*) Amplifier gain (dB when I5 (pflag) = 1)

   float dummy3[7];   // 53-59 Undefined

   float clk_cor;     // 60  (*) Clock correction (subtract from IST)
   float clk_cor_tim; // 61  Seconds since last clock correction
   float voltage;     // 62  Voltage
   float d_trig_rat;  // 63  Desired trigger ratio (real value=2**I55)
   float act_sta;     // 64  Actual value of STA at trigger
   float act_lta;     // 65  Actual value of LTA at trigger
   float max_sta_lta; // 66  Maximum value of STA/LTA during event

   float dummy4[62];  // 67-128 Undefined
} rhdr;
```

GRM2ST - CUSP GRM file to SUDS converter version 1.00
ST2GRM - SUDS to CUSP GRM file converter version 1.00

Sun 16-Aug-1992 21:27, RB

These two programs convert data files between CUSP GRM and SUDS format.

These programs were written using Microsoft C 6.00AX and the makefile provided is for the PWB.

Additional libraries required:
HSUDS.LIB - SUDS data file library version 1.31.
Available from:
Small Systems Support
2 Boston Harbor Place
Big Water, UT 84741-0205
(801) 675-5827 Voice
(801) 675-3730 FAX

Once built, the programs offer command line help by typing only the name of the executable (e.g., ST2GRM) and pressing return.

GRM2ST processes a single GRM file to produce a single and equivalent SUDS data file. ST2GRM performs the process in reverse.

Source files included:

GRM2ST	C	14214	03-07-92	7:18p
ST2GRM	C	12997	03-07-92	7:22p
GRM_HEAD	H	2154	03-05-92	12:25p <- C GRM structure definitions
GRM_HEAD	INC	6205	03-04-92	4:08p <- FORTRAN GRM structure definitions
GRM2ST	MAK	2281	03-07-92	3:59p
ST2GRM	MAK	2307	03-07-92	6:51p

```
// GRM2ST - Sat 07-Mar-1992 13:07, RB
// Copyright (C) Robert Banfill 1992. All rights reserved.

// Converts CUSP .GRM files to SUDS 1

#define VERSION "1.00"

#include <stdio.h>
#include <conio.h>
#include <stdlib.h>
#include <string.h>
#include <malloc.h>

#include "grm_head.h"
#include <suds.h>

// Prototypes
int build_db( FILE *grmfil );
int write_file( FILE *grmfil, char *sudsfspec );
int dump_grm_head( FILE *grmfil );
int get_blk( char *ptr, FILE *grmfil );
long *l_unxdr( long *val );
float *f_unxdr( float *val );
int *i_unxdr( int *val );
MS_TIME grm2ms_time( TIME *grm );
void s_upper( char *buffer );

// Defines
#define BLOCK_SIZE  512
#define STRUCT_SIZE 64
#define CHUNK 32767

// TypeDefs
typedef unsigned char UCHAR;

typedef struct {
   char stn[5];  // Station name
   char comp;    // Component descriptor
   MS_TIME ist;  // Initial sample time
   long offset;  // Byte offset into file for first sample
   long length;  // Number of samples
   long chan;    // Channel or "Pin" number
} REC;

// Globals
REC *db;
long num_recs, rec_num;
char net[4];
float rate;
char d_type = '_';
MS_TIME base_time;
ST_TIME eff_time;

int dump = 0;
int verbose = 1;

char *progname;

//---------------------------------------------------------------------
main( int argc, char *argv[] ) {
   FILE *grmfil;
   char grmfspec[_MAX_PATH], sudsfspec[_MAX_PATH], buf[_MAX_PATH];
```

```
   progname = argv[0];

   fprintf( stderr, "GRM2ST - CUSP GRM to SUDS data file converter - Version %.4s\n",
VERSION );
   fprintf( stderr, "Copyright (c) Robert Banfill 1992. All rights reserved.\n\n" );

   if( argc < 3 ) {
      printf( "Usage: GRM2ST grmfilespec { sudsfilespec | /Dump }\n" );
      printf( "    /Dump = Dump header values to stdout\n" );
      exit( 1 );
   }

   strcpy( grmfspec, argv[1] );
   _fullpath( buf, grmfspec, _MAX_PATH );
   strcpy( grmfspec, buf );
   s_upper( grmfspec );

   s_upper( argv[2] );
   if( strncmp( argv[2], "/D", 2 ) == 0 ) {
      dump = 1;
      strcpy( sudsfspec, "Dump header to stdout" );
   }
   else {
      strcpy( sudsfspec, argv[2] );
      _fullpath( buf, sudsfspec, _MAX_PATH );
      strcpy( sudsfspec, buf );
      s_upper( sudsfspec );
   }

   if( verbose ) {
      printf( "Input GRM filespec  : %s\n",  grmfspec );
      printf( "Output SUDS filespec: %s\n\n", sudsfspec );
   }

   if( ( grmfil = fopen( grmfspec, "r+b" ) ) == NULL ) {
      fprintf( stderr, "\nERROR: Unable to open: %s\n", grmfspec );
      exit( 1 );
   }

   if( dump ) {
      if( ! dump_grm_head( grmfil ) ) {
         fprintf( stderr, "\nERROR: Unable to dump header: %s\n", grmfspec );
         exit( 1 );
      }
   }
   else {
      if( ! build_db( grmfil ) ) {
         fprintf( stderr, "\nERROR: Unable to read header: %s\n", grmfspec );
         exit( 1 );
      }
      if( ! write_file( grmfil, sudsfspec ) )
         exit( 1 );
   }

   fclose( grmfil );

   exit( 0 );
}

//---------------------------------------------------------------------
int write_file( FILE *grmfil, char *sudsfspec ) {
   register i;
   long l, m, bytes, read_bytes, total;
```

```
   REC *rec;
   SUDS_STATIONCOMP sc;
   SUDS_DESCRIPTRACE _huge *dt;
   FILE *sudsfil;
   char _huge *ptr;

   sudsfil = st_open( sudsfspec, "w+b" );

   for( l=0; l<num_recs; l++ ) {
      rec = db+l;

      if( verbose )
         printf( "\rProcessing %ld of %ld records...", l+1, num_recs );

      if( fseek( grmfil, rec->offset, SEEK_SET ) != 0 ) {
         fprintf( stderr, "\nERROR: Unable to seek record in input file\n" );
         st_close( sudsfil );
         return( 0 );
      }

      bytes = (rec->length * sizeof(int) ) + sizeof(SUDS_DESCRIPTRACE);
      if( ( dt = (SUDS_DESCRIPTRACE _huge *)halloc( bytes, 1 ) ) == NULL ) {
         fprintf( stderr, "\nERROR: Not enough memory for trace buffer: ld bytes neede
d\n", bytes );
         st_close( sudsfil );
         return( 0 );
      }

      ptr = (char _huge *)dt + sizeof(SUDS_DESCRIPTRACE);
      read_bytes = bytes - sizeof(SUDS_DESCRIPTRACE);
      total = 0;
      while( total+CHUNK <= read_bytes ) {
         if( fread( ptr, 1, CHUNK, grmfil ) != CHUNK ) {
            fprintf( stderr, "\nERROR: Read error in input file\n" );
            st_close( sudsfil );
            return( 0 );
         }
         ptr += CHUNK;
         total += CHUNK;
      }
      if( fread( ptr, 1, (size_t)(read_bytes-total), grmfil ) != read_bytes-total ) {
         fprintf( stderr, "\nERROR: Read error in input file\n" );
         st_close( sudsfil );
         return( 0 );
      }

      st_init( STATIONCOMP, &sc );
      strcpy( sc.sc_name.st_name, rec->stn );
      sc.sc_name.component = rec->comp;
      sc.data_type = d_type;
      sc.data_units = 'd';
      sc.channel = rec->chan;
      sc.effective = eff_time;

      st_init( DESCRIPTRACE, dt );
      strcpy( dt->dt_name.st_name, rec->stn );
      dt->dt_name.component = rec->comp;
      dt->begintime = rec->ist;
      dt->datatype = d_type;
      dt->length = rec->length;
      dt->rate = rate;
      dt->mindata = -32767.0;
      dt->maxdata = 32767.0;
```

```
      dt->avenoise = 0.0;
      dt->time_correct = 0.0;
      dt->rate_correct = 0.0;

      st_put( &sc, STATIONCOMP, sizeof(sc), sudsfil );
      st_put( dt, DESCRIPTRACE, bytes, sudsfil );

      hfree( dt );
   }
   st_close( sudsfil );
   return( 1 );
}

//---------------------------------------------------------------------
int build_db( FILE *grmfil ) {
   register i;
   long bytes, header_len;
   char *ptr;
   TAG *tag;
   HID *hid;
   HST *hst;
   HPN *hpn;
   REC *rec;

   if( ( ptr = (char *)malloc( BLOCK_SIZE ) ) == NULL ) {
      fprintf( stderr, "\nERROR: Unable to allocate memory!\n" );
      return( 0 );
   }
   bytes = num_recs = rec_num = 0;

   do {
      if( get_blk( ptr, grmfil ) <= 0 )
         break;

      for( i=0; i<8; i++ ) {
         bytes += STRUCT_SIZE;

         tag = (TAG *)(ptr+(i*STRUCT_SIZE));
         l_unxdr( &tag->ltid );

         if( strncmp( tag->ctid, "HID", 3 ) == 0 ) {
            hid = (HID *)tag;

            header_len = *l_unxdr( &hid->bytes );
            if( verbose ) {
               hid->time.sec = 0.000001;
               f_unxdr( &hid->time.sec );
               eff_time = (ST_TIME)grm2ms_time( &hid->time );
               printf( "Header created: %s\n", list_mstime( (MS_TIME)eff_time, 4 ) );
            }
         }
         else if( strncmp( tag->ctid, "HST", 3 ) == 0 ) {
            hst = (HST *)tag;

            base_time = grm2ms_time( &hst->time );
            rate = 1.0 / *f_unxdr( &hst->rate );
            strncpy( net, hst->net.str, 4 );
            if( *l_unxdr( &hst->inc ) != 2 ) {
               fprintf( stderr, "\nERROR: Unsupported data type: %ld bytes/sample\n",
hst->inc );
               return( 0 );
            }
            d_type = 'i';
```

```
         }
         else if( strncmp( tag->ctid, "HPN", 3 ) == 0 ) {
            hpn = (HPN *)tag;

            num_recs++;
            rec_num = num_recs - 1;
            if( ( db = (REC *)realloc( db, (size_t)num_recs*sizeof(REC) ) ) == NULL )
{
               fprintf( stderr, "\nERROR: Unable to allocate memory!\n" );
               return( 0 );
            }
            rec = db+rec_num;

            strncpy( rec->stn, hpn->name.str, 4 );
            rec->stn[4] = '\0';
            rec->comp = hpn->type.str[0];
            rec->ist = base_time + ((MS_TIME)(*l_unxdr( &hpn->rtc )-1) / (MS_TIME)rate
);
            rec->offset = *l_unxdr( &hpn->key ) + header_len;
            rec->length = *l_unxdr( &hpn->n ) / sizeof(int);
            rec->chan = *l_unxdr( &hpn->pin );
      }

         if( bytes >= header_len )
            break;
      }
   } while( bytes < header_len );

   free( ptr );

   return( 1 );
}
//---------------------------------------------------------------------
int dump_grm_head( FILE *grmfil ) {
   register i;
   long bytes, str, header_len;
   char *ptr;
   TAG *tag;
   HID *hid;
   HST *hst;
   HPN *hpn;

   if( ( ptr = (char *)malloc( BLOCK_SIZE ) ) == NULL ) {
      fprintf( stderr, "\nERROR: Unable to allocate memory!\n" );
      return( 0 );
   }
   bytes = str = 0;

   do {
      if( get_blk( ptr, grmfil ) <= 0 )
         break;

      for( i=0; i<8; i++ ) {
         bytes += STRUCT_SIZE;
         str++;

         tag = (TAG *)(ptr+(i*STRUCT_SIZE));
         l_unxdr( &tag->ltid );

         printf( "\nStructure %ld: %s\n", str, tag->ctid );

         if( strncmp( tag->ctid, "HID", 3 ) == 0 ) {
            hid = (HID *)tag;
```

```
            printf( "   Bytes in header   : %ld\n", *l_unxdr( &hid->bytes ) );
            printf( "   Mem ID#           : %ld\n", *l_unxdr( &hid->id ) );
            printf( "   Analyst ID        : %.4s, %ld chars\n", hid->who.str, *l_unxdr
( &hid->who.n ) );
            header_len = hid->bytes;
         }
         else if( strncmp( tag->ctid, "HST", 3 ) == 0 ) {
            hst = (HST *)tag;
            printf( "   Set number        : %ld\n", *l_unxdr( &hst->set ) );
            printf( "   MSTIME            : %s\n", list_mstime( grm2ms_time( &hst->tim
e ), 4 ) );
            printf( "   Year, month, day  : %ld\n", hst->time.date );
            printf( "   Hour, minute      : %ld\n", hst->time.time );
            printf( "   Second            : %f\n", hst->time.sec );
            printf( "   Network name      : %.4s, %ld chars\n", hst->net.str, *l_unxdr
( &hst->net.n ) );
            printf( "   Device name       : %.4s, %ld chars\n", hst->dev.str, *l_unxdr
( &hst->dev.n ) );
            printf( "   Sampling rate     : %f\n", *f_unxdr( &hst->rate ) );
            printf( "   Bytes per sample  : %ld\n", *l_unxdr( &hst->inc ) );
            printf( "   Digitizer bits    : %ld\n", *l_unxdr( &hst->bits ) );
            printf( "   Max counts        : %ld\n", *l_unxdr( &hst->mc ) );
            printf( "   Max volts         : %f\n", *f_unxdr( &hst->vm ) );
            printf( "   Time sync code    : %ld\n", *l_unxdr( &hst->sync ) );
         }
         else if( strncmp( tag->ctid, "HPN", 3 ) == 0 ) {
            hpn = (HPN *)tag;
            printf( "   Set number        : %ld\n", *l_unxdr( &hpn->set ) );
            printf( "   Pin number        : %ld\n", *l_unxdr( &hpn->pin ) );
            printf( "   Station name      : %.8s, %ld chars\n", hpn->name.str, *l_unxd
r( &hpn->name.n ) );
            printf( "   Component         : %.4s, %ld chars\n", hpn->type.str, *l_unxd
r( &hpn->type.n ) );
            printf( "   Time to 1st sample: %ld\n", *l_unxdr( &hpn->rtc ) );
            printf( "   GRM offset        : %ld\n", *l_unxdr( &hpn->key ) );
            printf( "   Length of trace   : %ld\n", *l_unxdr( &hpn->n ) );
            printf( "   Triggering mask   : %ld\n", *l_unxdr( &hpn->mask ) );
            printf( "   Time to trigger   : %ld\n", *l_unxdr( &hpn->trc ) );
         }
         else {
            printf( "   Unrecognized structure\n" );
         }

         if( bytes >= header_len )
            break;
      }
   } while( bytes < header_len );

   fclose( grmfil );
   free( ptr );

   return( 1 );
}

//---------------------------------------------------------------------
// Un-XDR four byte reals
float *f_unxdr( float *val ) {
   UCHAR c, *p;
                              // Byte orders:
   p = (UCHAR *)val;          //    XDR = 4321
                              //  80x86 = 1234
   // Swap bytes 1 & 4
   c = *p;
```

```
   *p = *(p+3);
   *(p+3) = c;

   // Swap bytes 2 & 3
   c = *(p+1);
   *(p+1) = *(p+2);
   *(p+2) = c;

   return( val );
}

//-----------------------------------------------------------------
// Un-XDR four byte integers
long *l_unxdr( long *val ) {
   UCHAR c, *p;
                              // Byte orders:
   p = (UCHAR *)val;          //    XDR = 4321
                              //  80x86 = 1234
   // Swap bytes 1 & 4
   c = *p;
   *p = *(p+3);
   *(p+3) = c;

   // Swap bytes 2 & 3
   c = *(p+1);
   *(p+1) = *(p+2);
   *(p+2) = c;

   return( val );
}

//-----------------------------------------------------------------
// Un-XDR two byte integers
int *i_unxdr( int *val ) {
   UCHAR c, *p;
                              // Byte orders:
   p = (UCHAR *)val;          //    XDR = 21
                              //  80x86 = 12
   // Swap bytes 1 & 2
   c = *p;
   *p = *(p+1);
   *(p+1) = c;

   return( val );
}

//-----------------------------------------------------------------
int get_blk( char *ptr, FILE *grmfil ) {

   // Read the next block in the stream
   if( ( fread( ptr, sizeof(char), BLOCK_SIZE, grmfil ) ) != BLOCK_SIZE ) {
      if( ferror( grmfil ) ) {
         fprintf( stderr, "\nERROR: Read error on input file!\n" );
         return( -1 );
      }
      else if( feof( grmfil ) )
         return( 0 );
   }

   return( 1 );
}

//-----------------------------------------------------------------
```

```
// Convert GRM file TIME struct to MSTIME
MS_TIME grm2ms_time( TIME *grm ) {
   long date, time;
   int year, mon, day, hour, min;
   double sec;

   // GRM date is stored as long int YYYYMMDD
   date = *l_unxdr( &grm->date );
   year = date / 10000;
   mon = (date / 100) - (year * 100);
   day = date - ((year * 10000) + (mon * 100));

   // GRM time is stored as long int HHMM
   time = *l_unxdr( &grm->time );
   hour = time / 100;
   min = time - (hour * 100);

   // GRM second is stored as float
   sec = (double)*f_unxdr( &grm->sec );

   // See if year include century
   if( year < 100 )
      year += 1900;

   return( make_mstime( year, mon, day, hour, min, sec ) );
}

//---------------------------------------------------------------------
long die( int in ) {
   exit(in);
}

//---------------------------------------------------------------------
void s_upper( char *buffer ) {
   char *p;
   for( p = buffer; *p; p++ )
      *p = toupper( *p );
}
```

```
// ST2GRM - Sat 07-Mar-1992 13:07, RB
// Copyright (C) Robert Banfill 1992. All rights reserved.

// Converts SUDS 1 files CUSP .GRM

#define VERSION "1.00"

// Includes
#include <stdio.h>
#include <conio.h>
#include <stdlib.h>
#include <string.h>
#include <malloc.h>
#include <ctype.h>

#include "grm_head.h"
#include <suds.h>

// Prototypes
int build_db( void );
int wr_grm_head( void );
int wr_traces( void );
long *l_xdr( long *val );
float *f_xdr( float *val );
int *i_xdr( int *val );
void s_upper( char *buffer );
void ms2grm_time( TIME *grm, MS_TIME ms );

// Defines
#define BLOCK_SIZE  512
#define STRUCT_SIZE 64
#define CHUNK 32767

// TypeDefs
typedef unsigned char UCHAR;

typedef struct {
   char stn[5];
   char comp;
   MS_TIME ist;
   long grm_offset;
   long length;
   long st_offset;
   long chan;
} REC;

typedef struct {
   char stn[5];
   char comp;
   ST_TIME eff;
   long chan;
} STATI;

// Globals
REC *db;
long num_recs, rec_num;
char net[4];
float rate;
char d_type = '_';
MS_TIME base_time;
ST_TIME eff_time;

int verbose = 1;
```

```
char *progname;
FILE *sudsfil, *grmfil;

//---------------------------------------------------------------------
main( int argc, char *argv[] ) {
   char grmfspec[_MAX_PATH], sudsfspec[_MAX_PATH], buf[_MAX_PATH];

   progname = argv[0];

   fprintf( stderr, "ST2GRM - SUDS to CUSP GRM data file converter - Version %.4s\n",
VERSION );
   fprintf( stderr, "Copyright (c) Robert Banfill 1992. All rights reserved.\n\n" );

   if( argc < 3 ) {
      printf( "Usage: ST2GRM sudsfilespec grmfilespec\n" );
      exit( 1 );
   }

   strcpy( sudsfspec, argv[1] );
   _fullpath( buf, sudsfspec, _MAX_PATH );
   strcpy( sudsfspec, buf );
   s_upper( sudsfspec );

   strcpy( grmfspec, argv[2] );
   _fullpath( buf, grmfspec, _MAX_PATH );
   strcpy( grmfspec, buf );
   s_upper( grmfspec );

   if( verbose ) {
      printf( "Input SUDS filespec: %s\n",  sudsfspec );
      printf( "Output GRM filespec: %s\n\n", grmfspec );
   }

   sudsfil = st_open( sudsfspec, "r+b" );

   if( ( grmfil = fopen( grmfspec, "w+b" ) ) == NULL ) {
      fprintf( stderr, "\nERROR: Unable to open: %s\n", grmfspec );
      exit( 1 );
   }


   if( ! build_db( ) )
      exit( 1 );

   if( ! wr_grm_head( ) )
      exit( 1 );

   if( ! wr_traces( ) )
      exit( 1 );

   st_close( sudsfil );
   fclose( grmfil );

   exit( 0 );
}

//---------------------------------------------------------------------
int wr_traces( void ) {
   long l, total;
   int typ;
   long inp, len;
   REC *rec;
```

```
   char _huge *ptr, _huge *data;

   for( l=0; l<num_recs; l++ ) {

      printf( "\rProcessing %ld of %ld traces", l+1, num_recs );

      rec = db+l;

      st_seek( sudsfil, rec->st_offset, 0 );

      inp = st_get( &ptr, &typ, &len, sudsfil );

      if( typ != DESCRIPTRACE ) {
         fprintf( stderr, "\nERROR: Fatal error accessing SUDS file!\n" );
         return( 0 );
      }

      data = ptr + sizeof(SUDS_DESCRIPTRACE);
      total = 0;
      while( total+CHUNK <= rec->length ) {
         if( fwrite( data, CHUNK, 1, grmfil ) != 1 ) {
            fprintf( stderr, "\nERROR: Write error to output file\n" );
            return( 0 );
         }
         data += CHUNK;
         total += CHUNK;
      }
      if( fwrite( data, (size_t)(rec->length-total), 1,  grmfil ) != 1 ) {
         fprintf( stderr, "\nERROR: Write error to output file\n" );
         return( 0 );
      }

      st_free( ptr, (int)inp );
   }

   return( 1 );
}

//---------------------------------------------------------------------
int wr_grm_head( void ) {
   long l, offset = 0, bytes, bytes_written;
   HID hid;
   HST hst;
   HPN hpn;
   REC *rec;
   long header_bytes;
   char *ptr;

   printf( "\rWriting GRM header          " );

   // Header must be multiple of 512 bytes (8 structs)
   header_bytes = ((((num_recs - 1) / 8) + 1) * 8 ) * STRUCT_SIZE;

   // Output the HID struct
   strcpy( hid.tag.ctid, "HID " );
   l_xdr( &hid.tag.ltid );
   hid.bytes = header_bytes;
   l_xdr( &hid.bytes );
   hid.id = 0;
   hid.who.n = 0;
   strncpy( hid.who.str, "    ", 4 );
   ms2grm_time( &hid.time, eff_time );
```

```
   if( fwrite( &hid, sizeof(hid), 1, grmfil ) != 1 ) {
      fprintf( stderr, "\nERROR: Write error to output file\n" );
      return( 0 );
   }
   bytes_written = sizeof(hid);

   // Output the HST struct
   strcpy( hst.tag.ctid, "HST " );
   l_xdr( &hst.tag.ltid );
   hst.set = 0;
   ms2grm_time( &hst.time, base_time );
   hst.net.n = 4;
   l_xdr( &hst.net.n);
   strncpy( hst.net.str, net, 3 );
   hst.net.str[3] = ' ';
   hst.dev.n = 0;
   strncpy( hst.dev.str, "    ", 3 );
   hst.rate = 1.0 / rate;
   f_xdr( &hst.rate );
   hst.inc = 2;
   l_xdr( &hst.inc );
   hst.bits = 0;
   hst.mc = -99;
   l_xdr( &hst.mc );
   hst.vm = -99.0;
   f_xdr( &hst.vm );
   hst.sync = 0;
   l_xdr( &hst.sync );

   if( fwrite( &hst, sizeof(hst), 1, grmfil ) != 1 ) {
      fprintf( stderr, "\nERROR: Write error to output file\n" );
      return( 0 );
   }
   bytes_written += sizeof(hid);

   // Write out HPN structs
   for( l=0; l<num_recs; l++ ) {
      rec = db+l;

      strcpy( hpn.tag.ctid, "HPN " );
      l_xdr( &hpn.tag.ltid );
      hpn.set = 0;
      hpn.pin = rec->chan;
      l_xdr( &hpn.pin );
      hpn.name.n = 8;
      l_xdr( &hpn.name.n );
      strncpy( hpn.name.str, rec->stn, 4 );
      strncpy( &hpn.name.str[4], "    ", 4 );
      hpn.type.n = 4;
      l_xdr( &hpn.type.n );
      strncpy( hpn.type.str, "    ", 4 );
      hpn.type.str[0] = toupper( rec->comp );

      hpn.rtc = (long)((rec->ist - base_time) / rate) + 1;

      hpn.key = offset;
      l_xdr( &hpn.key );
      offset += rec->length;

      hpn.n = rec->length;
      l_xdr( &hpn.n );
      hpn.mask = 0;
      hpn.trc = -2147483647;
```

```
      l_xdr( &hpn.trc );

      if( fwrite( &hpn, sizeof(hpn), 1, grmfil ) != 1 ) {
         fprintf( stderr, "\nERROR: Write error to output file\n" );
         return( 0 );
      }
      bytes_written += sizeof(hid);
   }

   // Fill out the header
   bytes = header_bytes-bytes_written;
   if( bytes > 0 ) {
      if( ( ptr = (char *)malloc( (size_t)bytes ) ) == NULL ) {
         fprintf( stderr, "\nERROR: Unable to allocate memory!\n" );
         return( 0 );
      }
      memset( ptr, 0, (size_t)bytes );
      if( fwrite( ptr, 1, (size_t)bytes, grmfil ) != bytes ) {
         fprintf( stderr, "\nERROR: Write error to output file\n" );
         return( 0 );
      }
      free( ptr );
   }

   return( 1 );
}

//-----------------------------------------------------------------------
// Convert MSTIME to GRM file TIME struct
void ms2grm_time( TIME *grm, MS_TIME ms ) {
   long date, time;
   int year, mon, day, hour, min;
   double sec;

   decode_mstime( ms, &year, &mon, &day, &hour, &min, &sec );

   // GRM date is stored as long int YYYYMMDD
   date = (long)year * 10000L;
   date = date + ((long)mon * 100L);
   date = date + (long)day;
   grm->date = *l_xdr( &date );

   // GRM time is stored as long int HHMM
   time = (long)hour * 100L;
   time = time + (long)min;
   grm->time = *l_xdr( &time );

   // GRM second is stored as float
   grm->sec = *f_xdr( &(float)sec );

   return;
}
//-----------------------------------------------------------------------
int build_db( void ) {
   long l, j;
   char _huge *ptr;
   int typ, first, flag;
   long inp, len;
   SUDS_DESCRIPTRACE _huge *dt;
   SUDS_STATIONCOMP _huge *sc;
   REC *rec;
   long srecs, srec;
   STATI *sdb = NULL, *sr;
```

```
   srecs = 0;
   first = 1;
   num_recs = 0;
   while( ( inp = st_get( &ptr, &typ, &len, sudsfil ) ) != EOF ) {
     switch( typ ) {
        case MUXDATA:
           fprintf( stderr, "\nERROR: SUDS file contains multiplexed data!\n" );
           return( 0 );
           break;

        case STATIONCOMP:
           sc = (SUDS_STATIONCOMP _huge *)ptr;
           srecs++;
           srec = srecs - 1;
           if( ( sdb = (STATI *)realloc( sdb, (size_t)srecs*sizeof(STATI) ) ) == NULL
) {
              fprintf( stderr, "\nERROR: Unable to allocate memory!\n" );
              return( 0 );
           }
           sr = sdb+srec;

           strncpy( sr->stn, sc->sc_name.st_name, 4 );
           sc->sc_name.st_name[4] = '\0';
           sr->comp = sc->sc_name.component;
           sr->eff = sc->effective;
           sr->chan = sc->channel;
           break;

        case DESCRIPTRACE:
           dt = (SUDS_DESCRIPTRACE _huge *)ptr;
           flag = 0;

           if( first ) {
              first = 0;
              rate = dt->rate;
              if( dt->datatype == 'i' || dt->datatype == 's' ||
                  dt->datatype == 'u' || dt->datatype == 'q' )
                 d_type = dt->datatype;
              else {
                 fprintf( stderr, "\nERROR: Unsupported data type: %c\n", dt->datatyp
e );
                 return( 0 );
              }
              strncpy( net, dt->dt_name.network, 3 );
              dt->dt_name.network[3] = '\0';
           }
           else {
              if( rate != dt->rate ) {
                 fprintf( stderr, "WARNING: Data has different sampling rate: %s\n",
dt->dt_name.st_name );
                 flag = 1;
              }
              if( d_type != dt->datatype ) {
                 fprintf( stderr, "WARNING: Data has different data type: %s\n", dt->
dt_name.st_name );
                 flag = 1;
              }
              if( strncmp( net, dt->dt_name.network, 3 ) != 0 ) {
                 fprintf( stderr, "WARNING: Network name has changed: %s -> %s at %s\
n",
                    net, dt->dt_name.network, dt->dt_name.st_name );
                 flag = 1;
```

```
            }
            if( flag )
               fprintf( stderr, "Trace ignored!\n" );
         }

         num_recs++;
         printf( "\rBuilding database %ld", num_recs );
         rec_num = num_recs - 1;
         if( ( db = (REC *)realloc( db, (size_t)num_recs*sizeof(REC) ) ) == NULL )
{
            fprintf( stderr, "\nERROR: Unable to allocate memory!\n" );
            return( 0 );
         }
         rec = db+rec_num;

         rec->length = dt->length * sizeof(int);
         rec->ist = dt->begintime;
         strncpy( rec->stn, dt->dt_name.st_name, 4 );
         dt->dt_name.st_name[4] = '\0';
         rec->comp = dt->dt_name.component;
         rec->st_offset = st_tell( sudsfil ) - 1;
         break;
      }
      st_free( ptr, (int)inp );
   }

   base_time = 1.0e+31;
   eff_time = 2147483647;
   for( l=0; l<num_recs; l++ ) {
      rec = db+l;

      // Find earliest time
      if( rec->ist < base_time )
         base_time = rec->ist;

      // Look at other data
      for( j=0; j<srecs; j++ ) {
         sr = sdb+j;
         if( strncmp( rec->stn, sr->stn, 4 ) == 0 &&
             rec->comp == sr->comp ) {
            if( sr->eff < eff_time )
               eff_time = sr->eff;
            rec->chan = sr->chan;
            break;
         }
      }
   }

   free( sdb );

   return( 1 );
}

//---------------------------------------------------------------------
// XDR four byte reals
float *f_xdr( float *val ) {
   UCHAR c, *p;
                              // Byte orders:
   p = (UCHAR *)val;          //    XDR = 4321
                              //  80x86 = 1234
   // Swap bytes 1 & 4
   c = *p;
   *p = *(p+3);
```

```
   *(p+3) = c;

   // Swap bytes 2 & 3
   c = *(p+1);
   *(p+1) = *(p+2);
   *(p+2) = c;

   return( val );
}

//---------------------------------------------------------------------
// XDR four byte integers
long *l_xdr( long *val ) {
   UCHAR c, *p;
                              // Byte orders:
   p = (UCHAR *)val;          //    XDR = 4321
                              //  80x86 = 1234
   // Swap bytes 1 & 4
   c = *p;
   *p = *(p+3);
   *(p+3) = c;

   // Swap bytes 2 & 3
   c = *(p+1);
   *(p+1) = *(p+2);
   *(p+2) = c;

   return( val );
}

//---------------------------------------------------------------------
// XDR two byte integers
int *i_xdr( int *val ) {
   UCHAR c, *p;
                              // Byte orders:
   p = (UCHAR *)val;          //    XDR = 21
                              //  80x86 = 12
   // Swap bytes 1 & 2
   c = *p;
   *p = *(p+1);
   *(p+1) = c;

   return( val );
}

//---------------------------------------------------------------------
long die( int in ) {
   exit( in );
}

//---------------------------------------------------------------------
void s_upper( char *buffer ) {
   char *p;
   for( p = buffer; *p; p++ )
      *p = toupper( *p );
}
```

```
// GRM_HEAD.H - Wed 04-Mar-1992 16:09, RB

// GRM file header definitions


// Tag, strings & dates ------------------------------------------
typedef struct {
   union {
      long ltid;      // Tag ID as long integer
      char ctid[4];   // Tag ID as string
   };
} TAG;

typedef struct {
   long  n;           // Number of chars in string
   char  str[4];      // String
} C4;

typedef struct {
   long  n;           // Number of chars in string
   char  str[8];      // String
} C8;

typedef struct {
   long  date;        // Year, month & day concatenated
   long  time;        // Hour & minute concatenated
   float sec;         // Seconds
} TIME;

// HID, Event summary data structure ----------------------------------
typedef struct {
   TAG   tag;         // Tag
   long  bytes;       // Number of bytes in header
   long  id;          // Mem ID
   C4    who;         // ID of analyst
   TIME  time;        // Time of header creation
   long  fill[8];     // Fill
} HID;

// HST, Event set specific data structure -----------------------------
typedef struct {
   TAG   tag;         // Tag
   long  set;         // Set number
   TIME  time;        // Set start time
   C4    net;         // Network name
   C4    dev;         // Device name
   float rate;        // Samples / second
   long  inc;         // Bytes / sample
   long  bits;        // Digitizer resolution (bits)
   long  mc;          // Maximum counts
   float vm;          // Maximum volts
   long  sync;        // Time code sync ID
   long  fill[1];     // Fill
} HST;

// HPN, Event pin specific data structure -----------------------------
typedef struct {
   TAG   tag;         // Tag
   long  set;         // Set number
   long  pin;         // Pin number (channel)
   C8    name;        // Station name
   C4    type;        // Component
   long  rtc;         // Digitizer time counts to 1st sample
```

```
   long  key;          // grm offset in file (bytes)
   long  n;            // Length of trace (bytes)
   long  mask;         // Triggering mask
   long  trc;          // Digitizer time counts of trigger
   long  fill[3];      // Fill
} HPN;
```

```
C    All header structures contain 64-bytes ; the entire header is constructed
C    such that it is a multiple of 512 bytes; padding is added if necessary.
C .. Note that all data is to be stored in XDR format.
C .. Generic structures definitions used within header types defined below:
        INTEGER*4 TAG_HID, TAG_HST, TAG_HPN              ! type identifier
        PARAMETER (TAG_HID = 4475208)                          ! = 'HID'
        PARAMETER (TAG_HST = 5526344)                          ! = 'HST'
        PARAMETER (TAG_HPN = 5132360)                          ! = 'HPN'

        STRUCTURE /TAG/
          UNION
            MAP
              INTEGER*4          TID                  ! structure id ; type
            END MAP
            MAP
              BYTE               C4(4)                ! 3-character id
            END MAP
          END UNION
        END STRUCTURE

        STRUCTURE /CSTR4/
          INTEGER*4      NC                           ! characters
          BYTE           C4(4)                        ! 4-byte string
        END STRUCTURE

        STRUCTURE /CSTR8/
          INTEGER*4      NC                           ! characters
          BYTE           C8(8)                        ! 8-byte string
        END STRUCTURE

        STRUCTURE /DATETIME/
          INTEGER*4              DATE                 ! Year,month,day concatenated
          INTEGER*4              HRMN                 ! hour,minute concatenated
          REAL*4                 SEC                  ! seconds
        END STRUCTURE

C .. HID: Event id summary data structure
        STRUCTURE /HID/
          RECORD /TAG/           TAG                  ! Structure id
          INTEGER*4              NB                   ! total bytes in header
          INTEGER*4              ID                   ! mem id
          RECORD /CSTR4/         WHO                  ! id of analyst
          RECORD /DATETIME/      T                    ! time of header creation
          INTEGER*4              FILL(8)              ! undefined fields
        END STRUCTURE

C .. HST: Event set specific data structure
        STRUCTURE /HST/
          RECORD /TAG/           TAG                  ! Structure id
          INTEGER*4              SET                  ! set number
          RECORD /DATETIME/      T                    ! set start date-time
          RECORD /CSTR4/         NET                  ! network name
          RECORD /CSTR4/         DEV                  ! device name
          REAL*4                 DT                   ! secs/sample
          INTEGER*4              INC                  ! bytes/sample
          INTEGER*4              B                    ! digitizer bits
          INTEGER*4              MC                   ! max counts
          REAL*4                 VM                   ! max volts
          INTEGER*4              SYN                  ! time code synch identifier
          INTEGER*4              FILL(1)              ! undefined fields
        END STRUCTURE

C .. HPN: Event pin specific data structure
```

```
        STRUCTURE /HPN/
          RECORD /TAG/          TAG                     ! Structure id
          INTEGER*4             SET                     ! set number
          INTEGER*4             PIN                     ! pin number
          RECORD /CSTR8/        NAM                     ! site name 6 bytes
          RECORD /CSTR4/        TYP                     ! component 3 bytes
          INTEGER*4             RTC                     ! digitizer time cnts to 1st s
ample
          INTEGER*4             KEY                     ! grm offset in file (bytes)
          INTEGER*4             N                       ! length of trace (bytes)
          INTEGER*4             MSK                     ! triggering mask
          INTEGER*4             TRC                     ! digitizer time cnts of trigg
er
          INTEGER*4             FILL(3)                 ! undefined fields in .grm hea
der
        END STRUCTURE                                   ! otherwise stn LAT, LON, ELEV
 go here

C .. These following data structures are auxillary; to be filled by routines used to
C    filter CUSP data into other formats. These are not part of the GRM file header.
        INTEGER*4 TAG_HHY, TAG_HMG, TAG_HPX, TAG_HCD, TAG_HAP   ! type identifier
        PARAMETER (TAG_HHY = 5851208)           ! = 'HHY'
        PARAMETER (TAG_HMG = 4672840)           ! = 'HMG'
        PARAMETER (TAG_HPX = 5787720)           ! = 'HPX'
        PARAMETER (TAG_HCD = 4473672)           ! = 'HCD'
        PARAMETER (TAG_HAP = 5259592)           ! = 'HAP'
C .. HHY : Event hypocenter summary structure
        STRUCTURE /HHY/
          RECORD /TAG/          TAG                     ! Structure id
          INTEGER*4             MSK                     ! event type; fix
          RECORD /DATETIME/     T                       ! origin time
          REAL*4                LAT                     ! latitude
          REAL*4                LON                     ! longitude
          REAL*4                Z                       ! depth (- down)
          REAL*4                RMS                     ! rms of solution
          INTEGER*4             NP                      ! number of phases in solution
          REAL*4                GAP                     ! azimuthal gap
          REAL*4                DMN                     ! distance to closest stn
          REAL*4                ELT                     ! error lat
          REAL*4                ELN                     ! error lon
          REAL*4                EZ                      ! error depth
          REAL*4                ET                      ! error time
        END STRUCTURE

C .. HMG: Event magnitude summary structure
        STRUCTURE /MAG/
          REAL*4                M                       ! magnitude
          INTEGER*4             NP                      ! number of phases
          REAL*4                RMS                     ! rms of calculation
        END STRUCTURE

        STRUCTURE /HMG/
          RECORD /TAG/          TAG                     ! Structure id
          RECORD /MAG/          MD                      ! coda duration magnitude
          RECORD /MAG/          MC                      ! coda amplitude magnitude
          RECORD /MAG/          ML                      ! wood-anderson magnitude
          RECORD /MAG/          MH                      ! helicorder magnitude
          RECORD /MAG/          M                       ! other defined magnitude
        END STRUCTURE

C .. HPX: Pin phase arrival time data structure
        STRUCTURE /HPX/
          RECORD /TAG/          TAG                     ! Structure id
```

```
          INTEGER*4             SET                 ! set number
          INTEGER*4             PIN                 ! pin number
          INTEGER*4             RTC                 ! sample count of arrival
          RECORD/DATETIME/      T                   ! time of arrival
          RECORD/CSTR8/         PHZ                 ! phase descriptor
          INTEGER*4             AZ                  ! azimuth to station ( 0 = N)
          INTEGER*4             IA                  ! take-off angle ( 0 = UP)
          REAL*4                X                   ! distance to stn  (KM)
          REAL*4                TTR                 ! traveltime residual (S)
          REAL*4                TC                  ! delay time correction (S)
          REAL*4                ERR                 ! timing error estimate (S)
        END STRUCTURE

C .. HCD: Pin coda duration data structure
        STRUCTURE /HCD/
          RECORD /TAG/          TAG                 ! Structure id
          INTEGER*4             SET                 ! set number
          INTEGER*4             PIN                 ! pin number
          INTEGER*4             RTC                 ! sample count at start of cod
a
          INTEGER*4             NEQ                 ! number of coda windows
          REAL*4                AMP                 ! amplitude of S in digital co
unts
          REAL*4                AFX                 ! nominal minimun amplitude
          REAL*4                QFX                 ! fixed coda decay constant
          REAL*4                AFR                 ! free amplitude
          REAL*4                QFR                 ! free coda decay constant
          REAL*4                RMS                 ! residual of fit
          REAL*4                TAU                 ! length of coda
          REAL*4                RBB                 ! amplitude of final sample
          RECORD/CSTR8/         PHZ                 ! phase descriptor
        END STRUCTURE

C .. HAP: Pin amplitude/period data structure
        STRUCTURE /HAP/
          RECORD /TAG/          TAG                 ! Structure id
          INTEGER*4             SET                 ! set number
          INTEGER*4             PIN                 ! pin number
          INTEGER*4             RTC0                ! sample count at start of win
dow
          INTEGER*4             A1                  ! min/max amp at RTC2 (DC/micr
ons)
          INTEGER*4             A2                  ! min/max amp at RTC4 (DC/micr
ons)
          INTEGER*4             DC                  ! window bias offset digital c
ounts
          INTEGER*4             WIN                 ! offset from start of window
          INTEGER*4             RTC1                ! 1st zero crossing sample cou
nt
          INTEGER*4             RTC2                ! 1st max/min sample count
          INTEGER*4             RTC3                ! 2nd zero crossing sample cou
nt
          INTEGER*4             RTC4                ! 2nd max/min sample count
          INTEGER*4             RTC5                ! 3rd zero crossing sample cou
nt
          RECORD /CSTR8/        PHZ                 ! phase descriptor
        END STRUCTURE           ! period = (RTC5 - RTC1) * HST
.DT
C .. End of data
```

PCEQ2ST - Old style PC Quake to SUDS file converter version 1.00
PCQ2ST - PC Quake to SUDS file converter version 1.00

Sun 16-Aug-1992 21:27, RB

These two programs convert data files from two versions of PC-Quake to SUDS format.

These programs were written using Microsoft C 6.00AX and the makefile provided is for the PWB.

Additional libraries required:
HSUDS.LIB - SUDS data file library version 1.31.
Available from:
Small Systems Support
2 Boston Harbor Place
Big Water, UT 84741-0205
(801) 675-5827 Voice
(801) 675-3730 FAX

Once built, the programs offer command line help by typing only the name of the executable (e.g., PCQ2ST) and pressing return.

PCEQ2ST processes a single old-style PC-Quake data file and generates the equivalent SUDS data file.

PCQ2ST processes a single PC-Quake data file and generates the equivalent SUDS data file.

Source files included:

PCEQ2ST	C	13416	05-20-92	12:26p
PCQ2ST	C	12818	03-09-92	5:36p
PCEQ2ST	MAK	2082	05-20-92	12:03p
PCQ2ST	MAK	2467	03-09-92	5:36p

```
/*
 * PCEQ2ST - Old PCEQ waveform format to SUDS data file conversion routine
 *
 * Filespec: D:\CODE\PCEQ2ST\PCEQ2ST.C
 * Last edit: 20-May-1992, RB
 *
 * Written by R. Banfill, (801) 675-5827
 *    Small Systems Support, Box 410205, Big Water  UT 84741-0205.
 *
 * Current version built using Microsoft C/C++ 7.0
 *
 * Revision history:
 *    Version 1.00 13-May-1992
 *       First working version
 *
 */

// Version number
char version[] = "1.00";

// Standard include files
#include <malloc.h>
#include <math.h>
#include <stdio.h>
#include <process.h>
#include <string.h>
#include <stdlib.h>
#include <io.h>
#include <errno.h>

// SUDS include files
#include <suds.h>

// Local Prototypes
int pcq2suds( char *, char * );
int pcq_hdr_get( char * );
int suds_hdr_put( char * );
int copy_traces( void );
void upper( char * );

// Defined types
typedef struct {  // File header
        int dummy[2];   // Fill
        int yr;                 // Effective Year
        int mo;                 // Effective Month
        int dy;                 // Effective Day
        int hr;                 // Effective Hour
        int mi;                 // Effective Minute
        int sc;                 // Effective Second
        int ms;                 // Effective Millisecond
        int rate;               // Sampling rate (sample/second)
        int maxtra;     // Length of longest trace (samples)
        int nsta;               // Number of stations in file
} VHDR;
typedef struct {  // Station header
        int dummy[2];   // Fill
        char sta[4];    // Station identifier
        int gain;               // Channel gain
        int yr;                 // IST Year
        int mo;                 // IST Month
        int dy;                 // IST Day
        int hr;                 // IST Hour
        int mi;                 // IST Minute
```

```
        int sc;                 // IST Second
        int ms;                 // IST Millisecond
        int blk;                // Blocks
} CHDR;
typedef struct {  // Database record
   char  sta[5];  // Station identifier (null terminated)
   long  offset;  // Offset into file where data begins (bytes)
   int   blks;    // Number of 256 sample blocks in trace
   float rate;    // Sampling rate
   int   gain;    // Channel gain setting
   MS_TIME ist;   // Initial sample time
} REC;

// Global data
extern char *progname;      // Executable filespec (used by st_error())
int verbose, debug, htime; // Flags
FILE *infil, *outfil;       // File pointers
void *blk_ptr;              // Block pointer
REC *db;                    // Database base pointer
int nrecs;                  // Number of records in database

#define BLK_SIZE 516        // Number of bytes in a block

// ---------------------------------------------------------------------
main( int argc, char *argv[] ) {
   register i;
   char *j;
   char buffer[128], infile[128], outpath[128], outfile[128];

   // Defaults
   progname = argv[0];
   verbose = 1;
   debug = 0;
   htime = 0;
   outpath[0] = '\0';
   infile[0] = '\0';

   // Copyright notice
   printf( "Early PCEQ data to SUDS data file converter, Version %.4s\n", version );
   printf( "Copyright (c) Robert Banfill 1992. All rights reserved.\n\n" );

   // Help
   if( argv[1][0] == '?' || argv[1][1] == '?' || argv[1][1] == '?' || argc == 1 ) {
               printf( "Usage: PCEQ2ST [switches] inputfilespec <┘\n\n" );
      printf( "inputfilespec:\n");
      printf( "   A partial or full filespec of old style PCEQ data file(s).\n" );
      printf( "   This program performs wildcard expansion on the input file\n" );
      printf( "   specification.  Any DOS legal wildcard expression is\n" );
      printf( "   allowed (e.g. *.*, *.WV?, 900204??.WVG).\n\n" );
      printf( "switches:\n" );
      printf( "   /Opath  path = alternate destination directory for output\n" );
               printf( "              SUDS files. (same dir as inputfile)\n" );
      printf( "   /D      Dump file contents as ASCII to stdout. (NO)\n" );
      printf( "   /H      Use file header time stamp for all traces. (NO)\n" );
      printf( "   /Q      Quiet mode. (VERBOSE)\n\n" );
      printf( "() = default, [] = optional.  Arguments are not case sensitive.\n\n" );
      printf( "Output files are demultiplexed SUDS 1.3 files with the same name as\n"
);
      printf( "the input file and a .DMX extension.  This program returns error\n" );
      printf( "code 1 if an error occurs, 0 for successful execution.\n" );
      exit( 1 );
   }
```

```
   // Parse the command-line
   i = 1;
   while( argv[i] != NULL ) {
      if( argv[i][0] == '-' || argv[i][0] == '/' ) {
         // Switches
         switch( argv[i][1] ) {
            // Verbose
            case 'q':
            case 'Q':
               verbose = 0;
               break;
            // Use header time
            case 'h':
            case 'H':
               htime = 1;
               break;
            // Debug
            case 'd':
            case 'D':
               debug = 1;
               break;
            // Output path
            case 'o':
            case 'O':
               strcpy( outpath, &argv[i][2] );
               j = outpath+strlen( outpath )-1;
               if( strncmp( j, "\\", 1 ) != 0 )
                  strcat( outpath, "\\" );
               break;
            default:
               fprintf( stderr, "WARNING: unrecognized switch: %s\n", argv[i] );
               break;
         }
         i++;
         continue;
      }
      else {
         // Input filespec
         _fullpath( infile, argv[i], 128 );
         j = infile+strlen( infile )-4;
         if( *j != '.' )
            strcat( infile, ".WVM" );
         upper( infile );
      }

      // Check for input file
      if( infile[0] == '\0' )
         exit( 1 );

      // Build output filespec
      if( outpath[0] == '\0' )
         strcpy( outfile, infile );
      else {
         strcpy( outfile, outpath );
         j = strrchr( infile, '\\' )+1;
         strcat( outfile, j );
      }
      j = strrchr( outfile, '.' );
      strncpy( j, ".DMX", 4 );
      _fullpath( buffer, outfile, 128 );
      upper( outfile );

      // Process the current file
```

```
      if( pcq2suds( infile, outfile ) != 0 )
         exit( 1 );

      i++;
   }
   exit( 0 );
}

// ---------------------------------------------------------------------
int pcq2suds( char *infile, char *outfile ) {
   register i;

   if( verbose ) {
      printf( " Input filespec: %s\n", infile );
      printf( "Output filespec: %s\n\n", outfile );
   }
   else
      printf( "%s\n", infile );

   // Get the header
   if( pcq_hdr_get( infile ) != 0 )
      return( 1 );

   // Write out station info
   if( suds_hdr_put( outfile ) != 0 )
      return( 1 );

   // Copy the traces
   if( copy_traces( ) != 0 )
      return( 1 );

   free( db );
   free( blk_ptr );
   st_close( outfil );
   fclose( infil );

   return( 0 );
}

// ---------------------------------------------------------------------
int pcq_hdr_get( char *infile ) {
        register k, j, i;
        int bytes_read;
        VHDR vhdr;
        CHDR *chdr;
        REC *rec;

   // Allocate the block buffer
        if( ( blk_ptr = malloc( BLK_SIZE ) ) == NULL ) {
                fprintf( stderr, "ERROR: not enough memory\n" );
                return( 1 );
        }

   // Open data file
   if( ( infil = fopen( infile, "r+b" ) ) == NULL ) {
                fprintf( stderr, "ERROR: Unable to open: %s\n", infile );
      return( 1 );
   }

        // Read in the file header
        if( ( bytes_read = fread( blk_ptr, 1, BLK_SIZE, infil ) ) != BLK_SIZE ) {
                fprintf( stderr, "ERROR: read error: %s\n", infile );
      return( 1 );
```

```
   }

   // Save the file header
        memcpy( &vhdr, blk_ptr, sizeof(vhdr) );

   if( debug ) {
                printf( "\nPCEQ file header, %i bytes.\n", bytes_read );
                printf( "              Date: %02d-%02d-%04d %02d:%02d %02d.%03d\n", vh
dr.mo, vhdr.dy,
                                        vhdr.yr+1900, vhdr.hr, vhdr.mi, vhdr.sc, vhdr.
ms );
                printf( "     Sampling rate: %d\n", vhdr.rate );
                printf( "      Trace length: %d samples\n", vhdr.maxtra );
      printf( "Number of stations reported in header: %d\n", vhdr.nsta );
        }

        // Read through blank blocks (2 thru 5)
        for( i=2; i<=5; i++ ) {
                if( ( bytes_read = fread( blk_ptr, 1, BLK_SIZE, infil ) ) != BLK_SIZE
) {
                        fprintf( stderr, "ERROR: read error: %s\n", infile );
                        return( 1 );
                }
        }

   // Loop through file to find each channel
   nrecs = 0;
   db = NULL;
   while( 1 ) {

                // Read in the channel header
      if( ( bytes_read = fread( blk_ptr, 1, BLK_SIZE, infil ) ) != BLK_SIZE ) {
         break;
                }
                chdr = (CHDR *)blk_ptr;

      // Add a record to the database
      nrecs++;
      if( ( db = (REC *)realloc( db, sizeof(REC)*nrecs ) ) == NULL ) {
         fprintf( stderr, "ERROR: not enough memory\n" );
         return( 1 );
      }
      rec = db+nrecs-1;

                strncpy( rec->sta, chdr->sta, 4 );
                rec->sta[4] = '\0';
      rec->offset = ftell( infil );
                rec->blks = chdr->blk;
                rec->rate = (vhdr.rate == 0 ? 100.0 : (float)vhdr.rate );
      rec->gain = chdr->gain;
      if( htime ) {
         rec->ist = make_mstime( vhdr.yr+1900, vhdr.mo, vhdr.dy, vhdr.hr, vhdr.mi,
            (double)vhdr.sc+(((double)vhdr.ms)*0.001) );
      }
      else {
         rec->ist = make_mstime( chdr->yr+1900, chdr->mo, chdr->dy, chdr->hr, chdr->mi
,
            (double)chdr->sc+(((double)chdr->ms)*0.001) );
      }

                if( debug ) {
         printf( "\nChannel %d header, %d bytes.\n", nrecs, bytes_read );
                        printf( "Station identifier: %s\n", rec->sta );
```

```
                        printf( "            Offset: %ld\n", rec->offset );
                        printf( "            Blocks: %d\n", rec->blks );
                        printf( "               IST: %s\n", list_mstime( rec->ist, 4 )
 );
                        printf( "              Gain: %d\n", rec->gain );
                        printf( "     Sampling rate: %f\n", rec->rate );
                }

                // Read through data blocks
                for( j=0; j<rec->blks; j++ ) {
                        if( ( bytes_read = fread( blk_ptr, 1, BLK_SIZE, infil ) ) != B
LK_SIZE ) {
                                fprintf( stderr, "ERROR: read error: %s\n", infile );
                                return( 1 );
                        }
         if( debug ) {
                                for( k=0; k<255; k++ )
                                        printf( "%05d %d\n", j*256+k, *((int *)blk_ptr
+2+k) );
                        }
                }
        }

        if( debug )
                return( 1 );
        else
                return( 0 );
}

// ---------------------------------------------------------------------
int suds_hdr_put( char *outfile ) {
        register i;
   SUDS_DETECTOR sd;
        SUDS_STATIONCOMP sc;
        REC *rec;

   // Open output file
   outfil = st_open( outfile, "w+b" );

   // DETECTOR struct
        st_init( DETECTOR, &sd );

        sd.dalgorithm = 'm';
        strcpy( sd.net_node_id, "PCEQ2ST" );
        sd.versionnum = atof( version );

   st_put( &sd, DETECTOR, sizeof( SUDS_DETECTOR ), outfil );
   st_flush( outfil );

   // STATIONCOMP struct's
   for( i=0; i<nrecs; i++ ) {
                rec = db+i;
                st_init( STATIONCOMP, &sc );

                strcpy( sc.sc_name.st_name, rec->sta );
                sc.data_type = 'i';
      sc.data_units = 'd';
                sc.channel = i;
                sc.atod_gain = rec->gain;
                sc.clip_value = 32767;
                sc.effective = (ST_TIME)rec->ist;

      st_put( &sc, STATIONCOMP, sizeof( SUDS_STATIONCOMP ), outfil );
```

```
   }
   st_flush( outfil );

   return( 0 );
}

// ---------------------------------------------------------------------
int copy_traces( void ) {
        register i, j;
   int chan, dummy[2];
        long pos, bytes;
        SUDS_DESCRIPTRACE _huge *dt;
        char _huge *ptr;
        int _huge *ptr1;
        REC *rec;

        // Process each channel
        for( chan=0; chan<nrecs; chan++ ) {
                rec = db+chan;

      if( verbose )
         printf( "Processing channel: %i - station: %s - IST: %s\r",
            chan+1, rec->sta, list_mstime( rec->ist, 4 ) );

                // Create a DESCRIPTRACE structure
                bytes = (rec->blks*256*sizeof(int))+sizeof(SUDS_DESCRIPTRACE);
                if( ( ptr = halloc( bytes, 1 ) ) == NULL ) {
      printf( "ERROR: Not enough memory: %ld\n", bytes );
                        return( 1 );
                }
                dt = (SUDS_DESCRIPTRACE _huge *)ptr;

                // Initialize the structure
                st_init( DESCRIPTRACE, dt );

                strcpy( dt->dt_name.st_name, rec->sta );
                dt->begintime = rec->ist;
                dt->datatype = 'i';
                dt->rate = rec->rate;
                dt->time_correct = 0.;
                dt->rate_correct = 0.;
                dt->mindata = -32767.0;
                dt->maxdata = 32767.0;
                dt->avenoise = 0.;
      dt->length = rec->blks*256;

                // Reset the trace buffer pointer
                ptr1 = (int _huge *)(dt+1);
                pos = rec->offset;

                // Read in the  channel
      for( i=1; i<=rec->blks; i++ ) {
                        if( fseek( infil, pos, SEEK_SET ) != 0 ) {
                                printf( "ERROR: seek error in input file\n" );
                                return( 1 );
                        }

         if( fread( dummy, sizeof(int), 2, infil ) != 2 ) {
                                printf( "ERROR: read error in input file\n" );
                                return( 1 );
                        }
         if( fread( ptr1, sizeof(int), 256, infil ) != 256 ) {
                                printf( "ERROR: read error in input file\n" );
```

```
                                return( 1 );
                        }
                        ptr1 += 256;
            pos += BLK_SIZE;
         }

         // Trim off trailing zeros
         ptr1--;
         while( *ptr1 == 0 ) {
            dt->length--;
            bytes -= sizeof(int);
            ptr1--;
         }

                  // Write out SUDS structure
                  st_put( dt, DESCRIPTRACE, bytes, outfil );
                  st_flush( outfil );

                  // Free memory
                  hfree( ptr );
           }

   if( verbose )
      printf( "\n\n" );

   return( 0 );
}

// ---------------------------------------------------------------------
// Convert a string to upper case
void upper( char *buffer ) {
   char *p;
   for( p = buffer; *p; p++ )
      *p = toupper( *p );
}

// ---------------------------------------------------------------------
// SUDS library error exit routine
long die( int in ) {
   exit( in );
}
```

```
/*
 * PCQ2ST - PC-Quake to SUDS data file conversion routine
 *
 * Filespec: D:\CODE\PCQ2ST\PCQ2ST.C
 * Last edit: 05-May-1991, RB
 *
 * Written by R. Banfill, (801) 675-5827
 *    Small Systems Support, Box 410205, Big Water  UT 84741-0205.
 *
 * Revision history:
 *    Version 0.01.00 - 04-May-1991
 *       First effort.
 *    Version 1.00.00 - 05-May-1991
 *       First working version
 *    Version 1.00.01 - 05-May-1991
 *       Fixed time problem
 *    Version 1.00.02 - 05-May-1991
 *       Cleaned up.
 *    Version 1.01.00 - 28-Jul-1991
 *       Added 16 bit support
 *
 */

// Version number
char version[8] = "1.01.00";

// Standard include files
#include <malloc.h>
#include <stdio.h>
#include <process.h>
#include <string.h>
#include <stdlib.h>
#include <io.h>
#include <errno.h>
#include <sys\timeb.h>

// SUDS include files
#include <suds.h>

// Local Prototypes
int pcq2suds( char[128], char[128] );
int pcq_hdr_get( char[128] );
int suds_hdr_put( char[128] );
int copy_traces( void );
void upper( char[128] );

// Global data
extern char *progname;
int verbose, debug;
FILE *infil, *outfil;
struct PCQ_HDR {
   int magic;
   int chan_len;
   int scan_cnt;
   float srate;
   int chan_lst[16];
   int gain_lst[16];
   int chan_stat_lst[32];
   struct STN_LST {
      char stn_id[10];
      int chan;
      float lat;
      float lon;
```

```
      float elev;
   } stn_lst[16];
   struct timeb trig_tim;
   struct timeb ist;
   char cal_ist[10];
} hdr;

// ---------------------------------------------------------------------
main( int argc, char **argv ) {
   register i;
   char *j;
   char buffer[128], infile[128], outpath[128], outfile[128];

   progname = argv[0];
   verbose = 0;
   debug = 0;
   strcpy( outpath, "none" );
   strcpy( infile, "none" );

   printf( "PC-Quake -> SUDS data file converter, Version %.4s\n", version );
   printf( "Copyright (c) Robert Banfill 1991-92. All rights reserved.\n\n" );

   if( argv[1][0] == '?' || argv[1][1] == '?' || argc == 1 ) {
      printf( "Usage: PCQ2ST [switches] inputfilespec <↵\n\n" );
      printf( "inputfilespec:\n");
      printf( "   This program performs wildcard expansion on the input\n" );
      printf( "   file specification.  Any DOS legal wildcard expression\n" );
      printf( "   is allowed. (e.g. *.*, *.WVM, 900204??.WVM)\n\n" );
      printf( "switches:\n" );
      printf( "   /V      Verbose - display; sample rate, record length, initial\n" );
      printf( "           sample time and station names for each file. (off)\n" );
      printf( "   /Opath  Output path - alternate destination directory for output\n"
);
      printf( "           SUDS files. (same dir as inputfile)\n\n" );
      printf( "() indicates the default.  Arguments are not case sensitive.\n\n" );
      printf( "Output files are demultiplexed SUDS files with the same base name as\n"
 );
      printf( "the input file and a .DMX extension.  This program returns error\n" );
      printf( "code 1 if an error occurs, 0 for successful execution.\n" );
      exit( 1 );
   }

   i = 1;
   while( argv[i] != NULL ) {
      if( argv[i][0] == '-' || argv[i][0] == '/' ) {
         // Switches
         switch( argv[i][1] ) {
            // Verbose
            case 'v':
            case 'V':
               verbose = 1;
               break;
            // Debug
            case 'd':
            case 'D':
               debug = 1;
               verbose = 1;
               break;
            // Output path
            case 'o':
            case 'O':
               strcpy( outpath, &argv[i][2] );
               j = outpath+strlen( outpath )-1;
```

```
            if( strncmp( j, "\\", 1 ) != 0 )
               strcat( outpath, "\\" );
            break;
         default:
            printf( "Unrecognized switch: %s\n\n", argv[i] );
            break;
      }
      i++;
      continue;
   }
   else {
      // Input filespec
      strcpy( infile, argv[i] );
      j = infile+strlen( infile )-4;
      if( strncmp( j, ".", 1 ) != 0 )
         strcat( infile, ".WVM" );
   }

   if( strncmp( infile, "none", 4 ) == 0 )
      return( 1 );

   // Get the full input pathname
   if( _fullpath( buffer, infile, 128 ) != NULL )
      memcpy( infile, buffer, 128 );
   else {
      printf( "Invalid input filespec: %s\n", infile );
      exit( 1 );
   }
   upper( infile );

   // Build output filespec
   if( strncmp( outpath, "none", 4 ) == 0 )
      strcpy( outfile, infile );
   else {
      strcpy( outfile, outpath );
      j = strrchr( infile, '\\' )+1;
      strcat( outfile, j );
   }
   j = strrchr( outfile, '.' );
   strncpy( j, ".DMX", 4 );
   if( _fullpath( buffer, outfile, 128 ) != NULL )
      memcpy( outfile, buffer, 128 );
   else {
      printf( "Invalid output filespec: %s\n", outfile );
      exit( 1 );
   }
   upper( outfile );

   if( pcq2suds( infile, outfile ) != 0 )
      exit( 1 );

   i++;
  }
  exit( 0 );
}

// ---------------------------------------------------------------------
int pcq2suds( char infile[128], char outfile[128] ) {
  register i;

  if( verbose ) {
     printf( " Input filespec: %s\n", infile );
     printf( "Output filespec: %s\n\n", outfile );
```

```
   }
   else
      printf( "%s\n", infile );

   // Get the header
   if( pcq_hdr_get( infile ) != 0 )
      return( 1 );

   // Write out station info
   if( suds_hdr_put( outfile ) != 0 )
      return( 1 );

   // Copy the traces
   if( copy_traces( ) != 0 )
      return( 1 );

   st_close( outfil );
   fclose( infil );

   return( 0 );
}

// ---------------------------------------------------------------------
int pcq_hdr_get( char infile[128] ) {
   register i;
   int bytes_read;
   void *ptr;

   // Open data file
   if( ( infil = fopen( infile, "r+b" ) ) == NULL ) {
      printf( "ERROR: Unable to open: %s\n", infile );
      return( 1 );
   }

   // Allocate a buffer
   if( ( ptr = malloc( 1024 ) ) == NULL ) {
      printf( "ERROR: Not enough memory: pcq_hdr_read()\n" );
      return( 1 );
   }

   // Read in the header
   if( ( bytes_read = fread( ptr, sizeof(char), 1024, infil ) ) == 0 ) {
      printf( "ERROR: Unable to read: %s\n", infile );
      return( 1 );
   }

   // copy header, lose the buffer
   memcpy( &hdr, ptr, sizeof( hdr ) );
   free( ptr );

   // Make sure we are dealing with a PC-Quake file
   if( hdr.magic != 255 ) {
      printf( "ERROR: Bad magic number: %i in %s\n", hdr.magic, infile );
      printf( "       Not a PC-Quake format file!\n" );
      return( 1 );
   }

   if( debug ) {
      printf( "\nPC-Quake file header, %i bytes.\n", bytes_read );
      printf( "Magic number: %i\n", hdr.magic );
      printf( "Channel length: %i\n", hdr.chan_len );
      printf( "Scan count: %i\n", hdr.scan_cnt );
      printf( "Sample rate: %f\n", hdr.srate );
```

```
      for( i=0; i<=15; i++ )
         printf( "%i - chan_lst: %i - gain_lst: %i - stat_lst: %i\n",
                 i+1, hdr.chan_lst[i], hdr.gain_lst[i], hdr.chan_stat_lst[i] );
      for( i=0; i<=15; i++ )
         printf( "Station: %s - Chan: %i - Lat: %5.2f - Long: %6.2f - Elev: %5.2f\n",
                 hdr.stn_lst[i].stn_id, hdr.stn_lst[i].chan,
                 hdr.stn_lst[i].lat, hdr.stn_lst[i].lon,
                 hdr.stn_lst[i].elev );
      printf( "Trigger time: %li %u %i %i\n",
              hdr.trig_tim.time, hdr.trig_tim.millitm,
              hdr.trig_tim.timezone, hdr.trig_tim.dstflag );
      printf( "Channel start time: %li %u %i %i\n",
              hdr.ist.time, hdr.ist.millitm,
              hdr.ist.timezone, hdr.ist.dstflag );
      printf( "Cal chan str tim: %s\n", hdr.cal_ist );
   }
   return( 0 );
}

// ---------------------------------------------------------------------
int suds_hdr_put( char outfile[128] ) {
   register i, j;
   SUDS_DETECTOR sd;
   SUDS_STATIONCOMP sc;
   SUDS_TRIGGERS st;

   // Open output file
   outfil = st_open( outfile, "w+b" );

   // DETECTOR struct
   st_init( DETECTOR, &sd );
   sd.dalgorithm = 'm';
   strcpy( sd.net_node_id, "PCQ2ST" );
   st_put( &sd, DETECTOR, sizeof( SUDS_DETECTOR ), outfil );
   st_flush( outfil );

   // STATIONCOMP struct's
   for( i=0; i<=15; i++ ) {
      st_init( STATIONCOMP, &sc );
      strncpy( sc.sc_name.st_name, hdr.stn_lst[i].stn_id, 5 );
      sc.st_lat = (LONLAT)hdr.stn_lst[i].lat;
      sc.st_long = (LONLAT)hdr.stn_lst[i].lon;
      sc.elev = hdr.stn_lst[i].elev;
      sc.data_type = 's';
      sc.data_units = 'd';
      sc.channel = hdr.stn_lst[i].chan;
      for( j=0; j<=15; j++ ) {
         if( hdr.chan_lst[j] == hdr.stn_lst[i].chan ) {
            sc.atod_gain = hdr.gain_lst[j];
            break;
         }
      }
      st_put( &sc, STATIONCOMP, sizeof( SUDS_STATIONCOMP ), outfil );
   }
   st_flush( outfil );

   return( 0 );
}

// ---------------------------------------------------------------------
int copy_traces( void ) {
   register i, j;
   int handle, chan;
```

```
   long pos, flen, dlen, blks, blk;
   SUDS_DESCRIPTRACE _huge *dt;
   char _huge *ptr;
   int _huge *ptr1;
   MS_TIME tim, mtim;

   // Input file info
   handle = fileno( infil );
   flen = filelength( handle );
   blk = hdr.scan_cnt*hdr.chan_len;
   blks = (flen-1024)/(blk*sizeof(int));

   if( debug ) {
      printf( "\nInput file length: %li\n", flen );
      printf( "Data consists of: %i blocks of %i words\n\n", blks, blk );
   }

   // Create a DESCRIPTRACE structure
   dlen = blks*hdr.chan_len*sizeof(int);
   if( ( ptr = halloc( dlen+sizeof(SUDS_DESCRIPTRACE), 1 ) ) == NULL ) {
      printf( "ERROR: Not enough memory: %li\n",
              dlen+sizeof(SUDS_DESCRIPTRACE) );
      return( 1 );
   }
   dt = (SUDS_DESCRIPTRACE *)ptr;
   ptr1 = (int _huge *)(ptr+sizeof(SUDS_DESCRIPTRACE));

   // Initialize the structure
   st_init( DESCRIPTRACE, dt );
   tim = (MS_TIME)hdr.ist.time;
   mtim = (MS_TIME)(hdr.ist.millitm)/1000.;
   dt->begintime = tim+mtim;
   if( hdr.ist.dstflag == 0 )
      dt->localtime = -(hdr.ist.timezone*60.);
   else
      dt->localtime = -((hdr.ist.timezone*60.)-3600.);
   dt->datatype = 's';
   dt->length = hdr.chan_len*blks;
   dt->rate = hdr.srate;
   dt->time_correct = 0.;
   dt->rate_correct = 0.;
   dt->mindata = 0.;
   dt->maxdata = 4096.;
   dt->avenoise = 2048.;

   if( verbose ) {
      printf( "Sample rate: %f sps\n", dt->rate );
      printf( "Record length: %.2f seconds\n", (float)dt->length/dt->rate );
      if( debug )
         printf( "Local time = GMT + %i minutes\n", dt->localtime );
      if( hdr.ist.dstflag == 0 )
         printf( "IST: %s Local \n",
                 list_mstime( dt->begintime+dt->localtime, 4 ) );
      else
         printf( "IST: %s Local (daylight savings time in effect)\n",
                 list_mstime( dt->begintime+dt->localtime, 4 ) );
      printf( "IST: %s GMT\n\n", list_mstime( dt->begintime, 4 ) );
   }

   // Copy each channel
   for( j=0; j<=15; j++ ) {

      // Reset the input file
```

```
      rewind( infil );
      pos = 1024+(hdr.stn_lst[j].chan*hdr.chan_len*sizeof(int));

      // Reset the trace buffer pointer
      ptr1 = (int _huge *)(ptr+sizeof(SUDS_DESCRIPTRACE));

      // Read in a channel
      for( i=1; i<= blks; i++ ) {
         if( fseek( infil, pos, SEEK_SET ) != 0 ) {
            printf( "ERROR: Unable to seek record: %li\n", pos );
            return( 1 );
         }

         if( fread( ptr1, sizeof(int), hdr.chan_len, infil ) == 0 ) {
            printf( "ERROR: Unable to read: Input file\n" );
            return( 1 );
         }
         ptr1 += hdr.chan_len;
         pos += blk*sizeof(int);
      }

      // Copy station ID into SUDS structure
      strncpy( dt->dt_name.st_name, hdr.stn_lst[j].stn_id, 5 );

      if( verbose )
         printf( "Processing channel: %i - station: %s\r",
            hdr.stn_lst[j].chan, dt->dt_name.st_name );

      // Write out SUDS structure
      st_put( dt, DESCRIPTRACE, dlen+sizeof(SUDS_DESCRIPTRACE), outfil );
      st_flush( outfil );
   }

   if( verbose )
      printf( "\n\n" );

   hfree( ptr );
   return( 0 );
}

// ---------------------------------------------------------------------
void upper( char buffer[128] ) {
   char *p;
   for( p = buffer; *p; p++ )
      *p = toupper( *p );
}

// ---------------------------------------------------------------------
long die( int in ) {
   exit( in );
}
```

SEGY2ST - PASSCAL modified SEGY to SUDS file converter version 1.00

Sun 16-Aug-1992 21:27, RB

This program converts data files from the PASSCAL modified SEGY storage format to SUDS format.

This program was written using Microsoft C 6.00AX and the makefile provided is for the PWB.

Additional libraries required:
HSUDS.LIB - SUDS data file library version 1.31.
Available from:
Small Systems Support
2 Boston Harbor Place
Big Water, UT 84741-0205
(801) 675-5827 Voice
(801) 675-3730 FAX

Once built, the program offers command line help by typing only the name of the executable (e.g., SEGY2ST) and pressing return.

SEGY2ST processes a single SEGY data file and generates the equivalent SUDS data file.

Source files included:

SEGY2ST	C	6261	03-11-92	12:16p
SEGY	H	2507	03-11-92	10:56a
SEGY2ST	MAK	2389	03-11-92	12:14p

```
/*
 * SEGY2ST - PASSCAL modified SEGY to SUDS data file conversion routine
 *
 * Filespec: D:\CODE\SEGY2ST\SEGY2ST.C
 * Last edit: Wed 11-Mar-1992 10:58, RB
 *
 * Written by R. Banfill, (801) 675-5827
 *    Small Systems Support, Box 410205, Big Water  UT 84741-0205.
 *    Copyright (c) Robert Banfill 1991-92. All rights reserved.
 *
 * Revision history:
 *    Version 1.00 Wed 11-Mar-1992 10:58, RB
 *       First working version
 *
 */

// Version number
char version[] = "1.00";

// Standard include files
#include <malloc.h>
#include <stdio.h>
#include <string.h>
#include <stdlib.h>

// SUDS include files
#include <suds.h>

#include "segy.h"

// Local Prototypes
int segy2suds( char[128], char[128] );
int segy_hdr_get( char[128] );
int suds_hdr_put( char[128] );
int copy_traces( void );
void upper( char[128] );

char stn[32];
int verbose = 1;

extern char *progname;

FILE *infil, *outfil;

struct SegyHead hdr;

// ---------------------------------------------------------------------
main( int argc, char *argv[] ) {
   register i;
   char buffer[128], infile[128], outfile[128];

   progname = argv[0];

   printf( "SEGY to SUDS data file converter, Version %.4s\n", version );
   printf( "Copyright (c) Robert Banfill 1991-92. All rights reserved.\n\n" );

   if( argv[1][0] == '?' || argv[1][1] == '?' || argc < 3 ) {
      printf( "Usage: SEGY2ST inputfilespec outputfilespec\n\n" );
      exit( 1 );
   }

   strcpy( buffer, argv[1] );
   _fullpath( infile, buffer, 128 );
```

```
   upper( infile );

   strcpy( buffer, argv[2] );
   _fullpath( outfile, buffer, 128 );
   upper( outfile );

   if( !segy2suds( infile, outfile ) )
      exit( 1 );

   exit( 0 );
}

// ---------------------------------------------------------------------
int segy2suds( char infile[128], char outfile[128] ) {

   if( verbose ) {
      printf( " Input filespec: %s\n", infile );
      printf( "Output filespec: %s\n\n", outfile );
   }

   // Get the header
   if( !segy_hdr_get( infile ) )
      return( 0 );

   // Write out station info
   if( !suds_hdr_put( outfile ) )
      return( 0 );

   // Copy the traces
   if( !copy_traces( ) )
      return( 0 );

   st_close( outfil );
   fclose( infil );

   return( 1 );
}

// ---------------------------------------------------------------------
int segy_hdr_get( char infile[128] ) {
   register i;
   int bytes_read;
   void *ptr;

   // Open data file
   if( ( infil = fopen( infile, "r+b" ) ) == NULL ) {
      printf( "ERROR: Unable to open: %s\n", infile );
      return( 0 );
   }

   // Allocate a buffer
   if( ( ptr = malloc( 240 ) ) == NULL ) {
      printf( "ERROR: Not enough memory: segy_hdr_read()\n" );
      return( 0 );
   }

   // Read in the header
   if( ( bytes_read = fread( ptr, sizeof(char), 240, infil ) ) == 0 ) {
      printf( "ERROR: Unable to read: %s\n", infile );
      return( 0 );
   }

   // Copy header, lose the buffer
```

```
   memcpy( &hdr, ptr, sizeof( hdr ) );

   free( ptr );

   return( 1 );
}

// ---------------------------------------------------------------------
int suds_hdr_put( char outfile[128] ) {
   register i, j;
   int mon, day;
   SUDS_DETECTOR sd;
   SUDS_STATIONCOMP sc;
   SUDS_TRIGGERS st;

   // Open output file
   outfil = st_open( outfile, "w+b" );

   // DETECTOR struct
   st_init( DETECTOR, &sd );
   strcpy( sd.net_node_id, "SEGY2ST" );
   sd.versionnum = 1.00;
   st_put( &sd, DETECTOR, sizeof( SUDS_DETECTOR ), outfil );
   st_flush( outfil );

   // STATIONCOMP struct
   st_init( STATIONCOMP, &sc );
   sprintf( stn, "CHN%d", hdr.channel_number );
   strncpy( sc.sc_name.st_name, stn, 4 );
   sc.sc_name.st_name[4] = '\0';
   if( hdr.data_form != 0 ) {
      fprintf( stderr, "\nERROR: Unsupported data type: %d\n", hdr.data_form );
      return( 0 );
   }
   sc.data_type = 'i';
   sc.data_units = 'd';
   sc.channel = hdr.channel_number;
   sc.con_mvolts = hdr.scale_fac;
   sc.atod_gain = hdr.gainConst;
   mnday( hdr.day, isleap( hdr.year, 0 ), &mon, &day );
   sc.effective = (ST_TIME) make_mstime( hdr.year, mon, day,
      hdr.hour, hdr.minute, ((double)hdr.second +
      ((double)hdr.m_secs * 0.001 )) );
   st_put( &sc, STATIONCOMP, sizeof( SUDS_STATIONCOMP ), outfil );

   st_flush( outfil );

   return( 1 );
}

// ---------------------------------------------------------------------
int copy_traces( void ) {
   register i;
   int mon, day;
   long dlen;
   SUDS_DESCRIPTRACE _huge *dt;
   char _huge *ptr;
   int _huge *ptr1;

   // Create a DESCRIPTRACE structure
   dlen = hdr.sampleLength*sizeof(int);
   if( ( ptr = halloc( dlen+sizeof(SUDS_DESCRIPTRACE), 1 ) ) == NULL ) {
      printf( "ERROR: Not enough memory: %li\n",
```

```
            dlen+sizeof(SUDS_DESCRIPTRACE) );
      return( 0 );
   }
   dt = (SUDS_DESCRIPTRACE *)ptr;
   ptr1 = (int _huge *)(ptr+sizeof(SUDS_DESCRIPTRACE));

   // Initialize the structure
   st_init( DESCRIPTRACE, dt );
   strncpy( dt->dt_name.st_name, stn, 4 );
   dt->dt_name.st_name[4] = '\0';
   mnday( hdr.day, isleap( hdr.year, 0 ), &mon, &day );
   dt->begintime = make_mstime( hdr.year, mon, day,
      hdr.hour, hdr.minute, ((double)hdr.second +
      ((double)hdr.m_secs * 0.001 )) );
   dt->datatype = 'i';
   dt->length = hdr.sampleLength;
   dt->rate = 1.0 / ((float)hdr.deltaSample * 0.000001);
   dt->time_correct = 0.;
   dt->rate_correct = 0.;
   dt->mindata = -32767.0;
   dt->maxdata = 32767.0;
   dt->avenoise = 0.0;

   // Read in the data
   if( fread( ptr1, sizeof(int), (size_t)hdr.sampleLength, infil ) == 0 ) {
      printf( "ERROR: Unable to read: Input file\n" );
      return( 0 );
   }

   // Write out SUDS structure
   st_put( dt, DESCRIPTRACE, dlen+sizeof(SUDS_DESCRIPTRACE), outfil );
   st_flush( outfil );


   hfree( ptr );
   return( 1 );
}

// ---------------------------------------------------------------------
void upper( char buffer[128] ) {
   char *p;
   for( p = buffer; *p; p++ )
      *p = toupper( *p );
}

// ---------------------------------------------------------------------
long die( int in ) {
   exit( in );
}
```

```
/* This is the header for the segy trace header.  Reading bytes
  directly into this header will allow access to all of the fields.
*/
struct SegyHead {
  long lineSeq, reelSeq;  /* Sequence numbers within line and reel, resp. */
  long event_number;      /* Original field record number */
  long channel_number;    /* Trace number within the original field record.*/
  long energySourcePt;    /* Energy source point number */
  long cdpEns;            /* CDP ensemble number */
  long traceInEnsemble;   /* Trace number within CDP ensemble */
  short traceID;          /* Trace identification code:
        1 = seismic data        4 = time break  7 = timing
        2 = dead                5 = uphole      8 = water break
        3 = dummy               6 = sweep       9..., optional use */
  short vertSum, horSum;
  short dataUse;          /* 1 = production, 2 = test */
  long sourceToRecDist;
  long recElevation;
  long sourceSurfaceElevation;
  long sourceDepth;
  long datumElevRec, datumElemSource;
  long recWaterDepth, sourceWaterDepth;
  short elevationScale;
  short coordScale;
  long sourceLongOrX, sourceLatOrY;
  long recLongOrX, recLatOrY;
  short coordUnits;
  short weatheringVelocity;
  short subWeatheringVelocity;
  short sourceUpholeTime, recUpholeTime;
  short sourceStaticCor, recStaticCor;
  short totalStatic;
  short lagTimeA, lagTimeB;
  short delay;
  short muteStart, muteEnd;
  short sampleLength;     /* Number of samples in this trace */
  short deltaSample;      /* Sampling interval in microseconds. */
  short gainType;         /* 1 = fixed, 2 = binary, 3 = floating, 4... opt.*/
  short gainConst, initialGain;
  short correlated;       /* 1 = no, 2 = yes */
  short sweepStart, sweepEnd;
  short sweepLength;      /* in milliseconds */
  short sweepType;        /* 1 = linear, 2 = parabolic, 3 = exponential, 4... opt. */
  short sweepTaperAtStart, sweepTaperAtEnd;
  short taperType;
  short aliasFreq, aliasSlope;
  short notchFreq, notchSlope;
  short lowCutFreq, hiCutFreq;
  short lowCutSlope, hiCutSlope;
  short year, day, hour, minute, second;
  short timeBasisCode;
  short traceWeightingFactor;
  short phoneRollPos1, phoneFirstTrace, phoneLastTrace;
  short gapSize;
  short taperOvertravel;
  short extrash[10];
  long  samp_rate;
  short data_form, m_secs;
  short trigyear,trigday,trighour,trigminute,trigsecond,trigmills;
  float scale_fac;
  short inst_no;
  short not_to_be_used;
  long  num_samps;
```

```
  char extra[8];
}; /* end of segy trace header */
```

```
#include <stdio.h>
#include <math.h>
#include <string.h>
#include <malloc.h>
#include <memory.h>

#include <stdlib.h>
#include <direct.h>
#include <time.h>
#include "dmalloc.h"
#include "grstuff.h"
#include "DEMconst.h"
#include "DEMextrn.h"

#include "portab.h"          /* portability header */
#include "global.h"          /* definition of GLOBAL */
#include "globdef.h"         /* Global parameter definiton and declaration */
                             /* depending wether it is in MAIN or not */
                             /* Needs to know about window.h and GR???.h */
#include "message.h"
#include "data.h"
#include "datadef.h"
#include "nrutil.h"

#include "menu.h"
#include "note.h"
#define free(p) dfree(p)
#define malloc(p) dmalloc(p)
#define realloc(p,s) drealloc(p,s)
#define calloc(n,s) dcalloc(n,s)

#include "suds.h"
#include "mconst.h"

#define BLOCK_SIZE  4096

long current_position;


char *cvector();
void free_cvector();
unsigned int get_trace();
void decode_time ();
unsigned int get_suds ();
void wrt_org();
void wrt_evt();
void wrt_stcmp();
void sort();


/*-------------------------------------------------------------------------*/
/*
NAME
    get_trace

SYNOPSIS
    unsigned int get_trace(fp,length_ptr,str_ptr)
    FILE *fp;
    long *length_ptr, *str_ptr;

DESCRIPTION
    get_trace reads the tag of a suds struct and returns the data length
```

```
    and the struct length. The value of the function is the struct id on
    success, else the function has the value 0.
*/
/*-------------------------------------------------------------------------*/

unsigned int get_trace(fp,length_ptr,str_ptr)
FILE *fp;
long *length_ptr, *str_ptr;
{
    SUDS_STRUCTTAG suds_tag;

    current_position = ftell(fp);  /* current_position is a global variable */
    if (!fread(((void*) &suds_tag),sizeof(SUDS_STRUCTTAG),1,fp))
        return (0);

    *length_ptr = suds_tag.len_data;
    *str_ptr = suds_tag.len_struct;

    return(suds_tag.id_struct);
}



/*-------------------------------------------------------------------------*/
/*
NAME
    decode_time

SYNOPSIS
    void decode_time (begintime, year, month, day, hour, minute, second)
    double begintime;
    unsigned int *year, *month, *day, *hour, *minute;
    float *second;

DESCRIPTION
    decode_time converts the MS_time in year, month, day, hour, min and
    seconds.
*/
/*-------------------------------------------------------------------------*/

void decode_time (begintime, year, month, day, hour, minute, second)
double begintime;
unsigned int *year, *month, *day, *hour, *minute;
float *second;
{
   long time;
   struct tm *newtime;

   time = (long) begintime;
   newtime = gmtime (&time);
   *year = newtime->tm_year;
   *month = newtime->tm_mon + 1;
   *day = newtime->tm_mday;
   *hour = newtime->tm_hour;
   *minute = newtime->tm_min;
   *second = (float)newtime->tm_sec + (begintime - (float)time);
}

/*-------------------------------------------------------------------------*/
/*
NAME
    get_suds
```

```
SYNOPSIS
    unsigned int get_suds (input, length_ptr, suds_struct)
    FILE * input;
    long * length_ptr;
    void * *suds_struct;

DESCRIPTION
    Gets a suds structure from input stream.

*/
/*-------------------------------------------------------------------------*/

PRIVATE
unsigned int get_suds (input, length_ptr, suds_struct)
FILE * input;
long * length_ptr;
void * *suds_struct;
{
   SUDS_STRUCTTAG suds_tag;

   current_position = ftell (input);

   if (!fread (((void *) &suds_tag), sizeof(SUDS_STRUCTTAG), 1, input))
      return (0);

   if (*suds_struct != NULL) free (*suds_struct);
   *suds_struct = (void *) malloc ((int)suds_tag.len_struct);
   if (*suds_struct == NULL) {
      printf ("ERROR: Unable to allocate dynamic storage in GET_SUDS\n");
      exit(0);
   }

   if (!fread (*suds_struct, (int)suds_tag.len_struct, 1, input))
      return (0);

   *length_ptr = suds_tag.len_data;
   return (suds_tag.id_struct);
}

/*-------------------------------------------------------------------------*/
/*
NAME
    wrt_org

SYNOPSIS
    void wrt_org(data_in,index,org)
    DT_HEADER ***data_in;
    SUDS_ORIGIN *org;
    int index;

DESCRIPTION
    wrt_org puts the data from the struct SUDS_ORIGIN in the ISAM data base.

*/
/*-------------------------------------------------------------------------*/

void wrt_org(data_in,index,org)
DT_HEADER ***data_in;
SUDS_ORIGIN *org;
int index;
{
    int yer, mon, day, our, min;
    float sec;
```

```
    float x,y,z;
    TIME org_time;
    DT_HEADER **data;

    data = *data_in;
    decode_time(org->orgtime, &yer, &mon, &day, &our, &min, &sec);
    org_time.yr = yer;
    org_time.mo = mon;
    org_time.day= day;
    org_time.hr = our;
    org_time.mn = min;
    org_time.sec= sec;
    org_time.base = GMT;
    x = (float) org->or_long;
    y = (float) org->or_lat;
    z = - (float) org->depth;
    dt_head_access(data[index], EVT_TIME, (void*)&org_time);
    dt_head_access(data[index], EVT_X,(void*)&x);
    dt_head_access(data[index], EVT_Y,(void*)&y);
    dt_head_access(data[index], EVT_Z,(void*)&z);

    *data_in = data;
}

/*-------------------------------------------------------------------------*/
/*
NAME
    wrt_evt

SYNOPSIS
    void wrt_org(data_in,index,org)
    DT_HEADER ***data_in;
    SUDS_EVENT *evt;
    int index;

DESCRIPTION
    wrt_evt puts the data from the struct SUDS_EVENT in the ISAM data base.

*/
/*-------------------------------------------------------------------------*/

void wrt_evt(data_in,index,evt)
DT_HEADER ***data_in;
SUDS_EVENT *evt;
int index;
{
    char evtype[6];
    DT_HEADER **data;

    /* e=earthquake
       E=explosion
       n=nuclear
       i=icequake
       b=b_type
       n=net
       r=regional
       t=teleseism
       c=calibration
       n=noise        */

    data = *data_in;
    evtype[0] = evt->ev_type;
    dt_head_access(data[index], EVT_CTPE, (void*)&evtype[0]);
```

```
    *data_in = data;
}

/*-------------------------------------------------------------------------*/
/*
NAME
    wrt_stcmp

SYNOPSIS
    void wrt_stcmp(data_in,index,stcp)
    DT_HEADER ***data_in;
    int index;
    SUDS_STATIONCOMP *stcp;

DESCRIPTION
    wrt_stcmp puts the data from the struct SUDS_STATIONCOMP in
    the ISAM data base.

*/
/*-------------------------------------------------------------------------*/

void wrt_stcmp(data_in,index,stcp)
DT_HEADER ***data_in;
int index;
SUDS_STATIONCOMP *stcp;
{
    DT_HEADER **data;
    char code[6];
    float x,y,z;
    data = *data_in;

    strcpy(code,stcp->sc_name.st_name);
    dt_head_access(data[index], STA_CODE, (void*)&code[0]);

    x = stcp->st_long;
    y = stcp->st_lat;
    z = stcp->elev;
    /* convert it into km */
    z /= 1000.0;

    dt_head_access(data[index], STA_X, (void*)&x);
    dt_head_access(data[index], STA_Y, (void*)&y);
    dt_head_access(data[index], STA_Z, (void*)&z);

    *data_in = data;
}

/*-------------------------------------------------------------------------*/
/*
NAME
    sort

SYNOPSIS
    void sort(n,ra)
    int n,ra[];

DESCRIPTION
    sorts an array ra[1..n] into ascending numerical order using the
    Heapsort algorithm. n is input; ra is replaced on output by its
    sorted rearrangement.

*/
/*-------------------------------------------------------------------------*/
```

```
void sort(n,ra)
int n,ra[];
{
        int l,j,ir,i;
        int rra;

        l=(n >> 1)+1;
        ir=n;
        for (;;) {
                if (l > 1)
                        rra=ra[--l];
                else {
                        rra=ra[ir];
                        ra[ir]=ra[1];
                        if (--ir == 1) {
                                ra[1]=rra;
                                return;
                        }
                }
                i=l;
                j=l << 1;
                while (j <= ir) {
                        if (j < ir && ra[j] < ra[j+1]) ++j;
                        if (rra < ra[j]) {
                                ra[i]=ra[j];
                                j += (i=j);
                        }
                        else j=ir+1;
                }
                ra[i]=rra;
        }
}

/*-------------------------------------------------------------------------*/
/*
NAME
    io_read_suds

SYNOPSIS
    int io_read_suds(data_in)
    DT_HEADER ***data_in;

DESCRIPTION
    io_read_suds reads the data from a SUDS file into the isam data base.
*/
/*-------------------------------------------------------------------------*/
int io_read_suds(data_in)
DT_HEADER ***data_in;
{
    boolean io_done;          /* if true -> return */
    int io_res;
    char file_spec[20];
    char file_ret[FNAME_LEN];

    DT_HEADER **data;         /* array of pointers to DT_HEADERS */
    BYTE fname[FNAME_LEN];    /* input filename */
    FILE *fp;                 /* input file ptr */
    int sc_type;
    int mplx;
    char *string_end;         /* pointer to possible CR */
    float t_samp;
    int ch;                   /* char. buffer to get file length */
```

```
    int ndat;
    int max_ndat;
    int no_traces1;
    int i,j,k;
    int plot_on;
    int ax_type;
    int mrk_type;
    int index;
    AXIS axis;
    float x;
    float *tr;
    TIME time_start;
    AX_GNLBL toplabel;
    char text[30],text1[10];
    int yer, mon, day, our, min;
    float sec;

    int *cha_list,*cha_list1; /* channel list */
    int no_entries;           /* no. of channels selected */
    int chan,chan_1;

    MenuRecType note;

    char magic_number, machinetype;
    long datalength,st_length;
    unsigned int st_type;
    void *st_ptr;             /* pointer to suds_struct */
    SUDS_STATIONCOMP *stcp;
    SUDS_ORIGIN *org;
    SUDS_DESCRIPTRACE *strc;
    SUDS_EVENT *evt;
    int orgflag,evtflag,scflag;
    long blocksize, datasize, readsize, bytes_remaining;
    char *in_buffer;

    set_help("SUDS_IO");
    io_done =false;
    do {
       io_res = io_text(
       " Enter name of SUDS file\n*.sqz\n_\nCurrent data path is %s ",file_spec,
                                                                    Data_Path);
       if (io_res == ESCAPED) {
           return(ESCAPED);
       }

       ChooseFile(" SUDS ",file_spec,file_ret,&err);
       if(err) {
           men_message(" Not a valid selection ",ACKN);
       }
       else
           io_done = true;
    } while(!io_done);

    /* now, hopefully a valid filename has been chosen */
    strcpy(fname,file_ret);


   /* Check the magic number and machine type*/
   if ((fp = fopen(fname,"r+b"))!=NULL){
     fseek (fp, 0L, SEEK_SET);
     fread (((void *) &magic_number), sizeof(char), 1, fp);
     if (magic_number != ST_MAGIC) {
        men_message("ERROR: File must be in SUDS format.",ACKN);
```

```
      return(ERROR);
   }
   fread (((void *) &machinetype), sizeof(char), 1, fp);
   if (machinetype != MACHINE[0]) {
      men_message("ERROR: Invalid machine type.  Machine id = %c",ACKN,
                                                          machinetype);
      return(ERROR);
   }
   fseek (fp, 0L, SEEK_SET);

 /* count number of traces  */

   no_entries=0;

   while(st_type = get_trace(fp, &datalength, &st_length)) {
        if (st_type == DESCRIPTRACE) no_entries++;

        fseek(fp,st_length ,SEEK_CUR);
        fseek(fp,datalength,SEEK_CUR);
   }
   fseek (fp, 0L, SEEK_SET);

   if (no_entries == 0) {
       men_message("No traces in SUDS file!",ACKN);
       fclose(fp);
       return(ESCAPED);
   }

   data=*data_in;

   io_res = io_get_chalist("
   Enter channel list to load:(;loads all)\n;\n_\nfile: %s\nno. of channels: %d",
                           &cha_list,no_entries,no_entries,fname,no_entries);
   if (io_res == ESCAPED) {
       goto do_escape_2;
   }

   no_traces1 = *(cha_list);

   cha_list1 = (int*)calloc(no_traces1+1,sizeof(int));

   for (i=0;i<=no_traces1;i++) cha_list1[i] = cha_list[i];

   if(dt_free_all(data) == ERROR) {
        return(ERROR);
   }

   data = dt_alloc_head(no_traces1);
   if(data == NULL) {
       free((char*)cha_list);
       free((char*)cha_list1);
       data = dt_alloc_head(0);
       *data_in=data;
       if(data == NULL) {
           men_message("FATAL ALLOCATION ERROR! EXIT PITSA",ACKN);
           exit(1);
       }
       else
           goto do_escape_2;
   }

  /*
   now the environment has changed, we will need a plot all when
```

```
 we get back to the pitsa main routine.
*/
Need_Plot_All = TRUE;

/* sort the channel list into ascending order */
if (no_traces1 > 1)
    sort(no_traces1,cha_list);

max_ndat = Max_trlen1;

/* Now get the suds_structs from input file */
st_ptr = NULL;
index = 0;
chan = 1;

orgflag = FALSE;
scflag = FALSE;
evtflag = FALSE;

note = men_note_init("Reading channel: 0000");

while ((st_type = get_suds(fp, &datalength, &st_ptr)) &&
       (index < (no_entries))) {
    switch (st_type) {
        case SDS_ORIGIN:
            org = (SUDS_ORIGIN*) st_ptr;
            orgflag = TRUE;
            break;

        case STATIONCOMP:
            stcp = (SUDS_STATIONCOMP*) st_ptr;
            scflag = TRUE;
            break;

        case EVENT:
            evt = (SUDS_EVENT*) st_ptr;
            evtflag = TRUE;
            break;

        case DESCRIPTRACE:
            strc = (SUDS_DESCRIPTRACE*) st_ptr;
            index++;
            if ((index)==cha_list[chan] && chan<=no_traces1) {
                k=0;
                do {
                    k++;
                } while (cha_list1[k] != cha_list[chan]);

                chan_1 = k-1;
                men_note_write("Reading channel: %4d",note,chan_1+1);

                /* work up event */
                if (orgflag) wrt_org(&data,chan_1, org);
                if (evtflag) wrt_evt(&data,chan_1, evt);

                /* work up stationcomp */
                if (scflag) wrt_stcmp(&data,chan_1,stcp);

                decode_time(strc->begintime, &yer, &mon, &day,
                                                   &our, &min, &sec);
                time_start.yr = yer;
                time_start.mo = mon;
                time_start.day= day;
```

```
time_start.hr = our;
time_start.mn = min;
time_start.sec= sec;
time_start.base = GMT;
dt_head_access(data[chan_1], RCD_TIME, (void*)&time_start);
strcpy(text,"SUDS: 19");
sprintf(text1,"%d/",yer);
strcat(text,text1);
sprintf(text1,"%02d/",mon);
strcat(text,text1);
sprintf(text1,"%02d",day);
strcat(text,text1);
sprintf(text1," %02d:",our);
strcat(text,text1);
sprintf(text1,"%02d:",min);
strcat(text,text1);
sprintf(text1,"%02d.",(int)sec);
strcat(text,text1);
sprintf(text1,"%03d",(int)(1000*(sec-(int)sec)));
strcat(text,text1);
strcpy(toplabel.text,text);
toplabel.path = toRIGHT;
dt_head_access(data[chan_1], PLT_LBT,(void*)&toplabel);
ndat = (int)strc->length;
if (ndat>max_ndat) {
    men_message("trace has more than max_ndat samples",
                                                    ACKN);
    goto do_escape;
}

dt_head_access(data[chan_1],RCD_NDAT, (void*)&ndat);
t_samp = 1.0/strc->rate;
dt_head_access(data[chan_1],RCD_SMP, (void*)&t_samp);
plot_on = 1;

dt_head_access(data[chan_1],PLT_ON,(void *)&plot_on);
ax_type = FRAME;
dt_head_access(data[chan_1],PLT_XTP,(void *)&ax_type);
mrk_type = SLDLNE;
dt_head_access(data[chan_1],PLT_MRK,(void *)&mrk_type);
sc_type = LINLIN;
dt_head_access(data[chan_1],PLT_STP,(void *)&sc_type);
mplx = SNGTS;
dt_head_access(data[chan_1],RCD_MPX,(void *)&mplx);

plt_init_axis(&axis);

strcpy(axis.ax_gnlbl.text,stcp->sc_name.st_name);
axis.ax_gnlbl.path = toDOWN;

dt_head_access(data[chan_1],Y_AXIS,(void *)&axis);

axis.ax_gnlbl.path = toRIGHT;

strcpy(axis.ax_sclbl.fmt,"%5.0f");

dt_time_axis(&axis);

dt_head_access(data[chan_1],X_AXIS,(void *)&axis);

if(dt_alloc_trace(data[chan_1]) == ERROR) {
    men_message("ERROR: can't allocate memory for trace %d",
                ACKN,index);
```

```
                    for(i=0; i<(chan_1);i++)
                        dt_free_trace(data,i);
                    /* re-initialize data structures and leave */
                    goto do_escape;
                }

                tr = dt_trace_access(data,chan_1,0);
                j = 0;
                in_buffer = (char*)calloc(BLOCK_SIZE,sizeof(char));

                datasize = datalength/strc->length;
                blocksize = ((BLOCK_SIZE) / datasize) * datasize;
                for (bytes_remaining = datalength;
                            bytes_remaining;bytes_remaining-=readsize) {
                    readsize = (bytes_remaining >= blocksize)?
                                                    blocksize:bytes_remaining;
                    readsize = fread(in_buffer,sizeof(char),(int)readsize,
                                                                    fp);
                    for (i = 0; i<readsize; i+=datasize, j++) {
                        if (j < ndat) {
                           switch (strc->datatype) {
                              case 'r':
                                 *(tr+j)=*((short *)&(in_buffer[i]))/16.0;
                                 break;
                              case 's':
                                 *(tr+j)=(float)*((short *)&(in_buffer[i]));
                                 break;
                              case 'l':
                                 *(tr+j)=(float)*((long *)&(in_buffer[i]));
                                 break;
                              case 'f':
                                 *(tr+j)=*((float *)&(in_buffer[i]));
                                 break;
                              default:
                                 printf("ERROR: Unknown datatype in ");
                                 printf("struct descriptrace of %c\n",
                                                        strc->datatype);
                                 exit(0);
                           }
                           *(tr+j) -= strc->avenoise;
                        }
                    }
                }
                free((char*)in_buffer);
                chan++;
            }
            else
                fseek(fp,datalength,SEEK_CUR);
            break;
        }
    }

  men_note_erase(note);
  fclose(fp);
 }
 *data_in = data;
 free((char*)cha_list);
 free((char*)cha_list1);
 return(GOOD);

do_escape:
    dt_free_head(data,0);
```

```
    free((char*)data);
    data = dt_alloc_head(0);
    free((char*)cha_list);
    free((char*)cha_list1);
    *data_in = data;
    return(ESCAPED);

do_escape_2:
    free((char*)cha_list);
    free((char*)cha_list1);
    *data_in = data;
    return(ESCAPED);
}
```

```
#include <stdio.h>
#include <ctype.h>
#include <math.h>
#include <string.h>
#include <memory.h>
#include <malloc.h>

#include <direct.h>
#include <stdlib.h>
#include "grstuff.h"
#include "dmalloc.h"
#include "DEMconst.h"
#include "DEMextrn.h"
#define free(p) dfree(p)
#define malloc(p) dmalloc(p)
#define realloc(p,s) drealloc(p,s)
#define calloc(n,s) dcalloc(n,s)

#include "portab.h"
#include "global.h"
#include "globdef.h"
#include "message.h"


#include "menu.h"
#include "note.h"

#include "data.h"
#include "datadef.h"
#include "suds.h"

#define BASEJDN         2440588                 /* 1 January 1970       */

/*--------------------------------------------------------------------------*/
/*
NAME
    time2mstime

SYNOPSIS
    MS_TIME time2mstime(t)
    TIME *t;

DESCRIPTION
    converts time from Pitsa time format to MS_TIME.
*/
/*--------------------------------------------------------------------------*/

MS_TIME time2mstime(t)
TIME *t;
{
    long dt;
    MS_TIME daytime;
    extern long julday();

    if (t->yr < 1900) t->yr += 1900;

    dt = julday(t->mo,t->day,t->yr);
    dt -= BASEJDN;      /* days */
    dt *= 86400;        /* seconds */
    daytime = (MS_TIME)t->sec + (MS_TIME)(t->mn*60 + t->hr*3600);
    daytime += dt;
    return (daytime);
}
```

```
/*-------------------------------------------------------------------------*/
/*
NAME
    io_write_suds

SYNOPSIS
    int io_write_suds(data_in)
    DT_HEADER ***data_in;

DESCRIPTION
    io_write_suds writes PITSA data into a SUDS file.
*/
/*-------------------------------------------------------------------------*/

int io_write_suds(data_in)
DT_HEADER ***data_in;
{
    boolean io_done;          /* if true -> return */
    int io_res;

    DT_HEADER **data;     /* array of pointers to DT_HEADERS */
    char fname[FNAME_LEN]; /* filename */
    int *cha_list;        /* channel list */
    int no_chan_tot;      /* no. of channels in filesystem */
    int no_chan;          /* no of channels selected */
    int i,j;
    int count;            /* counter for processed files */
    int index;
    FILE *fo;             /* output file data trace */
    int ndat;             /* number of points in trace */
    float *trace;         /* pointer to data */
    int size;
    char answer;
    int ctype;            /* coordinate type in isam format */
    TIME t;
    SUDS_ORIGIN org;
    SUDS_DESCRIPTRACE dstr;
    SUDS_STATIONCOMP stcmp;
    SUDS_STRUCTTAG tag;
    MS_TIME time2mstime();


    data = *data_in;

    set_help("WR_SUDS");
    io_res = io_text(
    " Enter name of SUDS file \n\n_\nWithout extension!\nCurrent data path is %s ",
                                                        fname, Data_Path);

    if (io_res == ESCAPED) {
        return(ESCAPED);
    }

    /* if there is an extension present, strip it off */
    i = 0;
    while((fname[i] != '\0') && (fname[i] != '.')) {
        i++;
    }
    fname[i] = '\0';

    strcat(fname,".sqz");
```

```
    strcpy(Long_Fname,Data_Path);
    strcat(Long_Fname,fname);


    if(access(Long_Fname,00) >= 0) {
        set_help("WR_OVWRITE");
        io_done =false;
        do {
           io_res = io_text("File %s already exists. Overwrite (y/n) ?\nn",
                            text,Long_Fname);
           if (io_res == ESCAPED) {
               goto do_escape;
           }
           answer = text[0];
           if (islower(answer)) answer = toupper(answer);

           if(answer != 'Y') {
              /* either ignore or non valid selection */
              goto do_escape;
           } else {
               if(answer == 'Y') io_done = true;
           }
        } while(!io_done);

    }

    strcpy(Long_Fname,Data_Path);
    strcat(Long_Fname,fname);

    if ((fo = fopen(Long_Fname,"wb")) == NULL) {
        men_message("Error on opening %s",ACKN,Long_Fname);
        goto do_escape;
    }

    no_chan_tot = dt_no_channels(data);

    io_res = io_get_chalist(
    " Enter channel list to write: (; writes all)\n;\n_\nfile: %s\nno. of channels: %d
",
                        &cha_list,no_chan_tot,no_chan_tot,Long_Fname,no_chan_tot);

    if (io_res == ESCAPED) {
        goto do_escape;
    }
    no_chan = *(cha_list);

    tag.sync = ST_MAGIC;
    tag.machine = MACHINE[0];
    /* write the struct SUDS_ORIGIN */

    t = *((TIME*)dt_head_access(data[0],EVT_TIME,(void*)NULL));
    if (t.yr != 0) {
        ctype = *((int*)dt_head_access(data[0],EVT_CTPE,(void*)NULL));
        if (ctype != DGDGKM) {
            men_message(" Coordinates are not in (deg, deg, km),\nset to zero",
                                                                    ACKN);
            org.or_lat = 0.0;
            org.or_long = 0.0;
            org.depth = 0.0;
        } else {
            org.or_lat =(double)*((float*)dt_head_access(data[0],EVT_Y,
                                                    (void*)NULL));
            org.or_long = (double)*((float*)dt_head_access(data[0],EVT_X,
```

```
                                                                (void*)NULL));
        org.depth = -*((float*)dt_head_access(data[0],EVT_Z,(void*)NULL));
    }
    org.orgtime = time2mstime(&t);

    tag.id_struct = SDS_ORIGIN;
    tag.len_struct = sizeof(SUDS_ORIGIN);
    tag.len_data = 0.0;

    size = sizeof(SUDS_STRUCTTAG);
    fwrite(&tag,size,1,fo);
    size = tag.len_struct;
    fwrite(&org,size,1,fo);
}

/* write the selected traces */
for (i=1;i<=no_chan;i++) {
    index = cha_list[i] - 1;
    dstr.length = *((int*)dt_head_access(data[index],RCD_NDAT,(void*)NULL));
    dstr.rate = *((float*)dt_head_access(data[index],RCD_SMP,(void*)NULL));
    dstr.datatype = 'f';
    t = *((TIME*)dt_head_access(data[index],RCD_TIME,(void*)NULL));
    dstr.begintime = time2mstime(&t);

    trace = dt_trace_access(data,index,0);
    dstr.mindata = *trace;
    dstr.maxdata = dstr.mindata;
    for (j=1;j<=dstr.length;j++) {
        dstr.mindata = min(dstr.mindata,*(trace+j));
        dstr.maxdata = max(dstr.maxdata,*(trace+j));
    }

    strcpy(dstr.dt_name.st_name,(char*)dt_head_access(data[index],STA_CODE,
                                                        (void*)NULL));
    strcpy(stcmp.sc_name.st_name,dstr.dt_name.st_name);
    strcpy(stcmp.sc_name.network," ");

    ctype = *((int*)dt_head_access(data[0],STA_CTPE,(void*)NULL));
    if (ctype != DGDGKM) {
        men_message(" Coordinates are not in (deg, deg, km),\nset to zero",
                                                                    ACKN);
        stcmp.st_lat = 0.0;
        stcmp.st_long = 0.0;
        stcmp.elev = 0.0;
    } else {
        stcmp.st_lat = (double)*((float*)dt_head_access(data[index],STA_Y,
                                                        (void*)NULL));
        stcmp.st_long = (double)*((float*)dt_head_access(data[index],STA_X,
                                                        (void*)NULL));
        stcmp.elev = *((float*)dt_head_access(data[index],STA_Z,
                                                        (void*)NULL));
        stcmp.elev *= 1000.0; /* in meters */
    }
    stcmp.data_type = 'f';


    /* write the suds file */
    tag.id_struct = STATIONCOMP;

    tag.len_struct = sizeof(SUDS_STATIONCOMP);
    tag.len_data = 0.0;

    size = sizeof(SUDS_STRUCTTAG);
```

```
        fwrite(&tag,size,1,fo);

        size = sizeof(SUDS_STATIONCOMP);
        fwrite(&stcmp,size,1,fo);

        tag.id_struct = DESCRIPTRACE;

        tag.len_struct = sizeof(SUDS_DESCRIPTRACE);
        tag.len_data = dstr.length*sizeof(float);

        size = sizeof(SUDS_STRUCTTAG);
        fwrite(&tag,size,1,fo);

        size = sizeof(SUDS_DESCRIPTRACE);
        fwrite(&dstr,size,1,fo);

        size = tag.len_data;
        fwrite(trace,sizeof(float),(int)dstr.length,fo);
    }
    fclose(fo);
    free((char*)cha_list);
    *data_in = data;
    return(GOOD);

do_escape:
    fclose(fo);
    free((char*)cha_list);
    *data_in = data;
    return(ESCAPED);
}
```

```
#include <stdio.h>          /* C library  */
#include <string.h>
#include <math.h>
#include <malloc.h>

#include <stdlib.h>
#include "grstuff.h"

#include "portab.h"
#include "global.h"    /* macros needed for globdef.h */
#include "globdef.h"   /* all PITSA globals defined */
#include "data.h"
#include "message.h"


/*-------------------------------------------------------------------------*/
/*
NAME
    plt_particle_motion

SYNOPSIS
    void plt_particle_motion(data_in,motion_type)
    DT_HEADER ***data_in; pointer to our data
    int motion_type;      what kind of particel motion plot to do,
                          0 = 2D snake
                          1 = 3D snake
                          2 = windows
                          3 = vectors

AUTHOR(S):
    J. Johnson, F. Scherbaum

DESCRIPTION
    This routine will get ready to do a particel motion plot.  First it has
    to get the two traces to use for x and y, and then needs to zoom up on
    those two traces.  Then one of the particle motion plot routines will
    be called with this information.

*/
/*-------------------------------------------------------------------------*/

void plt_particle_motion(data_in,motion_type)
DT_HEADER ***data_in;
int motion_type;          /* 0=2d-snake, 1=3d-snake, 2=windows, 3=vectors  */
{
    DT_HEADER **data;     /* array of pointers to DT_HEADERS               */
    int *cha_list;        /* channel list of traces to use for x and y     */
    int no_channels;      /* no channels currently loaded                  */
    int status;           /* used for ESCAPED and NEED_HELP                */
    int done;             /* boolean for input loops                       */
    int i1,i2;            /* index of zoom up on traces                    */
    int selection_method; /* selection method used for zooming in on trace */
    int need_chan;

    dmalloc_name("plt_pmtn");

    /* are there at least two channels? */
    data = *data_in;
    no_channels = dt_no_channels(data);
    if (motion_type != 1) {
        if (no_channels < 2) {
            men_message("There are only %d channel(s)\nmust be at least 2",
                                                        ACKN, no_channels);
```

```
            return;
        }
    } else {
        if (no_channels < 3) {
            men_message("There are only %d channel(s)\nmust be at least 3",
                                                        ACKN, no_channels);
            return;
        }
    }

    /* get the two channels for the x and y (and z if 3D-snake) */
    status = 0;
    done = FALSE;
    while (!done) {
        if (motion_type != 1) {
            status = io_get_chalist("Enter channels for x and y:",
                                    &cha_list, no_channels, no_channels);
        } else {
            status = io_get_chalist("Enter channels for x, y and z:",
                                    &cha_list, no_channels, no_channels);
        }
        if (status == ESCAPED) {
            free((char*)cha_list);
            return;
        }

        /* make sure we have two channels to work with */
        need_chan = 2;
        if (motion_type == 1)
            need_chan = 3;
        if (cha_list[0] != need_chan) {
            free((char*)cha_list);
            men_message("Please enter %d and ONLY %d channels!",
                                            need_chan,need_chan,ACKN);
        } else {
            /* everything is ok, we can continue on */
            done = TRUE;
        }
    }

    /* now we zoom up on the traces */
    spr_just_zoom(data_in,&cha_list,&i1,&i2,&selection_method,&status);
    if (status == ESCAPED) {
        free((char*)cha_list);
        return;
    }

    /* now we are ready to do the particle motion */
    switch (motion_type) {
        case 0: { /* 2d-snake */
            plt_particle_snake(data_in,cha_list,i1,i2);
            break;
        }
        case 1: { /* 3d-snake */
            plt_particle_snake(data_in,cha_list,i1,i2);
            break;
        }
        case 2: { /* windows */
            /*
            plt_particle_windows(data_in,cha_list,i1,i2);
            */
            break;
        }
```

```
        case 3: { /* vectors */
            /*
            plt_particle_vectors(data_in,cha_list,i1,i2);
            */
            break;
        }
    }
    free((char *)cha_list);
    return;
}
```

```
#include <stdio.h>          /* C library  */
#include <string.h>
#include <memory.h>
#include <malloc.h>
#include <math.h>

#include  <stdlib.h>
#include <direct.h>
#include "dmalloc.h"
#include "grstuff.h"
#include "DEMconst.h"
#include "DEMextrn.h"


#include "portab.h"         /* portability header */
#include "global.h"         /* definition of GLOBAL */
#include "globdef.h"        /* Global parameter definiton and declaration */
                            /* depending wether it is in MAIN or not */
                            /* Needs to know about window.h and GR???.h */
#include "nr.h"
#include "nrutil.h"
#include "message.h"

#include "menu.h"
#include "note.h"
#define free(p) dfree(p)
#define malloc(p) dmalloc(p)
#define realloc(p,s) drealloc(p,s)
#define calloc(n,s) dcalloc(n,s)

#include "data.h"
#include "datadef.h"

extern float utl_cova();
extern float spr_cosine();

static MenuItem pol_filt_Items[3] =
     {
         {M_ITEM,"2 components (N,E)",0,MRT_NULL,MRT_NULL,M_NULL},
         {M_ITEM,"3 components (Z,N,E)",0,MRT_NULL,MRT_NULL,M_NULL},
         {M_LAST,"",0,MRT_NULL,MRT_NULL,M_NULL}
     };

static MenuItem what2show_Items[4] =
     {
         {M_ITEM,"eigenvector + Rect.-lin.",0,MRT_NULL,MRT_NULL,M_NULL},
         {M_ITEM,"eigenvector + LEV Projection",0,MRT_NULL,MRT_NULL,M_NULL},
         {M_ITEM,"Filtered Traces + Rect.-lin.",0,MRT_NULL,MRT_NULL,M_NULL},
         {M_LAST,"",0,MRT_NULL,MRT_NULL,M_NULL}
     };

/*-------------------------------------------------------------------------*/
/*
NAME
    spr_polar_flt

SYNOPSIS
    COUNT spr_polar_flt(data_in)
    DT_HEADER ***data_in;

DESCRIPTION
    Polarization filtering
    Ref.: Montalbetti, J. F., and E. R. Kanasewich, Enhamcement of
```

```
    teleseismic body phases with a polarization filter, Geophys. J.
    R. astr. Soc., 21, 119 - 129, 1970.

*/
/*-------------------------------------------------------------------------*/

COUNT spr_polar_flt(data_in)
DT_HEADER ***data_in;
{

    int which;
    int show;

    COUNT menu_poll;      /* return value of menu polling routine            */
                          /* number of submenus used for selection + 1,or    */
                          /* ERROR on error, ESCAPED on ESC, NEED_HELP on F1,*/
                          /* DO_HCPY on F2                                   */

    BOOL show_automatic=TRUE; /* TRUE -> menu shows up immediately           */
                              /* FALSE -> menu shows only after event        */

    BOOL allow_cmd_mode=TRUE; /* TRUE -> menu can be turned off through      */
                                  /* program mode */
                              /* FALSE -> menu is always active              */

    MenuRecType note;

    DT_HEADER **buffer;  /* array of pointers to DT_HEADERS */
    DT_HEADER **data;    /* array of pointers to DT_HEADERS */
    FRCT view_temp;
    int *cha_list1;      /* channel list1 */
    int cha_list2[6];    /* channel list2 */
    int io_res;
    int no_entries;
    int io_done;
    int index;           /* channel index */
    int i,j,k;
    int ndat;            /* number of points in data trace */
    int int_arg;
    int note_on = FALSE;
    int win_len;         /* window length in data points   */
    float frac;          /* taper fraction defining the center fraction   */
                         /* of the window unaffected by possible tapering */
    int n1,n2,n3,n4;     /* taper fix points */
    float taper;
    float exp_n, exp_j, exp_k; /* eigenvalue, rectilinearity, eigenvector */
                               /* exponents (see referenced paper)        */

    float *temp1;        /* buffer containing windowed and tapered data   */
    float *temp2;        /* buffer containing windowed and tapered data   */
    float *temp3;        /* buffer containing windowed and tapered data   */
    float cov_z_z, cov_n_n, cov_e_e;  /* covariance values */
    float cov_z_n, cov_z_e, cov_n_e;
    float **eig_vect;    /* eigenvector matrix     */
    float ** cov_mat;    /* covariance matrix      */
    float *eig_val;      /* eigenvalues            */
    float *diag_val;     /* diagonal elements of the tri-diagonal matrix  */
    float *offdiag_val;  /* off diagonal elements  */
    float *diag_val2;    /* diagonal elements of the tri-diagonal matrix  */
    float *offdiag_val2; /* off diagonal elements  */
    float **eig_vect2;   /* eigenvector matrix     */
    float ** cov_mat2;   /* covariance matrix      */
    float *eig_val2;     /* eigenvalues            */
```

```
    float *tr1;          /* temporary trace buffer */
    float *tr2;          /* temporary trace buffer */
    float *tr3;          /* temporary trace buffer */
    float *b0;           /* temporary trace buffer */
    float *b1;           /* temporary trace buffer */
    float *b2;           /* temporary trace buffer */
    float *b3;           /* temporary trace buffer */

    float rct_lin;       /* rectilinearity         */
    float dir_1, dir_2, dir_3; /* directivity      */
    int half_width;      /* half width of moving average */
    float ra;
    float a,b,c,d,e,f;
    int win_shift;       /* no. of points to shift analysis window */
                         /* between filter steps   */
    float x1,x2;         /* boundary values cfor interpolation */

    AXIS axis;


    data = *data_in;

    if (data[0] == NULL) {
        men_message("No data in memory",ACKN);
        return(ERROR);
    }

    no_entries = dt_no_channels(data);

    /* code to select 2 or 3 comp filter */

    set_help("POLFILTER_TYPE");
    if((menu_poll = men_pu_get(" POLARIZATION FILTER ",
        pol_filt_Items,show_automatic,allow_cmd_mode)) == ESCAPED)
            return(menu_poll);
    which = Menu_Path[1];

    if (which == 1) {
        if (no_entries < 2) {
            men_message("There are only %d channels",ACKN,no_entries);
            return(ERROR);
        }
    } else {
        if (no_entries < 3) {
            men_message("There are only %d channels",ACKN,no_entries);
            return(ERROR);
        }
    }

    ra = 180.0/PI;

    io_done = FALSE;
    while (!io_done) {
        if (which == 1) { /* 2 comp */
            io_res = io_get_chalist(
                " Enter X and Y channels for polarization filter\n1,2\n_\nno. of chann
els %d",
                &cha_list1,no_entries,no_entries,no_entries);

            if (io_res == ESCAPED) {
                goto do_escape_2;
            }
```

```
            if (cha_list1[0] == 2) {
                io_done = TRUE;
            } else {
                men_message("Please enter two and only two channels",ACKN);
            }
        } else {
            io_res = io_get_chalist(
                " Enter X, Y, Z channels for polarization filter\n1,2,3\n_\nno. of cha
nnels %d",
                &cha_list1,no_entries,no_entries,no_entries);

            if (io_res == ESCAPED) {
                goto do_escape_2;
            }

            if (cha_list1[0] == 3) {
                io_done = TRUE;
            } else {
               men_message("Please enter three and only three channels",ACKN);
            }
        }
    }

    ndat = *((int *)dt_head_access(data[cha_list1[1]-1],RCD_NDAT,(void *)NULL));

    set_help("POL_FILTER_WIN_LEN");
    io_res = io_meta_text(
        "Enter window length in points (-1 takes all)\n25\n_\nno. of points %d",
        text,"re-enter window length",io_ver_int_gt_x_lt_y, &int_neg_inf, &ndat,
                                                                    ndat);

    if (io_res == ESCAPED) {
        goto do_escape_2;
    }

    win_len = atoi(text);
    win_shift =1;
    if(win_len != -1)
    {
        set_help("POL_FILTER_SHIFT");
        io_res = io_meta_text(
            "Enter shift/filter step in points\n1",
            text,
            "re-enter shift/filter step",
            io_ver_int_gt_x_lte_y, &int_zero, &win_len);

        if (io_res == ESCAPED) {
            goto do_escape_2;
        }
        win_shift = atoi(text);
    }
    if (win_shift < 1) win_shift = 1;

    set_help("POL_FILTER_TAPER_FRAC");
    io_res = io_meta_text(
        "Enter taper fraction (0.0 to 1.0)\n.2",text,"re-enter taper fraction",
                            io_ver_float_gte_x_lte_y, &float_zero, &float_one);
    if (io_res == ESCAPED) {
        goto do_escape_2;
    }
    frac = atof(text);
    /* undistorted portion in center part of window */
```

```
    frac = 1.0 - frac;

    set_help("POL_FILTER_EXP_N");
    int_arg = 10;
    io_res = io_meta_text(
        "Enter eigenvalue exponent n\n1",text,"re-enter eigenvalue exponent n",
                                    io_ver_int_gte_x_lte_y, &int_zero, &int_arg);
    if (io_res == ESCAPED) {
        goto do_escape_2;
    }

    exp_n = atof(text);

    set_help("POL_FILTER_EXP_J");
    io_res = io_meta_text(
        "Enter rectilinearity exponent j\n1",text,
        "re-enter rectilinearity exponent j",io_ver_int_gte_x_lte_y,&int_zero,
                                                                    &int_arg);

    if (io_res == ESCAPED) {
        goto do_escape_2;
    }

    exp_j = atof(text);
    exp_k = 1.0;
    if(win_len != ndat) {
        set_help("POL_FILTER_EXP_K");
        io_res = io_meta_text(
            "Enter eigenvector exponent k\n1",text,
            "re-enter eigenvector exponent k",io_ver_int_gte_x_lte_y,&int_zero,
                                                                    &int_arg);

        if (io_res == ESCAPED) {
            goto do_escape_2;
        }

        exp_k = atof(text);
    }

    note_on = TRUE;

    /* allocate storage                */
    /* which ==1 : 3 buffers:          */
    /* 2 eigenvector + rectilinearity */
    /* which ==2: 4 buffers:           */
    /* 3 eigenvector + rectilinearity */

    buffer = dt_alloc_head(which + 2);
    if(buffer == NULL) {
            men_message("Can't allocate buffers!",ACKN);
            goto do_escape_2;
    }
    index = cha_list1[1]-1;

    /* allocate storage for data trace */
    for(i=0; i<which+2; i++) {
       /* copy header information from N trace*/
       dt_copy_header(buffer[i],data[index]);
       if(dt_alloc_trace(buffer[i]) == ERROR) {
           men_message("ERROR: can't allocate memory for traces",ACKN);
           goto do_escape_1;
       }
    }
```

```
    ndat = *((int *)dt_head_access(data[index],RCD_NDAT,(void *)NULL));
    if((win_len > ndat) || (win_len == -1))
       win_len = ndat;

       if(ndat > Max_trlen2)
          men_message("WARNING! (any key to continue) \nNumber of points in current t
race [%d] exceeds\nwarning level set for current operation [%d].\n\nResult may be mean
ingless!!!",ACKN,ndat,Max_trlen2);

    switch (which) {
        case 1: /* 2 components */
        {
            temp1 = vector(0,win_len);
            temp2 = vector(0,win_len);
            cov_mat2 = matrix(1,2,1,2);
            eig_vect2 = matrix(1,2,1,2);
            eig_val2 =  vector(1,2);
            diag_val2 =  vector(1,2);
            offdiag_val2 =  vector(1,2);
            break;
        }
        case 2: /* 3 components  */
        {
            temp1 = vector(0,win_len);
            temp2 = vector(0,win_len);
            temp3 = vector(0,win_len);
            cov_mat = matrix(1,3,1,3);
            eig_vect = matrix(1,3,1,3);
            eig_val =  vector(1,3);
            diag_val =  vector(1,3);
            offdiag_val =  vector(1,3);
            break;
        }
        default:
            break;
    }

    /*
     calculate covariance, may want to put a note here if
     it is taking a long time.
    */

    note = men_note_init("        Working....             ",note);

    for (i=0; i<=(ndat-win_len); i=i+win_shift) {

        /* get window boundaries */
        n1 = i;
        n4 = i + win_len - 1;
        /* taper margins */
        if(frac >= 1.0) {
            n2 = n1;
            n3 = n4;
        }
        if(frac <= 0.0) {
            n2=(int)(((float)(win_len-1))/2.0)+n1;
            n3=n2+1;
        } else {
            n2=(int)(((float)(win_len-1))/2.0) - 0.5*frac*win_len + n1;
            if(n2<n1) n2 = n1;
            n3=(int)(((float)(win_len-1))/2.0) + 0.5*frac*win_len + n1;
            if(n3 > n4) n3 = n4;
```

```
        }

        /* perform tapering */
        b1 = dt_trace_access(data,cha_list1[1]-1,0);
        for (j=n1; j<n1+win_len; j++) {
            taper = spr_cosine(j,n1,n2,n3,n4);
            temp1[j-n1] = (*(b1+j))*taper;
        }

        b2 = dt_trace_access(data,cha_list1[2]-1,0);
        for (j=n1; j<n1+win_len; j++) {
            taper = spr_cosine(j,n1,n2,n3,n4);
            temp2[j-n1] = (*(b2+j))*taper;
        }


        switch (which) {
            case 1: { /* 2 components */
                cov_n_e = utl_cova(temp1,temp2,win_len);
                cov_n_n = utl_cova(temp1,temp1,win_len);
                cov_e_e = utl_cova(temp2,temp2,win_len);

                cov_mat2[1][1] = cov_n_n;
                cov_mat2[2][1] = cov_n_e;
                cov_mat2[1][2] = cov_n_e;
                cov_mat2[2][2] = cov_e_e;
                if(win_len == ndat) {
                    men_message("cov[1][1] %g cov[1][2] %g",ACKN,cov_mat2[1][1],
                                                            cov_mat2[1][2]);
                    men_message("cov[2][1] %g cov[2][2] %g",ACKN,cov_mat2[2][1],
                                                            cov_mat2[2][2]);
                }
                /* determine eigenvectors/values */

                balanc(cov_mat2,2);
                tred2(cov_mat2,2,eig_val2,offdiag_val2);
                tqli(eig_val2,offdiag_val2,2,cov_mat2);
                for(j=1;j <=2 ;j++)
                    for(k=1;k<=2;k++)
                        eig_vect2[j][k] = cov_mat2[j][k];
                eigsrt(eig_val2,eig_vect2,2);

                if(win_len == ndat) {
                    men_message("eig_val1 %g eig_val2 %g",ACKN,eig_val2[1],
                                                          eig_val2[2]);
                    men_message("eig[1][1] %g eig[1][2] %g",ACKN,
                                          eig_vect2[1][1],eig_vect2[1][2]);
                    men_message("eig[2][1] %g eig[2][2] %g",ACKN,
                                         eig_vect2[2][1], eig_vect2[2][2]);
                }
                /*
                 eigenvectors are now sorted
                 according to decreasing eigenvalue
                 largest eigenvector now in:
                 x = eig_vect2[1][1]
                 y = eig_vect2[2][1]
                 rectilinearity
                */
                if (eig_val2[1] != 0.0)
                    rct_lin = 1.0 - pow((eig_val2[2]/eig_val2[1]),exp_n);
                rct_lin = pow(rct_lin,exp_j);

                /* directivity */
```

```
            dir_1 = pow(eig_vect2[1][1],exp_k);
            dir_2 = pow(eig_vect2[2][1],exp_k);

            b0 = dt_trace_access(buffer,0,0);
            *(b0 + i + win_len/2) = dir_1;
            if((win_shift > 1) && (i > 0)) {
                x1 = *(b0 + i + win_len/2 - win_shift);
                x2 = *(b0 + i + win_len/2);

                for(j = 0;j<win_shift;j++) {
                    *(b0 + i + win_len/2 - win_shift + j) =
                        x1 + ((float)j/(float)win_shift)*(x2-x1);
                }
            }

            b1 = dt_trace_access(buffer,1,0);
            *(b1 + i + win_len/2) = dir_2;
            if((win_shift > 1) && (i > 0)) {
                x1 = *(b1 + i + win_len/2 - win_shift);
                x2 = *(b1 + i + win_len/2);

                for(j = 0;j<win_shift;j++) {
                    *(b1 + i + win_len/2 - win_shift + j) =
                        x1 + ((float)j/(float)win_shift)*(x2-x1);
                }
            }

            b2 = dt_trace_access(buffer,2,0);
            *(b2 + i + win_len/2) = rct_lin;
            if((win_shift > 1) && (i > 0)) {
                x1 = *(b2 + i + win_len/2 - win_shift);
                x2 = *(b2 + i + win_len/2);

                for(j = 0;j<win_shift;j++) {
                    *(b2 + i + win_len/2 - win_shift + j) =
                        x1 + ((float)j/(float)win_shift)*(x2-x1);
                }
            }
            break;
        }
        case 2: { /* 3 comp */

            for(j = n1; j < n1 + win_len; j++) {
                taper = spr_cosine(j,n1,n2,n3,n4);
                temp3[j-n1] =
                    (*dt_trace_access(&data[cha_list1[3]-1],0,j))*taper;
            }

            cov_z_n = utl_cova(temp1,temp2,win_len);
            cov_z_e = utl_cova(temp1,temp3,win_len);
            cov_z_z = utl_cova(temp1,temp1,win_len);
            cov_n_e = utl_cova(temp2,temp3,win_len);
            cov_n_n = utl_cova(temp2,temp2,win_len);
            cov_e_e = utl_cova(temp3,temp3,win_len);

            cov_mat[1][1] = cov_z_z;
            cov_mat[2][1] = cov_z_n;
            cov_mat[3][1] = cov_z_e;

            cov_mat[1][2] = cov_z_n;
            cov_mat[2][2] = cov_n_n;
            cov_mat[3][2] = cov_n_e;
```

```
cov_mat[1][3] = cov_z_e;
cov_mat[2][3] = cov_n_e;
cov_mat[3][3] = cov_e_e;

if(win_len == ndat) {
    men_message("cov[1][1] %g cov[1][2] %g cov[1][3] %g",ACKN,
                    cov_mat[1][1], cov_mat[1][2], cov_mat[1][3]);
    men_message("cov[2][1] %g cov[2][2] %g cov[2][3] %g",ACKN,
                    cov_mat[2][1], cov_mat[2][2], cov_mat[2][3]);
    men_message("cov[3][1] %g cov[3][2] %g cov[3][3] %g",ACKN,
                    cov_mat[3][1], cov_mat[3][2], cov_mat[3][3]);
}

/* determine eigenvectors/values */
balanc(cov_mat,3);
tred2(cov_mat,3,eig_val,offdiag_val);
tqli(eig_val,offdiag_val,3,cov_mat);
for (j=1; j<=3; j++)
    for (k=1; k<=3; k++)
        eig_vect[j][k] = cov_mat[j][k];
eigsrt(eig_val,eig_vect,3);

if(win_len == ndat) {
    men_message("eig_val1 %g eig_val2 %g eig_val3 %g",ACKN,
                            eig_val[1], eig_val[2], eig_val[3]);
    men_message("eig[1][1] %g eig[1][2] %g eig[1][3] %g",ACKN,
                eig_vect[1][1], eig_vect[1][2], eig_vect[1][3]);

    men_message("eig[2][1] %g eig[2][2] %g eig[2][3] %g",ACKN,
                eig_vect[2][1], eig_vect[2][2], eig_vect[2][3]);
    men_message("eig[3][1] %g eig[3][2] %g eig[3][3] %g",ACKN,
                eig_vect[3][1], eig_vect[3][2], eig_vect[3][3]);
}

/* eigenvectors are now sorted          */
/* according to decreasing eigenvalue */
/* largest eigenvector now in:          */
/* x = eig_vect[1][1]                   */
/* y = eig_vect[2][1]                   */
/* z = eig_vect[3][1]                   */
/* rectilinearity                       */
if (eig_val[1] != 0.0)
    rct_lin = 1.0 - pow((eig_val[2]/eig_val[1]),exp_n);
rct_lin = pow(rct_lin,exp_j);

/* directivity                          */
dir_1 = pow(eig_vect[1][1],exp_k);
dir_2 = pow(eig_vect[2][1],exp_k);
dir_3 = pow(eig_vect[3][1],exp_k);


b0 = dt_trace_access(buffer,0,0);
*(b0 + i + win_len/2) = dir_1;
if((win_shift > 1) && (i > 0)) {
    x1 = *(b0 + i + win_len/2 - win_shift);
    x2 = *(b0 + i + win_len/2);

    for (j=0; j<win_shift; j++) {
        *(b0 + i + win_len/2 - win_shift + j) =
            x1 + ((float)j/(float)win_shift)*(x2-x1);
    }
}
```

```
                b1 = dt_trace_access(buffer,1,0);
                *(b1 + i + win_len/2) = dir_2;
                if((win_shift > 1) && (i > 0)) {
                    x1 = *(b1 + i + win_len/2 - win_shift);
                    x2 = *(b1 + i + win_len/2);

                    for (j=0; j<win_shift; j++) {
                        *(b1 + i + win_len/2 - win_shift + j) =
                            x1 + ((float)j/(float)win_shift)*(x2-x1);
                    }
                }

                b2 = dt_trace_access(buffer,2,0);
                *(b2 + i + win_len/2) = dir_3;
                if((win_shift > 1) && (i > 0)) {
                    x1 = *(b2 + i + win_len/2 - win_shift);
                    x2 = *(b2 + i + win_len/2);

                    for (j=0; j<win_shift; j++) {
                        *(b2 + i + win_len/2 - win_shift + j) =
                            x1 + ((float)j/(float)win_shift)*(x2-x1);
                    }
                }

                b3 = dt_trace_access(buffer,3,0);
                *(b3 + i + win_len/2) = rct_lin;
                if((win_shift > 1) && (i > 0)) {
                    x1 = *(b3 + i + win_len/2 - win_shift);
                    x2 = *(b3 + i + win_len/2);

                    for (j=0; j<win_shift; j++) {
                        *(b3 + i + win_len/2 - win_shift + j) =
                            x1 + ((float)j/(float)win_shift)*(x2-x1);
                    }
                }
                break;
            }
        }

        men_note_write("Position % 5d out of % 5d....",note,i,ndat-win_len);

    } /* end of big loop */

    men_note_erase(note);
    note_on = FALSE;

    /* now all first calculations have been done. */

    switch (which) {
        case 1: { /* 2 comp */

            if(win_len == ndat) {
                men_message(
                    "Largest Eigenvector: alpha [deg]\nN: %g: %g\nE: %g: %g\nRectiline
arity: %g",ACKN,
                    eig_vect2[1][1],acos(eig_vect2[1][1])*ra,
                    eig_vect2[2][1],acos(eig_vect2[2][1])*ra,
                    rct_lin);
            }

            free_vector(temp1,0,win_len);
            free_vector(temp2,0,win_len);
            free_matrix(cov_mat2,1,2,1,2);
```

```
            free_matrix(eig_vect2,1,2,1,2);
            free_vector(eig_val2,1,2);
            free_vector(diag_val2,1,3);
            free_vector(offdiag_val2,1,3);

            break;
        }
        case 2: { /* 3 comp */

            if(win_len == ndat) {
                men_message(
                    "Largest Eigenvector: alpha [deg]\nZ: %g: %g\nN: %g: %g\nE: %g: %g
\nRectilinearity: %g",ACKN,
                    eig_vect[1][1],acos(eig_vect[1][1])*ra,
                    eig_vect[2][1],acos(eig_vect[2][1])*ra,
                    eig_vect[3][1],acos(eig_vect[3][1])*ra,
                    rct_lin);
            }
            free_vector(temp1,0,win_len);
            free_vector(temp2,0,win_len);
            free_vector(temp3,0,win_len);
            free_matrix(cov_mat,1,3,1,3);
            free_matrix(eig_vect,1,3,1,3);
            free_vector(eig_val,1,3);
            free_vector(diag_val,1,3);
            free_vector(offdiag_val,1,3);

            break;
        }
    }

    /*
     now get ready to plot.  If win_len is ndat, we don't plot.
     The end of the following if is all the way at the end.
    */

    if(win_len != ndat) {
        set_help("POL_FILTER_SMOOTH");
        io_res = io_meta_text(
            "Enter half window length for smoothing\n5",
            text,
            "re-enter half window length",
            io_ver_int_gte_x_lt_y,&int_zero,&ndat);

        if (io_res == ESCAPED) {
            goto do_escape_1;
        }

        half_width = atoi(text);
        if(half_width > 0) {
            note = men_note_init("Working....");
            men_note_write("Working....",note);
            note_on = TRUE;
            spr_smooth_time(buffer,0,half_width);
            spr_smooth_time(buffer,1,half_width);
            spr_smooth_time(buffer,2,half_width);
            if(which == 2) /* 3 components */
                spr_smooth_time(buffer,3,half_width);

            men_note_erase(note);
            note_on = FALSE;
        }
```

```
        set_help("POL_FILTER_SHOW");

        if((menu_poll = men_pu_get(" What To Show? ",
            what2show_Items,show_automatic,allow_cmd_mode)) == ESCAPED)
                goto do_escape_1;

        show = Menu_Path[1];

        switch (show) {
            case 1:
            case 2: { /* Eigenvector + */
                switch (which) {
                    case 1: { /* 2 components */
                        axis = *((AXIS *) dt_head_access(
                            buffer[0],Y_AXIS,(void *)NULL));
                        strcpy(axis.ax_gnlbl.text,"E-1");
                        dt_head_access(buffer[0],Y_AXIS,(void *)&axis);

                        axis = *((AXIS *) dt_head_access(buffer[1],Y_AXIS,
                                                         (void *)NULL));
                        strcpy(axis.ax_gnlbl.text,"E-2");
                        dt_head_access(buffer[1],Y_AXIS,(void *)&axis);

                        axis = *((AXIS *) dt_head_access(buffer[2],Y_AXIS,
                                                         (void *)NULL));
                        strcpy(axis.ax_gnlbl.text,"RCT");
                        dt_head_access(buffer[2],Y_AXIS,(void *)&axis);

                        break;
                    }
                    case 2: { /* 3 components */
                        axis = *((AXIS *) dt_head_access(buffer[0],Y_AXIS,
                                                         (void *)NULL));
                        strcpy(axis.ax_gnlbl.text,"E-1");
                        dt_head_access(buffer[0],Y_AXIS,(void *)&axis);

                        axis = *((AXIS *) dt_head_access(buffer[1],Y_AXIS,
                                                         (void *)NULL));
                        strcpy(axis.ax_gnlbl.text,"E-2");
                        dt_head_access(buffer[1],Y_AXIS,(void *)&axis);

                        axis = *((AXIS *) dt_head_access(buffer[2],Y_AXIS,
                                                         (void *)NULL));
                        strcpy(axis.ax_gnlbl.text,"E-3");
                        dt_head_access(buffer[2],Y_AXIS,(void *)&axis);

                        axis = *((AXIS *) dt_head_access(buffer[3],Y_AXIS,
                                                         (void *)NULL));
                        strcpy(axis.ax_gnlbl.text,"RCT");
                        dt_head_access(buffer[3],Y_AXIS,(void *)&axis);

                        break;
                    }
                }

                if (show == 2) {      /* Eigenvector + Largest Eigenv Proj. */
                    if(which == 1) { /* 2 components       */
                        tr1= dt_trace_access(data,cha_list1[1]-1,0);
                        b0 = dt_trace_access(buffer,0,0);
                        b2 = dt_trace_access(buffer,2,0);
                        for (i=0; i<ndat; i++)
                           *(b2+i) = (*(tr1+i))*(*(b0+i));
```

```
                tr2= dt_trace_access(data,cha_list1[2]-1,0);
                b1 = dt_trace_access(buffer,1,0);
                b2 = dt_trace_access(buffer,2,0);
                for (i=0; i<ndat; i++)
                   *(b2+i) += (*(tr2+i))*(*(b1+i));

            }
            else if(which == 2) { /* 3 components */
                tr1= dt_trace_access(data,cha_list1[1]-1,0);
                b0 = dt_trace_access(buffer,0,0);
                b3 = dt_trace_access(buffer,3,0);
                for (i=0; i<ndat; i++)
                   *(b3+i) = (*(tr1+i))*(*(b0+i));

                tr2= dt_trace_access(data,cha_list1[2]-1,0);
                b1 = dt_trace_access(buffer,1,0);
                b3 = dt_trace_access(buffer,3,0);
                for (i=0; i<ndat; i++)
                   *(b3+i) += (*(tr2+i))*(*(b1+i));

                tr3= dt_trace_access(data,cha_list1[3]-1,0);
                b2 = dt_trace_access(buffer,2,0);
                b3 = dt_trace_access(buffer,3,0);
                for (i=0; i<ndat; i++)
                   *(b3+i) += (*(tr3+i))*(*(b2+i));
            }

            if (which == 1) { /* 2 comp */
                axis = *((AXIS *) dt_head_access(buffer[2],Y_AXIS,
                                                 (void *)NULL));
                strcpy(axis.ax_gnlbl.text,"E-P");
                dt_head_access(buffer[2],Y_AXIS,(void *)&axis);
            }
            else if(which == 2) { /* 3 components */
                axis = *((AXIS *) dt_head_access(buffer[3],Y_AXIS,
                                                 (void *)NULL));
                strcpy(axis.ax_gnlbl.text,"E-P");
                dt_head_access(buffer[3],Y_AXIS,(void *)&axis);
            }
        }
        break;
    }
    case 3: { /* filtered traces */
        if (which == 1) { /* 2 components */
            axis = *((AXIS *) dt_head_access(buffer[0],Y_AXIS,
                                             (void *)NULL));
            strcpy(axis.ax_gnlbl.text,"F-1");
            dt_head_access(buffer[0],Y_AXIS,(void *)&axis);
            axis = *((AXIS *) dt_head_access(buffer[1],Y_AXIS,
                                             (void *)NULL));
            strcpy(axis.ax_gnlbl.text,"F-2");
            dt_head_access(buffer[1],Y_AXIS,(void *)&axis);
            axis = *((AXIS *) dt_head_access(buffer[2],Y_AXIS,
                                             (void *)NULL));
            strcpy(axis.ax_gnlbl.text,"RCT");
            dt_head_access(buffer[2],Y_AXIS,(void *)&axis);
        }
        else if(which == 2) { /* 3 components */
            axis = *((AXIS *) dt_head_access(buffer[0],Y_AXIS,
                                             (void *)NULL));
            strcpy(axis.ax_gnlbl.text,"F-1");
            dt_head_access(buffer[0],Y_AXIS,(void *)&axis);
            axis = *((AXIS *) dt_head_access(buffer[1],Y_AXIS,
```

```
                                                        (void *)NULL));
                strcpy(axis.ax_gnlbl.text,"F-2");
                dt_head_access(buffer[1],Y_AXIS,(void *)&axis);
                axis = *((AXIS *) dt_head_access(buffer[2],Y_AXIS,
                                                        (void *)NULL));
                strcpy(axis.ax_gnlbl.text,"F-3");
                dt_head_access(buffer[2],Y_AXIS,(void *)&axis);
                axis = *((AXIS *) dt_head_access(buffer[3],Y_AXIS,
                                                        (void *)NULL));
                strcpy(axis.ax_gnlbl.text,"RCT");
                dt_head_access(buffer[3],Y_AXIS,(void *)&axis);
            }

            if(which == 1) { /* 2 components */
               tr1= dt_trace_access(data,cha_list1[1]-1,0);
               b0 = dt_trace_access(buffer,0,0);
               b2 = dt_trace_access(buffer,2,0);
               for (i=0; i<ndat; i++)
                  *(b0+i) *= (*(tr1+i))*(*(b2+i));

               tr1= dt_trace_access(data,cha_list1[2]-1,0);
               b1 = dt_trace_access(buffer,1,0);
               b2 = dt_trace_access(buffer,2,0);
               for (i=0; i<ndat; i++)
                  *(b1+i) *= (*(tr1+i))*(*(b2+i));
            }
            else if (which == 2) { /* 3 components */
               tr1= dt_trace_access(data,cha_list1[1]-1,0);
               b0 = dt_trace_access(buffer,0,0);
               b3 = dt_trace_access(buffer,3,0);
               for (i=0; i<ndat; i++)
                  *(b0+i) *= (*(tr1+i))*(*(b3+i));

               tr1= dt_trace_access(data,cha_list1[2]-1,0);
               b1 = dt_trace_access(buffer,1,0);
               b3 = dt_trace_access(buffer,3,0);
               for (i=0; i<ndat; i++)
                  *(b1+i) *= (*(tr1+i))*(*(b3+i));

               tr1= dt_trace_access(data,cha_list1[3]-1,0);
               b2 = dt_trace_access(buffer,2,0);
               b3 = dt_trace_access(buffer,3,0);
               for (i=0; i<ndat; i++)
                  *(b2+i) *= (*(tr1+i))*(*(b3+i));
            }
        }
        break;
    }

    /* select only the corrected trace to be copied */
    if(which == 1) { /* 2 components */
        cha_list2[0] = 3;
        cha_list2[1] = 1;
        cha_list2[2] = 2;
        cha_list2[3] = 3;
        cha_list2[4] = -1;

        for (i=0; i<=2; i++)
            plt_set_view(buffer[i],ALL_X,ALL_Y,RESCALE_X,RESCALE_Y);
    }
    else if(which == 2) { /* 3 components */
        cha_list2[0] = 4;
        cha_list2[1] = 1;
```

```
        cha_list2[2] = 2;
        cha_list2[3] = 3;
        cha_list2[4] = 4;
        cha_list2[5] = -1;

        for (i=0; i<=3; i++)
            plt_set_view(buffer[i],ALL_X,ALL_Y,RESCALE_X,RESCALE_Y);
    }
    if ((show == 1) || (show == 2)) {

        /*
         set all eig vect to +/- 1 and rect lin to +1
         buffer[0] +/-
         buffer[1] +/-
         buffer[2] + 2comp +/- for 3comp
         buffer[3] + 3comp only
        */

        /* buffer[0] */
        view_temp = *((FRCT *)dt_head_access(buffer[0],PLT_VWP,
                                                    (void *)NULL));
        view_temp.ymax = 1.0;
        view_temp.ymin = -1.0;
        dt_head_access(buffer[0],PLT_VWP,(void *)&view_temp);

        /* buffer [1] */
        view_temp = *((FRCT *)dt_head_access(buffer[1],PLT_VWP,
                                                    (void *)NULL));
        view_temp.ymax = 1.0;
        view_temp.ymin = -1.0;
        dt_head_access(buffer[1],PLT_VWP,(void *)&view_temp);

        /* buffer[2] */
        view_temp = *((FRCT *)dt_head_access(buffer[2],PLT_VWP,
                                                    (void *)NULL));
        view_temp.ymax = 1.0;

        if (which == 2)
            view_temp.ymin = -1.0;

        dt_head_access(buffer[2],PLT_VWP,(void *)&view_temp);

        /* buffer[3] */
        if ((which == 2) && (show != 2)) { /* 3 comp */
            view_temp = *((FRCT *)dt_head_access(buffer[3],PLT_VWP,
                                                    (void *)NULL));
            view_temp.ymax = 1.0;
            dt_head_access(buffer[3],PLT_VWP,(void *)&view_temp);
       }
    }

    for (i=0; i<=cha_list2[0]-1; i++) {
            plt_do_full_scale (buffer[i],DO_XAXIS,RESCALE_X);
            plt_do_small_scale(buffer[i],DO_YAXIS,RESCALE_Y);
    }

    /* plot the result */

    plt_chalist(deskTop,buffer,-1,-1,cha_list2,NO_COLOR_BREAK);
    /* accept new/selected traces ? */
    data = dt_accept(data,buffer,cha_list1,cha_list2,-1,-1,&io_res,DO_ASK);
}
```

```
    /*
     now the environment has changed, we will need a plot all when
     we get back to the pitsa main routine.
    */
    Need_Plot_All = TRUE;

    dt_free_all(buffer);
    free((char *)cha_list1);
    *data_in = data;
    return(GOOD);

do_escape_1:
    dt_free_all(buffer);

do_escape_2:
    free((char *)cha_list1);
    *data_in = data;
    if (note_on)
        men_note_erase(note);

    return(ESCAPED);

}

/*-------------------------------------------------------------------------*/
/*
NAME
    matrix

SYNOPSIS
    float **matrix(nrl,nrh,ncl,nch)
    int nrl,nrh,ncl,nch;

DESCRIPTION
    Allocates memory for a matrix[nrl..nrh][ncl..nch]
*/
/*-------------------------------------------------------------------------*/

float **matrix(nrl,nrh,ncl,nch)
int nrl,nrh,ncl,nch;
{
        int i;
        float **m;
        m=(float **) malloc((unsigned) (nrh-nrl+1)*sizeof(float*));
        if (!m) nrerror("allocation failure 1 in matrix()");
        m -= nrl;

        for(i=nrl;i<=nrh;i++) {
                m[i]=(float *) malloc((unsigned) (nch-ncl+1)*sizeof(float));
                if (!m[i]) nrerror("allocation failure 2 in matrix()");
                m[i] -= ncl;
        }
        return m;
}

/*-------------------------------------------------------------------------*/
/*
NAME
    free_matrix

SYNOPSIS
    void free_matrix(m,nrl,nrh,ncl,nch)
    float **m;
```

```
    int nrl,nrh,ncl,nch;

DESCRIPTION
    free the memory allocated by matrix

*/
/*-------------------------------------------------------------------------*/

void free_matrix(m,nrl,nrh,ncl,nch)
float **m;
int nrl,nrh,ncl,nch;
{
        int i;

        for(i=nrh;i>=nrl;i--) free((char*) (m[i]+ncl));
        free((char*) (m+nrl));
}
```

File: readme (8/17/92)

I. Title Page.

U. S. DEPARTMENT OF THE INTERIOR

U. S. GEOLOGICAL SURVEY

DESCRIPTIONS OF SEISMIC ARRAY COMPONENTS:

PART 3. SOFTWARE MODULES FOR DATA CONVERSIONS

Compiled by

W. H. K. Lee

MS 977, 345 Middlefield Road

Menlo Park, CA 94025

Open-File Report 92-598-B

August, 1992

II. Disclaimer.

This report is preliminary and has not been reviewed for conformity with U. S. Geological Survey editorial standards. Any use of trade, firm, or product names is for descriptive purposes only and does not imply endorsement by the U. S. Government.

Although these programs have been used by the U.S. Geological Survey, no warranty, expressed or implied, is made by the USGS as to the accuracy and functioning of the programs and related program material, nor shall the fact of distribution constitute any such warranty, and no responsibility is assumed by the USGS in connection therewith.

III. Date of Latest Revision.

August 17, 1992.

IV. Text.

INTRODUCTION

In the summer of 1990, funding was available to design and implement two portable seismic arrays for the volcano program. The approach was based on Lee et al. (1989). Several contracts were awarded to commercial companies to design and implement various components needed to build the portable arrays. The purpose of this report to present the software modules for data conversion purposes -- DR2ST/ST2DR, GRM2ST/ST2GRM, PCEQ2ST/PCQ2ST, SEGY2ST, IO_RSDS, and IO_WRSDS, and two miscellaneous modules -- PLT_PMTN, and SPR_PLFL -- in detail as submitted by the contractors. Source code on PC-DOS/MS-DOS diskette for this report is presented in U. S. Geological Survey Open-File Report 92-598-B.

DR2ST/ST2DR

DR2ST/ST2DR are two computer programs for converting data from DR-100 (GEOS) data format (Borcherdt et. al.) to the SUDS data format (Ward, 1989; Banfill, 1992) and vice versa, respectively.

GRM2ST/ST2GRM

GRM2ST/ST2GRM are two computer programs for converting data from CUSP's GRM data format (Allan Walter, personal communication, 1992) to the SUDS data format (Ward, 1989; Banfill, 1992) and vice versa, respectively.

PCEQ2ST/PCQ2ST

PCEQ2ST/PCQ2ST are two computer programs for converting data from the PCEQ data format (Valdes, 1989) and from the PC-Quake data format (Tottingham et al., 1989) to the SUDS data format (Ward, 1989; Banfill, 1992), respectively.

SEGY2ST

SEGY2ST is a computer program for converting data from the SEGY data format (Barry et al., 1975) to the SUDS data format (Ward, 1989; Banfill, 1992).

MISC

The MISC directory contains four source code modules that may be added to the PITSA program package (Scherbaum and Johnson, 1992) in order to (1) read and write data files in the SUDS format (Ward, 1989; Banfill, 1992): IO_RDSDS.C and IO_WRSDS.C, (2) plot particle motion: PLT_PMTN.C, and (3) perform polarization filtering: SPR_PLFL.C.

REFERENCES

Banfill, R., (1992). SUDS: Seismic Unified Data System, version 1.31, Small Systems Support, Big Water, Utah.

Barry, K. M., D. A. Cavers, and C. W. Kneale, (1975). Recommended standards for digital tape format, Geophysics, 40, 344-352.

Borcherdt R.D., J. B. Fletcher, E. G. Jensen, G. L. Maxwell, J. R. Vanschaack, (1985). A general earthquake-observation system (GEOS) Bull. Seism. Soc. Am., 75, 1783-1825.

Lee, W. H. K., D. M. Tottingham, and J. O. Ellis (1989). Design and implementation of a PC-based seismic data acquisition, processing, and analysis system, IASPEI Software Library, 1, 21-46.

Scherbaum, F., and J. Johnson, (1992). ``Programmable Interactive Toolbox for Seismological Analysis (PITSA)'', IASPEI Software Library, 5, in preparation.

Tottingham, D. M., W. H. K. Lee, and J. A. Rogers, (1989). User manual for MDETECT, IASPEI Software Library, 1, 49-88.

Valdes, C. M., (1989). User manual for PCEQ, IASPEI Software Library, 1, 175-201.

Ward, P. L. (1989). SUDS: Seismic Unified Data System, U. S. Geol. Surv. Open-file Report 89-188.

V. Diskette Contents.

This diskette contains seven directories:

(1) DR2ST -- source code for the DR2ST program written by Small Systems Support.

(2) ST2DR -- source code for the ST2DR program written by Small Systems Support.

(3) GRM2ST -- source code for the GRM2ST program written by Small Systems Support.

(4) ST2GRM -- source code for the ST2GRM program written by Small Systems Support.

(5) PCEQ2ST -- source code for the PCEQ2ST program written by Small Systems Support.

(6) PCQ2ST -- source code for the PCQ2ST program written by Small Systems Support.

(7) MISC -- source code for the IO_RSDS.C, IO_WRSDS.C, PLT_PMTN.C, and SPR_PLFL.C modules written by F. Scherbaum.

Open-File Report 92-598A and -B. Description of Seismic Array Components: Part 3. Software Modules for Data Conversion. 1992. 106 p. and one 3.5-in. diskette.

This report describes software modules for data conversion purposes – – DR2ST/ST2DR, GRM2ST/ST2GRM, PCEQ2ST/PCQ2ST, SEGY2ST, IO_RSDS, and IO_WRSDS, and two miscellaneous modules: PLT_PMTN, and SPR_PLFL – –as submitted by the contractors. Open-File Report 92-598A (106 p.) contains the software documentation (including listings of source code). Open-File Report 92598B is a 3.5-in. diskette containing the source code for the above mentioned software modules.

Requirements for part B: IBM 386 or 486 PC or compatible; minimum 1 MB RAM; math coprocessor; VGA graphics board and monitor; minimum 40 MB hard disk; 3.5-inch floppy disk drive; PC or MS DOS 4.01 or later; and Microsoft C Compiler.